Mark Pilgrim

Ponořme se do HTML5

Edice CZ.NIC

Mark Pilgrim
PONOŘME SE DO HTML5

Vydavatel:
CZ.NIC, z. s. p. o.
Milešovská 5, 130 00 Praha 3
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2014
Kniha vyšla jako 10. publikace v Edici CZ.NIC.
ISBN 978-80-905802-6-8

CZ.NIC | SPRÁVCE DOMÉNY CZ

ISBN 978-80-905802-6-8

— Mark Pilgrim

Ponořme se do HTML5

— Edice CZ.NIC

Předmluva vydavatele

— Předmluva vydavatele

Vážení čtenáři,

po sérii knížek zaměřených převážně na běžného uživatele Internetu se v Edici CZ.NIC opět vracíme k tématu, které bude blízké zejména vývojářům internetových aplikací. A nevybrali jsme si pro tento návrat nic menšího než jazyk HTML, jeden ze základních stavebních kamenů konceptu WWW, který byl bezpochyby hlavním strůjcem rozmachu Internetu na přelomu tisíciletí. Zdá se, že po krátkém období nejistoty, jakým směrem se vývoj HTML bude ubírat, chytá tento jazyk druhý dech. Na podzim loňského roku došlo ke standardizaci HTML5 a o této nejnovější verzi jazyka pojednává právě tato kniha. V roce, kdy HTML slaví 25. výročí své existence, je již zřejmé, že nová verze bude akceptována hlavními hráči a postupně vytlačí některé proprietární technologie, které v mezičase zaplnily uvolněný prostor.

Přestože originál knihy Ponořme se do HTML5 vznikl již v roce 2011, věříme, že poskytne čtenáři dostatečný vhled jak do historie tohoto jazyka, tak do všech jeho podstatných vlastností. Zárukou je pro nás osobnost autora Marka Pilgrima, který je autorem i jiné knihy v naší edici, a to Ponořme se do Pythonu 3. Tato kniha je v Edici CZ.NIC jednou z nejoblíbenějších, a to zejména díky čtivému stylu, kterým autor čtenáře doslova vtáhne do popisovaného tématu. Autor, který v roce 2011 trochu tajemně zmizel z veřejného internetového prostoru, neustoupil ze svého originálního stylu ani v této knize.

Přeji Vám pěkné čtení.

Jaromír Talíř, CZ.NIC
Praha, 23. června 2015

— Předmluva vydavatele

Obsah

— Obsah

Obsah

Pět věcí, které byste měli vědět o HTML5

1. Není to jedna velká věc

Možná se ptáte: „Jak můžu začít používat HTML5, pokud ho nepodporují starší prohlížeče?“ Ale ta otázka je sama o sobě zavádějící. HTML5 není jedna veličina, ale soubor jednotlivých prvků. Nemůžete detekovat „podporu HTML5“, protože to by nedávalo žádný smysl. Můžete ale detekovat podporu jednotlivých prvků, jako jsou plátno, video nebo geolokace.

HTML si lze představovat jako tagy a špičaté závorky. Ty jsou jistě jeho důležitou součástí, ale není to všechno. Specifikace HTML5 také určují, jak tyto špičaté závorky fungují s JavaScriptem prostřednictvím Document Object Model (DOM). Není to tak, že by HTML5 pouze definoval tag `<video>`, ale v DOMu se nachází také odpovídající DOM API pro videoobjekty. Toto API můžete použít pro detekci podpory jednotlivých formátů videa, přehrání videa, jeho pozastavení, vypnutí zvuku, zjištění, kolik procent videa se stáhlo, a všeho dalšího, co potřebujete, abyste uživateli spolu s tagem `<video>` poskytli dostatečný komfort.

Kapitola 2 a Příloha A vás naučí jak správně detekovat podporu pro každý z nových prvků HTML5.

2. Nemusíte nic vyhazovat

Ať už ho milujete, nebo nenávidíte, nemůžete popřít, že HTML4 je nejúspěšnější značkovací formát všech dob. Na tomto úspěchu staví HTML5. Nemusíte vyhazovat už jednou napsané kódy, ani se nemusíte znova učit věci, které už znáte. Pokud vaše webové aplikace včera běžela v HTML4, bude běžet také dnes v HTML5. Opravdu.

Ovšem pokud chcete své webové aplikace *vylepšit*, jste tady správně. Uveďme si jeden příklad: HTML5 podporuje všechny formulářové prvky z HTML4, ale také obsahuje nové vstupní prvky. Mezi ně patří doplňky, na které jsme čekali už dlouho, jako posuvníky nebo prvky pro výběr data, zatímco jiné jsou méně nápadné. Například vstupní prvek typu `email` vypadá jako obyčejné textové pole, ale mobilní prohlížeče upraví klávesnici na displeji tak, aby se daly snadněji vkládat e-mailové adresy. Starší prohlížeče, které nepodporují vstupní prvek typu `email`, s tímto prvkem budou zacházet jako s klasickým textovým polem a formulář bude fungovat i nadále bez změny kódu nebo skriptování. To znamená, že můžete začít vylepšovat svůj web už dnes, i když budou někteří vaši návštěvníci stále ještě používat IE6.

Všechny šťavnaté detaily o formulářích HTML se dozvíte v kapitole 9.

3. Začít je snadné

„Upgradovat“ na HTML5 jde i pouhou změnou *doctype*. Doctype by měl být přítomen na prvním řádku kódu každé stránky HTML. Předchozí verze HTML definovaly velké množství doctype a vybrat si ten správný může být problematické. V HTML5 je pouze jediný doctype:

```
<!DOCTYPE html>
```

Upgradování na doctype HTML nerozbije váš stávající kód, protože zastaralé prvky, které předtím definovalo HTML 4, se budou zobrazovat i nadále v HTML5. Ale umožní vám to použít – a validovat – nové sémantické prvky, jako jsou `<article>`, `<section>`, `<header>` a `<footer>`. Všechno o těchto nových prvcích se dozvíte v kapitole 3.

4. Už to funguje

Ať už chcete něco vykreslit na canvasu, přehrát video, navrhnout lepší formuláře nebo vytvořit webové aplikace, které fungují i offline, zjistíte, že HTML5 má už teď velmi dobrou podporu. Firefox, Safari, Chrome, Opera a mobilní prohlížeče už teď podporují plátno (kapitola 4), video (kapitola 5) a geolokaci (kapitola 6), místní úložiště (kapitola 7) a další. Google již nyní podporuje anotaci pomocí mikrodat (kapitola 10). Dokonce Microsoft, který těžko může někdo osočovat z průkopnictví v oblasti podpory standardů, v Internet Exploreru 9 podporuje většinu prvků HTML5.

Každá kapitola této knihy obsahuje až příliš dobře známé tabulky kompatibility prohlížečů. Ale co je podstatnější, v každé kapitole najdete také otevřenou diskuzi o možnostech, které máte, když potřebujete podporu pro starší prohlížeče. Prvky HTML5 jako geolokace (kapitola 6) a video (kapitola 5) se poprvé objevily v podobě prohlížečových zásuvných modulů typu Gears nebo Flash. Další funkce, jako je plátno (kapitola 4), lze plně emulovat v JavaScriptu. Tato kniha vás naučí jak využívat nové funkce moderních prohlížečů, aniž byste zapomínali na prohlížeče starší.

5. Už tu s námi zůstane

Tim Berners-Lee vynalezl web na začátku devadesátých let. Později založil sdružení W3C, aby fungovalo jako strážce internetových standardů, což tato organizace dělá už déle než 15 let. V červenci roku 2009 sdružení W3C o budoucnosti internetových standardů řeklo následující:

> Ředitel dnes oznámil, že až pracovní skupina XHTML 2 skončí podle plánu na konci roku 2009, nebude obnovena. Tímto krokem a navýšením zdrojů pro pracovní skupinu

HTML chce W3C urychlit vývoj HTML5 a vyjasnit postoj organizace ohledně budoucnosti HTML.

HTML5 tu s námi už zůstane. Tak pojďme na to.

1. Jak jsme se sem dostali

1. Jak jsme se sem dostali – 21

1.1 Začínáme

Nedávno jsem narazil na něco, co řekl jeden z vývojářů Mozilly ohledně napětí, které nutně vzniká při tvoření standardů:

> Implementace a specifikace spolu musejí jít ruku v ruce. Nechcete, aby implementace proběhla dřív, než jsou hotové specifikace, protože lidé začnou být závislí na detailech implementací, což omezuje specifikace. Také ovšem nechcete, aby byly specifikace hotové dřív, než vzniknou implementace a s nimi spojené zkušenosti, protože potřebujete zpětnou vazbu. Proto vzniká nevyhnutelné napětí, ale my se s ním zkrátka musíme nějak poprat.

Na tento citát myslete, když vám budu vysvětlovat, jak vzniklo HTML5.

1.2 Typy MIME

Tato kniha pojednává o HTML5, nikoliv o předchozích verzích HTML, ani o žádné verzi XHTML. Ale abychom pochopili historii HTML5 a důvody pro jeho vytvoření, musíme nejprve pochopit několik technických detailů. Konkrétně jsou to typy MIME.

Pokaždé, když váš webový prohlížeč vyšle požadavek na stránku předtím, než pošle skutečný zdrojový kód, posílá webový server tzv. hlavičku. Tyto hlavičky jsou obvykle neviditelné, ale existují nástroje pro tvorbu webových stránek, které je zviditelňují, pokud si to přejete. Ale tyto hlavičky jsou důležité, protože říkají vašemu prohlížeči, jak má interpretovat zdrojový kód stránky, který bude následovat. Nejdůležitější hlavička se jmenuje `Content-Type` a vypadá takto:

```
Content-Type: text/html
```

„`text/html`“ se nazývá „typ obsahu“ nebo „typ MIME“ stránky. Tato hlavička je jedinou věcí, která určuje, jak vypadá konkrétní zdroj a jak by měl být interpretován. Obrázky mají své vlastní typy MIME (`image/jpeg` pro formát JPEG, `image/png` pro formát PNG apod.). Soubory JavaScript mají své vlastní typy MIME stejně jako kaskádové styly. Své vlastní typy MIME má zkrátka úplně všechno. Celý Internet funguje na základě typů MIME.

Ve skutečnosti je to samozřejmě mnohem komplikovanější. První generace webových serverů (mluvím teď o serverech z roku 1993) nevysílala hlavičku `Content-Type`, protože ta tehdy ještě neexistovala (byla vymyšlena až v roce 1994). Z důvodů týkajících se kompatibility, které se datují až do roku 1993, budou některé oblíbené prohlížeče za určitých podmínek hlavičku `Content-Type` ignorovat (říká se tomu „content sniffing“, tedy „očichání obsahu“). Ale obvykle platí pravidlo, že všechno, co jste si kdy na webu prohlíželi – HTML stránky, obrázky, skripty,

videa, dokumenty PDF, všechno s URL adresou – vám bylo naservírováno s konkrétním typem MIME v hlavičce `Content-Type`.

Zapište si to za uši, později se k tomu vrátíme.

1.3 Delší odbočka na téma tvoření standardů

Proč vlastně máme prvek `<img>`? To asi není otázka, kterou byste slyšeli každý den. Je jasné, že ho *někdo* musel vytvořit. Sám od sebe asi nevznikl. Každý prvek, každý atribut, každá funkce HTML je výtvorem někoho, kdo rozhodl, jak budou fungovat, a všechno to zapsal. Tito lidé nejsou bohové, ani nejsou dokonalí. Jsou to jen lidé. Chytří lidé, to ano, ale stále jen lidé.

Jedna ze skvělých věcí na standardech, které se vyvinuly „v otevřeném prostoru" je, že se můžete vrátit v čase a na všechny takové otázky odpovědět. Diskuse se nacházejí v e-mailových konferencích, které jsou obvykle archivovány a dají se veřejně vyhledat. Proto jsem se rozhodl dát si trochu „e-mailové archeologie", abych našel odpověď na otázku „Proč máme prvek `<img>`?" Musel jsem se vrátit v čase do doby ještě předtím, než vznikla organizace nazvaná World Wide Web Consortium (W3C). Vrátil jsem se do internetového pravěku, kde jste mohli spočítat počet webových serverů na prstech obou rukou a možná ještě tak jedné nohy.

25. února roku 1993 Marc Andreessen napsal:

> Rád bych navrhl nový volitelný tag HTML:
>
> IMG
>
> Povinným parametrem je `SRC="url"`.
>
> Tímto tagem se pojmenovává bitmapový nebo pixmapový soubor, který si pak prohlížeč pokusí ze sítě stáhnout, interpretovat ho jako obrázek a vložit ho do textu v místě výskytu tagu.
>
> Příklad:
>
> ```
> <IMG SRC="file://foobar.com/foo/bar/blargh.xbm">
> ```
>
> (Zavírací tag není potřeba, jedná se o samostatný tag.)
>
> Tento tag může být zabudován do odkazu jako cokoliv jiného. V takovém případě se stane ikonou, která je citlivá na aktivaci stejně jako klasický textový odkaz.

> Prohlížeče by měly mít volnost ohledně jimi podporovaných formátů obrázků. Bylo by například dobré, aby podporovaly Xbm a Xpm. Pokud prohlížeč nedokáže daný formát přečíst, může s ním místo toho udělat cokoliv chce (v X Mosaic vyskočí jako zástupný symbol výchozí bitmapa).
>
> Tohle je požadovaná funkce pro X Mosaic. Funguje nám to a budeme to používat minimálně interně. Určitě se nebudu bránit nápadům, jak by se to mělo zvládnout v HTML. Pokud máte nějaký lepší nápad než to, co tu teď předkládám, dejte mi prosím vědět. Vím, že je to neurčité, pokud jde o formát obrázku, ale nevidím jinou alternativu než si říct „nechte prohlížeč dělat, co umí" a počkat si na to, až někdo přijde s dokonalým řešením (MIME, snad jednou v budoucnu).

Xbm a Xpm byly oblíbené grafické formáty používané v operačním systému Unix.

„Mosaic" byl jeden z nejstarších webových prohlížečů („X Mosaic" byla jeho verze, která běžela pod Unixem). Marc Andreessen napsal tuto zprávu počátkem roku 1993, tedy ještě předtím, než založil firmu, která ho proslavila – „Mosaic Communications Corporation" – a začal pracovat na vlajkové lodi této společnosti, „Mosaic Netscape" (asi je znáte lépe pod jejich pozdějšími názvy „Netscape Corporation" a „Netscape Navigator").

„MIME, snad jednou v budoucnu" je odkazem na vyjednávání obsahu (content negotiation), funkci HTTP, kde klient (např. webový prohlížeč) sdělí serveru (např. webovému serveru), jaké typy zdrojů podporuje (např. `image/jpeg`), takže server může vrátit informaci ve formátu upřednostňovaném klientem. Původní protokol HTTP, tak jak byl definován v roce 1991 (jehož jediná verze byla realizována v únoru 1993), neumožňoval klientům žádným způsobem serverům sdělit, jaké grafické formáty podporují, a proto čelil Marc Andreessen takovému designovému dilematu.

O několik hodin později Tony Johnson odpověděl:

> Mám něco hodně podobného v Midas 2.0 (ten používáme u nás v SLAC a chystáme se ho zveřejnit co nejdříve), ovšem všechny názvy se odlišují, a má to parametr navíc: `NAME="name"`. A funguje to téměř úplně stejně jako tebou navržený tag `IMG`, např.:
>
> ```
> <ICON name="NoEntry" href="http://note/foo/bar/NoEntry.xbm">
> ```
>
> Smyslem parametru name (název) je umožnit prohlížeči, aby měl sadu „zabudovaných" obrázků. Pokud se název shoduje se „zabudovaným" obrázkem, bude použit tento obrázek, aniž by bylo potřeba obrázek brát z vnějšího zdroje. Název by také mohl „line mode" (textovým) prohlížečům napovědět, jaký symbol umístit namísto obrázku.

> Je mi celkem jedno, jak se budou parametry a tagy jmenovat, ale bylo by rozumné, kdybychom používali totéž. Nejsem moc fanda zkratek, takže proč nepoužít `IMAGE=` a `SOURCE=`? Více se mi líbí `ICON`, protože to naznačuje, že by měl `IMAGE` být spíše menší, ale možná je `ICON` už příliš významově zatížené slovo?

Midas byl další ze starých webových prohlížečů, současník „X Mosaic“. Tento multiplatformní prohlížeč běžel pod Unixem a VMS. SLAC je zkratka pro Stanford Linear Accelerator Center, nyní SLAC National Accelerator Laboratory, centrum, které provozovalo první webový server ve Spojených státech (a zároveň první webový server mimo Evropu). Když Tony Johnson psal tuto zprávu, byl SLAC na poli WWW už zkušeným harcovníkem, jelikož na svém webovém serveru hostoval pět stránek po celých 441 dní.

Tony pokračoval:

> Když už mluvíme o nových tazích, mám další, v některých ohledech podobný tag, pro který bych chtěl podporu v Midas 2.0. V zásadě jde o:
>
> ```
> <INCLUDE HREF="...">
> ```
>
> Záměrem je, aby byl druhý dokument obsažen v prvním dokumentu v místě, kde se vyskytuje tag. V zásadě může být odkazovaným dokumentem cokoliv, ale hlavním účelem je umožnit, aby byly do dokumentů vkládány obrázky (v tomto případě libovolně velké). S příchodem HTTP2 by se pak o formátu vkládaného dokumentu zvlášť jednalo.

„HTTP2“ je odkazem na Základní protokol HTTP, jak byl definován v roce 1992. V tomto okamžiku, tj. na počátku roku 1993, byl stále z větší části nezrealizován. Návrh známý jako „HTTP2“ byl dále vyvíjen a nakonec byl standardizován jako „HTTP 1.0“ (ovšem až za další tři roky). HTTP 1.0 už zahrnoval hlavičky požadavku pro vyjednání obsahu neboli „MIME, snad jednou v budoucnu.“

Tony pokračoval:

> Jako alternativa mě napadlo:
>
> ```
> <A HREF="..." INCLUDE>Zobrazit fotografii</A>
> ```
>
> Příliš se mi nelíbí přidávat další funkce do tagu `<A>`, ale cílem je zde zachovat kompatibilitu i pro prohlížeče, které si neporadí s parametrem `INCLUDE`. Záměrem je, aby prohlížeče, které rozumí `INCLUDE`, nahradily text odkazu (v tomto případě „Zobrazit fotografii“) vloženým dokumentem (obrázkem), zatímco starší nebo „hloupější“ prohlížeče budou tag `INCLUDE` zcela ignorovat.

Tento návrh nebyl nikdy uskutečněn, ale myšlenka nahrazení obrázku textem, pokud obrázek chybí, je důležitou technikou pro zvýšení přístupnosti, která v Marcově původním návrhu `<IMG>` chyběla. O několik let později byla tato funkce přidána jako atribut `<img alt>`, což Nescape okamžitě pokazil tím, že s ním omylem zacházel jako s popiskem.

Za několik hodin poté, co Tony odeslal svou zprávu, odpověděl Tim Berners-Lee:

> Představoval jsem si, že obrázky budou znázorněny jako:
>
> ```
> <a name="fig1" href="fghjkdfghj" REL="EMBED, PRESENT">Obrázek</a>
> ```
>
> kde relační hodnoty znamenají:
>
> ```
> EMBED     Vložte zde při předložení
> PRESENT   Předložte vždy, když je předložen zdrojový dokument
> ```
>
> Vezměte na vědomí, že můžete použít různé kombinace těchto hodnot, a pokud prohlížeč nepodporuje žádný z nich, kód se nerozbije.
>
> Chápu, že když se to použije jako metoda pro volitelné ikony, znamená to vkládání odkazů do sebe. Hmm. Ale speciální tag jsem nechtěl.

Tento návrh nebyl nikdy zrealizován, ale s atributem `rel` se můžeme setkat dodnes.

Jim Davis dodal:

> Bylo by fajn, kdyby se dal nějak specifikovat typ obsahu, např.:
>
> ```
> <IMG HREF="http://nsa.gov/pub/sounds/gorby.au" CONTENT-TYPE=audio/
> basic>
> ```
>
> Ale jsem zcela ochoten se podřídit požadavku, abych určoval typ obsahu příponou souboru.

Ani tento návrh nebyl nikdy zrealizován, ale Netscape později přidal podporu vkládání mediálních objektů pomocí prvku `<embed>`.

Jay C. Weber se zeptal:

> I když mému žebříčku typů médií, u kterých bych chtěl, aby je podporoval WWW prohlížeč, vévodí obrázky, nemyslím si, že bychom měli vymýšlet speciální techniky pro

> každé médium zvlášť. Co se stalo s nadšením pro používání mechanismu založeného na typech MIME?

Marc Andreessen odpověděl:

> Toto není náhrada za blížící se používání MIME jako standardního dokumentového mechanismu. Jedná se o nezbytnou a jednoduchou implementaci funkce, která je potřebná nezávisle na MIME.

Jay C. Weber odpověděl:

> Pojďme na chvíli zapomenout na MIME, aby nás to nepletlo. Moje námitka byla vedena proti diskuzi na téma „jak budeme podporovat vložené obrázky" spíše než na téma „jak budeme podporovat vložené objekty v různých médiích".
>
> Jinak příští týden někdo přijde s návrhem na nový tag `<AUD SRC="file://foobar.com/foo/bar/blargh.snd">` pro audio.
>
> Používat něco, co se dá uplatnit obecněji, by neměl být takový problém.

Když se ohlédneme nazpět, vypadá to, že Jayovy obavy byly zcela opodstatněné. Trvalo to o trochu víc než týden, ale HTML5 konečně přidalo nové prvky `<video>` a `<audio>`.

V odpovědi na Jayovu původní zprávu Dave Raggett řekl:

> Přesně tak! Chci uvážit celou řadu možných typů obrázků/čárové grafiky, spolu s možností vyjednávání formátu. Timova poznámka o podpoře oblastí uvnitř obrázků s možností kliknutí je také důležitá.

O něco později v roce 1993 Dave Raggett navrhl HTML+ jako evoluci standardu HTML. Tento návrh nebyl nikdy realizován a byl překonán HTML 2.0. HTML 2.0 bylo „retrospektivní", což znamená, že formalizovalo prvky, které se už v té době běžně používaly. „Tato specifikace spojuje, vyjasňuje a formalizuje řadu prvků, které zhruba odpovídají schopnostem HTML, jak se běžně používalo do června 1994."

Dave později napsal HTML 3.0 založené na jeho předchozím návrhu HTML+. HTML 3.0 nebylo využito nikde kromě Areny, vlastního prohlížeče W3C, a bylo překonáno HTML 3.2, které bylo opět „retrospektivní". „HTML 3.2 přidává prvky s širokou škálou využití, jako jsou tabulky, aplety a tok textu kolem obrázku, a zároveň poskytuje plnou zpětnou kompatibilitu s existujícím standardem HTML 2.0."

Později se Dave stal spoluautorem HTML 4.0, vyvinul HTML Tidy a podílel se i na XHTML, XForms, MathML a dalších moderních specifikacích W3C.

Vraťme se do roku 1993, kdy Mark odpověděl Daveovi:

> Možná bychom se vlastně měli zamyslet nad univerzálním procedurálním grafickým jazykem, ve kterém bychom mohli vkládat libovolné hypertextové odkazy připojené k ikonám, obrázkům nebo k textu, k čemukoliv. Má někdo představu, co v tomhle ohledu dokáže Intermedia?

Intermedia byl hypertextový projekt Brownovy univerzity. Byl vyvíjen mezi lety 1985 a 1991 a běžel pod A/UX, což byl operační systém podobný Unixu určený pro starší typy počítačů Macintosh.

Myšlenka „univerzálního procedurálního grafického jazyka“ se nakonec uchytila. Moderní prohlížeče podporují jak SVG (značkovací jazyk s vloženými skripty), tak `<canvas>` (procedurální grafický API s přímým vykonáváním instrukcí), ovšem `<canvas>` se původně objevil jako proprietární rozšíření, než se stal „retrospektivním“ díky WHATWG.

Bill Janssen odpověděl:

> Další systémy, jimiž se můžeme inspirovat, které obsahují tento (celkem cenný) koncept, jsou Andrew a Slate. Andrew je postavený pomocí _insetů_, z nichž každý má nějaký zajímavý typ, jako je text, bitmapa, kresba, animace, zpráva, tabulka atd. Funguje zde koncept arbitrárního rekurzivního vkládání, takže jakýkoliv inset jakéhokoliv druhu může být vložen do jakéhokoliv jiného druhu, který podporuje vkládání. Například může být inset vložen do libovolného místa textu v textovém widgetu nebo do jakékoliv obdélníkové oblasti v kreslícím widgetu či do libovolné buňky v tabulce.

„Andrew“ odkazuje na Andrew User Interface System (v té době ovšem známý prostě jenom jako Andrew Project).

Mezitím dostal Thomas Fine jiný nápad:

> Podle mě je nejlepším způsobem, jak zacházet s obrázky ve WWW, používání MIME. Jsem si jistý, že postscript je už v tuto chvíli v MIME podporovaný podtyp, a právě on si dokáže velmi dobře poradit s mícháním textu a grafiky.
>
> Namítáte, že se na něj nedá klikat? Ano, máte pravdu. Myslím, že s tímhle si už teď dokáže poradit display poscript. A i kdyby náhodou ne, tak doplnění standardního postscriptu je jednoduchá záležitost. Definujte příkaz s odkazem, který specifikuje URL, a použije

> současnou cestu jako uzavřenou oblast pro tlačítko. Jelikož to postscript tak dobře umí s cestami, budou libovolné tvary tlačítka brnkačka.

Display Postscript byla zobrazovací technologie pro obrazovkové servery, na jejímž vývoji se podílely firmy Adobe a NeXT.

Tento návrh nebyl nikdy zrealizován, ale myšlenka, že nejlepším způsobem jak vylepšit HTML je nahradit ho něčím úplně jiným, se stále čas od času objevuje.

Tim Berners-Lee, 2. března 1993:

> HTTP2 umožňuje dokumentu, aby obsahoval jakýkoliv typ, o kterém uživatel řekl, že ho zvládne, a nikoliv pouze registrované typy MIME. Proto můžeme experimentovat. Ano, myslím, že existuje dost argumentů pro spojení postscriptu s hypertextem. Nevím, zda stačí Display Postcsript. Vím, že se Adobe snaží vytvořit svoje vlastní na postcriptu založené „PDF", které bude čitelné v jejich vlastním druhu prohlížecích programů.
>
> Myslel jsem, že obecný zastřešovací jazyk pro odkazy (založený na Hytime?) by umožnil standardům pro hypertext a grafiku/video se vyvíjet samostatně, což by pomohlo obojímu.
>
> Místo tagu `IMG` použijme `INCLUDE` a nechme ho odkazovat k libovolnému typu dokumentu. Nebo použijme `EMBED`, pokud `INCLUDE` zní jako příkaz v C++, od kterého by lidé čekali, že poskytne zdrojový kód SGML, aby byl analyzován inline – což rozhodně nebyl záměr.

HyTime byl raný, na SGML založený hypertextový dokumentový systém. Často se přetřásal v časných diskuzích na téma HTML a později XML.

Timův návrh na tag `<INCLUDE>` nebyl nikdy realizován, i když jeho ozvěny můžeme zaznamenat v prvcích `<object>`, `<embed>` a `<iframe>`.

Konečně 12. března 1993 se Marc Andreessen vrátil k tomuto diskuznímu vláknu:

> Zpátky k vláknu o vkládání obrázků – už brzo vydám Mosaic v 0.10, která bude podporovat vkládání GIFů a XMB obrázků/bitmap, jak jsem psal dříve. (…)
>
> V tuto chvíli nejsme připraveni na podporu `INCLUDE/EMBED`. (…) Takže nejspíš použijeme `<IMG SRC="url">` (nikoliv `ICON`, protože ne všechny vkládané obrázky je tak možné nazývat). Prozatím u vkládaných obrázků nebude uveden content-type; do budoucna plánujeme jeho podporu (spolu s obecným přijetím MIME). Rutiny pro čtení obrázků,

které teď používáme, ve skutečnosti dokážou poznat formát obrázku za pochodu, takže na příponě souboru nezáleží.

1.4 Nepřerušená linie

Všechny aspekty této téměř 19 let staré konverzace vedoucí ke vzniku prvku HTML, který se používá snad na každé webové stránce, jež kdy byla vytvořena, mě nesmírně fascinují. Uvažte následující:

- Protokol HTTP stále existuje. HTTP se úspěšně vyvinul z 0.9 do 1.0 a později 1.1. A vyvíjí se dál.

- HTML stále existuje. Ten primitivní datový formát – který ani nepodporoval vložené obrázky! – se úspěšně vyvinul do verzí 2.0, 3.2, 4.0. HTML tvoří nepřerušenou linii. Určitě je to linie pokroucená, zauzlovaná a zasukovaná. Na vývojovém stromě HTML bylo spousta „uschlých ratolestí", míst, kde se lidé, kteří měli standardy na starosti, příliš rozjeli (a to na úkor autorů a realizátorů). Ale i přesto. Jsme v roce 2012 a webové stránky z roku 1990 se stále zobrazují v moderních prohlížečích. Právě jsem jednu načetl ve svém nejnovějším mobilu s Androidem, a ani mi přitom nevyskočilo upozornění „prosím čekejte, než se převede historický formát…".

- HTML bylo vždy konverzací mezi tvůrci prohlížečů, autory stránek, tvůrci standardů a dalšími lidmi, kteří se vynořili bůhví odkud, aby si povídali o špičatých závorkách. Většina úspěšných verzí HTML byla „retrospektivní" – dohánělа technický svět, a přitom se ho snažila pošoupnout správným směrem. Každý, kdo vám říká, že by mělo HTML zůstat „čisté" (nejspíš ignorováním tvůrců prohlížečů nebo ignorováním autorů, případně obojím), je jednoduše vedle. HTML nebylo nikdy čisté, a všechny pokusy ho očistit byly natolik katastrofální, že se jim vyrovnaly pouze pokusy HTML něčím nahradit.

- Žádný z prohlížečů z roku 1993 neexistuje v podobě, v jaké bychom ho mohli poznat. Nestcape Navigator byl opuštěn v roce 1998 a od základů přepsán, aby vznikla Mozilla Suite, od které se pak odtrhl Firefox. Internet Explorer si prošel skromnými začátky v „Microsoft Plus! pro Windows 95", kde se ocitl v jednom balíčku s motivy plochy a pinballem. (Ale samozřejmě po jeho stopách můžeme pátrat také v ještě vzdálenější minulosti.)

- Některé z operačních systémů z roku 1993 stále existují, ale žádné z nich nemají vztah k modernímu Internetu. Většina lidí, kteří chodí na web, k tomu využívají PC s operačním systémem Windows 2000 nebo s pozdější verzí, Mac s Mac OS X, PC s určitou odnoží Linuxu nebo kapesní zařízení jako iPhone. V roce 1993 Windows běžel pod verzí

3.1 (a soutěžil s OS/2), Macy používaly System 7 a Linux se šířil přes Usenet. (Chcete si užít trochu legrace? Najděte nějakého veterána a pošeptejte mu „Trumpet Winsock" nebo „MackPPP.")

- Někteří z těchto lidí stále pracují v oboru a podílejí se na tom, čemu teď říkáme jednoduše „webové standardy". A to i po téměř 20 letech. A někteří z nich se podíleli už na předchůdcích HTML, s datací do 80. let i dříve.

- Když je řeč o předchůdcích… Díky výsledné popularitě HTML a webu je snadné zapomínat na další tehdejší formáty a systémy, které ovlivnily jejich výslednou podobu. Andrew? Intermedia? HyTime? HyTime přitom nebyl žádný obskurní akademický projekt – byla to norma ISO. Schválili ho pro vojenské použití. Zkrátka důležitá záležitost. Ostatně si o něm můžete přečíst sami… na téhle stránce, HTML ve svém webovém prohlížeči[1].

Ale nic z toho neodpovídá na původní otázku: proč máme prvek `<img>`? Proč ne prvek `<icon>`? Nebo prvek `<include>`? Proč nemáme hypertextový odkaz s atributem `<include>` nebo nějakou kombinaci hodnot `rel`? Proč zrovna prvek `<img>`? Docela jednoduše proto, že Marc Andreessen svůj kód oficiálně vydal, a vydané kódy vyhrávají.

To samozřejmě neznamená, že *všechny* vydané kódy vyhrávají; Andrew, Intermedia a HyTime své kódy přece také vydaly. Kód je nezbytnou podmínkou pro úspěch, ale sám o sobě nestačí. A už *vůbec* nechci říct, že vydání kódu před vznikem standardu je nejlepší řešení. Marcův prvek `<img>` nestanovil společný grafický formát, neurčoval, jak bude kolem něho proudit text, a nepodporoval textové alternativy ani nouzový obsah pro starší prohlížeče. A o sedmnáct let později stále ještě bojujeme s očicháváním obsahu a je to stále zdrojem neuvěřitelných bezpečnostních děr. A tohle všechno můžete vysledovat do doby před 17 lety, napříč velkými válkami prohlížečů, k 25. únoru 1993, kdy Marc Andreessen jen tak mimoděk prohodil „MIME, snad jednou v budoucnu," a pak svůj kód stejně vydal.

Ti, kdo vyhrávají, jsou ti, kdo vydávají.

1.5 Historie vývoje HTML mezi lety 1997 a 2004

V prosinci 1997 vydalo sdružení World Wide Web Consortium (W3C) HTML 4.0 a okamžitě ukončilo pracovní skupinu HTML. Ani ne o dva měsíce později jiná pracovní skupina W3C vydala XML 1.0. Pouhé tři měsíce nato provozovatelé W3C uspořádali workshop na téma „Utváření budoucnosti HTML", aby odpověděli na otázku: „Vykašlala se W3C na HTML?"

1 http://www.sgmlsource.com/history/hthist.htm

Tohle byla jejich odpověď:

> Diskuze dospěly k názoru, že další rozšiřování HTML 4.0 by bylo obtížné stejně jako převod 4.0 na aplikaci XML. Navrhovaný způsob, jak se lze vypořádat s těmito omezeními, je začít znovu s novou generací HTML založenou na balíku sad tagů XML.

Sdružení W3C obnovilo pracovní skupinu HTML a pověřilo ji vytvořením tohoto „balíku sad tagů XML". Prvním krokem byl v prosinci roku 1998 návrh prozatímní specifikace, která jednoduše převedla HTML do XML bez přidávání nových prvků či atributů. Tato specifikace se později stala známou pod názvem „XHTML 1.0". Definovala nový typ MIME pro dokumenty XHTML `application/xhtml+xml`. Pro usnadnění přechodu pro existující stránky HTML4 byl také zahrnut Dodatek C, který „shrnuje pokyny pro tvorbu stránek pro autory, kteří chtějí, aby se jejich dokumenty XHTML zobrazovaly v existujících uživatelských agentech HTML." Dodatek C říkal, že jste mohli tvořit takzvané „XHTML stránky", ale vydávat je přitom s typem MIME `text/html`.

Jejich dalším cílem byly webové formuláře. V roce 1999 tatáž pracovní skupina HTML vydala první návrh „Rozšířených formulářů XHTML". Svá očekávání popsala v prvním odstavci:

> Po pečlivém zvážení se pracovní skupina HTML rozhodla, že cíle pro další generaci formulářů jsou neslučitelné se zachováním zpětné kompatibility s prohlížeči vytvořenými pro dřívější verze HTML. Naším cílem je vytvořit nový čistý model formuláře („Rozšířené formuláře XHTML") založený na jasně definovaných požadavcích. Požadavky popsané v tomto dokumentu se zakládají na zkušenostech s celou škálou formulářových aplikací.

O několik měsíců později byly „Rozšířené formuláře XHTML" přejmenovány na „XForms" a dostaly svoji vlastní pracovní skupinu. Tato skupina pracovala paralelně s pracovní skupinou HTML a konečně vydala první verzi XForms 1.0 v říjnu 2003.

Mezitím se po dokončení přechodu na XML pracovní skupina HTML zaměřila na vytvoření „nové generace HTML". V květnu 2001 vydali první verzi XHTML 1.1, která přidávala k XHTML jen několik málo prvků navíc, ale také rušila zadní vrátka v Dodatku C. Od verze 1.1 měly všechny dokumenty XHTML obsahovat typ MIME `application/xhtml+xml`.

1.6 Všechno, co víte o XHTML, je špatně

Proč jsou typy MIME důležité? Proč se k nim neustále vracím? Jen tři slova: „draconian error handling", tj. přísné zacházení s chybami. Prohlížeče se vůči HTML vždycky chovaly benevolentně. Pokud vytvoříte HTML stránku a zapomenete na tag `</head>`, prohlížeče stránku i přesto zobrazí. (Některé tagy implicitně vyvolají konec `<head>` a začátek `<body>`.) Tagy byste

měli uzavírat hierarchicky, to znamená od posledního k prvnímu, ale když napíšete kód ve stylu `<b><i></b></i>`, prohlížeče si s tím (nějak) poradí a zobrazí stránku bez chybového hlášení.

Jak se dá očekávat, skutečnost, že „vadné" HTML stránky v prohlížečích fungovaly, vedla autory k vytváření vadných HTML stránek. Velké spousty vadných stránek. Podle některých odhadů má až 99 % všech HTML stránek na Internetu alespoň jednu chybu. Ale protože na tyto chyby prohlížeče nevyhazují žádná viditelná chybová hlášení, nikdo je nikdy neopraví.

W3C tohle vnímalo jako základní problém Internetu a rozhodlo se to napravit. XML vydané v roce 1997 porušilo tradici benevolentních klientů, když stanovilo podmínku, že všechny programy, které konzumují XML, musí zacházet s tzv. „well-formedness" chybami (chyby nesprávně strukturovaného kódu) jako s fatálními. Konceptu selhání u první chyby se začalo říkat „drakonické trestání chyb" podle starořeckého státníka Drakóna, který zavedl trest smrti i za relativně mírné porušení jeho zákonů. Když v W3C přeformulovali HTML jako slovník XML, zavedli podmínku, že všechny dokumenty vytvářené s novým typem MIME `application/xhtml+xml` budou podléhat drakonickému trestání chyb. Pokud by vaše XHTML stránka obsahovala byť jen jednu chybu v struktuře kódu – např. opomenutí tagu `</head>` nebo uzavírání tagů ve špatném pořadí – nebudou mít webové prohlížeče jinou možnost než přestat stránku zpracovávat a zobrazit koncovému uživateli chybové hlášení.

Tato myšlenka si nezískala všeobecnou oblibu. S ohledem na odhadovaných 99 % chyb v existujících stránkách a všudypřítomnou hrozbu, že se koncovému uživateli bude zobrazovat chyba, a zároveň ovšem nedostatek nových funkcí v XHTML 1.0 a 1.1, které by takové riziko vyvážily, tvůrci webů `application/xhtml+xml` v podstatě ignorovali. To ale neznamená, že by úplně ignorovali XHTML. To rozhodně ne. Dodatek C ve specifikaci XHTML 1.0 poskytl autorům webů zadní vrátka: „Použijte něco, co vypadá podobně jako syntax XHTML, ale zvolte typ MIME `text/html`." A to je přesně to, co udělaly tisíce vývojářů webových stránek – „upgradovali" na syntax XHTML, ale používali typ MIME `text/html`.

I dnes o sobě miliony webových stránek tvrdí, že jsou XHTML. Začínají s doctypem XHTML na prvním řádku, používají malá písmena pro názvy tagů a uvozovky kolem hodnot atributů a přidávají koncové lomítko po prázdných prvcích jako `<br />` a `<hr />`. Ale pouze nepatrná část těchto stránek používá typ MIME `application/xhtml+xml`, který by spustil drakonické trestání chyb XML. Jakákoliv stránka využívající typ MIME `text/html` – bez ohledu na doctype, syntax a styl kódování – bude analyzována za použití „benevolentního" analyzátoru HTML, který bude tiše ignorovat případné chyby v kódu a nikdy nebude upozorňovat koncové uživatele (ani kohokoliv jiného), i když je stránka z technického hlediska vadná.

Verze XHTML 1.0 v sobě měla tato zadní vrátka, ale verze XHTML 1.1 je zavřela, a nikdy nedokončená verze XHTML 2.0 pokračovala v tradici vyžadování drakonického trestání chyb. A to je důvod, proč máme miliony stránek, které o sobě tvrdí, že jsou XHTML 1.0, a jenom

pár, které se prohlašují za XHTML 1.1 (nebo XHTML 2.0). Je to, co používáte, opravdu XHTML? Zkontrolujte svůj typ MIME. (Pokud nevíte, jaký typ MIME používáte, můžu vám zaručit, že stále používáte `text/html`). Pokud na svých stránkách nepoužíváte `application/xhtml+xml`, vaše takzvané XHTML je XML pouze podle jména.

1.7 Konkurenční vize

V červnu 2004 pořádalo sdružení W3C workshop o webových aplikacích a složených dokumentech. Tohoto workshopu se zúčastnili zástupci tří prodejců prohlížečů, firmy zabývající se tvořením webů a další členové W3C. Skupina zúčastněných stran, včetně Mozilly Foundation a Opery Software, vystoupila s prezentací konkurenční vize budoucnosti webu: evoluce stávajícího standardu HTML 4, aby zahrnul nové funkce pro vývojáře moderních webových aplikací.

Následujících sedm principů představuje podle nás nejdůležitější požadavky pro tuto práci.

Zpětná kompatibilita, volná cesta k přechodu
: Webové aplikace by měly být založeny na technologiích, které tvůrci stránek znají, jako jsou HTML, CSS, DOM a JavaScript.

  Základní prvky webových aplikací by měly být v dnešním IE6 realizovatelné pomocí vzorců chování, skriptů a stylopisů (stylesheets), aby měli autoři volnou cestu k přechodu. Žádné řešení, které nemůže být použito s některým z uživatelských agentů, které mají v současné době vysoký podíl na trhu, bez nutnosti používání binárních zásuvných modulů, nemá příliš velkou šanci na úspěch.

Dobře definované zacházení s chybami
: Zacházení s chybami ve webových aplikacích musí být definováno do té míry, aby uživatelské agenty nemusely vytvářet své vlastní mechanismy zacházení s chybami nebo zpětně analyzovat mechanismy zacházení s chybami jiných uživatelských agentů.

Uživatelé by neměli být vystaveni chybám autorů
: Specifikace musí určit přesný způsob nápravy chyb ve všech možných chybových scénářích. Zacházení s chybami by mělo z velké části spočívat v hladké nápravě chyb (jako v CSS) spíše než v katastrofálním a do očí bijícím selhání (jako v XML).

Praktické využití
: Každý prvek ve specifikacích webových aplikací musí být odůvodněn případem praktického využití. Naopak to vždy platit nemusí: každý případ nutně nevyžaduje nový prvek.

  Případy použití by v ideálním případě měly být založeny na skutečných webech, kde autoři předtím použili nějaké špatné řešení toho, jak se poprat s nějakým omezením.

Skriptování se jen tak nezbavíme
: Ale měli bychom se mu vyhnout tam, kde může být použit praktičtější značkovací jazyk. Skriptování by mělo být nezávislé na zařízení a prezentaci, pokud není přizpůsobeno konkrétnímu zařízení (např. pokud není obsaženo v XBL).

Je třeba se vyhnout profilování pro konkrétní zařízení
: Autoři by měli mít možnost spoléhat na to, že do desktopové i mobilní verze stejného UA budou implementovány stejné funkce.

Otevřený proces
: Internet těžil z toho, že se vyvíjel v otevřeném prostředí. Webové aplikace budou tvořit jádro Internetu a jejich vývoj by se měl také odehrávat otevřeně. E-mailové konference, archivy a specifikace návrhů by měly být neustále přístupné veřejnosti.

Mezi účastníky workshopu proběhla anketa na téma „Má W3C vyvíjet deklarativní rozšíření HTML a CSS a imperativní rozšíření DOM, aby odpověděla na středně náročné požadavky webových aplikací, jako protiváhu k sofistikovaným, plně rozvinutým API na úrovni operačních systémů?“ (Anketu navrhl Ian Hickson z Opery Software.) Hlasování dopadlo 11 ku 8. Ve shrnutí průběhu workshopu sdružení W3C napsalo: „V současné době se W3C nechystá vynaložit žádné zdroje na téma třetí ankety: rozšíření HTML a CSS pro webové aplikace, kromě technologií, které jsou vyvíjeny v rámci současných pracovních skupin W3C.“

Tváří v tvář tomuto rozhodnutí měli lidé, kteří navrhovali vyvíjení HTML a formulářů HTML, pouze dvě možnosti: vzdát to, nebo pokračovat ve své práci mimo W3C. Vybrali si druhou možnost a zaregistrovali doménu `whatwg.org`. V červnu roku 2004 se pak zrodila pracovní skupina WHAT.

1.8 Pracovní skupina WHAT?

Co je vlastně zač pracovní skupina WHAT? Nechám je, aby vám to řekli sami:

> Web Hypertext Applications Technology Working Group je volné, neoficiální a otevřené sdružení výrobců webových prohlížečů a zúčastněných stran. Tato skupina usiluje o vývoj specifikací založených na HTML a souvisejících technologiích, aby se usnadnilo nasazení interoperabilních webových aplikací, s úmyslem předkládání výsledků normalizační organizaci. Toto předkládání by pak tvořilo základ práce na oficiálním rozšíření HTML v oblasti standardů.
>
> Vytvoření tohoto fóra následovalo po několika měsících práce na specifikacích pro takové technologie, která probíhala prostřednictvím e-mailu. Hlavním cílem až do tohoto bodu bylo rozšiřování Formulářů HTML4, aby podporovaly funkce požadované autory webů, bez porušení zpětné kompatibility u stávajícího obsahu. Tato skupina byla vytvořena, aby zajistila, že další vývoj těchto specifikací bude probíhat naprosto otevřeně, prostřednictvím veřejně archivované a přístupné e-mailové konference.

Klíčovou frází je zde „bez porušení zpětné kompatibility“. XHTML (bez zadních vrátek v podobě Dodatku C) není zpětně kompatibilní s HTML. Vyžaduje úplně nový typ MIME a vynucuje si drakonické trestání chyb pro veškerý obsah s tímto typem MIME. XForms nejsou zpětně kompatibilní s formuláři HTML, protože můžou být použity pouze v dokumentech, které používají nový typ MIME XHTML, což znamená, že si XForms rovněž vynucují drakonické trestání chyb. Všechny cesty vedou k MIME.

Místo toho, aby zahodila více než dekádu investování do HTML a způsobila, že by se 99 % existujících webových stránek stalo nepoužitelnými, se pracovní skupina WHAT rozhodla pro jiný přístup: zdokumentovala ty „benevolentní“ algoritmy pro zacházení s chybami, které prohlížeče v praxi používaly. Webové prohlížeče odjakživa ignorovaly chyby HTML, ale nikdo se nikdy neobtěžoval zapsat, jak to vlastně dělaly. Prohlížeč NCSA Mosaic měl své vlastní algoritmy pro nakládání s vadnými stránkami a Netscape se jim pokoušel přizpůsobit. Opera a Firefox se pokoušely přizpůsobit Internet Exploreru. Pak se Safari pokusil přizpůsobit Firefoxu. A tak to šlo dále a dále až dodnes. Vývojáři strávili tisíce hodin tím, že se snažili, aby jejich produkty byly kompatibilní s produkty konkurence.

Pokud vám to přijde jako ohromná spousta práce, tak to máte pravdu. Ohromná spousta práce to opravdu byla. Trvalo to pět let, ale (s výjimkou několika obskurních mezních případů) pracovní skupina WHAT nakonec úspěšně zdokumentovala, jak analyzovat syntax HTML způsobem, který by byl kompatibilní se stávajícím webovým obsahem. V konečném algoritmu není žádný krok, který by nařizoval, že by prohlížeče musely přestat načítat stránku a zobrazit koncovému uživateli chybové hlášení.

Mezitím co probíhalo všechno tohle reverzní inženýrství, pracovní skupina WHAT tiše pracovala i na několika dalších věcech. Jednou z nich byla specifikace, původně pojmenovaná Webové formuláře 2.0, která přidávala do formulářů HTML nové typy prvků. (Více o webových formulářích se dočtete v kapitole Forma šílenství.) Dalším projektem byla specifikace nazvaná „Webové aplikace 1.0“, která přinesla několik nových důležitých funkcí, jako vykreslovací plátno s přímým vykonáváním instrukcí a nativní podporu audia a videa bez zásuvných modulů.

1.9 Zpátky k W3C

Dva a půl roku se W3C a pracovní skupina WHAT víceméně vzájemně ignorovaly. Zatímco pracovní skupina WHAT se zaměřovala na webové formuláře a nové prvky HTML, pracovní skupina HTML W3C měla plné ruce práce s verzí XHTML 2.0. Ale v říjnu 2006 bylo jasné, že pracovní skupina WHAT nabrala na obrátkách, na rozdíl od XHTML2, které se stále ještě plácalo ve formě návrhu, přičemž ho nevyužíval žádný z vedoucích prohlížečů. V říjnu 2006 sám zakladatel W3C Tim Berners-Lee prohlásil, že W3C bude spolupracovat s pracovní skupinou WHAT na vývoji HTML.

Některé věci jsou jasnější, když se na ně díváme s několikaletým odstupem. HTML je nutné vyvíjet postupně. Pokus přimět svět, aby přešel na XML se všemi uvozovkami kolem hodnot atributů a lomítky v prázdných tazích a jmenných prostorech najednou, nefungoval. Většina veřejnosti vytvářející stránky HTML se nepohnula, z velké části proto, že si prohlížeče nestěžovaly. Některé velké komunity přešly a teď sklízejí ovoce dobře strukturovaných systémů, ale nejsou to zdaleka všechny. Je důležité postupně vylepšovat HTML a zároveň pokračovat v přechodu do dobře strukturovaného světa a získávat v tomto světě větší váhu.

V plánu je ustanovit úplně novou skupinu pro HTML. Na rozdíl od té předchozí bude pověřena postupným vylepšováním HTML a paralelně XHTML. Bude mít jiného předsedu a personální složení. Bude pracovat zároveň na HTML a XHTML. Tato skupina má velkou podporu u mnoha lidí, s kterými jsme mluvili, včetně tvůrců prohlížečů.

Skupina bude také pracovat na formulářích. To je složitá oblast, jelikož stávající formuláře a XForms jsou obojí formulářové jazyky. HTML formuláře jsou široce rozšířené a zároveň existuje i mnoho realizací a uživatelů XForms. Webforms navrhly rozumná rozšíření HTML formulářů. Podle vzoru Webforms chceme rozšířit HTML formuláře i my.

Jednou z prvních věcí, pro kterou se nově ustanovená pracovní skupina HTML W3C rozhodla, bylo přejmenovat „Webové aplikace 1.0“ na „HTML5“. No a jsme doma a můžeme začít s HTML5.

1.10 Postskriptum

V říjnu roku 2009 sdružení W3C zrušilo pracovní skupinu XHTML 2 a jako odůvodnění svého rozhodnutí vydalo následující prohlášení:

> Když jsme v březnu 2007 ohlásili pracovní skupiny HTML a XHTML 2, naznačili jsme, že budeme dále sondovat, zda je na trhu zájem o XHTML 2. Sdružení W3C si je vědomo důležitosti toho vyslat komunitě ohledně budoucnosti HTML jasný signál.
>
> Ačkoliv uznáváme hodnotu příspěvků pracovní skupiny XHTML 2 za léta její činnosti, po diskusi s účastníky se vedení W3C rozhodlo nechat pověření skupiny vypršet na konci roku 2009 a dále už ho neobnovovat.

Ti, kdo vyhrávají, jsou ti, kdo vydávají.

1.11 K dalšímu čtení

- The History of the Web[2] (Historie Internetu), starý stručný text Iana Hicksona
- HTML/History[3] (HTML/Historie) Michaela Smitha, Henriho Sivonena a dalších
- A Brief History of HTML[4] (Stručná historie HTML) Scotta Reynena

2 http://hixie.ch/commentary/web/history
3 http://www.w3.org/html/wg/wiki/History
4 http://atendesigngroup.com/blog/brief-history-of-html

2. Detekce prvků HTML5

2. Detekce prvků HTML5 – 41

2.1 Začínáme

Možná se ptáte: „Jak můžu začít používat HTML5, pokud ho nepodporují starší prohlížeče?“ Ale ta otázka je sama o sobě zavádějící. HTML5 není jedna velká věc, ale soubor jednotlivých prvků. Proto nemůžete detekovat „podporu HTML5“, protože to by nedávalo žádný smysl. Ale můžete detekovat podporu jednotlivých prvků, jako jsou plátno, video nebo geolokace.

2.2 Techniky detekce

Když prohlížeč načítá webovou stránku, vytváří Document Object Model (*DOM*), sadu objektů, které představují prvky HTML na stránce. Každý prvek – každé `<p>`, `<div>` či `<span>` – je v DOMu reprezentován jiným objektem. (Existují také globální objekty jako `window` a `document`, které nejsou svázány s konkrétními prvky.)

Všechny objekty v DOMu sdílejí sadu společných vlastností, ale některé objekty jich mají víc než jiné. V prohlížečích, které podporují prvky HTML5, budou mít některé objekty jedinečné vlastnosti. Z letmého nahlédnutí do DOMu se dozvíte, které prvky jsou podporovány.

Existují čtyři základní techniky toho, jak zjistit, zda prohlížeč podporuje nějaký prvek. Od nejjednodušší k nejsložitější:

- Zkontrolujte, zda u globálního objektu (jako je `window` nebo `navigator`) určitá vlastnost existuje.

  Příklad: testování podpory geolokace

- Zkontrolujte prvek a pak zkontrolujte, zda u tohoto prvku existuje určitá vlastnost.

  Příklad: testování podpory plátna

- Vytvořte prvek, zkontrolujte, zda pro tento prvek existuje určitá metoda, pak metodu vyvolejte a zkontrolujte hodnotu, kterou vrátí.

  Příklad: testování podporovaných formátů videa

- Vytvořte prvek, nastavte vlastnost na určitou hodnotu a pak zkontrolujte, zda si daná vlastnost hodnotu uchovala.

  Příklad: testování podporovaných typů `<input>`

2.3 Modernizr, detekční knihovna

Modernizr je open-source javascriptová knihovna licencovaná MIT, která detekuje podporu mnoha prvků HTML5 a CSS3. Měli byste vždy používat její nejnovější verzi. Pro její využití použijte v horní části vaší stránky prvek `<script>`.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Dive Into HTML5</title>
    <script src="modernizr.min.js"></script>
</head>                                    ↖ Tohle použijte v <head> své stránky
<body>
    ...
</body>
</html>
```

Modernizr se spouští automaticky. Nevyvolává se žádná funkce `modernizr_init()`. Když se spustí, vytvoří globální objekt jménem `Modernizr`, jenž obsahuje sadu booleovských vlastností pro každý prvek, který dokáže detekovat. Pokud například váš prohlížeč podporuje API plátna, bude vlastnost `Modernizr.canvas` mít hodnotu true. Pokud váš prohlížeč plátno API nepodporuje, hodnota vlastnosti `Modernizr.canvas` bude `false`.

```
if (Modernizr.canvas) {
    // vykresleme si nějaké tvary!
} else {
    // nativní podpora plátna není k dispozici :(
}
```

2.4 Plátno

HTML5 definuje prvek `<canvas>` jako „bitmapové plátno závislé na rozlišení, které může být použito pro vykreslování grafů, herní grafiky nebo jiných vizuálních obrazů v reálném čase.“ *Plátno* je obdélník na vaší stránce, kde můžete pomocí JavaScriptu vykreslit, cokoli chcete. HTML5 definuje sadu funkcí („API plátna“) pro kreslení tvarů, definování cest, vytváření přechodů a používání transformací.

Při detekci API plátna se využívá detekční technika č. 2. Pokud váš prohlížeč podporuje API plátna, objekt DOM, který vytváří pro reprezentaci prvku `<canvas>`, bude mít metodu

`getContext()`. Pokud váš prohlížeč API plátna nepodporuje, objekt DOM, který vytváří pro prvek `<canvas>`, bude mít pouze soubor společných vlastností, ale nic specifického pro plátno.

```
function supports_canvas() {
    return !!document.createElement('canvas').getContext;
}
```

Tato funkce začíná vytvořením fiktivního prvku `<canvas>`. Ale tento prvek není nikdy připojen k vaší stránce a nikdo ho nikdy neuvidí. Jenom se tak vznáší v paměti, nikam nemíří a nic nedělá, jako kánoe na líně tekoucí řece.

```
return !!document.createElement('canvas').getContext;
```

Hned jak vytvoříte fiktivní prvek `<canvas>`, otestujete přítomnost metody `getContext()`. Tato metoda bude přítomna pouze v případě, že váš prohlížeč podporuje API plátna.

```
return !!document.createElement('canvas').getContext;
```

Konečně použijete trik s dvojitou negací, abyste si vynutili výsledek v booleovské hodnotě (`true` nebo `false`).

```
return !!document.createElement('canvas').getContext;
```

Tato funkce detekuje podporu většiny součástí API plátna včetně tvarů, cest, přechodů a vzorů. Nedetekuje knihovnu explorercanvas třetí strany, která implementuje API plátna v Microsoft Internet Exploreru. Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory API plátna použít Modernizr.

Zkontrolujte podporu plátna

```
if (Modernizr.canvas) {
    // vykresleme si nějaké tvary!
} else {
    // nativní podpora plátna není k dispozici :(
}
```

Pro API textu na plátně existuje samostatný test, který vám ukážu jako další.

2.5 Text na plátně

I když váš prohlížeč podporuje API plátna, nemusí ještě podporovat *API textu na plátně*. Prvek API plátna se postupně rozšiřoval a později k němu byly připojeny i textové funkce. Některé prohlížeče připravily podporu plátna dříve, než bylo hotové API textu.

Kontrola podpory API textu na plátně využívá detekční techniku č. 2. Pokud váš prohlížeč podporuje API plátna, objekt DOM, který vytváří pro reprezentaci prvku `<canvas>`, bude mít metodu `getContext()`. Pokud váš prohlížeč API plátna nepodporuje, objekt DOM, který vytváří pro prvek <canvas>, bude mít pouze soubor společných vlastností, ale nic specifického pro plátno.

```
function supports_canvas_text() {
    if (!supports_canvas()) { return false; }
    var dummy_canvas = document.createElement('canvas');
    var context = dummy_canvas.getContext('2d');
    return typeof context.fillText == 'function';
}
```

Tato funkce začíná kontrolou podpory plátna, přičemž používá funkci `supports_canvas()`, kterou jste mohli vidět v předchozím oddíle. Pokud váš prohlížeč nepodporuje API plátna, určitě nebude podporovat ani API textu na plátně!

```
if (!supports_canvas()) { return false; }
```

Dále vytvoříte fiktivní prvek `<canvas>` a získáte jeho vykreslovací kontext. Bude to zaručeně fungovat, protože funkce `supports_canvas()` už zkontrolovala, že u všech objektů plátna existuje metoda `getContext()`.

```
var dummy_canvas = document.createElement('canvas');
var context = dummy_canvas.getContext('2d');
```

Konečně zkontrolujete, zda má vykreslovací kontext funkci `fillText()`. Pokud ano, máte API textu na plátně k dispozici. Hurá!

```
return typeof context.fillText == 'function';
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory API textu na plátně použít Modernizr.

Zkontrolujte podporu textu na plátně

```
if (Modernizr.canvastext) {
    // vykresleme si nějaký text!
} else {
    // nativní podpora textu na plátně není k dispozici :(
}
```

2.6 Video

HTML5 definuje nový prvek pro vkládání videí na vaše webové stránky nazvaný `<video>`. Vkládání videí dříve nebylo možné bez použití zásuvných modulů třetích stran, jako jsou Apple QuickTime® nebo Adobe Flash®.

Prvek `<video>` je navržen tak, aby se dal používat bez detekčních skriptů. Můžete určit několik videosouborů a prohlížeče, které podporují HTML5, si vyberou jeden podle toho, které videoformáty podporují. (Pokud se chcete dozvědět více o různých videoformátech, podívejte se na průvodce kódováním videa „A gentle introduction to video encoding" part 1: container formats[1] a part 2: lossy video codecs[2]).

Prohlížeče, které nepodporují HTML5 video, budou prvek `<video>` úplně ignorovat, ale i toho můžete využít a přikázat jim, aby místo toho video přehrály v zásuvném modulu třetí strany. Kroc Camen vytvořil řešení nazvané Video for Everybody!, které používá HTML5 video všude, kde je to možné, ale ve starších prohlížečích se vrací ke QuickTime nebo Flash. Toto řešení nepoužívá vůbec žádný JavaScript a funguje v prakticky všech prohlížečích, včetně mobilních.

Pokud chcete s videem udělat něco více, než ho jenom dát na stránku a přehrát, budete muset použít JavaScript. Kontrola podpory videa využívá detekční techniku č. 2. Pokud váš prohlížeč podporuje HTML5 video, objekt DOM, který vytváří pro reprezentaci prvku `<video>`, bude mít metodu `canPlayType()`. Pokud váš prohlížeč HTML5 video nepodporuje, objekt DOM, který vytváří pro prvek `<video>`, bude mít pouze soubor vlastností společných pro všechny prvky. Podporu videa můžete zkontrolovat pomocí této funkce:

```
function supports_video() {
    return !!document.createElement('video').canPlayType;
}
```

1 http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats
2 http://diveintomark.org/archives/2008/12/19/give-part-2-lossy-video-codecs

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory HTML5 videa použít Modernizr.

Zkontrolujte podporu HTML5 videa

```
if (Modernizr.video) {
    // přehrajme si nějaké video!
} else {
    // nativní podpora videa není k dispozici :(
    // co takhle se místo toho podívat po QuickTime nebo Flash?
}
```

V kapitole Video vám ukážu ještě jedno řešení, které používá tyto detekční techniky ke konverzi prvků `<video>` na přehrávače založené na Flash pro prohlížeče, které nepodporují HTML5 video.

Pro detekci videoformátů, které váš prohlížeč dokáže přehrát, existuje samostatný test, který vám ukážu jako další.

2.7 Formáty videa

Formáty videa jsou jako psané jazyky. Noviny psané anglicky můžou obsahovat stejné informace jako noviny psané španělsky, ale pokud mluvíte pouze anglicky, budou pro vás užitečné pouze první z nich. Aby váš prohlížeč mohl přehrát video, musí porozumět „jazyku“, ve kterém bylo toto video napsáno.

„Jazyku“ videa se říká „kodek“ – je to algoritmus použitý pro zakódování videa v tok bitů. Po celém světě se používají desítky kodeků. Který byste měli používat vy? Nevýhodou HTML5 videa je, že se prohlížeče nedokážou shodnout na jednom jediném kodeku. Povedlo se jim to ovšem omezit na dva. Jeden kodek stojí peníze (kvůli patentové licenci), ale funguje v Safari a na iPhonu. (Tento kodek bude také fungovat ve Flash, pokud použije řešení typu Video for Everybody!) Druhý kodek je zadarmo a funguje v open-source prohlížečích, jako jsou Chromium a Mozilla Firefox.

Kontrola podpory formátů videa využívá detekční techniku č. 3. Pokud váš prohlížeč podporuje HTML5 video, objekt DOM, který vytváří pro reprezentaci prvku `<video>`, bude mít metodu `canPlayType()`. Tato metoda vám prozradí, zda daný prohlížeč podporuje určitý formát videa.

Tato funkce kontroluje podporu patentovaného formátu, který podporují Macy a iPhony.

```
function supports_h264_baseline_video() {
    if (!supports_video()) { return false; }
    var v = document.createElement("video");
    return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

Funkce začíná kontrolou podpory HTML5 videa a používá funkci `supports_video()`, kterou jste mohli vidět v předchozím oddíle. Pokud váš prohlížeč nepodporuje HTML5 video, určitě nebude podporovat ani žádné formáty videa!

```
if (!supports_video()) { return false; }
```

Poté funkce vytvoří fiktivní prvek `<video>` (ale nepřipojí ho ke stránce, takže nebude vidět) a vyvolá metodu `canPlayType()`. Tato metoda je určitě přítomna, protože ji funkce `supports_video()` právě zkontrolovala.

```
var v = document.createElement("video");
```

„Formát videa“ je ve skutečnosti tvořen kombinací různých věcí. Z technického hlediska se ptáte prohlížeče, jestli dokáže přehrát video H.264 Baseline a audio AAC LC v kontejneru MPEG-4. (Co to všechno znamená, vám vysvětlím v kapitole Video. Také by vás mohl zaujmout průvodce kódováním videa – A gentle introduction to video encoding.)

```
return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
```

Funkce `canPlayType()` nevrací hodnoty `true` nebo `false`. S ohledem na to, jak složité formáty videa jsou, vrací řetězec:

- `"probably"`, pokud si je prohlížeč celkem jistý, že tento formát dokáže přehrát
- `"maybe"`, pokud si prohlížeč myslí, že tento formát možná dokáže přehrát
- `""` (prázdný řetězec), pokud si je prohlížeč jistý, že tento formát nedokáže přehrát

Tato druhá funkce kontroluje podporu otevřeného formátu videa podporovaného Mozilla Firefox a dalšími open-source prohlížeči. Proces je úplně stejný; jediným rozdílem je řetězec, který předáte funkci `canPlayType()`. Z technického hlediska se ptáte prohlížeče, jestli dokáže přehrát video Theora a audio Vorbis v kontejneru Ogg.

```
function supports_ogg_theora_video() {
    if (!supports_video()) { return false; }
    var v = document.createElement("video");
    return v.canPlayType('video/ogg; codecs="theora, vorbis"');
}
```

Konečně WebM je nový open-source (a nepatentovaný) video kodek, který bude obsažen v dalších verzích nejpoužívanějších prohlížečů včetně Chrome, Firefoxu a Opery. Pro detekci podpory videa WebM můžete použít stejnou techniku.

```
function supports_webm_video() {
    if (!supports_video()) { return false; }
    var v = document.createElement("video");
    return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory různých formátů videa v HTML5 videa použít Modernizr (verze 1.5 nebo pozdější).

Zkontrolujte podporu formátů videa v HTML5

```
if (Modernizr.video) {
    // přehrajme si nějaké video! ale jakého druhu?
    if (Modernizr.video.webm) {
        // zkuste WebM
    } else if (Modernizr.video.ogg) {
        // zkuste Ogg Theora + Vorbis v kontejneru Ogg
    } else if (Modernizr.video.h264){
        // zkuste video H.264 + audio AAC v kontejneru MP4
    }
}
```

2.8 Místní úložiště

Úložiště HTML5 poskytuje webovým stránkám možnost, aby ukládaly data na vašem počítači a později se k nim vrátily. Tento koncept se podobá cookies, ale je určen pro větší množství dat. Cookies jsou omezeny co do velikosti a váš prohlížeč je posílá zpět na webový server pokaždé, když vyšle požadavek na novou stránku (což vyžaduje čas navíc a zabírá vzácnou šířku pásma). Úložiště HTML5 zůstává na vašem počítači a webové stránky se k němu mohou dostat pomocí JavaScriptu poté, co se stránka načte.

PTEJTE SE PROFESORA ZNAČKY

Q: Je místní úložiště skutečnou součástí HTML5? Proč je ve zvláštní specifikaci?

A: Stručně řečeno: ano, místní úložiště je součástí HTML5. O trochu delší odpověď zní, že místní úložiště bylo původně součástí hlavní specifikace HTML5, ale pak bylo přesunuto do své vlastní specifikace, protože si někteří členové pracovní skupiny HTML5 stěžovali, že je HTML5 moc velké. Pokud vám to přijde stejné, jako rozdělit koláč na několik kousků a čekat, že se tím sníží celkový počet kalorií… no, tak vítejte v bláznivém světě webových standardů.

Kontrola podpory úložiště HTML5 využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje úložiště HTML5, bude mít globální objekt `window` vlastnost `localStorage`. Pokud váš prohlížeč úložiště HTML5 nepodporuje, zůstane vlastnost `localStorage` nedefinovaná. Kvůli nešťastné závadě u starších verzí Firefoxu bude tento test vyvolávat výjimku, pokud jsou vypnuty cookies, takže je celý test nutné vložit do bloku `try..catch`.

```
function supports_local_storage() {
    try {
        return 'localStorage' in window && window['localStorage'] !==
        null;
    } catch(e){
        return false;
    }
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory místního úložiště HTML5 použít Modernizr (verze 1.1 nebo pozdější).

Zkontrolujte podporu místního úložiště HTML5

```
if (Modernizr.localstorage) {
    // window.localStorage je k dispozici!
} else {
    // nativní podpora pro místní úložiště není k dispozici :(
    // zkuste záchranný plán nebo jiné náhradní řešení třetí strany
}
```

Uvědomte si, že JavaScript je citlivý na rozlišování malých a velkých písmen. Atribut v Modernizru se jmenuje `localstorage` (všechno malým), ale vlastnost DOMu se jmenuje `window.localStorage` (jedno z písmen je velké).

PTEJTE SE PROFESORA ZNAČKY

Q: Jak bezpečná je databáze úložiště HTML5? Může si ji prohlížet každý?

A: Každý, kdo má fyzický přístup k vašemu počítači, si pravděpodobně může prohlédnout (nebo dokonce měnit) vaši databázi úložiště HTML5. V rámci vašeho prohlížeče může kterákoliv webová stránka číst a upravovat své vlastní hodnoty, ale nebude mít přístup k hodnotám uloženým jinými stránkami. Říká se tomu omezení stejného původu.

2.9 Web Workers

Web Workers nabízejí standardizovanou cestu, jak můžou prohlížeče na pozadí spouštět JavaScript. S web workers můžete vytvořit několik „vláken“, které poběží více méně ve stejnou dobu. (Představte si, jak váš počítač může najednou spustit několik aplikací, a budete celkem blízko.) Tato „vlákna na pozadí“ mohou provádět složité matematické výpočty, vysílat síťové žádosti nebo se připojit k místnímu úložišti, zatímco hlavní webová stránka reaguje na posouvání, klikání nebo psaní uživatele.

Kontrola podpory web workers využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje Web Worker API, bude mít globální objekt window vlastnost `Worker`. Pokud váš prohlížeč Web Worker API nepodporuje, zůstane vlastnost `Worker` nedefinovaná.

```
function supports_web_workers() {
    return !!window.Worker;
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory web workers použít Modernizr (verze 1.1 nebo pozdější).

Zkontrolujte podporu web workers

```
if (Modernizr.webworkers) {
    // window.Worker je k dispozici!
} else {
```

```
    // nativní podpora pro web workers není k dispozici :(
    // zkuste záchranný plán nebo jiné náhradní řešení třetí strany
}
```

JavaScript je citlivý na rozlišování malých a velkých písmen. Atribut v Modernizru se jmenuje `webworkers` (všechno malým), ale objekt DOM se jmenuje `window.Worker` (s velkým „W“ ve slově „Worker“).

2.10 Offline webové aplikace

Čtení statických webových stránek offline je snadné: připojíte se na Internet, načtete webovou stránku, odpojíte se od Internetu, uchýlíte se na samotu u lesa a pak můžete stránku pročítat tak dlouho, jak je vám libo. (Pokud chcete ušetřit čas, možná můžete vynechat tu samotu u lesa.) Ale co s webovými aplikacemi, jako je Gmail nebo Google Docs? Díky HTML5 teď může každý (nejen Google!) vytvořit webové aplikace, které budou fungovat i offline.

Offline webové aplikace začínají jako online webové aplikace. Když poprvé navštívíte webovou stránku, která podporuje prohlížení offline, webový server vašemu prohlížeči sdělí, které soubory potřebuje, aby fungoval offline. Těmito soubory může být cokoliv – HTML, JavaScript, obrázky, dokonce i videa. Jakmile si váš prohlížeč stáhne všechny tyto potřebné soubory, můžete se na stránku vrátit, i když zrovna nebudete připojeni k Internetu. Váš prohlížeč zjistí, že jste offline, a použije soubory, které už si předtím stáhl. Až se vrátíte online, veškeré změny, které jste provedli, se nahrají na vzdálený server.

Kontrola podpory offline využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje offline webové aplikace, bude mít globální objekt window vlastnost `applicationCache`. Pokud váš prohlížeč offline webové aplikace nepodporuje, zůstane vlastnost `applicationCache` nedefinovaná. Podporu offline můžete zkontrolovat pomocí následující funkce:

```
function supports_offline() {
    return !!window.applicationCache;
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory offline webových aplikací použít Modernizr (verze 1.1 nebo pozdější).

Zkontrolujte podporu offline

```
if (Modernizr.applicationcache) {
    // window.applicationCache je k dispozici!
```

```
} else {
    // nativní podpora pro offline není k dispozici :(
    // zkuste záchranný plán nebo jiné náhradní řešení třetí strany
}
```

Uvědomte si, že JavaScript je citlivý na rozlišování malých a velkých písmen. Atribut v Modernizru se jmenuje `applicationcache` (všechno malým), ale objekt DOM se jmenuje `window.applicationCache` (s velkým „C“ ve slově „Cache“).

2.11 Geolokace

Geolokace je umění zjistit, kde ve světě jste, a (případně) sdílet tuto informaci s lidmi, kterým důvěřujete. Zjistit, kde právě jste, jde více způsoby – z vaší IP adresy, z vašeho wi-fi připojení, z toho, s jakým vysílačem komunikuje váš mobil, anebo pomocí speciálních zařízení GPS, jež vypočítávají zeměpisnou šířku a délku z informací, které jim posílají satelity.

PTEJTE SE PROFESORA ZNAČKY

Q: Je geolokace součástí HTML5? Proč o ní vůbec mluvíme?

A: Podpora geolokace se přidává do prohlížečů právě teď, spolu s podporou nových prvků HTML5. Přísně vzato se geolokace nachází v procesu standardizace pracovní skupinou Geolokace, která není součástí pracovní skupiny HTML5. Ale i přesto budu o geolokaci v této knize mluvit, protože je to součást vývoje webu, který právě teď probíhá.

Kontrola podpory geolokace využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje geolokační API, bude mít globální objekt navigator vlastnost `geolocation`. Pokud váš prohlížeč geolokační API nepodporuje, vlastnost `geolocation` v `navigator` nebude přítomna. Podporu geolokace můžete zkontrolovat následovně:

```
function supports_geolocation() {
    return 'geolocation' in navigator;
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory geolokační API použít Modernizr.

Zkontrolujte podporu geolokace

```
if (Modernizr.geolocation) {
    // pojďte zjistit, kde jste!
} else {
    // nativní podpora pro geolokaci není k dispozici
    // zkuste geoPosition.js nebo jiné náhradní řešení třetí strany
}
```

I když váš prohlížeč neobsahuje nativní podporu geolokační API, stále ještě máte šanci. GeoPosition.js je javascriptová knihovna, jejímž cílem je poskytnutí podpory geolokace ve starších prohlížečích, jako je Blackberry, Palm OS a Microsoft Internet Explorer 6, 7 a 8. Není to úplně totéž, co `navigator.geolocation` API, ale plní tentýž účel.

Existují také geolokační API vytvořené pro konkrétní zařízení na starších mobilních platformách, jako jsou BlackBerry, Nokia, Palm a OMTP BONDI.

V kapitole věnované geolokaci najdete to, jak všechny tyto různé API používat, rozepsáno do nejmenších podrobností.

2.12 Vstupní typy

O webových formulářích víte všechno, že? Uděláte `<form>`, přidáte pár prvků `<input type="text">` a možná taky `<input type="password">`, a zakončíte to tlačítkem `<input type="submit">`.

Ve skutečnosti máte ve svých znalostech pořádné mezery. HTML5 definuje více než tucet nových vstupních typů, které můžete ve vašich formulářích používat.

- `<input type="search">` pro vyhledávací pole
- `<input type="number">` pro přepínací číselné seznamy
- `<input type="range">` pro posuvníky
- `<input type="color">` pro výběr barev
- `<input type="tel">` pro telefonní čísla
- `<input type="url">` pro webové adresy
- `<input type="email">` pro e-mailové adresy
- `<input type="date">` pro výběr data v kalendáři
- `<input type="month">` pro výběr měsíce
- `<input type="week">` pro výběr týdne
- `<input type="time">` pro výběr času

- `<input type="datetime">` pro absolutní, přesný údaj datum + čas
- `<input type="datetime-local">` pro místní datum a čas

Kontrola podpory vstupních typů HTML5 využívá detekční techniku č. 4. Nejdříve v paměti vytvoříte fiktivní prvek. Výchozí vstupní typ pro všechny prvky `<input>` je `"text"`. To má zásadní důležitost.

```
var i = document.createElement("input");
```

Dále nastavíte atribut `type` na fiktivním prvku `<input>` na vstupní typ, který chcete detekovat.

```
i.setAttribute("type", "color");
```

Pokud váš prohlížeč podporuje daný vstupní typ, vlastnost `type` si uchová hodnotu, kterou jste nastavili. Pokud váš prohlížeč daný vstupní typ nepodporuje, bude ignorovat vámi nastavenou hodnotu a vlastnost `type` zůstane jako `"text"`.

```
return i.type !== "text";
```

Místo toho, abyste všech 13 oddělených funkcí psali vy, můžete pro detekci podpory všech nových vstupních typů definovaných v HTML5 použít Modernizr. Modernizr používá jediný prvek `<input>` pro efektivní detekci podpory všech 13 vstupních typů. Pak vytváří asociativní pole zvané `Modernizr.inputtypes`, který obsahuje 13 klíčů (atributy HTML5 type) a 13 booleovských hodnot (`true`, pokud jsou podporovány, a `false`, pokud ne).

Zkontrolujte podporu nativního výběru data

```
if (!Modernizr.inputtypes.date) {
    // nativní podpora pro <input type="date"> není k dispozici :(
    // můžete si ji vytvořit sami s Dojo nebo jQueryUI
}
```

2.13 Placeholder (zástupný text)

Kromě nových vstupních typů HTML5 obsahuje i pár drobných zlepšováků k už existujícím formulářům. Jedním ze zlepšení je možnost vložit do vstupního pole zástupný text. Zástupný text se ve vstupním poli zobrazuje tak dlouho, dokud je pole prázdné a nemá fokus. Jakmile kliknete (nebo přepnete tabulátorem) do vstupního pole, zástupný text zmizí. Pokud máte problém si to představit, podívejte se na snímky obrazovky v kapitole o webových formulářích.

Kontrola podpory zástupného textu používá detekční techniku č. 2. Pokud prohlížeč podporuje zástupný text ve vstupních polích, objekt DOM, který vytváří pro reprezentaci prvku `<input>`, bude mít vlastnost `placeholder` (i když v HTML nepoužijete atribut `placeholder`). Pokud váš prohlížeč zástupný text nepodporuje, objekt DOM, který vytváří pro prvek `<input>`, nebude mít vlastnost placeholder.

```
function supports_input_placeholder() {
    var i = document.createElement('input');
    return 'placeholder' in i;
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory zástupného textu použít Modernizr (verze 1.1 nebo pozdější).

Zkontrolujte podporu zástupného textu

```
if (Modernizr.input.placeholder) {
    // váš zástupný text by už měl být vidět!
} else {
    // zástupný text není podporován :(
    // použijte skriptované řešení
}
```

2.14 Autofokus formulářů

Webové stránky mohou používat JavaScript, aby automaticky daly fokus na první vstupní pole webového formuláře. Například domovská stránka Google.com má autofokus na vstupním poli, takže můžete napsat klíčová slova vyhledávání bez toho, abyste museli umístit kurzor do vyhledávacího pole. Tohle je sice pro většinu lidí pohodlné, ale pro pokročilé uživatele nebo lidi se speciálními potřebami to může být otrava. Pokud zmáčknete mezerník s tím, že chcete stránku posouvat, stránka se vám posouvat nebude, protože fokus už je ve vstupním poli formuláře. (Místo posouvání tak napíšete do pole mezeru.) Pokud dáte fokus na jiné vstupní pole, zatímco se stránka ještě načítá, skript pro autofokus stránky vám „pomůže“ tím, že po úplném načtení přemístí fokus zpátky do původního vstupního pole, což naruší váš rytmus a způsobí, že začnete psát do špatného místa.

Protože autofokus funguje pomocí JavaScriptu, může být obtížné zvládnout všechny mezní příklady, a pro lidi, kteří nechtějí, aby webová stránka fokus „kradla“, moc východisek není.

Pro vyřešení tohoto problému HTML5 přichází s atributem autofocus pro webové formuláře. Atribut `autofocus` dělá přesně to, co říká: přesouvá fokus do určitého vstupního pole. Ale protože se jedná pouze o součást značkovacího jazyka, a nikoliv o skript, jeho chování bude na všech webových stránkách konzistentní. Dodavatelé prohlížečů (nebo autoři rozšíření) také můžou uživatelům nabídnout způsob jak autofokus vypnout.

Kontrola podpory autofokusu využívá detekční techniku č. 2. Pokud váš prohlížeč podporuje autofokus webových formulářů, objekt DOM, který vytváří pro reprezentaci prvku `<input>`, bude mít vlastnost `autofocus` (dokonce i pokud v HTML nepoužijete atribut autofocus). Pokud váš prohlížeč autofokus webových formulářů nepodporuje, objekt DOM, který vytváří pro prvek `<input>`, vlastnost autofocus mít nebude. Podporu autofokusu můžete detekovat pomocí této funkce:

```
function supports_input_autofocus() {
    var i = document.createElement('input');
    return 'autofocus' in i;
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory formulářových polí s autofokusem použít Modernizr (verze 1.1 nebo pozdější).

Zkontrolujte podporu autofokusu

```
if (Modernizr.input.autofocus) {
    // autofokus funguje!
} else {
    // autofokus není podporován :(
    // použijte skriptované řešení
}
```

2.15 Mikrodata

Mikrodata jsou standardizovaným způsobem, jak přidat vašim webovým stránkám sémantiku navíc. Například můžete mikrodata použít pro prohlášení, že je daná fotografie k dispozici v rámci licence Creative Commons. Jak uvidíte v kapitole o distribuované rozšiřitelnosti, můžete použít mikrodata pro kód stránky „O mně“. Prohlížeče, rozšíření prohlížečů a vyhledávače můžou proměnit váš kód pro mikrodata HTML5 ve vizitku vCard, což je standardní formát pro sdílení kontaktních údajů. Můžete také definovat své vlastní slovníky mikrodat.

Standard pro mikrodata HTML5 zahrnuje jak kód HTML (hlavně pro vyhledávače), tak sadu funkcí DOM (hlavně pro prohlížeče). Použít kód pro mikrodata vašim webovým stránkám rozhodně neublíží. Není to nic jiného než pár dobře umístěných atributů a vyhledávače, které atributům pro mikrodata nerozumí, je budou jednoduše ignorovat. Ale pokud se potřebujete dostat k mikrodatům nebo je upravovat pomocí DOM, potřebujete zkontrolovat, zda daný prohlížeč podporuje mikrodata DOM API.

Kontrola podpory HTML5 mikrodat API využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje HTML5 mikrodata API, bude mít globální objekt `document` funkci `getItems()`. Pokud váš prohlížeč mikrodata nepodporuje, funkce `getItems()` nebude definovaná.

```
function supports_microdata_api() {
    return !!document.getItems;
}
```

Modernizr zatím kontrolu podpory mikrodat API nepodporuje, takže musíte použít funkci podobnou té uvedené výše.

2.16 History API

HTML5 history API je standardizovaným způsobem, jak upravit historii prohlížeče pomocí skriptu. Část tohoto API (navigování v historii) bylo k dispozici už v předchozích verzích HTML. Novinkou v HTML5 je možnost přidávat do historie prohlížeče položky a reagovat na to, když jsou tyto položky odstraněny z historie tak, že uživatel zmáčkne tlačítko zpět. To znamená, že URL může i nadále fungovat jako jedinečný identifikátor aktuálního zdroje, a to i v hodně skriptovaných aplikacích, které ani neprovádějí úplné obnovení stránky.

Kontrola podpory HTML5 history API využívá detekční techniku č. 1. Pokud váš prohlížeč podporuje HTML5 history API, bude mít globální objekt `history` funkci `pushState()`. Pokud váš prohlížeč history API nepodporuje, funkce `pushState()` nebude definovaná.

```
function supports_history_api() {
    return !!(window.history && history.pushState);
}
```

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory HTML5 history API použít Modernizr (verzi 1.6 nebo pozdější).

Zkontrolujte podporu historie API

```
if (Modernizr.history) {
    // správa historie funguje!
} else {
    // historie není podporovaná :(
    // použijte skriptované řešení, jako je History.js
}
```

2.17 K dalšímu čtení

Specifikace a standardy:

- prvek `<canvas>`[3]
- prvek `<video>`[4]
- typy `<input>`[5]
- atribut `<input placeholder>`[6]
- atribut `<input autofocus>`[7]
- HTML5 úložiště[8]
- Web Workers[9]
- Offline webové aplikace[10]
- Geolokační API[11]
- Historie relací a navigace[12]

Javascriptové knihovky:

- Modernizr, detekční knihovna pro HTML5[13]
- geo.js, geolokační API wrapper[14]
- HTML5 Cross-browser Polyfills[15]

3 https://html.spec.whatwg.org/multipage/scripting.html#the-canvas-elementa
4 https://html.spec.whatwg.org/multipage/embedded-content.html#video
5 https://html.spec.whatwg.org/multipage/forms.html#states-of-the-type-attribute
6 https://html.spec.whatwg.org/multipage/forms.html#common-input-element-attributes
7 https://html.spec.whatwg.org/multipage/forms.html#association-of-controls-and-forms
8 https://w3c.github.io/webstorage/
9 https://html.spec.whatwg.org/multipage/workers.html
10 https://html.spec.whatwg.org/multipage/browsers.html#offline
11 http://www.w3.org/TR/geolocation-API/
12 https://html.spec.whatwg.org/multipage/browsers.html#history
13 http://modernizr.com/
14 https://code.google.com/p/geo-location-javascript/
15 https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills

Další články a návody:

- Video for Everybody![16]
- A gentle introduction to video encoding[17]
- Video type parameters[18]
- The All-In-One Almost-Alphabetical Guide to Detecting Everything[19]
- Internet Explorer 9 Guide for Developers[20]

16 http://camendesign.com/code/video_for_everybody
17 http://diveintomark.org/tag/give
18 http://wiki.whatwg.org/wiki/Video_type_parameters
19 http://diveintohtml5.info/everything.html
20 http://msdn.microsoft.com/en-us/ie/ff468705.aspx

3. Co to všechno znamená?

3. Co to všechno znamená? – 63

3.1 Začínáme

Tato kapitola si vezme do parády stránku HTML, na které není vůbec nic špatného, a vylepší ji. Některé její části se zkrátí. Jiné se zase prodlouží. Celkově získá více sémantiky. A bude to super.

Tady je ta stránka[1]. Učte se z ní. Žijte jí. Zamilujte si ji. Otevřete si ji v novém panelu a nevracejte se, dokud aspoň jednou nekliknete na „Zobrazit zdrojový kód".

3.2 Doctype

Odshora:

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Tomuto se říká „doctype". Za doctypem se skrývá mnoho historie – a taky alchymie. Při práci na Internet Explorer 5 pro Mac se vývojáři v Microsoftu ocitli před nečekaným problémem. Nová verze jejich prohlížeče zlepšila svou podporu standardů natolik, že se starší stránky přestaly správně zobrazovat. Nebo lépe řečeno, ony se sice zobrazovaly správně (podle specifikací), ale lidé očekávali, že se budou zobrazovat nesprávně. Ty stránky byly totiž vytvořeny tak, aby vyhovovaly vrtochům tehdejších předních prohlížečů, hlavně Netscape 4 a Internet Explorer 4. IE5 pro Mac byl natolik pokročilý, že vlastně rozbil web.

Microsoft přišel s novátorským řešením. Před zobrazením stránky se IE5/Mac podíval na „doctype", který obvykle tvoří první řádek zdrojového kódu HTML (ještě před prvkem `<html>`). Starší stránky (které se spoléhaly na zobrazovací vrtochy starších prohlížečů) obvykle doctype vůbec neměly. IE5 pro Mac zobrazoval tyto stránky stejně jako starší prohlížeče. Aby autoři stránek mohli „aktivovat" podporu nových standardů, museli přejít na novou technologii tím, že před prvek `<html>` přidali správný doctype.

Tato myšlenka se velmi rychle rozšířila a brzo měly všechny hlavní prohlížeče dva režimy: „vrtošivý režim" a „standardní režim". Brzy se to vymklo z rukou. Když se v Mozille pokusili vydat verzi 1.1 jejich prohlížeče, zjistili, že se ve „standardním režimu" zobrazovaly i některé stránky, které se ve skutečnosti spoléhaly na jeden konkrétní vrtoch. Mozilla jednoduše upravila svůj zobrazovací engine tak, aby tento vrtoch zmizel, a najednou přestaly fungovat tisíce stránek. Proto byl vytvořen – a to si nevymýšlím – „téměř standardní režim".

1 http://diveintohtml5.info/examples/blog-original.html

Ve své zásadní práci Activating Browser Modes with Doctype (Aktivace režimů prohlížeče pomocí doctype) Henri Sivonen podává přehled jednotlivých režimů:

Vrtošivý režim
: Ve vrtošivém režimu prohlížeče porušují současné webové specifikace, aby se vyhnuly „rozbíjení" stránek vytvořených podle zvyklostí, které panovaly v druhé polovině 90. let.

Standardní režim
: Ve standardním režimu se prohlížeče pokoušejí k vyhovujícím dokumentům chovat tak, aby to odpovídalo specifikacím, do takové míry, do jaké to konkrétní prohlížeč dovoluje. HTML5 tomuto režimu říká „režim bez vrtochů".

Téměř standardní režim
: Firefox, Safari, Chrome, Opera (od verze 7.5) a IE8 mají také režim známý jako „téměř standardní režim", který provádí vertikální dimenzování buněk tabulky tradičně a ne přísně podle specifikací CSS2. HTML5 tomuto režimu říká „režim s omezenými vrtochy".

(Měli byste si přečíst i zbytek Henriho článku, protože to tady říkám velmi zjednodušeně. Dokonce i v IE5 pro Mac se objevovalo několik starších doctypů, které se nepočítaly jako přechod na podporu standardů. Seznam vrtochů se postupně rozšiřoval a spolu s tím přibývalo i doctypů, které spouštěly „vrtošivý režim". Když jsem to zkoušel počítat naposledy, napočítal jsem 5 doctypů, které spouštěly „téměř standardní režim" a 73, které spouštěly „vrtošivý režim". Ale asi jsem na nějaké zapomněl a nebudu se ani pouštět do popisu šíleností, které dělá Internet Explorer 8, aby mohl přepínat mezi čtyřmi – slovy: čtyřmi zobrazovacími režimy. Tady je to na vývojovém diagramu[2]. Brr. Odstup, satane!)

Tak, kde jsme to skončili? Aha, u doctypu:

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Tohle je zrovna jeden z 15 doctypů, které spouštějí „standardní režim" ve všech moderních prohlížečích. Není na něm nic špatného. Pokud se vám líbí, můžete u něho zůstat. Nebo ho můžete změnit na doctype HTML5, který je kratší a sympatičtější a rovněž spouští „standardní režim" ve všech moderních prohlížečích.

Tohle je doctype HTML5:

```
<!DOCTYPE html>
```

To je vše. Jenom 15 znaků. Je tak jednoduchý, že ho můžete napsat ručně a nesplést se přitom.

2 https://hsivonen.fi/doctype/ie8-mode.png

3.3 Kořenový prvek

Stránka HTML je sérií do sebe vnořených prvků. Celá struktura stránky je jako strom. Některé prvky jsou „sourozenci“, jako dvě větve, které vycházejí ze stejného kmene. Některé prvky jsou zase „potomky“ jiných prvků, jako dvě větvičky, které vycházejí ze stejné větší větve. (Funguje to i naopak: prvek, který obsahuje jiné prvky, se nazývá „mateřský“ uzel svých bezprostředních podřízených prvků a „předek“ svých vnuků.) Prvkům, které nemají potomky, se říká „listové“ uzly. Nejvzdálenější prvek, který je předkem všech ostatních prvků stránky, se nazývá „kořenový prvek“. Kořenovým prvkem stránky HTML je vždy `<html>`.

Na této příkladové stránce[3] vypadá kořenový prvek takto:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    lang="en"
    xml:lang="en">
```

Na tomto kódu není nic špatného. Zase platí, že pokud se vám líbí, můžete u něho zůstat. Je to regulérní HTML5. Ale některé jeho části už v HTML5 nejsou potřeba, takže můžete ušetřit pár bajtů tím, že je odstraníte.

První věcí k diskusi je atribut `xmlns`. Jedná se o pozůstatek XHTML 1.0. Říká nám, že prvky na této stránce jsou ve jmenném prostoru XHTML `http://www.w3.org/1999/xhtml`. Ale prvky v HTML5 jsou vždycky v tomto jmenném prostoru, takže už to nemusíte vysloveně prohlašovat. Vaše stránka HTML5 bude ve všech prohlížečích fungovat úplně stejně, ať už bude či nebude obsahovat tento atribut.

Když vypustíme atribut `xmlns`, zůstane nám tento kořenový prvek:

```
<html lang="en" xml:lang="en">
```

Oba zde použité atributy `lang` a `xml:lang` definují jazyk této stránky HTML. (`en` označuje angličtinu. Nepíšete anglicky? Najděte si kód svého jazyka[4].) Proč se pro jednu věc používají dva atributy? Opět se jedná o pozůstatek XHTML. V HTML5 hraje roli pouze atribut `lang`. Jestli chcete, můžete atribut `xml:lang` zachovat, ale pokud to uděláte, musíte zajistit, aby obsahoval stejnou hodnotu jako atribut `lang`.

> Aby se ujasnilo přecházení do a z XHTML, můžou autoři stránek v prvcích HTML v dokumentech HTML určit atribut, který nebude ležet v žádném jmenném prostoru, nebude obsahovat žádný prefix a bude mít doslovné localname (místní jméno) „xml:lang“. Tyto

3 http://diveintohtml5.info/examples/blog-original.html

4 http://www.w3.org/International/questions/qa-choosing-language-tags

atributy však musejí být určeny pouze v případě, že je určen i atribut `lang`, který neleží v žádném jmenném prostoru, a oba atributy musejí mít stejnou hodnotu při porovnání v kódové tabulce ASCII, které nerozlišuje velká a malá písmena. Atribut, který neleží v žádném jmenném prostoru, neobsahuje žádný prefix a má doslovné localname „xml:lang“, nemá žádný vliv na zpracování jazyka.

Jste připraveni se tohoto atributu zbavit? Je to v pořádku, prostě ho nechte plavat. Už se chystá, odchází… a je pryč! Takhle nám zůstal tento kořenový prvek:

```
<html lang="en">
```

A to je všechno, co vám o tom řeknu.

3.4 Prvek <head>

Prvním potomkem kořenového prvku je obvykle prvek `<head>`. Prvek `<head>` obsahuje metadata – informace o stránce spíše než tělo stránky samotné. (Tělo stránky je pochopitelně obsaženo v prvku `<body>`, tj. „tělo“.) Prvek `<head>` je sám o sobě docela nudný a v HTML5 neprodělal žádné zajímavé změny. Něco zajímavého najdeme ovšem uvnitř prvku `<head>`. K tomu si znovu otevřeme naši příkladovou stránku:

```
<head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
    <title>Můj blog</title>
    <link rel="stylesheet" type="text/css"
    href="style-original.css" />
    <link rel="alternate" type="application/atom+xml"
                          title="Můj blog - kanál"
                          href="/feed/" />
    <link rel="search" type="application/opensearchdescription+xml"
                          title="Můj blog - vyhledávání"
                          href="opensearch.xml" />
    <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

Nejdříve se podíváme na prvek `<meta>`.

3.5 Znakové sady

Když se řekne „text“, asi vás napadne něco jako „znaky a symboly, které vidím na obrazovce svého počítače.“ Ovšem počítače nepoužívají znaky a symboly, ale bity a bajty. Každý kousek textu, který jste kdy viděli na obrazovce počítače, je ve skutečnosti uložen v konkrétní *znakové sadě*. Existují stovky různých znakových sad, z nichž některé jsou optimalizovány pro jeden konkrétní jazyk, jako je ruština, čínština nebo angličtina, a jiné můžou být použity pro více jazyků. Jednoduše řečeno: znaková sada představuje propojení mezi tím, co vidíte na obrazovce, a tím, co si váš počítač skutečně ukládá do paměti a na disk.

Ve skutečnosti je to mnohem složitější. Stejný znak se může objevovat ve více sadách, ale každá sada může pro jeho uložení do paměti nebo na disk používat jinou sekvenci bajtů. Proto si znakovou sadu představte jako dešifrovací klíč pro text. Kdykoliv vám někdo bude strkat sekvenci bajtů a tvrdit o ní, že je to „text“, musíte vědět, jakou znakovou sadu autor použil, abyste mohli dekódovat bajty v znaky a zobrazit je (případně zpracovat apod.).

Jak váš prohlížeč určí znakovou sadu proudu bajtů, který vysílá webový server? Jsem rád, že se na to ptáte. Pokud znáte hlavičky HTTP, možná jste viděli hlavičku, která vypadala takhle:

```
Content-Type: text/html; charset="utf-8"
```

Stručně řečeno nám tento řádek sděluje, že si webový sever myslí, že vám posílá dokument HTML a že tento dokument používá znakovou sadu `UTF-8`. Bohužel v té celé velkolepé kaši, které se říká World Wide Web, má jen málo autorů kontrolu nad svým serverem HTTP. Vezměte si třeba službu Blogger: obsah sice vytvářejí jednotlivci, ale servery provozuje Google. Proto v HTML 4 existoval způsob, jak určit znakovou sadu přímo v dokumentu HTML. Tohle jste asi taky viděli:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Tento řádek nám v krátkosti říká, že si autor tohoto webu myslí, že vytvořil dokument HTML používající znakovou sadu `UTF-8`.

Obě tyto techniky fungují i v HTML5. Preferovanou metodou je hlavička HTTP, která přebíjí tag `<meta>`, pokud je přítomen. Ale ne každý umí vytvářet hlavičky HTTP, a proto se stále používá i tag `<meta>`. V HTML5 se dokonce trochu zjednodušil a vypadá teď takhle:

```
<meta charset="utf-8" />
```

Funguje to ve všech prohlížečích. Odkud se tato zkrácená syntax vzala? Tady je nelepší vysvětlení, co jsem našel:

Důvodem pro kombinaci atributů `<meta charset>` je, že už ji uživatelské agenty vlastně používají, protože lidé zapomínají dávat věci do uvozovek, například:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;"
charset="ISO-8859-1">
```

Můžete se podívat i na pár testovacích případů `<meta charset>`, pokud nevěříte tomu, že to už prohlížeče dělají.

PTEJTE SE PROFESORA ZNAČKY

Q: Nikdy nepoužívám neobvyklé znaky. Musím i přesto určit svou znakovou sadu?

A: Ano! Znakovou sadu byste měli určit *vždy*, u každé stránky HTML, kterou vytvoříte. Neurčení znakové sady může vést k chybám zabezpečení.

Abychom to shrnuli: znakové sady jsou komplikované a nijak je neusnadnily desítky let špatně napsaného softwaru, který používali autoři odkojení metodou „zkopíruj a vlož“. Znakovou sadu byste měli určit **vždy**, **u každého** dokumentu HTML, jinak se začnou dít nepěkné věci. Je jedno, jestli to uděláte pomocí hlavičky HTTP `Content-Type`, pomocí `<meta http-equiv>` nebo kratší verze `<meta charset>`, ale prostě to udělejte. Web vám poděkuje.

3.6 Vztahy k odkazu

Obyčejné odkazy (`<a href>`) jednoduše odkazují na jinou stránku. Vztahy k odkazu jsou způsobem, jak vysvětlit, *proč* na jinou stránku odkazujete. Dokončují větu „Odkazuji na jinou stránku, protože…“

- ... je to šablona kaskádových stylů obsahující pravidla, které by měl váš prohlížeč použít na tento dokument.
- ... je to kanál se stejným obsahem jako tato stránka, ale ve standardním odebíratelném formátu.
- ... je to překlad této stránky do jiného jazyka.
- ... je to tentýž obsah, ale ve formátu PDF.

- ... je to další kapitola online knihy, jejíž součástí je i tato stránka.

A tak dále. HTML5 rozděluje vztahy k odkazu na dvě kategorie:

> Pomocí prvku odkaz můžou být vytvořeny dvě kategorie odkazů. **Odkazy na vnější zdroje** jsou odkazy, které se používají pro vylepšení daného dokumentu, zatímco **hypertextové odkazy** odkazují na jiné dokumenty. (…)
>
> Konkrétní chování odkazů k vnějším zdrojům záleží na jejich konkrétním vztahu, jak ho definuje příslušný typ odkazu.

Z příkladů, které jsem právě uvedl, pouze první (`rel="stylesheet"`) odkazuje na vnější zdroj. Zbytek tvoří hypertextové odkazy na jiné dokumenty. Můžete na ně kliknout nebo nemusíte, ale každopádně nejsou potřeba k tomu, abyste si mohli prohlížet právě otevřenou stránku.

Nejčastěji můžeme vztahy k odkazu vidět u prvků `<link>` v rámci `<head>` stránky. Některé vztahy k odkazu se mohou používat i u prvků `<a>`, ale i tam, kde je to povoleno, to není obvyklé. HTML5 také umožňuje určité vztahy k odkazu u prvků `<area>`, ale to je snad ještě vzácnější. (HTML 4 neumožňovalo atribut `rel` u prvků `<area>`.) Podívejte se na kompletní tabulku vztahů k odkazu[5], abyste zjistili, kde můžete použít konkrétní hodnoty `rel`.

PTEJTE SE PROFESORA ZNAČKY

Q: Můžu vytvářet své vlastní vztahy k odkazu?

A: Nápadů na nové vztahy k odkazu je, zdá se, nevyčerpatelná zásoba. Ve snaze zabránit lidem, aby si něco vymýšleli jen tak, vede mikroformátová komunita seznam navrhnutých hodnot `rel` a specifikace HTML definuje proces, jak si tyto hodnoty nechat schválit.

3.7 Rel = stylesheet

Podívejme se na první vztah k odkazu na naší vzorové stránce:

```
<link rel="stylesheet" href="style-original.css" type="text/css" />
```

Tohle je nejčastěji používaný vztah k odkazu na světě (a to doslova). `<link rel="stylesheet">` slouží pro odkazování na pravidla CSS, která jsou uložena ve zvláštním souboru. Jedna z drob-

5 https://html.spec.whatwg.org/multipage/semantics.html#linkTypes

ných optimalizací, kterou můžete v HTML5 provést, je vynechání atributu `type`. Pro celý web existuje pouze jediný jazyk šablony stylů, CSS, takže je takto nastavena výchozí hodnota atributu `type`. Funguje to ve všech prohlížečích. (Domnívám se, že jednoho dne by někdo mohl přijít s novým jazykem šablon stylů, ale pokud se to stane, tak prostě vraťte atribut `type`).

```
<link rel="stylesheet" href="style-original.css" />
```

3.8 Rel = alternate

Pokračujme s naší vzorovou stránkou:

```
<link rel="alternate"
    type="application/atom+xml"
    title="Můj blog - kanál"
    href="/feed/" />
```

Tento vztah k odkazu je také docela běžný. V kombinaci s RSS nebo typem média Atom v atributu `type` umožňuje `<link rel="alternate">` něco, čemu se říká „automatické vyhledání kanálu“. To umožňuje čtečkám kanálu se syndikovaným obsahem (jako Google Reader) zjistit, že má stránka novinkový kanál nejnovějších článků. Některé prohlížeče také podporují automatické vyhledání kanálu tím, že vedle URL zobrazují speciální ikonu. (Na rozdíl od `rel="stylesheet"` je zde atribut `type` důležitý. Nezapomeňte na něj!)

Vztah k odkazu `rel="alternate"` byl vždy zvláštní směsí případů užití, a to i v HTML 4. V HTML5 byla jeho definice vyjasněna a rozšířena tak, aby přesněji popsala existující webový obsah. Jak jste právě viděli, použití `rel="alternate"` společně s `type=application/atom+xml` signalizuje, že má daná stránka Atom feed (kanál ve formátu Atom). Ale `rel="alternate"` můžete použít také spolu s jinými atributy `type`, abyste signalizovali, že je obsah v jiném formátu, např. PDF.

HTML5 také konečně řeší dlouhotrvající zmatek ohledně toho, jak odkazovat na překlady dokumentů. HTML 4 navrhuje použití atributu `lang` v kombinaci s `rel="alternate"` pro specifikaci jazyka dokumentu, na který se odkazuje, ale to je nesprávné. Dokument Errata HTML 4 uvádí čtyři vyložené chyby ve specifikaci HTML 4. Jednou z těchto chyb je specifikování jazyka dokumentu připojeného pomocí `rel="alternate"`. Správný způsob, popsaný v Erratech HTML 4 a nyní i v HTML5, je použít atribut `hreflang`. Bohužel tato errata nebyla nikdy začleněna do specifikace HTML 4, protože v té době už nikdo v pracovní skupině W3C HTML na HTML nepracoval.

3.9 Další vztahy k odkazu v HTML5

`rel="author"` se používá pro odkaz na informace o autorovi stránky. Může odkazovat na adresu `mailto:`, ale ne nutně. Může prostě odkazovat na kontaktní formulář nebo na stránku „O autorovi."

HTML 4 definovalo hodnoty `rel="start"`, `rel="prev"` a `rel="next"` pro vymezení vztahů mezi stránkami, které tvoří sérii (jako třeba kapitoly v knížce nebo i příspěvky na blogu). Jediná, která z nich kdy byla používána správně, byla hodnota `rel="next"`. Lidé používali `rel="previous"` místo `rel="prev"`, `rel="begin"` a `rel="first"` místo `rel="start"`, `rel="end"` místo `rel="last"`.

No a ještě si navíc – úplně sami – vymysleli `rel="up"` pro odkazování na „mateřskou" stránku. Nejlepší způsob, jak přemýšlet o `rel="up"`, je podívat se na vaši drobečkovou navigaci (nebo si ji aspoň představit). Vaše domovská stránka je pravděpodobně prvním „drobečkem" a právě otevřená stránka posledním. `rel="up"` odkazuje na předposlední stránku v drobečkové navigaci.

HTML5 obsahuje `rel="next"` a `rel="prev"`, stejně jako HTML 4, a podporuje `rel="previous"` pro zpětnou kompatibilitu. Specifikace původně obsahovala i `rel="first"`, `rel="last"` a `rel="up"`. „Na základě nedostatku zájmu realizátorů a uživatelů" se ovšem pracovní skupina HTML rozhodla tyto hodnoty ze specifikace vynechat.

`rel="icon"` je druhý nejoblíbenější vztah k odkazu po `rel="stylesheet"`. Obvykle se vyskytuje spolu s `shortcut`, jako zde:

```
<link rel="shortcut icon" href="/favicon.ico">
```

Všechny hlavní prohlížeče podporují toto použití, které stránce přiřazuje malou ikonu. Tato ikona se obvykle zobrazuje v adresním řádku prohlížeče vedle URL nebo v panelu prohlížeče, případně v obojím.

Další nová věc v HTML5: společně se vztahem `icon` může být použit atribut `sizes`, aby udal velikost odkazové ikony.

S `rel="license"` přišla mikroformátová komunita. Tato hodnota „udává, že odkazovaný dokument obsahuje autorskou licenci, pod níž je daná stránka poskytována."

`rel="nofollow"` udává, že odkaz nebyl schválen původním autorem nebo vydavatelem stránky, nebo že odkaz na dokument je přítomen v prvé řadě kvůli komerčnímu vztahu mezi autory daných dvou webů. Tuto hodnotu vymyslel Google a standardizovala se v rámci mikroformátové komunity. WordPress řadí `rel="nofollow"` mezi odkazy přidané komentátory. Myšlen-

ka byla taková, že když se „nofollow“ (nesledované) odkazy nebudou předávat do hodnocení PageRank, spameři přestanou na blogy přidávat spamující komentáře. To se sice nestalo, ale `rel="nofollow"` nám zůstal.

`rel="noreferrer"` „udává, že při klepnutí na odkaz nemají uniknout žádné informace o referrerovi (URL, ze kterého byla webová stránka navštívena).“ WebKit podporuje `rel="noreferrer"`, takže funguje v Google Chrome a Safari (a snad také v dalších prohlížečích založených na WebKit).

`rel="prefetch"` „udává, že je užitečné předběžně stáhnout daný zdroj a uložit ho do mezipaměti, protože je vysoce pravděpodobné, že uživatel bude tento zdroj potřebovat.“ Vyhledávače občas na stránku s výsledky hledání přidávají `<link rel="prefetch" href="`*`URL prvního výsledku hledání`*`">`, pokud mají pocit, že je první výsledek výrazněji populárnější než ostatní. Například: ve Firefoxu si otevřete Google a vyhledejte CNN, podívejte se na zdrojový kód stránky a hledejte klíčové slovo `prefetch`. Jediným prohlížečem je v současnosti Mozilla Firefox, který podporuje `rel="prefetch"`.

`rel="search"` „udává, že odkazovaný dokument poskytuje speciálně rozhraní pro prohledávání dokumentu a s ním spojených zdrojů.“ Pokud konkrétně chcete, aby `rel="search"` dělal něco užitečného, měl by odkazovat na dokument OpenSearch, který popisuje, jak má prohlížeč konstruovat URL, aby na právě otevřené stránce vyhledal dané klíčové slovo. OpenSearch (a odkazy s `rel="search"`, které okazují na dokumenty s popisy) je podporován Internet Explorerem od verze 7, Mozillou Firefoxem od verze 2 a Google Chromem.

`rel="tag"` „udává, že štítek, který odkazovaný dokument reprezentuje, platí pro právě otevřený dokument.“ Označování „štítků“ (klíčových slov pro kategorie) atributem `rel` vymysleli v Technorati, aby se jim lépe řadily do kategorií příspěvky na blogu. Starší blogy a návody o nich proto mluvily jako o „štítcích Technorati“. (Ano, čtete správně: komerční společnost přesvědčila celý svět, aby přidával metadata, která jí usnadňovala práci. Tomu říkám dobrá práce!) Tato syntax byla později standardizována v mikroformátové komunitě, kde se jmenovala prostě `rel="tag"`. Většina blogovacích systémů, které umožňují s jednotlivými příspěvky spojovat kategorie, klíčová slova nebo štítky, je budou označovat pomocí odkazů `rel="tag"`. Prohlížeče s nimi nic zvláštního nedělají; jsou vytvořené hlavně pro vyhledávače, kterým signalizují, o čem daná stránka je.

3.10 Nové sémantické prvky v HTML5

HTML5 není pouze o zkracování stávajících kódů (i když i toho dělá dost). Také definuje nové sémantické prvky.

`<section>`
: Prvek `section` představuje obecný dokument nebo sekci aplikace. Sekce v tomto kontextu znamená tematické zařazení obsahu do skupiny, obvykle s nadpisem. Jako příklady sekcí můžeme uvést kapitoly, panely v dialogovém okně nebo očíslované oddíly v diplomce. Domovská stránka webu může být rozdělena do sekcí pro úvod, novinky a kontaktní údaje.

`<nav>`
: Prvek `nav` představuje sekci stránky, která odkazuje na jiné stránky nebo části stránky: sekci s navigačními odkazy. Ne všechny skupiny odkazů na stránce musejí být v prvku `nav` – pro prvek `nav` jsou vhodné pouze sekce, které obsahují větší navigační bloky. Je třeba běžné, že se v zápatí nachází krátký seznam odkazů na některé části stránky, jako jsou podmínky použití, domovská stránka a stránka s autorskými právy. V takových případech stačí použít pouze prvek `footer` bez prvku `nav`.

`<article>`
: Prvek `article` představuje součást stránky, která se skládá z autonomní kompozice v dokumentu, na stránce, v aplikaci nebo na webu a která má být nezávisle distribuovatelná nebo znovupoužitelná, např. pomocí syndikace. Může to být příspěvek na fóru, článek v časopise nebo v novinách, příspěvek na blogu, komentář uživatele, interaktivní widget nebo gadget nebo jakýkoliv jiný nezávislý obsah.

`<aside>`
: Prvek `aside` představuje sekci stránky skládající se z obsahu, který okrajově souvisí s obsahem v okolí prvku `aside` a může se považovat za nezávislý na tomto obsahu. Takové sekce se v tištěné typografii často znázorňují jako postranní sloupce. Tento prvek může být použit pro typografické efekty, např. zvýrazněnou citaci nebo postranní sloupec, pro reklamu, pro skupiny prvků `nav` nebo pro jakýkoliv jiný obsah, který chcete oddělit od hlavního obsahu stránky.

`<hgroup>`
: Prvek `hgroup` představuje nadpis sekce. Tento prvek se používá pro seskupení sady prvků `h1`–`h6` v případě, že má záhlaví několik úrovní, jako jsou podnadpisy, alternativní názvy nebo slogany.

`<header>`
: Prvek `header` představuje skupinu úvodních či navigačních pomůcek. Prvek `header` by měl obvykle obsahovat nadpis sekce (prvek `h1`–`h6` ze skupiny `hgroup`), ale není to povinné. Prvek může být použit pro zabalení obsahu sekce, vyhledávacího formuláře nebo souvisejících log.

`<footer>`
: Prvek `footer` představuje zápatí pro nejbližší rodičovský strukturující obsah nebo strukturující kořenový prvek. Zápatí obvykle obsahuje informace o sekci: kdo ji napsal, odkazy na příbuzné dokumenty, údaje o autorských právech a podobně. Zápatí se nemusí nutně vyskytovat na konci sekce, i když většinou tomu tak je. Pokud obsahuje prvek `footer` celé sekce, jsou to obvykle dodatky, rejstříky, dlouhé kolofony, zdlouhavé licenční dohody a podobný obsah.

`<time>`
: Prvek `time` představuje buď čas na 24hodinových hodinách, nebo přesné datum v proleptickém gregoriánském kalendáři, volitelně doplněné o čas a časové pásmo.

`<mark>`
: Prvek `mark` představuje část textu v jednom dokumentu, která je označená nebo zvýrazněná pro referenční účely.

Chápu, že už se nemůžete dočkat, až tyto nové prvky začnete používat, jinak byste tuhle kapitolu nečetli. Ale nejdřív si ještě musíme udělat jednu odbočku.

3.11 Dlouhá odbočka na téma „jak prohlížeče zacházejí s neznámými prvky“

Každý prohlížeč má seznam prvků HTML, které podporuje. Například seznam Mozilly Firefox najdete uložený na `nsElementTable.cpp`. S prvky, které nejsou na seznamu, je nakládáno jako s „neznámými prvky.“ S neznámými prvky jsou dva zásadní problémy:

Jak by měl daný prvek vypadat? Ve výchozím nastavení má `<p>` mezery nahoře i dole, `<blockquote>` má odsazený levý okraj a `<h1>` se zobrazuje větším písmem. Ale jaké výchozí styly by měly být použity na neznámé prvky?

Jak by měl vypadat DOM daného prvku? `nsElementTable.cpp` od Mozilly obsahuje informace o tom, jaké druhy jiných prvků může každý prvek obsahovat. Pokud použijete např. tagy `<p><p>`, druhý prvek pro odstavec implicitně uzavírá první, takže se prvky chovají jako sourozenci, nikoliv jako rodič a dítě. Ale pokud napíšete `<p><span>`, `span` automaticky odstavec neuzavírá, protože Firefox ví, že je `<p>` blokový prvek, který může obsahovat řádkový prvek `<span>`. Proto se v DOMu `<span>` zobrazuje jako dítě `<p>`.

Různé prohlížeče se s tímto problémem vypořádávají různě. (To je ale překvapení, co?) Z nejpoužívanějších prohlížečů má nejproblematičtější řešení Microsoft Internet Explorer, ale každý prohlížeč tady potřebuje trochu píchnout.

Odpověď na první otázku by měla být jednoduchá: nepropůjčovat neznámým prvkům žádný zvláštní vzhled. Nechat je prostě převzít ty vlastnosti CSS, které platí, kdykoliv se objeví na stránce, a nechat autory určit všechny styly v CSS. Tohle víceméně funguje, ovšem s jedním drobným zádrhelem, o kterém byste měli vědět.

PROFESOR ZNAČKA ŘÍKÁ

Všechny prohlížeče zobrazují neznámé prvky v řádku, tj. jako kdyby měly pravidlo CSS `display:inline`.

HTML5 definuje několik nových blokových prvků. To znamená, že můžou obsahovat jiné blokové prvky a prohlížeče kompatibilní s HTML5 jim budou automaticky přiřazovat styl `display:block`. Pokud chcete tyto prvky používat ve starších prohlížečích, budete muset definovat styl zobrazení ručně:

```
article,aside,details,figcaption,figure,
footer,header,hgroup,menu,nav,section {
    display:block;
}
```

(Tento kód je převzat z HTML5 Reset Stylesheet Riche Clarka, kde se můžete dozvědět spoustu věcí, které jsou nad rámec této kapitoly.)

Ale počkejte, bude to ještě horší! Před verzí 9 nepřiřazoval Internet Explorer neznámým prvkům *vůbec žádné* styly. Například pokud jste měli tento kód:

```
<style type="text/css">
    article { display: block; border: 1px solid red }
</style>
...
<article>
<h1>Vítejte v Initechu</h1>
<p>Toto je váš <span>první den</span>.</p>
</article>
```

Internet Explorer (do verze 8 včetně) nebude s prvkem `<article>` zacházet jako s blokovým prvkem, ani kolem článku neudělá červenou čáru. Všechny styly budou prostě ignorovány. Internet Explorer 9 tento problém řeší.

Druhým problémem je DOM, který prohlížeče vytvářejí při setkání s neznámými prvky. Nejproblematičtějším prohlížečem je tady opět Internet Explorer ve svých starších verzích (před verzí 9, která řeší i tento problém.) Pokud IE8 vysloveně nerozpozná název prvku, vloží prvek do DOMu *jako prázdný uzel bez potomků*. Všechny prvky, od kterých byste čekali, že budou přímými potomky neznámého prvku, budou ve skutečnosti vloženy jako sourozenci.

Tady máte trochu umění ASCII, abyste viděli ten rozdíl. Tohle je DOM, tak jak ho diktuje HTML5:

```
article
|
+--h1 (potomek tagu article)
| |
| +--textový uzel "Vítejte v Initechu"
|
+--p (potomek tagu article, sourozenec h1)
   |
   +--textový uzel "Toto je váš"
   |
   +--span
   | |
   | +--textový uzel "první den"
   |
   +--textový uzel "."
```

Ovšem tohle je DOM, který ve skutečnosti vytvoří Internet Explorer:

```
article (bez potomků)
h1 (sourozenec tagu article)
|
+--textový uzel "Vítejte v Initechu"
p (sourozenec h1)
|
+--textový uzel "Toto je váš"
|
+--span
| |
| +--textový uzel "první den"
|
+--textový uzel "."
```

Existuje způsob, jak tenhle problém báječně obejít. Pokud vytvoříte fiktivní prvek `<article>` pomocí JavaScriptu předtím, než ho použijete na stránce, Internet Explorer jako mávnutím kouzelného proutku prvek `<article>` rozpozná a nechá vás určit jeho styl pomocí CSS. Dokonce ten fiktivní prvek ani nemusíte vkládat do DOMu. Stačí pouze prvek vytvořit jednou (na každé stránce), a IE se naučí přiřadit styl prvku, který nerozeznává.

```
<html>
<head>
<style>
    article { display: block; border: 1px solid red }
</style>
<script>document.createElement("article");</script>
</head>
<body>
<article>
<h1>Vítejte v Initechu</h1>
<p>Toto je váš<span>první den</span>.</p>
</article>
</body>
</html>
```

Tohle funguje ve všech verzích Internet Exploreru až zpět k verzi IE 6! Tuto techniku můžeme rozšířit na vytvoření fiktivních kopií všech nových prvků HTML5 najednou – jak říkám, do DOMu se vůbec nevkládají, takže tyto fiktivní prvky ani neuvidíte – a pak je můžete normálně začít používat, bez toho, abyste si museli dělat starosti s prohlížeči, které nezvládají HTML5.

Přesně tohle udělal Remy Sharp se svým skriptem výstižně pojmenovaným Skript umožňující HTML5. Od chvíle, co jsem začal psát tuto knihu, ten skript prošel už více než tuctem úprav, ale tohle je jeho základní myšlenka:

```
<!--[if lt IE 9]>
<script>
    var e = ("abbr,article,aside,audio,canvas,datalist,details," +
        "figure,footer,header,hgroup,mark,menu,meter,nav,output," +
        "progress,section,time,video").split(',');
    for (var i = 0; i < e.length; i++) {
        document.createElement(e[i]);
    }
</script>
<![endif]-->
```

Části `<!--[if lt IE 9]>` a `<![endif]-->` jsou podmíněné komentáře. Internet Explorer si je vykládá jako podmínku s `if`: „Pokud je daný prohlížeč Internet Explorer verze nižší než 9, pak se spustí tento blok." Všechny ostatní prohlížeče se budou k celému bloku chovat jako ke komentáři HTML. Výsledek je takový, že Internet Explorer (do verze 8 včetně) spustí tento skript, ale ostatní prohlížeče ho budou úplně ignorovat. Díky tomu se vaše stránka načte rychleji v prohlížečích, které tento hack nepotřebují.

Javascriptový kód sám o sobě je relativně přímočarý. Proměnná `e` skončí jako pole řetězců typu `"abbr"`, `"article"`, `"aside"` a tak dále. Následně toto pole projdeme cyklem a vytvoříme jednotlivé pojmenované prvky vyvoláním `document.createElement()`. Ale protože ignorujeme vrácenou hodnotu, prvky se nevloží do DOMu. Ovšem stačí to na to, aby Internet Explorer s těmito prvky zacházel tak, jak chceme, až prvky později doopravdy vložíme na stránku.

To „později" je tam důležité. Skript musí být na vrchu vaší stránky, nejlépe ve vašem prvku `<head>`, a nikoliv na konci. Internet Explorer tak spustí skript předtím, než analyzuje vaše tagy a atributy. Pokud dáte tento skript až na konec stránky, bude to příliš pozdě. Internet Explorer chybně interpretuje váš kód a vytvoří špatný DOM a skript umístěný na konci stránky ho nedonutí vrátit se a opravit to.

Remy Sharp tento kód „zmenšil" a umístil na Google Project Hosting. (Pokud vás to zajímá, tak skript samotný je open-source s licencí MIT, takže ho můžete použít pro libovolný projekt). Pokud chcete, můžete ze skriptu udělat i odkaz směřující přímo na hostovanou verzi, a to takto:

```
<head>
    <meta charset="utf-8" />
    <title>Můj blog</title>
    <!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
</script>
    <![endif]-->
</head>
```

Tak a teď jsme připraveni začít používat nové sémantické prvky v HTML5.

3.12 Hlavičky

Vraťme se k naší vzorové stránce. Zaměřme se teď pouze na hlavičky:

```
<div id="header">
```

```
    <h1>Můj blog</h1>
    <p class="tagline">Stálo mne mnoho úsilí, aby to bylo snadné.
    </p>
</div>

…

<div class="entry">
    <h2>Cestovní den</h2>
</div>

…

<div class="entry">
    <h2>Jedu do Prahy!<h2>
</div>
```

Na tomto kódu není nic špatného. Jestli se vám líbí, můžete u něho zůstat. Je to regulérní HTML5. Ale HTML5 nabízí pár nových sémantických prvků pro hlavičky a sekce.

Ze všeho nejdřív se zbavme toho `<div id="header">`. Je to běžná formule, ale nic neznamená. Prvek `div` nemá žádnou definovanou sémantiku, stejně jako atribut `id`. (Uživatelské agenty nemají povoleno z hodnoty atributu `id` vyvozovat žádný význam). Můžete ji klidně změnit na `<div id="shazbot">` a budete mít stále stejnou sémantickou hodnotu, to znamená žádnou.

HTML5 pro tento účel definuje prvek <header>. Specifikace HTML5 využívá příklady použití prvku `<header>` v praxi. Takhle by to vypadalo na naší vzorové stránce:

```
<header>
    <h1>Můj blog</h1>
    <p class="tagline">Stálo mne mnoho úsilí, aby to bylo snadné.
    </p>
    …
</header>
```

To je dobré. Všem, které to zajímá, říká, že je tohle hlavička. Ale co ten slogan (tagline)? Další běžná formule, pro kterou až dosud neexistoval standardizovaný kód. Zapsání do kódu je v tomto případě obtížné. Slogan je podobný podnadpisu, ale je „připevněn“ k hlavnímu nadpisu. Je to vlastně podnadpis, který nevytváří svou vlastní sekci.

Prvky hlavičky jako <h1> a <h2> vaši stránku strukturují. Spolu vytvářejí osnovu, která slouží k vizualizaci vaší stránky (nebo k orientaci na ní). Čtečky obrazovek používají osnovy dokumentů, aby pomohly nevidomým uživatelům orientovat se na stránce. Existují online nástroje a rozšíření prohlížečů, které vám můžou pomoct vizualizovat osnovu dokumentu.

V HTML 4 představovaly prvky <h1>–<h6> *jediný* způsob, jak vytvořit osnovu dokumentu. Osnova vzorové stránky vypadá následovně:

```
Můj blog (h1)
    |
    +--Cestovní den (h2)
    |
    +--Jedu do Prahy! (h2)
```

To je v pořádku, ale znamená to, že neexistuje žádný způsob, jak napsat kód pro slogan „Stálo mne mnoho úsilí, aby to bylo snadné.“ Pokud pro něj zkusíme použít tag <h2>, přidá to do osnovy dokumentu fantomový uzel:

```
Můj blog (h1)
    |
    +--Stálo mne mnoho úsilí, aby to bylo snadné. (h2)
    |
    +--Cestovní den (h2)
    |
    +--Jedu do Prahy! (h2)
```

Ale to neodpovídá struktuře dokumentu. Slogan nepředstavuje sekci; je to pouhý podnadpis.

Možná bychom mohli pro slogan použít <h2> a pro každý název článku <h3>? Ne, to je ještě horší:

```
Můj blog (h1)
    |
    +--Stálo mne mnoho úsilí, aby to bylo snadné. (h2)
        |
        +--Cestovní den (h3)
        |
        +--Jedu do Prahy! (h3)
```

V osnově našeho dokumentu i nadále zůstává fantomový uzel, ale teď navíc „ukradl“ potomky, které po právu náleží kořenovému uzlu. A v tom spočívá ten problém: HTML 4 nenabízí žádný

způsob, jak zapsat podnadpis, aniž by se přidal k osnově dokumentu. I kdybychom se rozkrájeli, „Stálo mne mnoho úsilí, aby to bylo snadné.“ v tom schématu prostě zůstane. A proto jsme skončili se zápisem bez sémantického smyslu, jako je `<p class="tagline">`.

HTML5 nabízí řešení: prvek `<hgroup>`. Prvek `<hgroup>` dokáže zabalit dva nebo více *propojených* nadpisových prvků. Co znamená „propojených“? Propojené prvky jsou takové, které spolu v osnově dokumentu vytvářejí pouze jediný uzel.

Podívejte se na tento kód:

```
<header>
    <hgroup>
        <h1>Můj blog</h1>
        <h2>Stálo mne mnoho úsilí, aby to bylo snadné.</h2>
    </hgroup>
    …
</header>

…

<div class="entry">
    <h2>Cestovní den</h2>
</div>

…

<div class="entry">
    <h2>Jedu do Prahy!</h2>
</div>
```

Tady je vytvořená osnova dokumentu:

```
Můj blog (h1 svého prvku hgroup)
    |
    +--Cestovní den (h2)
    |
    +--Jedu do Prahy! (h2)
```

Abyste se ujistili, že používáte nadpisové prvky správně, můžete své stránky otestovat pomocí HTML5 Outliner[6].

6 http://gsnedders.html5.org/outliner/

3.13 Články

Vraťme se k naší vzorové stránce a podívejme se, co můžeme udělat s tímhle kódem:

```
<div class="entry">
    <p class="post-date">22. října 2009</p>
    <h2>
        <a href="#"
            rel="bookmark"
            title="odkaz na tento příspěvek">
            Cestovní den
        </a>
    </h2>
    …
</div>
```

Opět se jedná o regulérní HTML5. HTML5 však nabízí specifičtější prvek pro běžné případy označování článku na stránce – prvek `<article>`.

```
<article>
    <p class="post-date">22. října 2009</p>
    <h2>
        <a href="#"
            rel="bookmark"
            title="odkaz na tento příspěvek">
            Cestovní den
        </a>
    </h2>
    …
</article>
```

Ale není to úplně tak prosté. Měli byste provést ještě jednu změnu. Nejdřív vám ji ukážu a pak vysvětlím:

```
<article>
    <header>
        <p class="post-date">22. října 2009</p>
        <h1>
            <a href="#"
                rel="bookmark"
                title="odkaz na tento příspěvek">
```

```
                Cestovní den
            </a>
        </h1>
    </header>
    …
</article>
```

Všimli jste si toho? Změnil jsem prvek <h2> na <h1> a zabalil ho do prvku <header>. Jak funguje prvek <header>, jste už viděli. Jeho účelem je zabalit všechny prvky, které tvoří hlavičku článku (v tomto případě datum vydání a název článku). Ale… ale… neměl by mít každý dokument jen jeden <h1>? Nerozbije to osnovu dokumentu? Nerozbije, ale abychom pochopili proč, musíme se o krok vrátit.

V HTML 4 jediný způsob jak vytvořit osnovu dokumentu byl pomocí prvků <h1>–<h6>. Pokud jste chtěli v osnově mít jenom jeden kořenový uzel, museli jste se v kódu omezit na jediný prvek <h1>. Ale specifikace HTML5 definuje algoritmus pro generování osnovy dokumentu, který využívá nové sémantické prvky v HTML5. Algoritmus HTML5 říká, že prvek <article> vytváří novou sekci, což znamená nový uzel v osnově dokumentu. A v HTML5 může mít každá sekce svůj vlastní prvek <h1>.

To je oproti HTML 4 docela drastická změna a my si hned řekneme, proč je to změna k lepšímu. Hodně webových stránek se v reálu generuje pomocí šablon. Trocha obsahu se vezme z jednoho zdroje a strčí se semhle na stránku, další trocha obsahu se vezme z jiného zdroje a strčí se zase támhle. Hodně návodů je postaveno podobně. „Tady je kousek HTML kódu. Prostě ho zkopírujte a vložte na svoji stránku." U malých kousků obsahu to nevadí, ale co když kopírujete kód pro celou sekci? V tom případě se v návodu dočtete něco jako: „Tady je kousek HTML kódu. Prostě ho zkopírujte, vložte do textového editoru a opravte tagy pro nadpisy, aby odpovídaly příslušné úrovni vnoření nadpisových tagů na stránce, kam ho kopírujete."

Řekněme si to ještě jinak. HTML 4 nemá žádný *obecný* nadpisový prvek. Má šest přísně očíslovaných nadpisových prvků <h1>–<h6>, které do sebe musejí být vnořeny přesně v tomto pořadí. To je docela hloupé, hlavně pokud je vaše stránka „poskládaná" spíš než „napsaná". A právě tento problém řeší HTML5 díky novým strukturující prvkům a novým pravidlům pro stávající nadpisové prvky. Pokud chcete použít nové strukturující prvky, můžu vám dát tento kód:

```
<article>
    <header>
        <h1>Syndikovaný příspěvek</h1>
    </header>
    <p>Lorem ipsum blah blah…</p>
</article>
```

a vy si ho můžete vložit do stránky, *kamkoli chcete*, a to bez úprav. To, že využívá prvek <h1>, není problém, protože je to celé vložené do <article>. Prvek <article> definuje autonomní uzel v osnově dokumentu, prvek <h1> tento uzel pojmenovává a všechny ostatní strukturující prvky na stránce zůstanou na takové úrovni vnoření, na jaké byly předtím.

PROFESOR ZNAČKA ŘÍKÁ

Jako se vším na webu je to ve skutečnosti o trochu komplikovanější, než vám tady říkám. Nové „explicitní" strukturující prvky (jako <h1> zabalený v <article>) na sebe se staršími „implicitními" strukturujícími prvky (<h1>–<h6>) mohou působit nepředvídatelným způsobem. Ulehčíte si život, pokud použijete jedno, nebo druhé, ale ne oboje najednou. Pokud musíte v rámci jedné stránky použít oboje, zkontrolujte výsledek v HTML5 Outliner a přesvědčte se, že osnova vašeho dokumentu dává smysl.

3.14 Data a časy

Je to celkem vzrušující, že jo? No dobře, není to tak vzrušující jako sjet nahý na lyžích z Mount Everestu a recitovat přitom americkou hymnu pozpátku, ale na HTML kód je to slušné. Pokračujme dál na naší vzorové stránce. Teď se chci zaměřit na tento řádek:

```
<div class="entry">
    <p class="post-date">22. října 2009</p>
    <h2>Cestovní den</h2>
</div>
```

Zase ta známá písnička, že? Běžná formule – označující datum vydání článku –, která nemá žádný sémantický kód, o nějž by se opřela, a proto se autoři uchýlili k obecnému kódu s vlastními atributy class. Opět se jedná o regulérní HTML5. Nikdo vás nenutí, abyste ho měnili. Ale HTML5 pro tento případ nabízí specifické řešení: prvek <time>.

```
<time datetime="2009-10-22" pubdate>22. října 2009</time>
```

Prvek <time> se skládá ze tří částí:

- Strojově čitelné časové razítko
- Textový obsah čitelný pro člověka

- Volitelný příznak `pubdate`

V tomto příkladu atribut `datetime` určuje pouze datum, nikoliv čas. Je ve formátu čtyř číslic pro rok, dvou číslic pro měsíc a dvou číslic pro den, přičemž rok, měsíc a den jsou odděleny pomlčkami:

```
<time datetime="2009-10-22" pubdate>22. října 2009</time>
```

Pokud chcete uvést i čas, přidejte za datum písmeno `T`, pak čas v 24 hodinovém formátu a nakonec časové pásmo.

```
<time datetime="2009-10-22T13:59:47-04:00" pubdate>
    22. října 2009 13:59 EDT
</time>
```

(Formát pro datum/čas je celkem flexibilní. Specifikace HTML5 obsahuje příklady platných řetězců pro datum a čas.)

Všimněte si, že jsem změnil textový obsah – to mezi `<time>` a `</time>` –, aby souhlasil se strojově čitelným časovým razítkem. To ve skutečnosti není potřeba. Textovým obsahem může být cokoliv chcete, pokud v atributu `datetime` uvedete strojově čitelné datové či časové razítko.

Tohle je regulérní HTML5:

```
<time datetime="2009-10-22">minulý čtvrtek</time>
```

A tohle je taky regulérní HTML5:

```
<time datetime="2009-10-22"></time>
```

Posledním kouskem do skládanky je atribut pubdate. Je to booleovský atribut, který jednoduše přidáte, pokud ho potřebujete, a to takhle:

```
<time datetime="2009-10-22" pubdate>22. října 2009</time>
```

Pokud se vám nelíbí „nahaté“ atributy bez hodnoty, může použít tento ekvivalent:

```
<time datetime="2009-10-22" pubdate="pubdate">22. října 2009</time>
```

Co znamená atribut `pubdate`? Může to být jedna ze dvou věcí. Pokud je prvek `<time>` obsažen v prvku `<article>`, znamená to, že časové razítko označuje datum vydání článku. Pokud

prvek `<time>` není obsažen v prvku `<article>`, znamená to, že toto časové razítko označuje datum vydání celého dokumentu.

Tady je celý článek přepsaný tak, aby plně využíval výhod HTML5:

```
<article>
    <header>
        <time datetime="2009-10-22" pubdate>
            22. října 2009
        </time>
        <h1>
            <a href="#"
                rel="bookmark"
                title="odkaz na tento příspěvek">
                Cestovní den
            </a>
        </h1>
    </header>
    <p>Lorem ipsum dolor sit amet…</p>
</article>
```

3.15 Navigace

Jednou z nejdůležitějších částí jakékoliv webové stránky je navigační lišta. CNN.com má na vrchu každé stránky „záložky“, které odkazují na různé zpravodajské sekce – „Technika“, „Zdraví“, „Sport“ atd. Výsledky vyhledávání v Googlu mají na vrchu každé stránky podobný proužek, který vám nabízí, abyste zkusili vyhledávat v jiných službách Google – „Obrázky“, „Videa“, „Mapy“ apod. A naše vzorová stránka má navigační lištu v hlavičce, která obsahuje odkazy na různé sekce naší hypotetické stránky – „domů“, „blog“, „galerie“ a „o mně“.

Takhle původně vypadal kód pro navigační lištu:

```
<div id="nav">
    <ul>
        <li><a href="#">domů</a></li>
        <li><a href="#">blog</a></li>
        <li><a href="#">galerie</a></li>
        <li><a href="#">o mně</a></li>
    </ul>
</div>
```

Opět se jedná o regulérní HTML5. Ovšem tento kód sice představuje seznam čtyř položek, ale nic na tom seznamu nevypovídá o tom, že je to součást navigace stránky. Vizuálně to můžete odhadnout z toho, že je součástí hlavičky stránky, a tím, že si přečtete text odkazů. Ale sémanticky tento seznam odkazů nic neodlišuje od jiných.

Kdo se zajímá o sémantiku navigace stránky? Například lidé s hendikepy. Proč? Představte si následující situaci: vaše pohybové možnosti jsou omezené a používání myši je pro vás obtížné nebo dokonce nemožné. Abyste se s tím vyrovnali, můžete používat doplněk prohlížeče, který vám umožní přeskočit (na) hlavní navigační odkazy. Anebo si představte tohle: pokud máte zhoršený zrak, můžete používat speciální program zvaný „čtečka obrazovky", který převádí text webových stránek na mluvené slovo a nahlas je čte nebo shrne jejich obsah. Jakmile se dostanete přes název stránky, další důležité informace o stránce tvoří hlavní navigační odkazy. Pokud se chcete rychle zorientovat, řeknete čtečce obrazovky, aby přeskočila na navigační lištu a začala číst. Pokud chcete rychle prohlížet, řeknete čtečce obrazovky, aby navigační lištu přeskočila a začala číst hlavní obsah. Každopádně je možnost rozpoznat navigační odkazy v programech důležitá.

I když na označení navigace vaší stránky pomocí `<div id="nav">` není nic vyloženě špatného, není na tom taky nic vyloženě dobrého. Neoptimálnost tohoto řešení pociťují skuteční lidé. HTML5 nabízí sémantický způsob jak označit navigační sekce: prvek `<nav>`.

```
<nav>
    <ul>
        <li><a href="#">domů</a></li>
        <li><a href="#">blog</a></li>
        <li><a href="#">galerie</a></li>
        <li><a href="#">o mně</a></li>
    </ul>
</nav>
```

PTEJTE SE PROFESORA ZNAČKY

Q: Jsou přeskakovací odkazy kompatibilní s prvkem `<nav>`? Budu přeskakovací odkazy potřebovat i v HTML5?

A: Přeskakovací odkazy umožňují uživatelům přeskakovat navigační sekce. Pomáhají hendikepovaným uživatelům, kteří používají software třetích stran pro přečtení stránky nahlas a navigaci bez myši. (Naučte se jak a proč vytvářet přeskakovací odkazy*.)

* http://webaim.org/techniques/skipnav/

Až budou čtečky obrazovek aktualizovány, aby rozpoznávaly prvek <nav>, budou přeskakovací odkazy zbytečné, protože software pro čtení obrazovky bude moci automaticky nabízet přeskočení navigační sekce označené prvkem <nav>. Ovšem bude asi trvat nějakou dobu, než všichni hendikepovaní uživatelé na webu aktualizují na software pro čtení obrazovky znalý HTML5, takže byste zatím měli i nadále vytvářet vlastní přeskakovací odkazy pro přeskočení sekcí <nav>.

3.16 Zápatí

Konečně jsme se dostali na konec naší vzorové stránky. Poslední věc, o které chci mluvit, je poslední věcí na stránce: zápatí. Zápatí se původně zapisovalo takto:

```
<div id="footer">
    <p>&#167;</p>
    <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</div>
```

To je regulérní HTML5. Pokud chcete, můžete u něho zůstat. Ale HTML5 nabízí specifičtější prvek – prvek <footer>.

```
<footer>
    <p>&#167;</p>
    <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</footer>
```

Co je vhodné dát do prvku <footer>? Nejspíš všechno, co jste dosud dávali do . No dobrá, je to definice kruhem. Ale vážně je to tak. Specifikace HTML5 říká: „Zápatí obvykle obsahuje informace o tom, kdo napsal sekci, odkazy na související dokumenty, údaje o autorských právech a podobně.“ Přesně to máme i na této vzorové stránce: krátké prohlášení o autorských právech a odkaz na stránku s informacemi o autorovi. Když se dívám na některé oblíbené stránky, vidím tam slušný potenciál pro <footer>.

- CNN má zápatí, které obsahuje prohlášení o autorských právech, odkazy na překlady a odkazy na podmínky použití, stránku o ochraně informací, stránky „o nás“, „kontaktujte nás“ a „nápověda“. Všechno tohle je ideální materiál pro <footer>.

- Google je proslavený spartánskou strohostí své domovské stránky, ale dole na ní najdeme odkazy na „Reklamu“, „Firmu“ a „O společnosti Google“, dále prohlášení o autorských

právech a odkaz na ochranu soukromí v Google. Všechno tohle by se mohlo zabalit do `<footer>`.

- Můj blog má zápatí s odkazy na mé další stránky a prohlášení o autorských právech. To se pro prvek `<footer>` rozhodně hodí. (Všimněte si, že odkazy samotné by *neměly* být zabalené v prvku `<nav>`, protože to nejsou navigační odkazy stránky; je to jen soubor odkazů na mé jiné projekty na jiných stránkách.)

„Tučná zápatí" jsou dneska v módě. Podívejte se na zápatí na webu W3C[7]. Má tři sloupce pojmenované „Navigace", „Kontaktuje W3C" a „Aktualizace W3C". Kód vypadá víceméně takto:

```
<div id="w3c_footer">
    <div class="w3c_footer-nav">
        <h3>Navigation</h3>
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/standards/">Standards</a></li>
            <li><a href="/participate/">Participate</a></li>
            <li><a href="/Consortium/membership">Membership</a></li>
            <li><a href="/Consortium/">About W3C</a></li>
        </ul>
    </div>
    <div class="w3c_footer-nav">
        <h3>Contact W3C</h3>
        <ul>
            <li><a href="/Consortium/contact">Contact</a></li>
            <li><a href="/Help/">Help and FAQ</a></li>
            <li><a href="/Consortium/sup">Donate</a></li>
            <li><a href="/Consortium/siteindex">Site Map</a></li>
        </ul>
    </div>
    <div class="w3c_footer-nav">
        <h3>W3C Updates</h3>
        <ul>
            <li><a href="http://twitter.com/W3C">Twitter</a></li>
            <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
        </ul>
    </div>
    <p class="copyright">Copyright © 2009 W3C</p>
</div>
```

7 http://www.w3.org/

Pro konverzi na sémantické HTML5 bych provedl následující změny:

- Konverze `<div id="w3c_footer">` na prvek `<footer>`.
- Konverze prvních dvou případů `<div class="w3c_footer-nav">` na prvky `<nav>` a třetího případu na prvek `<section>`.
- Konverze nadpisů `<h3>` na `<h1>`, protože by teď byl každý z nich uvnitř strukturujícího prvku. Prvek `<nav>` vytváří sekci v osnově dokumentu, stejně jako prvek `<article>`.

Výsledný kód by mohl vypadat nějak takhle:

```
<footer>
    <nav>
        <h1>Navigation</h1>
    <ul>
        <li><a href="/">Home</a></li>
        <li><a href="/standards/">Standards</a></li>
        <li><a href="/participate/">Participate</a></li>
        <li><a href="/Consortium/membership">Membership</a></li>
        <li><a href="/Consortium/">About W3C</a></li>
    </ul>
    </nav>
    <nav>
        <h1>Contact W3C</h1>
    <ul>
        <li><a href="/Consortium/contact">Contact</a></li>
        <li><a href="/Help/">Help and FAQ</a></li>
        <li><a href="/Consortium/sup">Donate</a></li>
        <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
    </nav>
    <section>
        <h1>W3C Updates</h1>
    <ul>
        <li><a href="http://twitter.com/W3C">Twitter</a></li>
        <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
    </section>
    <p class="copyright">Copyright © 2009 W3C</p>
</footer>
```

3.17 K dalšímu čtení

Příkladové stránky použité v této kapitole:
- Původní (HTML 4)[8]
- Upravená (HTML5)[9]

O znakových sadách:
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)[10] (Absolutní minimum, které každý vývojář softwaru musí naprosto nezbytně vědět o Unicode a znakových sadách (bez pardonu!) od Joela Spolského
- On the Goodness of Unicode[11] (O užitečnosti Unicodu), On Character Strings[12] (O řetězcích znaků) a Characters vs. Bytes[13] (Znaky vs. bajty) od Tima Braye

O umožnění nových prvků HTML5 v Internet Exploreru:
- How to style unknown elements in IE[14] (Jak určit styl neznámých prvků v IE) od Sjoerda Visschera
- HTML5 shiv[15] od Johna Resiga
- HTML5 enabling script[16] (Skript umožňující HTML5) od Remyho Sharpa
- The Story of the HTML5 Shiv[17] (Příběh HTML5 shiv) od Paula Irishe

O standardních režimech a doctype sniffing (určování typu dokumentu):
- Activating Browser Modes with Doctype[18] (Aktivace režimů prohlížeče pomocí doctype) od Henriho Sivonena. Tohle je jediný článek, který si na tohle téma potřebujete přečíst. Jakýkoliv článek o doctypech, který by necitoval Henriho článek, je zaručeně zastaralý, neúplný nebo špatný.

Validátor HTML5:
- html5.validator.nu[19]

8 http://diveintohtml5.info/examples/blog-original.html
9 http://diveintohtml5.info/examples/blog-html5.html
10 http://www.joelonsoftware.com/articles/Unicode.html
11 http://www.tbray.org/ongoing/When/200x/2003/04/06/Unicode
12 http://www.tbray.org/ongoing/When/200x/2003/04/13/Strings
13 http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF
14 http://xopus.com/devblog/2008/style-unknown-elements.html
15 http://ejohn.org/blog/html5-shiv/
16 https://remysharp.com/2009/01/07/html5-enabling-script
17 http://www.paulirish.com/2011/the-history-of-the-html5-shiv/
18 https://hsivonen.fi/doctype/
19 https://html5.validator.nu/

4. Říkejme tomu vykreslovací prostor

4. Říkejme tomu vykreslovací prostor – 95

4.1 Začínáme

HTML5 definuje prvek plátno jako „bitmapové plátno závislé na rozlišení, které může být použito pro vykreslování grafů, herní grafiky nebo jiných vizuálních obrazů v reálném čase." *Plátno* je obdélník na stránce, do kterého můžete pomocí JavaScriptu vykreslit, cokoliv chcete.

Základní podpora `<canvas>`						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu prvku `<canvas>`.						

Jak vlastně plátno vypadá? Vlastně nijak. Prvek `<canvas>` nemá vlastní obsah ani ohraničení.

← Neviditelné plátno

Kód vypadá následovně:

```
<canvas width="300" height="225"></canvas>
```

Přidejme tečkované ohraničení, abychom viděli, s čím máme tu čest.

← Plátno s ohraničením

Na jedné stránce můžete mít více prvků `<canvas>`. Každé plátno se zobrazuje v DOMu a udržuje si vlastní stav. Pokud přiřadíte každému plátnu atribut `id`, budete k nim mít přístup jako k jakýmkoliv jiným prvkům.

Rozšiřme ten kód o atribut `id`:

```
<canvas id="a" width="300" height="225"></canvas>
```

Nyní můžete prvek `<canvas>` snadno najít v DOMu.

```
var a_canvas = document.getElementById("a");
```

4.2 Jednoduché tvary

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu tvarů prvku `<canvas>`.						

Každé plátno je na začátku prázdné. To je nuda. Pojďme na něj něco vykreslit!

Ovladač `onclick` vyvolal následující funkci:

```
function draw_b() {
    var b_canvas = document.getElementById("b");
    var b_context = b_canvas.getContext("2d");
    b_context.fillRect(50, 25, 150, 100);
}
```

První řádek této funkce není ničím zvláštním, pouze vyhledá prvek `<canvas>` v DOMu.

A pak tady máme toto →

```
function draw_b() {
    var b_canvas = document.getElementById("b");
    var b_context = b_canvas.getContext("2d");
    b_context.fillRect(50, 25, 150, 100);
}
```

Každé plátno má vykreslovací kontext, kde se odehrává celá ta legrace. Jakmile jste našli prvek `<canvas>` v DOMu (s použitím `document.getElementById()` nebo jiné vámi prefe-

rované metody), vyvoláte metodu `getContext()`. Metodě `getContext()` **musíte** předat řetězec `"2d"`.

PTEJTE SE PROFESORA ZNAČKY

Q: Existují trojrozměrná plátna?

A: Zatím ne. Někteří vývojáři experimentovali s API třírozměrných pláten vlastní výroby, ale žádné nebyly standardizovány. Specifikace HTML5 uvádí, že „pozdější verze této specifikace bude pravděpodobně definovat 3d kontext."

Takže máte prvek `<canvas>` a máte jeho vykreslovací kontext. Ve vykreslovacím kontextu se definují všechny metody a vlastnosti pro vykreslení. Vykreslovacím obdélníkům je věnován celý soubor vlastností a metod:

- Vlastnost `fillStyle` může být barvou v CSS, vzorem nebo přechodem. (Více o přechodech za chvíli.) Výchozí barva `fillStyle` je černá, ale můžete ji nastavit na jakoukoliv. Každý vykreslovací kontext si „pamatuje" nastavené vlastnosti, dokud je otevřena daná stránka nebo dokud je nějakým způsobem nevynulujete.

- `fillRect(x, y, šířka, výška)` vykresluje obdélník vyplněný podle aktuálního stylu.

- Vlastnost `strokeStyle` je podobná `fillStyle` — může to být barva v CSS, vzor nebo přechod.

- `strokeRect(x, y, šířka, výška)` vykresluje obdélník čárou podle aktuálního stylu. `strokeRect` nepřidává výplň, ale kreslí jen okraje.

- `clearRect(x, y, šířka, výška)` vymaže pixely v daném obdélníku.

PTEJTE SE PROFESORA ZNAČKY

Q: Můžu „vynulovat" plátno?

A: Ano. Nastavením šířky nebo výšky prvku `<canvas>` vymažete jeho obsah a vrátíte všechny vlastnosti jeho vykreslovacího kontextu na výchozí hodnoty. Nepotřebujete *měnit* šířku, stačí ji nastavit na aktuální hodnotu:

```
var b_canvas = document.getElementById("b");
b_canvas.width = b_canvas.width;
```

Vraťme se ke kódu z předchozího příkladu…

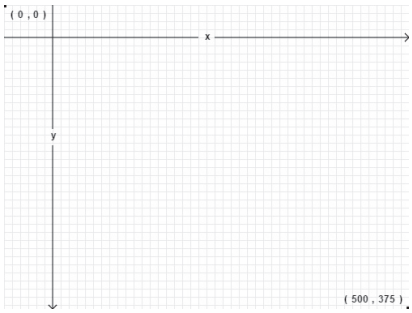
```
var b_canvas = document.getElementById("b");
var b_context = b_canvas.getContext("2d");
```

Vykreslete obdélník → `b_context.fillRect(50, 25, 150, 100);`

Vyvolaná metoda `fillRect()` vykreslí obdélník a vyplní ho aktuálním stylem, kterým je černá, dokud ho nezměníte. Obdélník bude vymezen levým horním rohem (50, 25), šířkou (150) a výškou (100). Pro lepší představu o tom, jak to funguje, se podívejme na plátno v souřadnicové soustavě.

4.3 Souřadnice plátna

Plátno je dvourozměrnou sítí. Bod (0, 0) se nachází v jejím levém horním rohu. Hodnoty osy x se zvětšují směrem doprava. Hodnoty osy y se zvětšují směrem ke spodku plátna.



← Znázornění souřadnic plátna

Toto znázornění souřadnic bylo vykresleno pomocí prvku `<canvas>`. Sestává se z:

- sady šedých svislých čar
- sady šedých vodorovných čar
- dvou černých vodorovných čar
- dvou malých černých diagonálních čar, které tvoří šipku
- dvou černých svislých čar
- dvou malých černých diagonálních čar, které tvoří další šipku
- písmena „x“
- písmena „y“
- textu „(0, 0)“ u levého horního rohu
- textu „(500, 375)“ u pravého dolního rohu
- jedné tečky v levém horním rohu a další v pravém dolním rohu

Zaprvé musíme definovat samotný prvek `<canvas>`. Prvek `<canvas>` pak definuje `width` (šířku) a `height` (výšku) a také `id`, abychom ho pak mohli vyhledat.

```
<canvas id="c" width="500" height="375"></canvas>
```

Pak potřebujeme skript, který vyhledá prvek `<canvas>` v DOMu a načte jeho vykreslovací kontext.

```
var c_canvas = document.getElementById("c");
var context = c_canvas.getContext("2d");
```

A teď můžeme začít s kresbou čar.

4.4 Cesty

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu cest prvku `<canvas>`.

Představte si, že kreslíte obrázek tuší. Pravděpodobně se však nepustíte rovnou do tušové malby pro případ, že byste udělali chybu. Místo toho si nastíníte přímky a křivky tužkou, a až budete s výsledkem spokojeni, obkreslíte skicu tuší.

Každé plátno obsahuje cestu. Vytyčení cesty se podobá kresbě tužkou. Můžete si nakreslit, co chcete, ale neprojeví se to ve výsledné podobě vaší práce, dokud se nechopíte štětce a neobkreslíte cestu tuší.

Pro vykreslení přímek tužkou se používají dvě metody:
1. `moveTo(x, y)` přesouvá tužku na udávanou výchozí pozici.
2. `lineTo(x, y)` vykresluje přímku do uvedené koncové pozice.

Čím více vyvoláváte `moveTo()` a `lineTo()`, tím větší je vaše cesta. Jsou to „metody tužky“ – můžete je vyvolávat tak často, jak budete chtít, ale na plátně se nic neobjeví, dokud nevyvoláte některou z „tušových“ metod.

Začněme tedy vykreslením šedé mřížky.

```
for (var x = 0.5; x < 500; x += 10) {
    context.moveTo(x, 0);
    context.lineTo(x, 375);            ← Vykreslete svislé čáry
}

for (var y = 0.5; y < 375; y += 10) {
    context.moveTo(0, y);
    context.lineTo(500, y);            ← Vykreslete vodorovné čáry
}
```

Toto byly „metody tužky“. Na plátně se ve skutečnosti zatím nic nakreslilo. Na to, abychom to zvěčnili, potřebujeme „tušovou“ metodu.

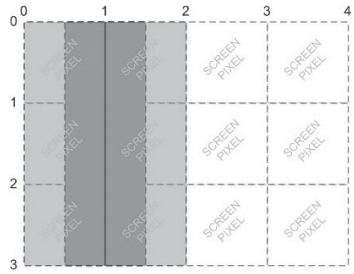
```
context.strokeStyle = "#eee";
context.stroke();
```

`stroke()` je jednou z takových „tušových“ metod. Bere celou komplikovanou cestu, kterou jste vytyčili metodami `moveTo()` a `lineTo()`, a vykresluje ji na plátně. `strokeStyle` odpovídá za barvu čar. Tady je výsledek:

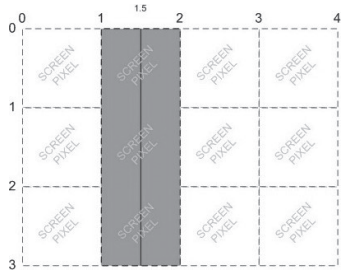
PTEJTE SE PROFESORA ZNAČKY

Q: Proč jste dal počátek x a y na `0.5`? Proč ne na `0`?

A: Představte si pixely jako velké čtverce. Souřadnice s celými čísly (0, 1, 2…) jsou jejich hranami. Pokud nakreslíte čáru s šířkou jeden pixel mezi souřadnicemi s celými čísly, přesáhne čára až na protější strany pixelového čtverce a ve výsledku bude mít šířku dva pixely. Abyste nakreslili čáru s šířkou jeden pixel, potřebujete posunout souřadnice o 0,5 kolmo ke směru čáry.



Například když se pokusíte nakreslit čáru od `(1, 0)` k `(1, 3)`, prohlížeč nakreslí čáru, která zakrývá po 0,5 pixelu obrazovky na každé straně od souřadnice `x=1`. Obrazovka neumí zobrazit půlku pixelu, proto roztáhne čáru až na šířku dvou pixelů.



Když se ovšem pokusíte nakreslit čáru od `(1.5, 0)` k `(1.5, 3)`, prohlížeč nakreslí čáru, která zakryje po 0,5 pixelu obrazovky na každé straně od souřadnice `x=1.5`, což vám dá čáru, která skutečně bude mít šířku jeden pixel.

Díky Jasonu Johnsonovi za poskytnutí těchto ilustrací.

Teď si nakresleme vodorovnou šipku. Všechny přímky a křivky cesty se vykreslují stejnou barvou (nebo přechodem – ano, k tomu se dostaneme již brzy).

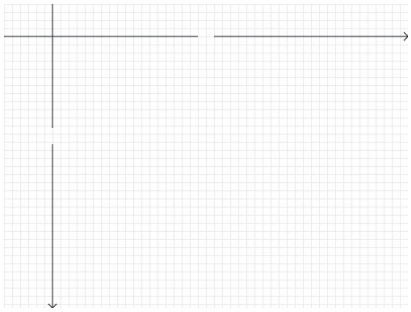
```
context.beginPath();
context.moveTo(0, 40);
context.lineTo(240, 40);
context.moveTo(260, 40);          ← Nová cesta
context.lineTo(500, 40);
context.moveTo(495, 35);
context.lineTo(500, 40);
context.lineTo(495, 45);
```

Svislá šipka vypadá velmi podobně. Jelikož svislá šipka má stejnou barvu jako ta vodorovná, **nepotřebujeme** pro ni kreslit novou cestu. Obě šipky budou součástí stejné cesty.

```
context.moveTo(60, 0);
context.lineTo(60, 153);
context.moveTo(60, 173);
context.lineTo(60, 375);          ← To není nová cesta
context.moveTo(65, 370);
context.lineTo(60, 375);
context.lineTo(55, 370);
```

Říkal jsem, že tyto šipky budou černé, ale barva `strokeStyle` je stále šedá. (Když zakládáte novou cestu, `fillStyle` ani `strokeStyle` se nevynulují.) To je v pořádku, protože teď jsme používali „metody tužky“. Ale než to vykreslíme skutečně, „tuší“, potřebujeme nastavit `strokeStyle` na černou. Jinak ty dvě šipky budou šedé a téměř je neuvidíme! Následující řádky mění barvu na černou a vykreslují čáry na plátno:

```
context.strokeStyle = "#000";
context.stroke();
```



← Tady je výsledek

4.5 Text

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu textu na plátně. † Mozilla Firefox 3.0 vyžaduje doplněk pro kompatibilitu.						

Kromě kreslení čar můžete na plátně vykreslit také text. Na rozdíl od ostatního textu na webové stránce zde není k dispozici blokový model. To znamená, že nejsou dostupné žádné techniky rozvržení známé z CSS: obtékání, vnější a vnitřní okraje, zalamování slov. (To pro vás možná nezní jako špatná zpráva!) Můžete nastavit několik atributů písma, pak vybrat bod na plátně a vykreslit tam svůj text.

Ve vykreslovacím kontextu jsou dostupné následující atributy písma:

- `font` může obsahovat cokoliv, co byste zahrnuli do pravidla pro písmo `font` v CSS, jako styl písma, kapitalizaci, ztučnění, velikost písma, výšku řádku a rodinu písem.

- `textAlign` odpovídá za zarovnání textu. Je podobný (ale ne úplně stejný) jako pravidlo `text-align` v CSS. Možnými hodnotami jsou `start`, `end`, `left`, `right`, a `center`.

- `textBaseline` řídí, kde je text vykreslen vzhledem k počátečnímu bodu. Možnými hodnotami jsou `top` (vrch), `hanging` (závěsná), `middle` (střed), `alphabetic` (abecední), `ideographic` (ideografická), nebo `bottom` (spodek).

`textBaseline` je záludný, protože text je záludný. (Ten anglický ani tak ne, ale na plátně můžete vykreslit jakýkoliv symbol Unicode, a Unicode je záludný). Specifikace HTML5 vysvětluje různé typy základních čar:

> Vrch čtverčíku se nachází přibližně na vrchu glyfů písma, závěsná základní čára je místem, kde jsou ukotveny glyfy jako आप, střed je v půlce úseku mezi vrchem a spodkem čtverčíku, na abecední základní čáře jsou ukotveny symboly jako Á, ÿ, f, a Ω, na ideografické čáře se kotví glyfy jako 私 a 達, spodek čtverčíku se nachází přibližně naspodu glyfů písma. Vrch a spodek ohraničujícího rámečku mohou být od těchto základních čar vzdálené kvůli symbolům, které se rozprostírají daleko mimo čtverčík.

V případě jednoduchých abeced, jako je anglická, můžete pro vlastnost `textBaseline` zůstat u parametrů `top`, `middle` a `bottom`.

Nakresleme nějaký text! Text vykreslený na plátně přebírá velikost písma a styl samotného prvku `<canvas>`, ale můžete to přenastavit tím, že určíte vlastnosti písma ve vykreslovacím kontextu.

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

← Změňte styl písma

Metoda `fillText()` vykresluje samotný text.

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

← Vykreslete text

PTEJTE SE PROFESORA ZNAČKY

Q: Můžu pro vykreslení textu na plátně použit relativní velikost písma?

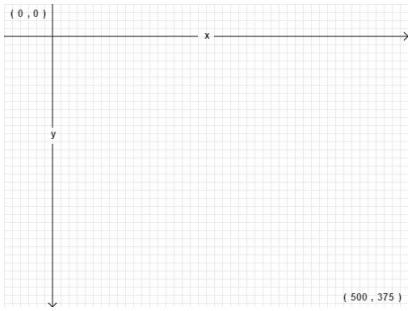
A: Ano. Stejně jako každý jiný prvek HTML na vaší stránce má prvek `<canvas>` velikost vypočítanou na základě pravidel CSS stránky. Pokud ve vlastnosti `context.font` nastavíte relativní velikost písma jako `1.5em` nebo `150%`, prohlížeč ji vynásobí vypočítanou hodnotou velikosti písma samotného prvku `<canvas>`.

Dejme tomu, že chci, aby vrch textu v levém horním rohu byl na úrovni `y=5`. Ale jsem líný a nechci měřit výšku textu a vypočítávat polohu základní čáry. Místo toho můžu umístit `textBaseline` nahoru a zadat souřadnice levého horního rohu ohraničujícího rámečku.

```
context.textBaseline = "top";
context.fillText("( 0 , 0 )", 8, 5);
```

Teď text v pravém dolním rohu. Dejme tomu, že chci, aby pravý spodní roh textu byl na souřadnicích (492,370), jen pár pixelů od pravého spodního rohu plátna, ale nechci měřit šířku a výšku textu. Můžu nastavit `textAlign` na `right` a `textBaseline` na `bottom`, potom vyvolat `fillText()` s pravou a spodní souřadnicí ohraničujícího rámečku textu.

```
context.textAlign = "right";
context.textBaseline = "bottom";
context.fillText("( 500 , 375 )", 492, 370);
```

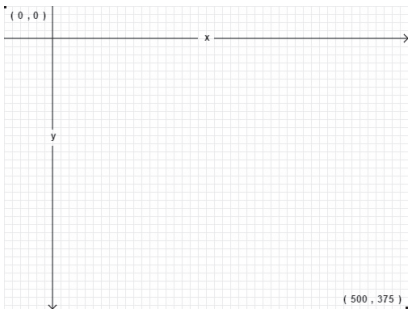


← Tady je výsledek

Ouha! Zapomněli jsme na samotné body v rozích. Jak se vykreslují kroužky, si ukážeme později. Zatím se uchýlím k malému podvodu a vykreslím je jako obdélníčky.

```
context.fillRect(0, 0, 3, 3);
context.fillRect(497, 372, 3, 3);
```

← Vykreslete dva body



← A to bude vše! Tady máte výsledek celé práce

4.6 Přechody

	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Lineární přechody	7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
Kruhové přechody	9.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu přechodů prvku `<canvas>`.

Dříve v této kapitole jste se naučili vykreslit obdélník vyplněný jednolitou barvou a také čáru provedenou jednolitou barvou. Ale tvary a čáry nemusí mít pouze jednolitou výplň. Můžete také různě čarovat s barevnými přechody. Podívejme se na příklad.

Kód vypadá stejně jako u jakékoliv jiného plátna:

```
<canvas id="d" width="300" height="225"></canvas>
```

Především potřebujeme najít prvek `<canvas>` a jeho vykreslovací kontext.

```
var d_canvas = document.getElementById("d");
var context = d_canvas.getContext("2d");
```

Jakmile máme vykreslovací kontext, můžeme začít definovat přechod. Přechod je postupná proměna mezi dvěma nebo vícero barvami. Vykreslovací kontext plátna podporuje dva typy přechodů:

- `createLinearGradient(x0, y0, x1, y1)` se vykresluje podél čáry z bodu (x0, y0) do bodu (x1, y1).

- `createRadialGradient(x0, y0, r0, x1, y1, r1)` se vykresluje podél kužele mezi dvěma kruhy. První tři parametry popisují počáteční kruh se středem (x0, y0) a poloměrem r0. Poslední tři popisují koncový kruh se středem (x1, y1) a poloměrem r1.

Vytvořme lineární přechod. Přechody mohou mít libovolné rozměry, tomuto ale přiřadím šířku 300, jako u plátna.

Vytvořte objekt přechodu ↷

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

Jelikož se obě hodnoty y (2. a 4. parametr) rovnají 0, přechod odstínu zleva doprava bude v tomto případě rovnoměrný.

Jakmile máme objekt přechodu, můžeme definovat jeho barvy. Přechod může mít dvě nebo více barevných hranic (color stops). Barevné hranice se mohou nacházet kdekoliv v rámci přechodu. Pro přidání barevné hranice musíte zadat její polohu v přechodu. Poloha přechodu se označuje číslem od 0 do 1.

Definujme přechod odstínů od černé k bílé.

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

Definice přechodu nic nevykresluje na plátně. Je to pouze objekt schovaný někam do paměti. Pro vykreslení přechodu musíte nastavit styl výplně `fillStyle` na tento přechod a nakreslit tvar, třeba obdélník nebo čáru.

Styl výplně je přechod ↷

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

← A tady je výsledek

Chcete například přechod seshora dolů. Při tvorbě objektu přechodu ponechte hodnoty `x` (1. a 3. parametr) konstantní a hodnoty `y` (2. a 4. parametr) nastavte v rozmezí od 0 do výšky plátna.

Hodnoty x se rovnají 0, hodnoty y jsou různé ↷

```
var my_gradient = context.createLinearGradient(0, 0, 0, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

← A tady je výsledek

Můžete také vytvářet diagonální přechody.

Liší se jak hodnoty x, tak i y ↷

```
var my_gradient = context.createLinearGradient(0, 0, 300, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

← A tady je výsledek

4.7 Obrázky

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+*	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
* Internet Explorer 7 a 8 vyžadují knihovnu třetí strany explorercanvas. Internet Explorer 9 má nativní podporu obrázků na prvku `<canvas>`.						

Toto je kocour:

← Prvek `<img>`

Toto je stejný kocour, vykreslený na plátně:

Prvek `<canvas>` →

Vykreslovací kontext plátna definuje metodu `drawImage()` pro vykreslení obrázků na plátně. Metoda může obsahovat tři, pět nebo devět argumentů.

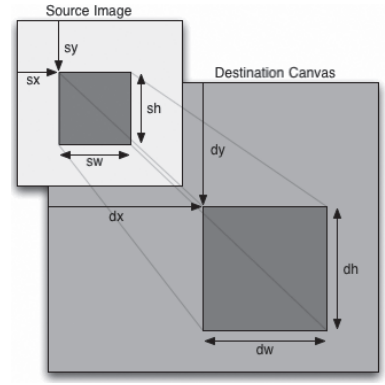
- `drawImage(image, dx, dy)` vezme obrázek a vykreslí ho na plátno. Souřadnice `(dx, dy)` udávají polohu levého horního rohu obrázku. Souřadnice (0, 0) vykreslí obrázek v levém horním rohu plátna.

- `drawImage(image, dx, dy, dw, dh)` vezme obrázek, změní jeho šířku na `dw` a výšku na `dh` a vykreslí ho na plátno v poloze (`dx`, `dy`).

- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)` vezme obrázek, ořízne ho na obdélník `(sx, sy, sw, sh)`, upraví ho na rozměry `(dw, dh)` a vykreslí ho na plátno v souřadnicích `(dx, dy)`.

Specifikace HTML5 vysvětluje parametry `drawImage()`:

> Zdrojový obdélník je obdélník [uvnitř zdrojového obrázku], jehož rohy leží v bodech `(sx, sy)`, `(sx+sw, sy)`, `(sx+sw, sy+sh)` a `(sx, sy+sh)`.
>
> Cílový obdélník je obdélník [uvnitř plátna], jehož rohy leží v bodech `(dx, dy)`, `(dx+dw, dy)`, `(dx+dw, dy+dh)` a `(dx, dy+dh)`.



Pro vykreslení obrázku na plátno potřebujete obrázek. Obrázkem může být existující prvek `<img>`, nebo můžete pomocí JavaScriptu vytvořit objekt `Image()`. V každém případě musíte zajistit, aby se obrázek úplně načetl před tím, než ho vykreslíte na plátno.

Používáte-li existující prvek `<img>`, můžete ho bezpečně vykreslit na plátno během události `window.onload`.

Použití prvku `<img>` ↷

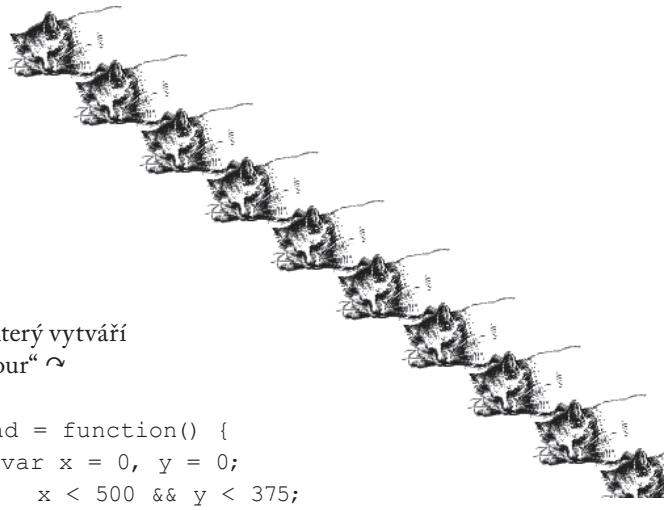
```
<img id="cat" src="images/cat.png" alt="sleeping cat" width="177"
height="113">
<canvas id="e" width="177" height="113"></canvas>
<script>
window.onload = function() {
    var canvas = document.getElementById("e");
    var context = canvas.getContext("2d");
    var cat = document.getElementById("cat");
    context.drawImage(cat, 0, 0);
};
</script>
```

Pokud objekt obrázku vytváříte kompletně v JavaScriptu, můžete obrázek bezpečně vykreslit na plátno během události `Image.onload`.

Použití objektu `Image()` ↷

```
<canvas id="e" width="177" height="113"></canvas>
<script>
    var canvas = document.getElementById("e");
    var context = canvas.getContext("2d");
    var cat = new Image();
    cat.src = "images/cat.png";
    cat.onload = function() {
        context.drawImage(cat, 0, 0);
    };
</script>
```

Volitelný 3. a 4. parametr v metodě `drawImage()` určují změnu velikosti obrázku. Toto je tentýž obrázek zmenšený na poloviční šířku a výšku a vykreslený opakovaně na různých souřadnicích v rámci jednoho plátna.



Tady je skript, který vytváří efekt „multikocour“ ↷

```
cat.onload = function() {
    for (var x = 0, y = 0;
            x < 500 && y < 375;
            x += 50, y += 37) {
        context.drawImage(cat, x, y, 88, 56); ← Změňte velikost obrázku
    }
};
```

Když vezmeme v potaz, kolik je s tím práce, pochopitelně se nabízí otázka: proč bychom vůbec měli kreslit obrázek na plátno? Jakou výhodu má složitý obrázek na plátně oproti prvku `<img>` a pár pravidlům CSS? Dokonce i „multikocour“ by přece šel vytvořit s 10 překrývajícími se prvky `<img>`.

Jednoduchá odpověď zní: ze stejného důvodu, z jakého bychom chtěli vykreslit na plátno text. Schéma souřadnic plátna obsahovalo text, čáry a tvary; text na plátně byl pouze součástí většího projektu. Složitější diagram by snadno mohl použít `drawImage()` pro ikony, sprity nebo další grafiku.

4.8 A co IE?

Verze Internet Exploreru starší než 9.0 nepodporují API plátna. (IE9 má plnou podporu API plátna). Starší verze Internet Exploreru ovšem podporují technologii Microsoftu zvanou VML, která dokáže hodně z toho, co umí prvek `<canvas>`. A tak se zrodil `excanvas.js`.

Explorercanvas (`excanvas.js`) je open-source javascriptová knihovna licencovaná Apache, která realizuje API plátna v Internet Exploreru. Pro její použití vložte navrch vaší stránky tento prvek `<script>`:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Dive Into HTML5</title>
    <!--[if lt IE 9]>
        <script src="excanvas.js"></script>
    <![endif]-->
</head>
<body>
    ...
</body>
</html>
```

Části `<!--[if lt IE 9]>` a `<![endif]-->` jsou podmíněné komentáře. Internet Explorer si je vykládá jako podmínku s `if`: „Pokud je daný prohlížeč verze Internet Exploreru nižší než verze 9, pak se spustí tento blok.“ Všechny ostatní prohlížeče se budou k celému bloku chovat jako k HTML komentáři. Výsledek je takový, že si Internet Explorer 7 a 8 stáhnou skript `excanvas.js` a spustí ho, ale ostatní prohlížeče ho budou úplně ignorovat (nebudou ho stahovat,

spouštět, ani nic jiného). Díky tomu se vaše stránka načte rychleji v prohlížečích, které mají nativní podporu API plátna.

Pokud v `<head>` stránky použijete `excanvas.js`, už nemusíte Internet Exploreru nijak vycházet vstříc. Prostě v kódu použijte prvky nebo je dynamicky vytvořte pomocí JavaScriptu. Řiďte se instrukcemi v této kapitole pro získání vykreslovacího kontextu prvku `<canvas>` a budete moct vykreslovat tvary, text a vzory.

No… ne tak úplně. Existuje několik omezení:

- Přechody mohou být pouze lineární. Kruhové přechody nejsou podporovány.
- Vzory se musejí opakovat v obou směrech.
- Nejsou podporovány oblasti ořízntutí.
- Nerovnoměrné škálování neškáluje tahy správně.
- Je to pomalé. To by pro nikoho neměl být zas takový šok, protože javascriptový analyzátor v Internet Exploreru je obecně pomalejší než v ostatních prohlížečích. Jakmile začnete vykreslovat složité tvary pomocí javascriptové knihovny, která překládá příkazy do úplně jiné technologie, bude to váznout. Poklesu výkonu si nevšimnete u jednoduchých příkladů, jako je vykreslení několika čar a transformace obrázku, ale uvidíte ho hned, jakmile začnete dělat animace založené na plátně a další šílenosti.

Používání `excanvas.js` má ještě jeden háček. Jde o problém, na který jsem narazil při vytváření příkladů pro tuto kapitolu. ExplorerCanvas spouští své vlastní prostředí pseudo-plátna automaticky vždy, když v HTML stránce použijete skript `excanvas.js`. To ovšem neznamená, že je Internet Explorer připraven ho okamžitě použít. Můžete se dostat do situace, kdy je prostředí pseudo-plátna *téměř* připraveno k použití – téměř, ale ne úplně. Hlavní známkou tohoto stavu je, že si Internet Explorer stěžuje slovy „`object doesn't support this property or method`“ (objekt nepodporuje tuto vlastnost nebo metodu), kdykoliv zkusíte něco udělat s prvkem `<canvas>`, například získat jeho vykreslovací kontext.

Nejjednodušším řešením je odložit všechny činnosti spojené s plátnem až do okamžiku, než dojde k události `onload`. To může chvíli trvat – pokud má vaše stránka hodně obrázků nebo videí, událost `onload` se tím zdrží –, ale poskytne to ExplorerCanvas čas na to, aby to rozbalil.

4.9 Úplný příklad ze života

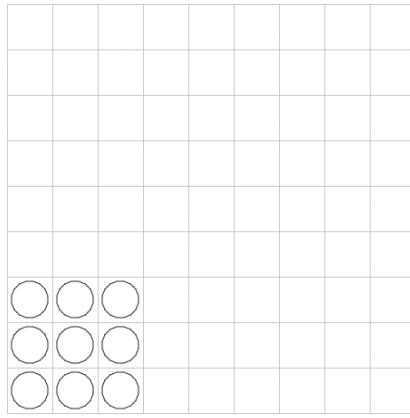
Halma je víc než sto let stará stolní hra, která má mnoho variant. V tomto příkladu jsem vytvořil verzi halmy pro jednoho s 9 hracími kameny na desce 9 x 9. Na začátku hry tvoří kameny čtverec v dolním levém rohu desky. Cílem hry je přesunout všechny kameny tak, aby tvořily čtverec 3 x 3 v pravém horním rohu desky pomocí nejmenšího možného počtu tahů.

V halmě jsou povoleny dva druhy tahů:

- Zvolte kámen a posuňte ho do libovolného sousedního pole. „Prázdné" pole je takové, kde právě není žádný kámen. „Sousední" pole se nachází vedle pole, ve kterém je kámen teď, směrem na sever, jih, východ, západ, severozápad, severovýchod, jihozápad nebo jihovýchod. (Deska se na koncích nespojuje. Pokud se kámen nachází v nejspodnější řadě, nemůže se pohybovat na jih, jihovýchod ani jihozápad.)

- Zvolte kámen a přeskočte sousední kámen, což můžete případně opakovat – tak, že přeskočíte sousední kámen a pak i *další* kámen, který sousedí s vaší novou pozicí, což se stále počítá jako jeden tah. Libovolný počet přeskočení se stále počítá jako jediný tah. (Jelikož je cílem minimalizovat celkový počet tahů, pro úspěch v halmě potřebujete vytvářet a pak využívat dlouhé řetězce šachovnicově uspořádaných kamenů, aby je pak další kameny mohly v dlouhých řadách přeskakovat.)

Tady tu hru máte. Pokud se v ní chcete trochu povrtat vývojářskými nástroji prohlížeče, můžete si ji zahrát na nové stránce[1].



Tahů: 0

1 http://diveintohtml5.info/examples/canvas-halma.html

Jak to funguje? Jsem rád, že se na to ptáte, ale celý kód vám tady neukážu. (Můžete se na něj podívat na diveintohtml5.info/examples/halma.js.) Přeskočím většinu kódu pro herní mechaniku a zaměřím se na několik úseků, které se věnují vykreslování na plátno a reakcím na klikání myši na plátno.

Během načítání stránky zahájíme hru určením rozměrů `<canvas>` samotného a uložením odkazu na jeho vykreslovací kontext.

```
gCanvasElement.width = kPixelWidth;
gCanvasElement.height = kPixelHeight;
gDrawingContext = gCanvasElement.getContext("2d");
```

Pak uděláme něco, co jste ještě neviděli: přidáme prvku `<canvas>` posluchač událostí, aby poslouchal události při kliknutí.

```
gCanvasElement.addEventListener("click", halmaOnClick, false);
```

Funkce `halmaOnClick()` se vyvolá vždy, když uživatel klikne někam na plátno. Jejím argumentem je objekt `MouseEvent`, který obsahuje informace o tom, kam uživatel kliknul.

```
function halmaOnClick(e) {
    var cell = getCursorPosition(e);

    // zbytek programu tvoří herní logika
    for (var i = 0; i < gNumPieces; i++) {
        if ((gPieces[i].row == cell.row) &&
            (gPieces[i].column == cell.column)) {
            clickOnPiece(i);
            return;
        }
    }
    clickOnEmptyCell(cell);
}
```

Dalším krokem je vzít objekt MouseEvent a vypočítat, na které pole halmové desky se právě kliknulo. Halmová deska zabírá celé plátno, takže kliknutí nutně směřuje někam na desku. Musíme jenom přijít na to kam. To je záludné, protože události při kliknutí myši se v každém prohlížeči provádějí různě.

```
function getCursorPosition(e) {
    var x;
```

```
    var y;
    if (e.pageX != undefined && e.pageY != undefined) {
        x = e.pageX;
        y = e.pageY;
    }
    else {
        x = e.clientX + document.body.scrollLeft +
            document.documentElement.scrollLeft;
        y = e.clientY + document.body.scrollTop +
            document.documentElement.scrollTop;
    }
```

V tuto chvíli máme souřadnice `x` a `y` vztahující se k dokumentu (tj. k celé HTML stránce). To nám ještě moc nepomůže. Chceme souřadnice, které se budou vztahovat k plátnu.

```
x -= gCanvasElement.offsetLeft;
y -= gCanvasElement.offsetTop;
```

Teď máme souřadnice x a y vztahující se k plátnu. To znamená, že pokud v tuto chvíli x je 0 a y je 0, víme, že uživatel kliknul na pixel v horním levém rohu plátna.

Z toho můžeme vypočítat, na které pole halmy uživatel kliknul, a zařídit se podle toho.

```
    var cell = new Cell(Math.floor(y/kPieceHeight),
                        Math.floor(x/kPieceWidth));
    return cell;
}
```

Uf! Události při kliknutí myši jsou pořádná fuška. Ale stejnou logiku (vlastně přesně tento kód) můžete použít ve všech aplikacích založených na plátnu. Zapamatujte si: kliknutí myši → souřadnice vztahující se k dokumentu → souřadnice vztahující se k plátnu → kód specifický pro aplikaci.

Podívejme se na hlavní vykreslovací rutinu. Protože je v tomto případě grafika tak jednoduchá, rozhodl jsem se desku znovu vykreslit pokaždé, když se ve hře něco změní. To není nezbytně nutné. Vykreslovací kontext plátna uchová vše, co jste tam předtím vykreslili, i když se uživatel posune na stránce tak, že plátno nebude vidět, nebo přepne na jiný panel a vrátí se později. Pokud vytváříte na plátně založenou aplikaci se složitější grafikou (jako je arkádová hra), můžete optimalizovat výkon tím, že zjistíte, které oblasti plátna jsou „špinavé“, a znovu pak vykreslíte pouze tyto oblasti. Ale to už je nad rámec této knihy.

```
gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);
```

Rutina vykreslení herní desky by vám měla přijít povědomá. Podobá se totiž tomu, jak jsme dříve v této kapitole vykreslovali schéma souřadnic plátna.

```
gDrawingContext.beginPath();

/* vertical lines */
for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
    gDrawingContext.moveTo(0.5 + x, 0);
    gDrawingContext.lineTo(0.5 + x, kPixelHeight);
}

/* horizontal lines */
for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
    gDrawingContext.moveTo(0, 0.5 + y);
    gDrawingContext.lineTo(kPixelWidth, 0.5 + y);
}
/* draw it! */
gDrawingContext.strokeStyle = "#ccc";
gDrawingContext.stroke();
```

Opravdová zábava nastane při vykreslování každého jednotlivého kamenu. Kámen má tvar kruhu, což jsme dosud nekreslili. Když si navíc uživatel vybere kámen a chystá se jím pohnout, chceme tento kámen znázornit jako vyplněný kruh. Argument p tady reprezentuje kámen s vlastnostmi `row` a `column`, které ukazují jeho současnou pozici na hrací desce. Použijeme nějaké herní konstanty, které přeloží (`column`, `row`) na souřadnice (`x`, `y`) vztahující se k plátnu, nakreslíme kruh a (pokud je kámen vybrán) vyplníme ho barvou.

```
function drawPiece(p, selected) {
    var column = p.column;
    var row = p.row;
    var x = (column * kPieceWidth) + (kPieceWidth/2);
    var y = (row * kPieceHeight) + (kPieceHeight/2);
    var radius = (kPieceWidth/2) - (kPieceWidth/10);
```

To je k herní logice vše. Teď máme souřadnice (`x`, `y`), které se vztahují k plátnu, pro střed kruhu, který chceme nakreslit. V API plátna není žádná metoda `circle()`, ale je tam metoda `arc()`. A co je vlastně kruh jiného než dokončený oblouk? Pamatujete si ze školy základy geometrie? Metoda `arc()` potřebuje středový bod (`x`, `y`), poloměr, počáteční a koncový úhel (v radiánech)

a směrový příznak (`false` je po směru hodinových ručiček, `true` je proti směru). Pro výpočet radiánů můžete použít javascriptový modul `Math`.

```
gDrawingContext.beginPath();
gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
gDrawingContext.closePath();
```

Ale moment! Ještě se nic nenakreslilo. Stejně jako `moveTo()` a `lineTo` je metoda `arc()` metodou „tužky". Aby se kruh skutečně nakreslil, musíme nastavit `strokeStyle` a vyvolat `stroke()` pro obkreslení „tuší".

```
gDrawingContext.strokeStyle = "#000";
gDrawingContext.stroke();
```

Co když je daný kámen zvolený uživatelem? Můžeme znovu použít stejnou cestu, kterou jsme použili pro vykreslení obrysu kamenu, a vyplnit kruh barvou.

```
if (selected) {
    gDrawingContext.fillStyle = "#000";
    gDrawingContext.fill();
}
```

A to je… vlastně všechno. Zbytek programu tvoří herní logika – rozlišování mezi možnými a nemožnými tahy, sledování počtu tahů, zjištění, kdy hra končí. S devíti kroužky, několika čárami a jedním ovladačem `onclick` jsme v `<canvas>` vytvořili celou hru. Hurá!

4.10 K dalšímu čtení

- Canvas tutorial[2] v Mozilla Developer Center
- HTML5 canvas — the basics[3] od Mihai Sucana
- CanvasDemos.com[4]: dema, nástroje a návody pro HTML prvek plátno
- The canvas element[5] v návrhu standardu HTML5
- Internet Explorer 9 Guide for Developers: HTML5 canvas element[6]

2 https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
3 https://dev.opera.com/articles/html5-canvas-basics/
4 http://box513.bluehost.com/suspended.page/disabled.cgi/nestedelements.com
5 https://html.spec.whatwg.org/multipage/scripting.html#the-canvas-element
6 https://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_canvas

5. Video na webu

5. Video na webu – 121

5.1 Začínáme

Každý, kdo v posledních čtyřech letech navštívil YouTube.com, ví, že na webovou stránku se dá vložit video. Před HTML5 ovšem neexistovaly standardy, které by toto umožňovaly. Prakticky každé video, které jste „na webu" zhlédli, bylo promítáno prostřednictvím zásuvného modulu třetí strany – mohl to být třeba QuickTime, RealPlayer nebo Flash. (YouTube používá Flash.) Tyto moduly se do vašeho prohlížeče integrují tak dobře, že si jejich přítomnost ani neuvědomujete. Tedy až do okamžiku, kdy se pokusíte přehrát video na platformě, která daný zásuvný modul nepodporuje.

HTML5 definuje standardní způsob vkládání videa na webovou stránku: prvek `<video>`. Podpora tohoto prvku se ještě vyvíjí, což je slušný způsob jak říct, že zatím nefunguje. Přinejmenším ne všude. Ale nezoufejte! Existuje spousta alternativ, náhradních řešení a možností.

Podpora prvku `<video>`						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
9.0+	3.5+	3.0+	3.0+	10.5+	1.0+	2.0+

Avšak podpora samotného prvku `<video>` je jen částí příběhu. Než se začneme bavit o videu v HTML5, musíte vědět něco o video samotném. (Pokud už se ve videu vyznáte, můžete přeskočit rovnou na Co funguje na webu).

5.2 Videokontejnery

Možná, že o videu přemýšlíte v kategoriích „souborů AVI" nebo „souborů MP4". Ve skutečnosti však jsou „AVI" a „MP4" pouze formáty kontejneru. Stejně jako soubor ZIP může obsahovat soubory různého druhů, kontejnery videa určují jenom *způsob* ukládání obsahu, a ne *druh* obsahu samotného. (Vlastně je to ještě o něco komplikovanější, jelikož každý proud videa není kompatibilní s každým formátem kontejneru, to ale prozatím nechme být.)

Soubor videa normálně obsahuje několik stop: videostopu (bez zvuku) a jednu nebo několik zvukových stop (bez videa). Stopy jsou obvykle vzájemně propojeny. Zvuková stopa obsahuje značky, které napomáhají synchronizaci audia s videem. Jednotlivé stopy mohou obsahovat metadata, jako je název videa, obrázek na přebalu, čísla epizod (u televizních seriálů) a tak dále.

Existují *mraky* formátů pro video kontejnery. Mezi ty nejpopulárnější patří:

- MPEG 4, obvykle s příponou `.mp4` nebo `.m4v`. Kontejner MPEG 4 je založen na starším kontejneru QuickTime od Apple (.mov). Upoutávky k filmům na stránkách Apple dosud

používají tento starší kontejner QuickTime, ale filmy k zapůjčení na iTunes se dodávají v kontejneru MPEG 4.

- Ogg, obvykle s příponou `.ogv`. Ogg je otevřený standard, přátelský k open source systémům a nechráněný žádnými známými patenty. Firefox 3.5, Chrome 4 a Opera 10.5 nativně (bez zásuvných modulů pro jednotlivé platformy) podporují kontejnerový formát Ogg, video Ogg (nazývané „Theora“) a audio Ogg (nazývané „Vorbis“). Co se týče stolních systémů, významné linuxové distribuce podporují Ogg samy od sebe a na Mac a Windows ho můžete používat po nainstalování komponent QuickTime, respektive filtrů DirectShow. Také ho na všech platformách dokáže přehrát vynikající přehrávač VLC.

- Flash Video, obvykle s příponou `.flv`. Asi nepřekvapí, že Flash Video používá Adobe Flash. Před nástupem Flash 9.0.60.184 (také známého jako Flash Player 9 Update 3) to byl jediný kontejnerový formát, který Flash podporoval. Všechny poslední verze Flash také podporují kontejner MPEG 4.

- WebM, obvykle s příponou `.webm`. WebM je bezplatná, k open source systémům přátelská komprese videa vyvinutá speciálně pro použití s videem HTML5, která využívá videokodek VP8 a audiokodek Vorbis. Z technického hlediska je podobná jinému formátu – Matroska. V novějších verzích Chromium, Google Chrome, Mozilla Firefox a Opera je její podpora nativní, bez nutnosti zásuvných modulů pro jednotlivé platformy.

- ASF, obvykle s příponou `.asf`. Kontejnerový formát ASF byl vyvinut Microsoftem pro vysílání videa. Obsahoval báječný systém řízení digitálních práv, který znemožňoval uživatelům vytvoření zálohy jejich poctivě pořízených licencí. Takže pokud jste ztratili licenci na obsah, byli jste nuceni si ji pořídit znovu.

- Audio Video Interleave, obvykle s příponou `.avi`. Kontejnerový formát AVI vynalezl Microsoft, a to v jednodušších časech, kdy samotný fakt, že počítač je schopen přehrát video, byl něčím ohromujícím. Oficiálně nepodporuje funkce novějších kontejnerových formátů jako vložená metadata. Oficiálně nepodporuje dokonce ani většinu dnes používaných video a audiokodeků. Jak šel čas, různé firmy se ho pokoušely vzájemně neslučitelnými způsoby rozšířit o podporu toho či jiného, a je to dosud výchozí kontejnerový formát pro řadu populárních kodérů, jako je MEncoder.

5.3 Videokodeky

Když mluvíte o „zhlédnutí videa“, myslíte tím patrně kombinaci jedné video stopy s jednou zvukovou stopou. Nemáte ale přitom dva soubory, máte jenom „video“. Třeba soubor AVI nebo MP4. Ty jsou pouze kontejnerovými formáty podobnými souboru ZIP, který obsahuje soubory

různého druhu. Kontejnerový formát rozhoduje o způsobu úschovy videostop a zvukových stop v jediném souboru.

Když „se díváte na video“, váš video přehrávač dělá alespoň tři věci najednou:

- Interpretuje kontejnerový formát a zjišťuje, jaké obsahuje video a zvukové stopy a jakým způsobem jsou uloženy v souboru, aby mohl najít data, která je třeba dekódovat v dalším kroku.
- Dekóduje datový proud videa a zobrazuje na obrazovce sérií obrazů.
- Dekóduje datový proud zvuku a posílá zvuk do reproduktorů.

Videokodek je algoritmus, kterým je kódován datový proud videa, tj. který určuje, jak provést bod 2 výše. (Slovo „kodek“ je splynulinou, kombinací slov „kodér“ a „dekodér“.) Videopřehrávač *dekóduje* proud videa podle *videokodeku* a pak na obrazovce ukazuje sérii obrazů, neboli „snímky“. Většina současných videokodeků používá nejrůznější triky, aby minimalizovala množství informace potřebné k zobrazení jednoho snímku za druhým. Například místo ukládání jednotlivých snímků (jako snímků obrazovky) ukládají pouze rozdíly mezi snímky. Většina videí se ve skutečnosti liší mezi snímky jdoucími po sobě velmi málo, díky čemuž se zvyšuje stupeň komprese a snižuje se velikost souboru.

Existují *ztrátové* a *bezeztrátové* videokodeky. Bezeztrátové video je příliš velké pro použití na webu, proto se soustředím na ztrátové kodeky. Název *ztrátový videokodek* poukazuje na to, že při kódování videa informace nenávratně mizí. Stejně jako při kopírování audiokazety se při každém kódování informace ze zdrojového videa ztrácí a kvalita se zhoršuje. Zatímco audiokazeta „syčí“, na videu kódovaném poněkolikáté se mohou objevit kostičky, zejména v akčních scénách. (Toto se vlastně může stát, i když budete kódovat přímo z originálního zdroje, pokud vyberete slabý videokodek nebo zadáte špatné parametry.) Výhodou ovšem je, že ztrátové videokodeky mohou nabídnout úžasný stupeň komprese cestou vyhlazování kostiček při přehrávání tak, že nebudou pro lidské oko znatelné.

Existují *tuny* videokodeků. Třemi nejvýznamnějšími z nich jsou H.264, Theora a VP8.

H.264

H.264 je také známý pod názvy „MPEG-4 part 10“, „MPEG-4 AVC“ a „MPEG-4 Advanced Video Coding“. H.264 byl také vyvinut skupinou MPEG a standardizován v roce 2003. Má za cíl poskytnout jednotný kodek pro přístroje se slabým procesorem a malou šířkou pásma (chytré telefony), přístroje s výkonným procesorem a velkou šířkou pásma (současné stolní počítače) a všechno mezi tím. Aby toho dosáhl, je standard H.264 rozdělen do „profilů“, z nichž každý

definuje sadu volitelných funkcí, které mění složitost na úkor velikosti souboru. Vyšší profily používají více volitelných funkcí, nabízí lepší vizuální kvalitu při menší velikosti souboru, déle se kódují a vyžadují větší výkon procesoru pro dekódování v reálném čase.

Pro hrubou představu o nabídce profilů: iPhone od Apple podporuje profil Baseline, set-top box AppleTV podporuje profily Baseline a Main a Adobe Flash pro stolní počítače podporuje profily Baseline, Main a High. YouTube momentálně používá video H.264 pro kódování videí s vysokým rozlišením, přehratelné pomocí Adobe Flash; YouTube také poskytuje video H.264 mobilním zařízením včetně iPhonu od Apple a mobilů s operačním systémem Android od Google. H.264 je také jedním z video kodeků obsažených ve specifikaci Blu-Ray; disky Blu-Ray, které ho používají, používají obvykle profil High.

Většina zařízení mimo PC, která umějí přehrát video H.264 (včetně iPhonů a samostatných přehrávačů Blu-Ray), ve skutečnosti provádí dekódování na vyhrazeném čipu, jelikož jejich hlavní procesor zdaleka není schopen dekódovat video v reálném čase. Dnes dokonce i grafické karty s nízkým výkonem podporují hardwarové dekódování H.264. Existují konkurenční kodéry H.264, jako open source knihovna x264. **Využití standardu H.264 vyžaduje licenci**; sjednat ji můžete prostřednictvím skupiny MPEG LA. Video H.264 se dá vložit do většiny rozšířených kontejnerových formátů včetně MP4 (používá ho hlavně obchod iTunes firmy Apple) a MKV (používají ho hlavně nekomerční nadšenci do videa).

THEORA

Kodek Theora se vyvinul z kodeku VP3 a je následně vyvíjen nadací Xiph.org. **Theora je bezplatným kodekem a není chráněn žádným známým patentem** kromě originálních patentů VP3, jejichž licence jsou ovšem udělovány bezplatně. Přestože standard byl od roku 2004 „zmrazen", projekt Theora (zahrnující referenční open source kodér a dekodér) vydal jenom verzi 1.0 v listopadu 2008 a verzi 1.1 v září 2009.

Video Theora se dá vložit do libovolného kontejnerového formátu, nejčastěji se ovšem vyskytuje v kontejneru Ogg. Každá významná distribuce Linuxu podporuje Theora sama od sebe a Mozilla Firefox 3.5 má nativní podporu videa Theora v kontejneru Ogg. Slovem „nativní" mám na mysli „dostupnou pro všechny platformy bez zásuvných modulů pro jednotlivé platformy". Video Theora můžete také přehrát ve Windows nebo Mac OS X po nainstalování dekódovacího softwaru od Xiph.org.

VP8

VP8 je dalším kodekem od On2, společnosti, která původně vyvinula VP3 (později Theora). Technicky má podobný výstup jako profil High kodeku H.264, ale složitost dekódování má nízkou, na úrovni profilu H.264 Baseline.

V roce 2010 Google koupil On2 a publikoval specifikace videokodeku a ukázkový kodér a dekodér jako open source. Zároveň s tím Google „otevřel" všechny licence, které On2 měla na VP8, tím, že je udělil bezplatně. (A to je maximum, co se dá s licencemi udělat: nelze je „uvolnit" nebo zrušit poté, co byly vydány. Aby se staly open source, musí být udělovány zdarma, a každý tak bude moci využívat technologií, na které se patent vztahuje, aniž by musel za to platit nebo vyjednávat licence.) Ke dni 19. května 2010 **je VP8 bezplatným moderním kodekem, jehož použití není omezeno žádnou známou licencí** kromě těch, které On2 (teď Google) již udělila bez poplatků.

5.4 Zvukové kodeky

Pokud se nechcete omezovat na sledování pouze filmů natočených někdy před rokem 1927, potřebujete k videím zvukové stopy. Podobně jako videokodeky jsou *zvukové kodeky* algoritmy kódování zvukových datových proudů. Stejně jako u videokodeků existují *ztrátové* a *bezeztrátové* zvukové kodeky. A stejně jako bezeztrátové video je bezeztrátové audio pro umístění na web opravdu příliš velké. Takže se soustředím na ztrátové zvukové kodeky.

Vlastně je ten okruh ještě užší, protože existují různé kategorie ztrátových zvukových kodeků. Audio se používá i tam, kde není video (např. v telefonii), a existuje také zvláštní kategorie pro zvukové kodeky optimalizované pro kódování mluveného slova. Těmito kodeky nezkopírujete hudbu z cédéčka, protože by zněla jako zpěv čtyřletého děcka z telefonního sluchátka. Naopak by se hodily v telefonní ústředně Asterisk, protože šířka pásma je vzácná a tyto kodeky dokážou zkomprimovat mluvené slovo na zlomek velikosti dosažené kodeky určenými pro všeobecné použití. Avšak kvůli absenci podpory v prohlížečích – jak nativní, tak i pomocí zásuvných modulů – se použití kodeků optimalizovaných pro mluvené slovo na webu nikdy nerozšířilo. Proto se budu věnovat *ztrátovým zvukovým kodekům pro všeobecné použití*.

Jak jsem už zmínil výše, když „se díváte na video", váš počítač dělá alespoň tři věci najednou:

- Interpretuje kontejnerový formát.
- Dekóduje datový proud videa.
- Dekóduje datový proud zvuku a posílá zvuk do reproduktorů.

Zvukový kodek určuje, jak provést bod 3: dekódování zvukového datového proudu a jeho převedení do digitálních křivek, které pak vaše reproduktory přemění ve zvuk. Stejně jako u videokodeků zde existuje spousta triků, jak zmenšit množství informací ukládaných ve zvukovém proudu. Jelikož mluvíme o *ztrátových* zvukových kodecích, v průběhu cyklu nahrávání → kó-

dování → dekódování → poslech se informace ztrácí. Různé zvukové kodeky se přitom zbavují různých věcí, ale všechny mají stejný účel: oklamat vaše uši, abyste si nevšimli, že něco chybí.

Jeden z konceptů audia, který video postrádá, jsou *kanály*. Posíláme zvuk do reproduktorů, že ano? No a kolik máte reproduktorů? Pokud sedíte u počítače, je pravděpodobné, že máte jenom dva: levý a pravý. Můj počítač má tři: levý, pravý a jeden na podlaze. Takzvané systémy „prostorového zvuku" mohou mít šest a více reproduktorů, strategicky rozmístěných po pokoji. Do každého z reproduktorů je pouštěn některý *kanál* originální nahrávky. Když sedíte uprostřed šesti reproduktorů, doslova obklíčeni šesti různými zvukovými kanály, teoreticky by je váš mozek měl sloučit a dodat vám pocit, že jste ve středu dění. Funguje to? Mnohomiliardový zábavní průmysl podle všeho věří, že ano.

Většina zvukových kodeků pro všeobecné použití zvládá dva zvukové kanály. Při nahrávání je zvuk rozdělen do levého a pravého kanálu; při kódování jsou oba kanály uloženy do jednoho zvukového proudu; při dekódování jsou oba kanály dekódovány a každý je pouštěn do příslušného reproduktoru. Některé zvukové kodeky zvládají více než dva kanály a sledují, kam který z nich patří, aby si váš přehrávač nespletl reproduktory.

Existuje *spousta* zvukových kodeků. Už jsem říkal, že existuje spousta zvukových kodeků? No, to je jedno. Zvukových kodeků jsou hromady a hromady, ale v souvislosti s webem potřebujete vědět jen o třech z nich: MP3, AAC a Vorbis.

MPEG-1 AUDIO LAYER 3

MPEG-1 Audio Layer 3 je lidově známý jako „MP3". Pokud jste neslyšeli o MP3, nevím, co s vámi. Walmart prodává přenosné hudební přehrávače a říká jim „MP3 přehrávače". Znáte Walmart? Každopádně…

MP3 mohou obsahovat **až 2 kanály** zvuku. Kódovány mohou být s různým *bitovým* tokem: 64 kbps, 128 kbps, 192 kbps a řadou dalších od 32 do 320. Čím je větší bitový tok, tím je větší soubor a lepší kvalita zvuku, i když není úměra mezi kvalitou a bitovým tokem přímá. (128 kbps zní více než dvakrát lépe než 64 kbps, ale 256 kbps nezní dvakrát lépe než 128 kbps.) Navíc formát MP3 umožňuje *kódování s variabilním bitovým tokem*, což znamená, že některé části kódovaného proudu jsou zkomprimovány více než jiné. Například ticho mezi tóny může být kódováno s nízkým bitovým tokem, který v dalším okamžiku vyskočí nahoru, když více nástrojů zahraje složitý akord. MP3 je možné také kódovat s konstantním bitovým tokem, čemuž se – nepřekvapivě – říká *kódování s konstantním bitovým tokem*.

MP3 standard neurčuje přesně, jak kódovat soubory MP3 (ale přesně určuje, jak je dekódovat); různé kodéry používají různé psychoakustické modely, jejichž výsledky se sice drasticky liší, ale dají se dekódovat stejnými přehrávači. Open source projekt LAME je nejlepším bezplatným

kodérem a podle některých je vůbec nejlepším kodérem pro všechny bitové toky kromě těch nejnižších.

Formát MP3 (standardizovaný v roce 1991) **je chráněn patentem**, což vysvětluje, proč Linux nemá zabudované prostředky pro jeho přehrávání. Téměř všechny přenosné hudební přehrávače podporují samostatné soubory MP3 a zvukové stopy MP3 je možné vložit do kteréhokoliv videokontejneru. Adobe Flash umí přehrát jak samostatné soubory MP3, tak i zvukové stopy MP3 ve videokontejneru MP4.

ADVANCED AUDIO CODING

Advanced Audio Coding je s láskou nazýván „AAC". Byl standardizován v roce 1997 a proslavil se poté, co si ho společnost Apple vybrala jako výchozí formát pro svůj obchod iTunes. Zpočátku byly všechny soubory AAC, které se „kupovaly" v obchodě iTunes, šifrovány pomocí vlastní technologie Apple pro správu digitálních práv, nazývané FairPlay. Teď jsou některé písničky v obchodě iTunes dostupné jakožto nechráněné soubory AAC, kterým Apple říká „iTunes Plus", protože to zní mnohem lépe, než kdyby říkal všemu ostatnímu „iTunes Minus". **Formát AAC je chráněn patentem**; ceny licencí můžete najít na Internetu[1].

AAC byl vyvinut, aby poskytl lepší kvalitu zvuku v porovnání s MP3 při stejném *bitovém toku*, a může kódovat audio s jakýmkoliv bitovým tokem. (MP3 má omezený počet hodnot bitového toku a horní hranici 320 kbps.) AAC může kódovat **až 48 zvukových kanálů**, v praxi to však nikdo nedělá. Také se formát AAC od MP3 liší tím, že má několik *profilů*, stejně jako H.264, a to ze stejných důvodů. „Jednodušší" profil je určen pro přehrávání v reálném čase na zařízeních s omezeným výkonem procesoru, kdežto složitější profily nabízejí lepší kvalitu zvuku při stejném bitovém toku za cenu pomalejšího kódování a dekódování.

Všechna současná produkce Apple, jako jsou iPody, AppleTV a QuickTime, podporuje některé profily AAC v samostatných zvukových souborech a zvukových proudech ve videokontejneru MP4. Adobe Flash podporuje všechny profily AAC v MP4, stejně jako open source přehrávače videa MPlayer a VLC. Pro kódování je možné použít open source knihovnu FAAC; její podporu můžete nastavit při kompilaci v utilitě mencoder nebo ffmpeg.

VORBIS

Vorbis se často nazývá „Ogg Vorbis", přestože to není technicky správně. („Ogg" je pouze kontejnerovým formátem a zvukové proudy Vorbis je možné vložit do jiných kontejnerů.) **Vorbis není chráněn žádným známým patentem**, a proto je podporován sám od sebe všemi významnými distribucemi Linuxu a přenosnými zařízeními, které běží na open source firmwaru Rockbox. Mozilla Firefox 3.5 podporuje zvukové soubory Vorbis v kontejneru Ogg nebo videa Ogg

1 http://www.vialicensing.com/licensing/aac-fees.aspx

se zvukovou stopou Vorbis. Mobily s Androidem také umějí přehrát samostatné zvukové soubory Vorbis. Zvukové proudy Vorbis se obvykle vkládají do kontejneru Ogg nebo WebM, ale mohou být také vloženy do kontejneru MP4 nebo MKV (a s provedením některých úprav také do AVI). Vorbis podporuje **libovolný počet zvukových kanálů**.

Existují open source kodéry a dekodéry pro Vorbis, např. OggConvert (kodér), ffmpeg (dekodér), aoTuV (kodér) a libvorbis (dekodér). Existují také komponenty QuickTime pro Mac OS X a filtry DirectShow pro Windows.

5.5 Co funguje na webu

Jestli se ještě nenudíte, vedete si lépe než většina ostatních. Jak vidíte, video (a audio) je náročné téma – a to ještě byla zkrácená verze! Určitě vás zajímá, jak toto všechno souvisí s HTML5. HTML5 nabízí prvek `<video>` umožňující vkládat na webové stránky videa. Pro svoje video můžete bez omezení použit jakýkoliv videokodek, zvukový kodek a kontejnerový formát. Jedním prvkem `<video>` můžete odkazovat na několik videosouborů a prohlížeč z nich vybere první videosoubor, který dokáže přehrát. **Musíte sami zjistit, které prohlížeče podporují které kontejnery a kodeky**.

V okamžiku napsání této knihy vypadá situace s videem v HTML5 takto:

- Firefox 3.5+ podporuje video Theora a audio Vorbis v kontejneru Ogg. Firefox 4+ navíc podporuje WebM.

- Opera 10.5+ podporuje video Theora a audio Vorbis v kontejneru Ogg. Opera 10.60 (a novější) podporuje také WebM.

- Chrome 3.0+ podporuje H.264, video Theora a audio Vorbis v kontejneru Ogg. Chrome 6.0+ podporuje také WebM.

- Safari 3.0+ pro Macintosh a Windows podporuje cokoliv, co podporuje QuickTime. Teoreticky můžete uživatele požádat o instalaci zásuvných modulů QuickTime třetí strany. V praxi to však udělá málokdo. Takže vám zbývají pouze formáty, které QuickTime podporuje sám od sebe. Jejich seznam je dlouhý, ale neobsahuje WebM, Theoru, Vorbis ani kontejner Ogg. Přesto má QuickTime podporu videa H.264 (profil Main) a audia AAC v kontejneru MP4.

- Chytré telefony jako iPhone od Apple a mobily s Google Android podporují video H.264 (profil Baseline) a audio AAC (nejjednodušší profil „low complexity“) v kontejneru MP4.

- Adobe Flash (9.0.60.184 a novější) podporuje video H.264 (všechny profily) a audio AAC (všechny profily) v kontejneru MP4.

- Internet Explorer 9+ podporuje všechny profily videa H.264 a audio AAC nebo MP3 v kontejneru MP4. Také přehraje video WebM, pokud nainstalujete kodek od třetí strany, který se nedodává spolu s žádnou verzí Windows. Jiné kodeky od třetích stran IE nepodporuje (na rozdíl od Safari, který přehraje cokoliv, co zvládne QuickTime).

- Internet Explorer 8 vůbec nemá podporu videa HTML5, ale skoro všichni uživatelé Internet Exploreru mají zásuvný modul Adobe Flash. Dále v této kapitole vám ukážu, jak můžete za použití videa HTML5 elegantně couvnout k Flash.

Možná to bude stravitelnější ve formě tabulky.

Podpora video kodeků v běžných prohlížečích							
Kodeky/kontejner	**IE**	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
Theora+Vorbis+Ogg	·	3.5+	†	5.0+	10.5+	·	·
H.264+AAC+MP4	9.0+	·	3.0+	5.0+‡	·	3.0+	2.0+
WebM	9.0+*	4.0+	†	6.0+	10.6+	·	2.3+
* Internet Explorer 9 bude podporovat WebM pouze tehdy, „když uživatel nainstaluje kodek VP8“.							
† Safari přehraje všechno, co přehraje QuickTime. QuickTime má zabudovanou podporu H.264/AAC/MP4. Je možné nainstalovat zásuvné moduly třetích stran, které přidají podporu Theora a WebM, ale každý uživatel potřebuje mít tyto moduly již nainstalované, aby Safari mohl tyto formáty rozpoznat.							
‡ Google Chrome v roce 2011 avizoval, že přestane podporovat H.264, ale dosud se to nestalo.							

A abych vás dorazil:

PROFESOR ZNAČKA ŘÍKÁ

Neexistuje ani jedna kombinace kontejnerů a kodeků, která by fungovala ve všech prohlížečích s podporou HTML5.

V dohledné době se to zřejmě nezmění.

Pro zajištění přehratelnosti videa na všech zařízeních a platformách ho budete muset kódovat více než jednou.

Vaše akce k zajištění maximální kompatibility pak budou vypadat přibližně takto:

- Uděláte verzi, která používá WebM (VP8 + Vorbis).
- Uděláte další verzi, která používá video H.264 baseline a audio AAC „low complexity“ v kontejneru MP4.
- Uděláte zase další verzi, která používá video Theora a audio Vorbis v kontejneru Ogg.*
- Odkážete se na všechny tři videosoubory v jednom prvku <video> a vrátíte se k videopřehrávači na bázi Flash.

* WebM a H.264 mají dostatečnou podporu. Takže pokud nepočítáte s Firefoxem 3.5 nebo Operou 10.5, můžete Theoru vypustit.

5.6 Problémy s licencováním videa H.264

Než budeme pokračovat, musím poukázat na to, že kódování videa dvakrát je spojeno s náklady. Samozřejmě samotné opakování tohoto úkonu je nákladné, protože zabírá více času a vyžaduje více výpočetních operací, než kdybyste to dělali jen jednou. Existují ale i reálné náklady spojené s videem H.264: náklady licenční.

Pamatujete si, jak jsem poprvé mluvil o videu H.264 a mimoděk jsem se zmínil o tom, že tento videokodek je chráněn patentem a že licence vydává skupina MPEG LA? Je to docela důležitá skutečnost. Abyste pochopili, proč je tak důležitá, odkážu vás na bludiště licencování H.264:

> MPEG LA ve svém portfoliu licencí H.264 rozlišuje licence dvou druhů: jedny pro výrobce kodérů a dekodérů a druhé pro distributory obsahu. (…)
>
> Licence pro distributory se dále dělí na čtyři klíčové poddruhy, z nichž dva (předplatné a nákup nebo placené použití jednotlivých položek) jsou pro situaci, kdy za videoslužby platí přímo koncový uživatel, a dva další („bezplatná“ televize nebo internetové vysílání) jsou pro úhradu z jiných zdrojů, než je cílový divák. (…)
>
> Licenční poplatek pro „bezplatnou“ televizi má dvě varianty. První je jednorázová úhrada 2 500 dolarů za každý kodér vysílání AVC, tedy kodér AVC „používaný pro vysílání videa AVC Držitelem licence přímo nebo v zastoupení pro Koncového uživatele,“ který si ho dekóduje a zhlédne. Pokud se ptáte, zda toto není dvojité účtování, máte pravdu: licenční poplatek již byl uhrazen výrobci kodéru, a vysílací stanice následně zaplatí jednu z variant licenčního poplatku.

> Druhou variantou je roční poplatek za vysílání. … Roční poplatek za vysílání je rozdělen podle počtu diváků:
>
> - 2 500 dolarů za kalendářní rok pro televizní trh s 100 000 až 499 999 domácnostmi
> - 5 000 dolarů za kalendářní rok pro televizní trh s 500 000 až 999 999 domácnostmi
> - 10 000 dolarů za kalendářní rok pro televizní trh s 1 000 000 a více domácnostmi
>
> … Proč by se ale někdo nezapojený do poskytování televizního vysílání měl starat o všechny tyto problémy kolem „bezplatné" televize? Jak jsem již zmínil výše, účastnické poplatky se vztahují na jakékoliv dodání obsahu. Po vymezení „bezplatné" televize jako vysílání nejen vzduchem MPEG LA dále rozšířila účastnické poplatky na vysílání přes Internet jako „video AVC dodávané prostřednictvím celosvětové sítě Internet koncovému uživateli, u nějž koncový uživatel nehradí poplatek za právo příjmu nebo prohlížení." Jinými slovy jakékoliv veřejnoprávní vysílání – vzduchem, přes kabel, satelit nebo Internet – podléhá účastnickým poplatkům. (…)
>
> Poplatky za internetové vysílání jsou poněkud přemrštěné, zřejmě s ohledem na očekávaný růst, který bude rychlejší než u vysílání vzduchem nebo „bezplatné" televize přes kabel či satelit. MPEG LA sečetla vysílací poplatek za „bezplatnou televizi" a poplatek navíc a na dobu prvního licenčního období, které končí 31. prosince 2010, poskytuje svého druhu odklad a uvádí, že „po skončení prvního období daný poplatek nepřesáhne výši poplatků splatných za stejnou dobu za bezplatnou televizi."

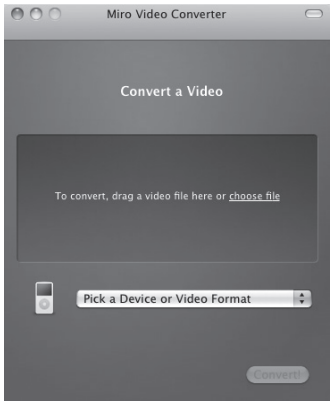
Poslední část o poplatkovém systému pro internetové vysílání už byla revidována. Skupina MPEG-LA se nedávno nechala slyšet, že internetové vysílání bude bez poplatků. To ale neznamená, že H.264 je bezplatný pro všechny uživatele. Zejména kodéry (například ten, co zpracovává videa nahraná na YouTube) a dekodéry (například ten zabudovaný v Microsoft Internet Explorer 9) stále podléhají licenčním poplatkům. Pro podrobnosti viz Free as in smokescreen[2].

5.7 Kódování videa pomocí videokonvertoru Miro

Existuje mnoho nástrojů pro kódování videa s mnoha funkcemi, které ovlivňují kvalitu videa. Jestliže nechcete trávit čas dozvídáním se čehokoliv okolo kódování videa, je tato kapitola právě pro vás.

2 http://shaver.off.net/diary/2010/08/27/free-as-in-smokescreen/

Video konvertor Miro je open source program pod licencí GPL určený pro kódování videa v různých formátech. Stáhnout si ho můžete pro Mac OS X i Windows[3]. Podporuje všechny výstupní formáty zmíněné v této kapitole. Z funkcí jsou dostupné pouze volba videosouboru a volba výstupního formátu. Jakožto vstupní soubor program přijme skoro každé video včetně videí DV natočených amatérskými videokamerami a z většiny videí vyprodukuje výsledek přiměřené kvality. Jelikož tento konvertor neumožňuje změnu nastavení, v případě, že by vám výsledek nevyhovoval, můžete jedině zkusit jiný program.



Pro začátek práce spusťte aplikaci Miro Video Converter.

← Hlavní obrazovka Miro Video Converter

Klikněte na „Choose file“ („Vyberte soubor“) a označte zdrojové video, které chcete kódovat.

← „Vyberte soubor“

Seznam v rozbalovací nabídce „Pick a Device or Video Format“ („Vyberte zařízení nebo formát videa“) obsahuje řadu zařízení a formátů. Pro účely této kapitoly se budeme zajímat jenom o tři z nich.

- *WebM (vp8)* je video WebM (video VP8 a audio Vorbis v kontejneru WebM).

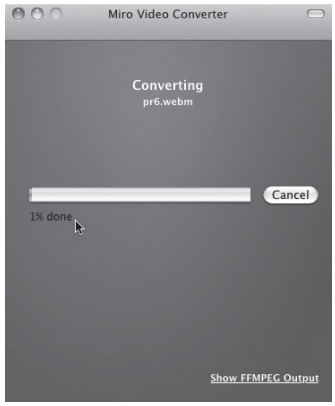
3 http://www.mirovideoconverter.com/

- *Theora* je video Theora a audio Vorbis v kontejneru Ogg.
- *iPhone* je video H.264 profilu Baseline a audio AAC low-complexity v kontejneru MP4.

Nejdříve zvolte „WebM“.

← Volba výstupu WebM

Klikněte na tlačítko „Convert“ („Konvertovat“) a Miro Video Converter okamžitě začne kódovat vaše video. Výstupní soubor se bude jmenovat `SOURCEFILE.webm` a bude uložen ve složce se zdrojovým videem.

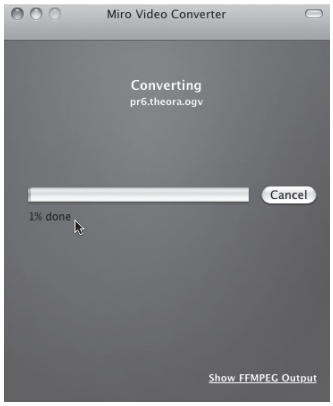


← Na tuto obrazovku se budete dívat dlouho

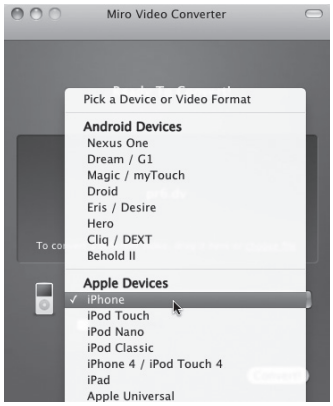
Jakmile bude kódování ukončeno, program se vrátí k hlavní obrazovce. Teď v seznamu Zařízení a formáty zvolte „Theora“.

← Na řadě je Theora

A je to. Znovu stiskněte „Konvertovat“ pro kódování videa Theora. Soubor se bude jmenovat `SOURCEFILE.theora.ogv` a bude uložen ve stejné složce jako zdrojové video.



← Teď si můžete dát kafe



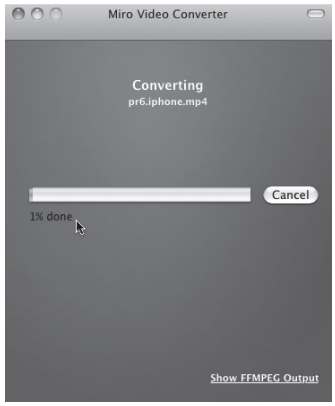
Nakonec dejte kódovat video H.264 kompatibilní s iPhone: zvolte v seznamu Zařízení a formáty položku „iPhone“.

← „iPhone“, ne „iPhone 4“

Při kódování videa kompatibilního s iPhone nabídne Miro Video Converter možnost poslat soubor do vaší knihovny iTunes. Nehodlám soudit, zda byste to měli udělat, či nikoliv, ale pro umístění videa na webu to není potřeba.

← Neposílejte do iTunes

Stiskněte kouzelné tlačítko „Konvertovat“ a počkejte. Kódovaný soubor se bude jmenovat `SOURCENAME.iphone.mp4` a bude uložen do stejné složky se zdrojovým videem.



← Třeba si zacvičte jógu

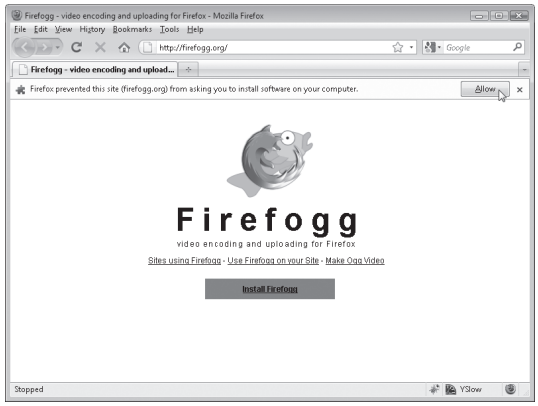
Teď byste měli mít tři videosoubory navíc k tomu původnímu. Pokud vám vyhovuje kvalita obrazu, přeskočte dále na „A konečně kód“, kde se dozvíte, jak je všechny spojit v jednom prvku `<video>`, který by fungoval v různých prohlížečích. Jestli se chcete dozvědět něco o dalších nástrojích nebo možnostech kódování videa, pokračujte ve čtení.

5.8 Kódování videa Ogg pomocí Firefoggu

(V této části používám pojem „video Ogg“ jako zkratku pro „video Theora a audio Vorbis v kontejneru Ogg“. Podpora této kombinace kodeků a kontejneru je zabudována v Mozilla Firefox a Google Chrome.)

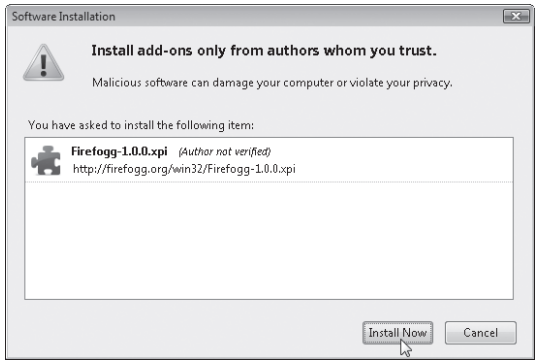
Firefogg je open source rozšíření pro Firefox pod licencí GPL, určené pro kódování videa Ogg. Pro jeho použití si potřebujete nainstalovat Mozilla Firefox 3.5 nebo novější verzi a pak navštívit firefogg.org.

← Domovská stránka Firefogg



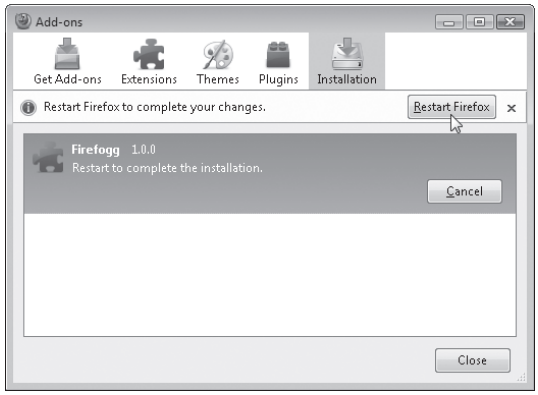
Klikněte na „Install Firefogg“ („Nainstalovat Firefogg“). Firefox se vás zeptá, zda si opravdu přejete umožnit stránce nainstalovat rozšíření. Pro pokračování klikněte na „Allow“ („Povolit“).

← Povolte instalaci Firefogg



Firefox zobrazí standardní okno instalace software. Pro pokračování klikněte na „Instalovat“.

← Nainstalujte Firefogg



Pro dokončení instalace klikněte na „Restartovat Firefox“.

← Restartujte Firefox

Po restartu Firefox potvrdí `firefogg.org`, že Firefogg byl úspěšně nainstalován.

← Úspěšně nainstalováno

Pro zahájení procesu kódování klikněte na „Vytvořit video Ogg“.

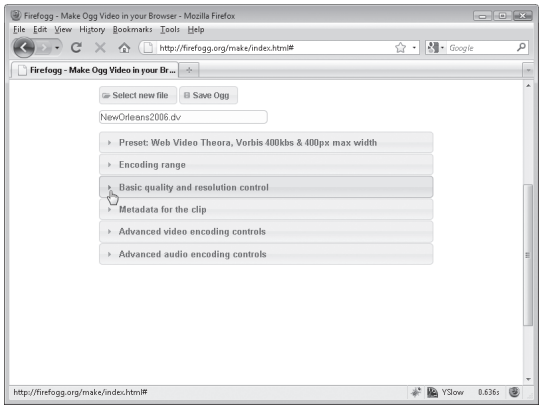
← Vytvořme si nějaké video!

Pro výběr zdrojového souboru klikněte na „Select file“ („Vybrat soubor“).

← Vyberte si videosoubor

Firefogg má šest „záložek“:

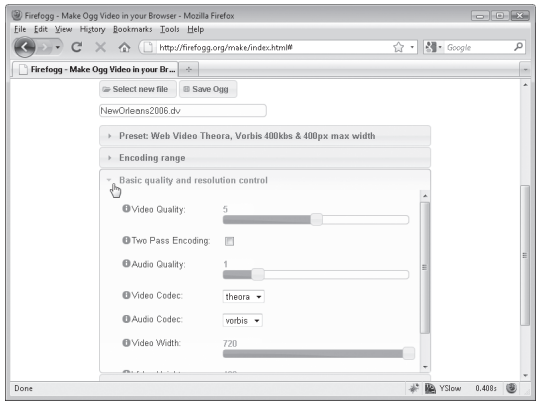
- Předvolby. Výchozí předvolba je „web video“, která se pro naše účely hodí.
- Rozsah kódování. Kódování videa může trvat dlouho. Když to děláte poprvé, je někdy vhodné, než najdete uspokojivou kombinaci nastavení, kódovat jen část videa (třeba prvních 30 vteřin).
- Základní nastavení kvality a rozlišení. Tady se nacházejí ta nejdůležitější nastavení.
- Metadata. Nebudu to tady rozepisovat, ale při kódování videa do něj můžete vložit popisky, např. název nebo jméno autora. Pravděpodobně jste už přidávali metadata do své hudební sbírky pomocí iTunes nebo jiného správce hudby. Tady je ta myšlenka stejná.
- Pokročilá nastavení kódování videa. Nehrajte si s tím, pokud nevíte, co to přesně udělá. (Firefogg poskytuje interaktivní nápovědu pro většinu z těchto možností. Klikněte na značku „i“ vedle příslušné možnosti, chcete-li o tom vědět více.)
- Pokročilá nastavení kódování zvuku. Nápodobně, nehrajte si s tím, pokud nevíte, co to přesně děláte.



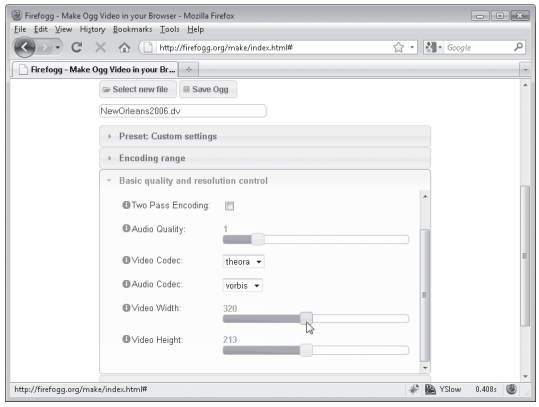
Tady popíšu pouze záložku Základní nastavení kvality a rozlišení. Ta obsahuje všechna důležitá nastavení:

- Kvalita videa. Měří se na stupnici s hodnotami od 0 (nejnižší kvalita) do 10 (nejvyšší kvalita). Vyšší hodnoty znamenají větší soubory, takže budete muset experimentovat, než naleznete optimální poměr mezi velikostí souboru a kvalitou.

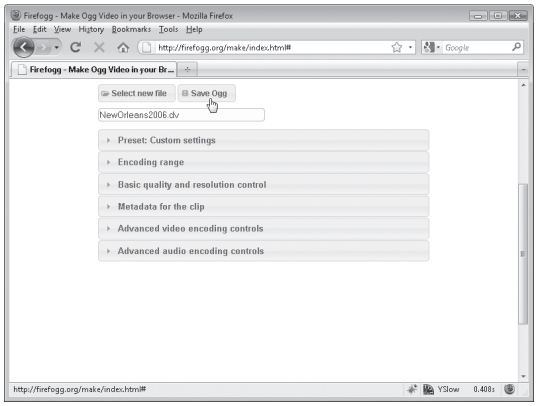
- Kvalita zvuku. Tato stupnice má hodnoty od -1 (nejnižší kvalita) do 10 (nejvyšší kvalita). Vyšší hodnoty znamenají větší soubory, stejně jako u nastavení kvality videa.
- Videokodek. Ten musí být nastavený na „Theora“.
- Zvukový kodek. Ten musí být nastavený na „Vorbis“.
- Šířka videa a výška videa. Výchozím nastavením jsou rozměry aktuálního zdrojového videa. Pokud pro kódování chcete změnit šířku nebo výšku, použijete tuto možnost. Firefogg automaticky přizpůsobí druhou hodnotu pro zachování poměru stran (aby se video neroztáhlo nebo nesrazilo).



V tomto příkladu změním šířku originálního videa na polovinu. Všimněte si, že Firefogg automaticky přizpůsobuje hodnotu výšky.

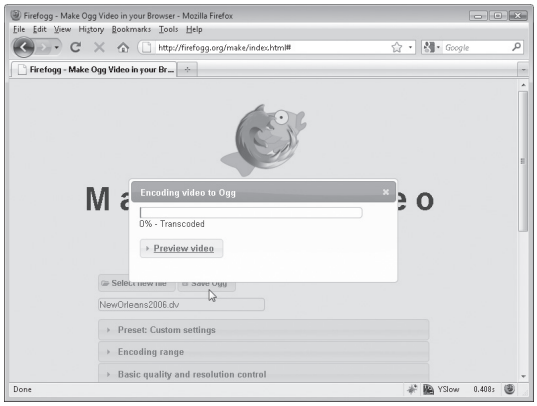


← Upravte šířku a výšku videa



Když jste si dost vyhráli se všemi knoflíky, klikněte na „Save Ogg“ („Uložit Ogg“) a proces kódování se spustí. Firefogg vás požádá o název souboru zakódovaného videa.

← „Uložit Ogg“



Firefogg při kódování videa zobrazí pěkný ukazatel průběhu. Teď už stačí jen čekat (a čekat, a čekat…)

← Probíhá kódování

5.9 Hromadné kódování videa Ogg pomocí ffmpeg2theora

(Stejně jako v předchozím oddílu tady budu používat pojem „video Ogg“ jako zkratku pro „video Theora a audio Vorbis v kontejneru Ogg“. Podpora této kombinace kodeků a kontejneru je zabudována v Mozille Firefox a Googlu Chrome.)

Pokud potřebujete kódovat velké množství videosouborů Ogg a chcete ten proces automatizovat, rozhodně musíte zkusit ffmpeg2theora[4].

ffmpeg2theora je open source aplikace pod licencí GPL určená pro kódování videa Ogg. Zkompilované binární soubory se dají stáhnout pro Mac OS X, Windows, a současné distribuce

4 http://v2v.cc/~j/ffmpeg2theora/

Linuxu[5]. Program přijme skoro každé video včetně DV videí natočených amatérskými videokamerami.

Pro začátek práce s ffmpeg2theora musíte vyvolat program pomocí příkazového řádku. (V Mac OS X otevřete Aplikace → Utility → Terminál. Ve Windows otevřete nabídku Start → Programy → Příslušenství → Příkazový řádek.)

ffmpeg2theora rozlišuje řadu přepínačů příkazového řádku. (Přečíst si o nich můžete, když použijete příkaz `ffmpeg2theora --help`.) Tady se budu zabývat jen třemi z nich.

- `--video-quality Q`, kde „Q" je číslo od 0 do 10.
- `--audio-quality Q`, kde „Q" je číslo od -2 do 10.
- `--max_size=WxH`, kde „W" a „H" jsou požadovaná maximální šířka a výška videa. („x" mezi nimi je prostě písmeno „x".) ffmpeg2theora změní rozměry videa přiměřeně k nastaveným hodnotám, takže výsledné video bude mít menší rozměry než W×H. Například kódováním videa s rozměry 720×480 do `--max_size 320x240` vytvoříte video s rozměry `320×213`.

Tady je příklad kódování videa se stejným nastavením jako v předchozím oddíle (kódování pomocí Firefogg).

```
you@localhost$ ffmpeg2theora --videoquality 5
                             --audioquality 1
                             --max_size 320x240
                             pr6.dv
```

Kódovaný soubor bude uložen do stejné složky jako zdrojové video a dostane příponu `.ogv`. Můžete určit jiné umístění a/nebo název souboru pomocí přepínače příkazového řádku `--output=/cesta/k/zakodovanemu/videu`.

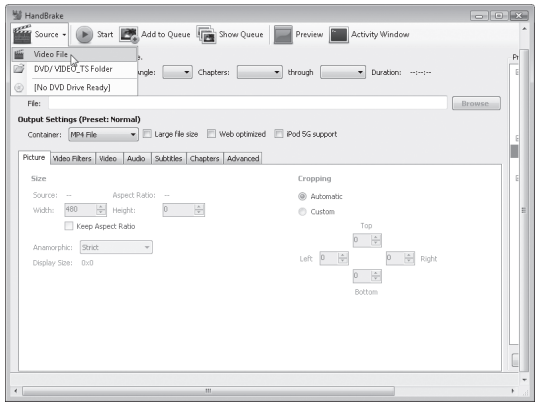
5.10 Kódování videa H.264 pomocí HandBrake

(V tomto oddílu budu používat pojem „video H.264" jako zkratku pro „video H.264 profilu baseline a audio AAC profilu low-complexity v kontejneru MPEG-4". Podpora této kombinace kodeků a kontejneru je zabudována v Safari, Adobe Flash, iPhonu a zařízeních s Google Android.)

5 http://v2v.cc/~j/ffmpeg2theora/download.html

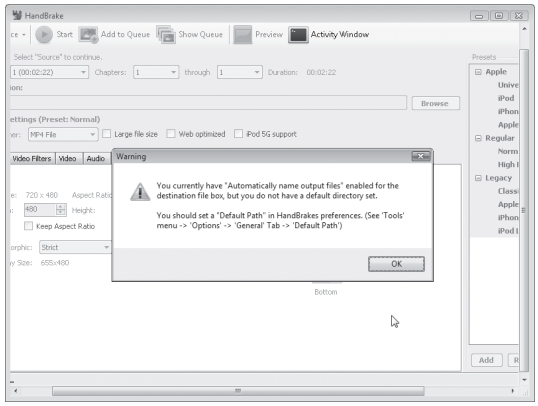
Když ponecháme stranou otázky licencování, nejjednodušším způsobem kódování videa H.264 je HandBrake. HandBrake je open source aplikace pod licencí GPL určená pro kódování videa H.264. (Dříve pracovala i s jinými formáty videa, ale v poslední verzi vývojáři ukončili podporu většiny z nich, aby se plně soustředili na video H.264.) Spustitelné soubory jsou dostupné pro Windows, Mac OS X a současné distribuce Linuxu.

HandBrake má dvě podoby: grafickou a příkazový řádek. Nejdříve vás seznámím s grafickým rozhraním a pak se podíváme, jak moje doporučená nastavení budou vypadat v příkazovém řádku.



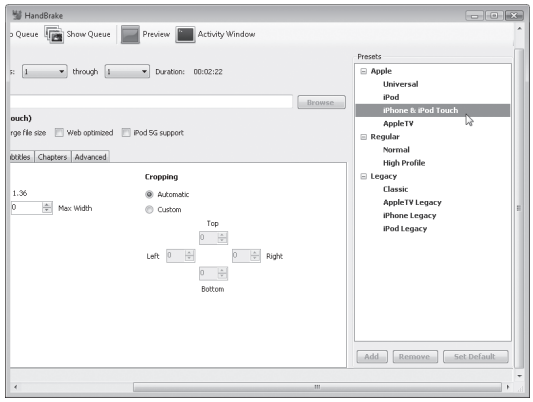
První, co musíte udělat po spuštění aplikace HandBrake, je vybrat zdrojové video. Klikněte na rozbalovací tlačítko „Source“ („Zdroj“) a pro výběr souboru zvolte „Video File“ („Videosoubor“). HandBrake je schopna zpracovat skoro každé video včetně DV videí natočených amatérskými videokamerami.

← Vyberte zdrojové video



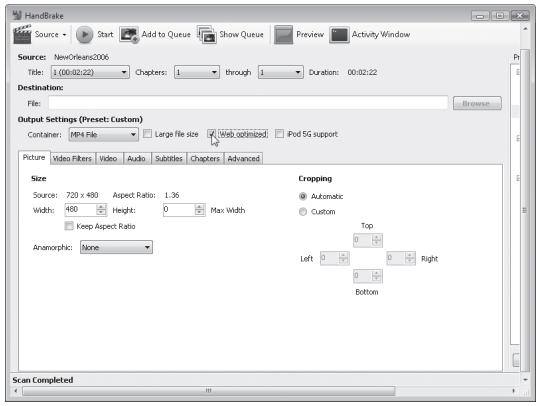
HandBrake si postěžuje, že jste nevybrali výchozí umístění pro zakódovaná videa. Můžete toto upozornění klidně ignorovat nebo otevřít okénko možností (z nabídky „Tools“ – „Nástroje“) a nastavit výchozí umístění pro výstupní soubory.

← Toto ignorujte



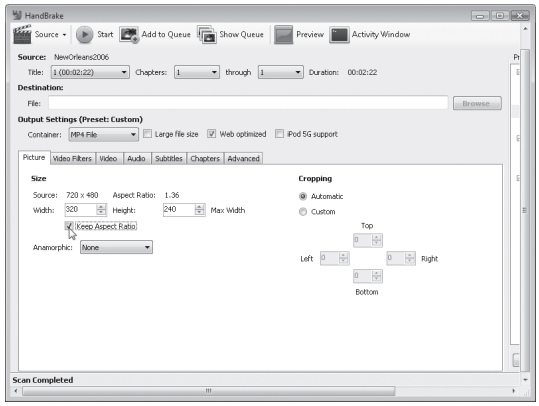
Vpravo se nachází seznam předvoleb. Volbou položky „iPhone & iPod Touch“ nastavíte většinu potřebných parametrů.

← Zvolte předvolbu iPhone



Jednou důležitou volbou, která je ve výchozím nastavení vypnutá, je „Web optimized“ („Optimalizovat pro web“). Aktivace této volby reorganizuje metadata v kódovaném videu tak, že je možné prohlížet začátek videa, zatímco zbytek se teprve stahuje. Vřele doporučuji vždy zapínat tuto možnost. Neovlivňuje to kvalitu ani velikost výsledného videa, takže nevidím důvod proč ji nepoužít.

← Vždy optimalizujte pro web



V záložce „Picture“ („Obraz“) můžete nastavit maximální šířku a výšku kódovaného videa. Také označte volbu „Keep Aspect Ratio“ („Zachovat poměr stran“), aby se video při změně velikosti nerozt áhlo nebo nesrazilo.

← Nastavte šířku a výšku

V záložce „Video“ můžete nastavit čtyři důležité volby.

- Videokodek. Ujistěte se, že máte zvolený „H.264 (x264)“.

- 2-Pass Encoding (Dvouprůchodové kódování). Když označíte tuto volbu, HandBrake spustí kodér videa dvakrát. Poprvé pro analýzu videa a zjištění věcí jako kompozice barev, pohyb a předěly scén. Podruhé provede samotné kódování s použitím zjištěných informací. Jak asi tušíte, trvá to přibližně dvakrát déle než jednoprůchodové kódování, ale poskytuje lepší kvalitu videa při stejné velikosti souboru. Při kódování videa H.264 vždy používám dvouprůchodové kódování. Pokud nebudujete nový YouTube a nekódujete videa 24 hodin denně, možná byste ho měli používat také.

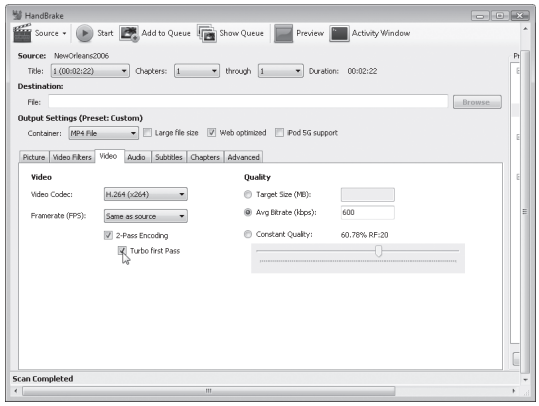
- Turbo First Pass (Zrychlený první průchod). Když zapnete dvouprůchodové kódování, můžete ušetřit trochu času tím, že zvolíte „zrychlený první průchod“. Ten zkracuje čas věnovaný první fázi (analýze videa) a snižuje kvalitu jen nepatrně. Co se mě týče, obvykle tuto funkci používám, ale je pokud pro vás kvalita nanejvýš důležitá, nechte ji vypnutou.

- Kvalita. „Kvalita“ kódovaného videa se dá stanovit různými způsoby. Můžete nastavit velikost výsledného souboru a HandBrake vám zajistí, že ji nepřesáhne. Můžete stanovit průměrný „bitový tok“, což je doslova počet bitů potřebných k ukládání jedné sekundy videa. (Říká se mu „průměrný“, protože některé sekundy potřebují více bitů než jiné.) Nebo můžete nastavit konstantní kvalitu na stupnici 0 až 100 %. Vyšší hodnoty znamenají vyšší kvalitu, ale i větší soubor. Kterou z možností použijete k určení kvality, je zcela na vás.

PTEJTE SE PROFESORA ZNAČKY

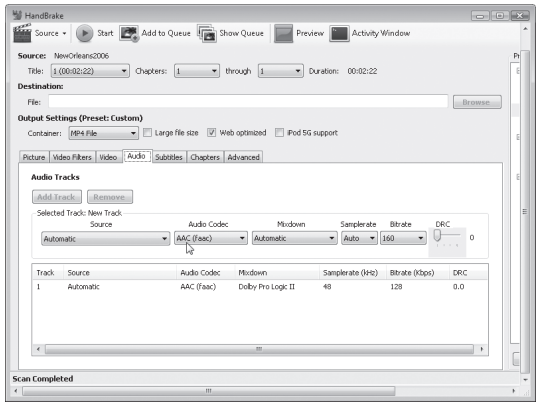
Q: Můžu použít dvouprůchodové kódování i u videa Ogg?

A: Ano, ale kvůli zásadním rozdílům ve fungování kodéru pravděpodobně nemusíte. Dvouprůchodové kódování H.264 vždy vede k lepší kvalitě videa. Dvouprůchodové kódování Ogg je užitečné, pouze když usilujete o konkrétní velikost souboru. (Možná že právě toto potřebujete, ale dané příklady se tím nezabývají a při kódování videa pro web ten čas navíc za to nestojí.) Pro nejlepší kvalitu videa Ogg používejte nastavení kvality videa a s dvouprůchodovým kódováním si nelamte hlavu.



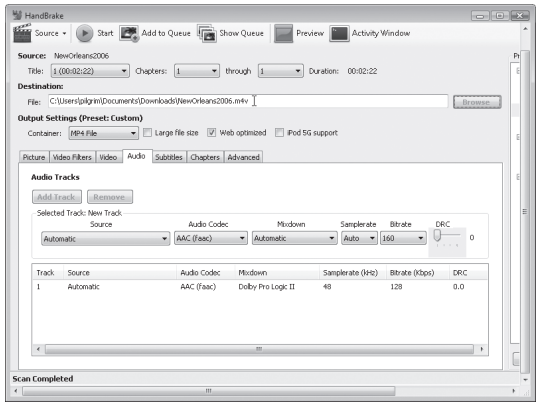
V tomto příkladu jsem vybral průměrný bitový tok 600 kbps, což se tak akorát hodí pro video s rozměry 320×240. (Dále v této kapitole vám ukážu vzor videa kódovaného s tokem 200 kbps.) Také jsem zapnul dvouprůchodové kódování se zrychleným prvním průchodem.

← Možnosti kvality videa



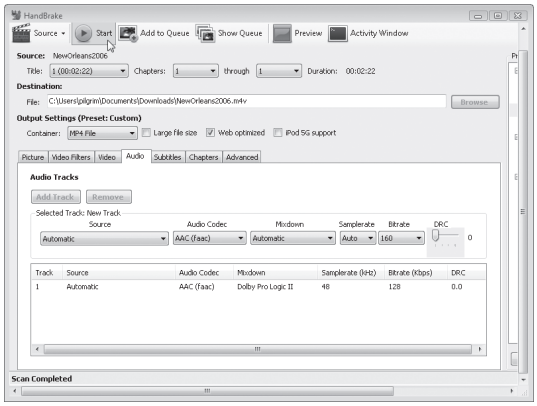
V záložce „Zvuk“ pravděpodobně nepotřebujete nic měnit. Pokud má vaše zdrojové video několik zvukových stop, musíte vybrat jednu, kterou použijete v kódovaném videu. Pokud je ve vašem videu jenom mluvená řeč (a ne hudba nebo obvyklé zvuky okolí), můžete snížit bitový tok třeba až na 96 kbps. Ostatní nastavení v předvolbě „iPhone“ mohou zůstat.

← Možnosti kvality zvuku



Dále stiskněte tlačítko „Browse“ („Procházet“) a vyberte umístění a název souboru vašeho kódovaného videa.

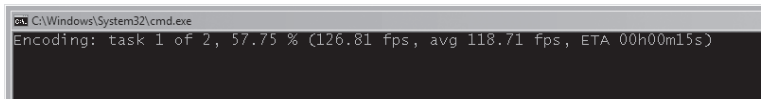
← Nastavte název výsledného souboru



Nakonec klikněte na „Start“, a tím se kódování zahájí.

← Pojďme si vytvořit video!

Během kódování videa HandBrake zobrazí informaci o průběhu.



← Trpělivost, cvrčku

5.11 Hromadné kódování videa H.264 pomocí HandBrake

(Stejně jako v předchozím oddílu tady budu používat pojem „video H.264“ jako zkratku pro „video H.264 profilu baseline a audio AAC profilu low-complexity v kontejneru MPEG-4“. Podpora této kombinace kodeků a kontejneru je zabudována v Safari, Adobe Flash, iPhonu a zařízeních s Google Android.)

HandBrake má také verzi v podobě příkazového řádku. Stejně jako ffmpeg2theora nabízí příkazový řádek HandBrake ohromné množství možností. (Přečíst si o nich můžete, když použijete příkaz `HandBrakeCLI --help`.) Soustředím se na několik z nich:

- `--preset "X"`, kde „X“ je název předvolby HandBrake. Předvolba, kterou potřebujete pro video H.264, se jmenuje „iPhone & iPod Touch“ a je důležité vložit celý název do uvozovek.
- `--width W`, kde „W“ je šířka kódovaného videa. HandBrake automaticky přizpůsobí výšku pro zachování poměru stran.
- `--vb Q`, kde „Q“ je průměrný bitový tok (v kilobitech za sekundu).
- `--two-pass`, která aktivuje dvouprůchodové kódování.
- `--turbo`, která aktivuje zrychlený první průchod při dvouprůchodovém kódování.
- `--input F`, kde „F“ je název zdrojového souboru videa.

- `--output E`, kde „E“ je název výsledného souboru kódovaného videa.

Tady je příklad práce s příkazovým řádkem HandBrake, kde se přepínače příkazového řádku shodují s nastaveními, které jsme použili v grafické verzi HandBrake.

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
                            --width 320
                            --vb 600
                            --two-pass
                            --turbo
                            --input pr6.dv
                            --output pr6.mp4
```

Od shora dolů: tento příkaz spouští HandBrake s předvolbou „iPhone & iPod Touch“, mění rozměry videa na 320×240, nastavuje průměrný bitový tok na 600 kbps, aktivuje dvouprůchodové kódování se zrychleným prvním průchodem, načítá soubor `pr6.dv` a kóduje ho jako `pr6.mp4`. Uf!

5.12 Kódování videa WebM pomocí ffmpeg

WebM má plnou podporu v `ffmpeg` od verze 0.6 výše. V příkazovém řádku spusťte bez parametrů příkaz ffmpeg a zjistěte, zda byl zkompilován s podporou VP8:

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg
developers
   built on May 19 2010 22:32:20 with gcc 4.4.3
   configuration: --enable-gpl --enable-version3 --enable-nonfree
--enable-postproc --enable-pthreads --enable-libfaac --enable
-libfaad --enable-libmp3lame --enable-libopencore-amrnb --enable
-libopencore-amrwb --enable-libtheora --enable-libx264 --enable
-libxvid --enable-x11grab --enable-libvorbis --enable-libvpx
```

Pokud neuvidíte kouzelná slůvka „`--enable-libvorbis`“ a „`--enable-libvpx`“, nemáte správnou verzi `ffmpeg`. (Pokud jste ffmpeg kompilovali sami, zkontrolujte, zda nemáte nainstalované dvě různé verze. Pokud máte, nevadí, neměly by si navzájem překážet. Jenom budete muset pro spuštění verze `ffmpeg` s podporou VP8 zadávat její plnou cestu.)

Provedu dvouprůchodové kódování. V prvním průchodu (pass 1) utilita jenom projede vkládaný videosoubor (`-i pr6.dv`) a zapíše některé statistiky do souboru protokolu (který bude automaticky pojmenován `pr6.dv-0.log`). Určím videokodek pomocí parametru `-vcodec`:

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16
-keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
-vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -an -y NUL
```

Většina příkazového řádku `ffmpeg` nemá nic společného s VP8 ani WebM. `libvpx` podporuje několik možností spojených s VP8, které můžete zadat do `ffmpeg`, ale zatím nevím, jakým způsobem kterákoliv z nich funguje. Jakmile přijdu s vhodným vysvětlením, dám ho sem a zapracuju ho do této knihy, pokud by mohlo být přínosné.

Během druhého průchodu (pass 2) načte `ffmpeg` statistiky uložené při prvním průchodu a začne kódovat video a zvuk. Výsledkem bude soubor s příponou `.webm`.

```
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16
-keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
-vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -acodec libvorbis
-y pr6.webm
```

Důležitých je tady pět parametrů:

- `-vcodec libvpx` určuje, že pro kódování používáme kodek videa VP8. WebM vždy používá video VP8.
- `-b 614400` určuje bitový tok. Na rozdíl od jiných formátů `libvpx` požaduje bitový tok přímo v bitech, a ne v kilobitech. Když chcete video s tokem 600 kbps, vynásobte 600x1024 a dostanete 614400.
- `-s 320x240` určuje cílové rozměry, šířka krát výška.
- `-aspect 4:3` určuje poměr stran videa. Video ve standardním rozlišení má obvykle poměr 4:3, ale většina videí ve vysokém rozlišení má poměr 16:9 nebo 16:10. Při testování jsem zjistil, že musím to výslovně uvést v příkazovém řádku a nespoléhat se na to, že `ffmpeg` detekuje poměr automaticky.
- `-acodec libvorbis` určuje, že pro kódování používáme zvukový kodek Vorbis. WebM vždy používá audio Vorbis.

5.13 A konečně kód

Jsem si jistý, že tohle měla být knížka o HTML. Takže kde je kód?

V HTML5 máte dvě možnosti jak na stránku vložit video. Obě využívají prvek `<video>`. Pokud máte pouze jeden video soubor, můžete na něho jednoduše odkázat v atributu `src`. Velmi se to podobá vkládání obrázku pomocí tagu `<img src="...">`.

Jeden videosoubor↷

```
<video src="pr6.webm"></video>
```

Z technického hlediska je to vše, co potřebujete. Ale stejně jako u tagu `<img>` byste měli v tagu `<video>` vždy používat atributy `width` a `height`. Atributy `width` a `height` se můžou shodovat s maximální šířkou a výškou, které jste zadali při procesu kódování. Nedělejte si starosti, jestli je jeden rozměr videa o něco menší. Váš prohlížeč video zarovná na střed do pole definovaného tagem `<video>`. Nikdy se nestane, že by bylo video příliš sražené nebo roztáhnuté.

```
<video src="pr6.webm" width="320" height="240"></video>
```

Prvek `<video>` nezobrazuje automaticky žádné ovládání přehrávače. Ovládání si můžete vytvořit sami se starým dobrým HTML, CSS a JavaScriptem. Prvek `<video>` má metody `play()` a `pause()` a vlastnost pro čtení a zápis nazvanou `currentTime`. Má také vlastnosti pro čtení a zápis `volume` a `muted`. Takže máte opravdu všechno pro to, abyste si vytvořili své vlastní rozhraní.

Pokud nechcete vytvářet vlastní rozhraní, můžete prohlížeči nařídit, aby zobrazoval zabudovanou sadu ovládání. Stačí, když do tagu `<video>` vložíte atribut `controls`.

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

Existují ještě další dva volitelné atributy, o kterých se chci zmínit, než budu pokračovat: `preload` a `autoplay`. Nemusíte se tak mračit – hned vám vysvětlím, k čemu jsou dobré. Atribut `preload` říká prohlížeči, že budete chtít začít stahovat video soubor hned, jak se stránka načte. To dává smysl v případě, kdy jediným účelem stránky je zobrazení videa. Pokud se ovšem jedná pouze o doplňující materiál, na který se podívá jen pár uživatelů, můžete `preload` nastavit na `none`, a nařídíte tak prohlížeči, aby minimalizoval zatížení sítě.

Tady je příklad videa, které se začne stahovat (ale ne přehrávat) hned, jakmile se stránka načte:

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

A tady je příklad videa, které se po načtení stránky stahovat nezačne:

```
<video src="pr6.webm" width="320" height="240" preload="none">
</video>
```

Atribut `autoplay` dělá přesně to, co byste podle názvu očekávali – říká prohlížeči, že budete chtít začít stahovat video soubor hned, jakmile se stránka načte, a *zároveň* budete chtít video co nejdříve automaticky přehrát. Některým lidem se to líbí, jiní to nenávidějí. Rád bych vám vysvětlil, proč je přítomnost takového atributu v HTML5 důležitá. Někteří lidé budou chtít přehrát videa automaticky, i když to jejich návštěvníky otravuje. Kdyby HTML5 *nedefinovalo* standardní způsob, jak automaticky přehrávat videa, lidé by se uchýlili k javascriptovým hackům. (Například by vyvolávali metodu videa `play()` během události `load` okna.) Ubránit se proti něčemu takovému by pro návštěvníky bylo mnohem těžší. Oproti tomu je jednoduché přidat do prohlížeče rozšíření (nebo si ho napsat, je-li to nutné), které bude říkat „ignoruj atribut `autoplay`, nechci videa přehrávat automaticky."

Tady je příklad videa, které se začne stahovat a přehrávat hned, jakmile se stránka načte:

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

A tady je skript Greasemonkey, který si můžete nainstalovat do vlastní kopie Firefoxu a který zabrání videu v HTML5, aby se přehrávalo automaticky. Používá DOM atribut `autoplay` definovaný HTML5, který je javascriptovou obdobou atributu `autoplay` v kódu HTML. `[disable_video_autoplay.user.js]`

```
// ==UserScript==
// @name         Vypne automatické přehrávání videa
// @namespace    http://diveintomark.org/projects/greasemonkey/
// @description  Zajišťuje, že se prvky HTML5 video nepřehrávají
                automaticky
// @include *
// ==/UserScript==

var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
    var elmVideo = arVideos[i];
    elmVideo.autoplay = false;
}
```

Ale moment… Pokud jste se řídili návodem v této kapitole, tak nemáte jen jeden videosoubor – máte tři. Jedním je soubor `.ogv`, který jste vytvořili pomocí Firefoggu nebo ffmpeg2theora.

Druhým je soubor `.mp4`, který jste vytvořili pomocí HandBrake. Třetím je soubor `.webm`, který jste vytvořili pomocí ffmpeg. HTML5 nabízí způsob, jak všechny tři tyto soubory spojit: prvek `<source>`. Jeden prvek `<video>` může obsahovat více prvků `<source>`. Váš prohlížeč bude po pořádku procházet seznam zdrojů videa a přehraje první, u kterého to dokáže.

Nabízí se další otázka: jak prohlížeč ví, které video dokáže přehrát? V nejhorším případě načte každé z videí a pokusí se ho přehrát. To ovšem znamená plýtvání šířkou pásma. Ušetříte síti hodně zatížení, pokud prohlížečům informace o každém videu předem dodáte. Můžete to udělat použitím atributu `type` v prvku `<source>`.

Takhle to bude vypadat celé:

Tři (!) videosoubory ↷

```
<video width="320" height="240" controls>
    <source src="pr6.mp4" type="video/mp4;
    codecs=avc1.42E01E,mp4a.40.2">
    <source src="pr6.webm" type="video/webm; codecs=vp8,vorbis">
    <source src="pr6.ogv" type="video/ogg; codecs=theora,vorbis">
</video>
```

Rozeberme si to. Prvek `<video>` určuje šířku a výšku videa, ale na videosoubor přímo neodkazuje. Uvnitř prvku `<video>` se nacházejí tři prvky `<source>`. Každý prvek `<source>` odkazuje na jeden soubor videa (pomocí atributu `src`) a také poskytuje informace o formátu videa (pomocí atributu `type`).

Atribut `type` vypadá komplikovaně – no dobře, taky *je* komplikovaný. Kombinuje tři různé informace: kontejnerový formát, videokodek a zvukový kodek. Začněme od konce. U videosouboru `.ogv` je kontejnerovým formátem Ogg, což je zde znázorněno jako `video/ogg`. (Z technického hlediska je to typ MIME pro videosoubory ve formátu Ogg.) Použitým videokodekem je zde Theora a zvukovým kodekem Vorbis. Je to celkem jednoduché, až na formát hodnoty atributu, který je trochu komplikovaný. Hodnota sama o sobě musí obsahovat uvozovky, což znamená, že musíte použít jiný typ uvozovek, do kterých se pak celá hodnota vloží.

```
<source src="pr6.ogv" type="video/ogg; codecs=theora,vorbis">
```

WebM je v podstatě totéž, ale má v parametru `codecs` uvedený jiný typ MIME (`video/webm` místo `video/ogg`) a jiný videokodek (`vp8` místo `theora`).

```
<source src="pr6.webm" type="video/webm; codecs=vp8,vorbis">
```

Video H.264 je ještě složitější. Pamatujete si, jak jsem říkal, že jak video H.264, tak audio AAC mohou mít různé „profily“? Kódovali jsme s profilem H.264 Baseline (základní) a profilem AAC Low-complexity (nízká složitost), a pak jsme to všechno zabalili do kontejneru MPEG-4. Všechny tyto informace jsou obsaženy v atributu `type`.

```
<source src="pr6.mp4" type="video/mp4; codecs=avc1.42E01E,mp4a.40.2">
```

Přínos takové práce spočívá v tom, že prohlížeč nejdřív zkontroluje atribut `type`, aby zjistil, zda může daný videosoubor přehrát. Pokud se prohlížeč rozhodne, že dané video přehrát nemůže, *nebude soubor stahovat*. Ani jeho část. Ušetříte šířku pásma a vaši uživatelé dříve uvidí video, za kterým na stránku přišli.

Pokud se řídíte návodem v této kapitole pro kódování videí, můžete jednoduše zkopírovat a vložit hodnoty atributu `type` z tohoto příkladu. Jinak budete muset nastavit parametry type sami.

PROFESOR ZNAČKA ŘÍKÁ

iPady s operačním systémem iOS 3.x obsahovaly chybu, kvůli které si dokázaly všimnout pouze zdroje videa uvedeného jako první v pořadí. iOS 4 (bezplatná aktualizace pro všechny iPady) tuto chybu vyřešil. Pokud chcete, aby se video zobrazovalo i uživatelům iPadů, kteří ještě neaktualizovali na iOS 4, budete muset jako první uvést soubor MP4, a až pak ostatní formáty videa. *Ach jo*.

5.14 Typy MIME pozvedají hlavu

Skládanka jménem video má tolik kousků, že se mi s tím ani nechce začínat. Ale je to důležité, protože špatně nastavený webový server může vést k tomu, že budete rudí vzteky, až se budete snažit přijít na to, proč vám videa hrají, dokud jsou v počítači, ale odmítají se přehrát, když je nahrajete na stránky. Pokud se setkáte s tímto problémem, jeho příčinou jsou pravděpodobně typy MIME.

Typy MIME jsem zmínil v kapitole o historii, ale nejspíš jste jimi jen tak prolistovali, aniž byste ocenili jejich význam. Tak tady to máte velkými písmeny:

PROFESOR ZNAČKA KŘIČÍ

VIDEOSOUBORY MUSEJÍ BÝT NABÍZENY SE SPRÁVNÝM TYPEM MIME!

Co je to správný typ MIME? Už jste ho viděli: je to část hodnoty atributu `type` v prvku `<source>`. Ale jenom nastavit atribut `type` v kódu HTML nestačí. Také musíte zajistit, aby váš webový server uvedl správný typ MIME v HTTP hlavičce `Content-Type`.

Pokud používáte webový server Apache, nebo něco, co je z Apache odvozené, můžete použít direktivu AddType v souboru `httpd.conf nebo .htaccess` ve složce, kam ukládáte video soubory. (Pokud používáte jiný webový server, podívejte se do jeho dokumentace na nastavení HTTP hlavičky `Content-Type` pro konkrétní typy souborů.)

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

První řádek je pro videa v kontejneru Ogg. Druhý řádek je pro videa v kontejneru MPEG-4. Třetí je pro WebM. Nastavte to jednou a můžete na to zapomenout. Ale když to zapomenete nastavit, vaše videa v některých prohlížečích nepůjdou přehrát, a to i přesto, že jste v HTML kódu v atributu `type` uvedli typ MIME.

Pokud se chcete dozvědět ještě více šťavnatých detailů o konfiguraci webového serveru, doporučuju vám tento vynikající článek na Mozilla Developer Center: Configuring servers for Ogg media[6]. (Rady v tomto článku se vztahují i na videa v MP4 a WebM.)

5.15 A co IE?

Internet Explorer 9 podporuje prvek `<video>` v HTML5, ale Microsoft veřejně slíbil, že konečná verze IE 9 bude podporovat i video H.264 a audio AAC v kontejneru MPEG-4, stejně jako Safari a iPhone.

Ale co starší verze Internet Exploreru? Však víte, všechny ty verze vydané před IE 9, včetně IE 8? Většina lidí používajících Internet Explorer má nainstalován také zásuvný modul Adobe Flash. Moderní verze Adobe Flash (od verze 9.0.60.184 výše) podporují video H.264 a audio AAC v kontejneru MPEG-4, stejně jako Safari a iPhone. Když zakódujete video H.264 pro Safari, můžete ho přehrát ve videopřehrávači založeném na Flash v případě, že přijdete na to, že prohlížeč některého z vašich návštěvníků nepodporuje HTML5.

FlowPlayer je open source video přehrávač založený na Flash a licencovaný GPL. (K dispozici jsou i komerční licence[7].) FlowPlayer o prvku `<video>` neví vůbec nic. Nedokáže kouzlem proměnit tag `<video>` v objekt Flash. Ale HTML5 je připraven si s tím poradit, protože mů-

6 https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media
7 https://flowplayer.org/pricing/

žete vnořit prvek <object> do prvku <video>. Prohlížeče, které nepodporují HTML5, budou prvek <video> ignorovat a přejdou místo toho k vnořenému <object>, což spustí zásuvný modul Flash a přehraje klip v FlowPlayeru. Prohlížeče, které podporují video HTML5, najdou zdroj videa, který umějí přehrát, a přehrají ho, *a budou úplně ignorovat vnořený prvek* <object>.

Ten poslední fakt je klíčem k celé skládance: HTML5 specifikuje, že všechny prvky (kromě prvků <source>), které jsou potomky prvku <video>, se musí úplně ignorovat. To vám umožní používat video HTML5 v novějších prohlížečích a elegantně se vracet k Flash v prohlížečích starších, bez nějakých zbytečných javascriptových hacků. Více o této technice se můžete dočíst zde: Video For Everybody[8].

5.16 Problémy na iPhonech a iPadech

iOS je operační systém Applu, na kterém běží iPhone, iPod Touch a iPad. iOS 3.2 má s videem HTML5 několik problémů.

- iOS nerozpozná video, pokud použijete atribut poster. Atribut poster prvku <video> vám umožní zobrazit vámi zvolený obrázek, než se video načte, nebo dokud uživatel nezmáčkne „přehrát“. iOS 4.0 tento problém řeší, ale bude nějakou dobu trvat, než uživatelé aktualizují na tuto verzi.

- Pokud používáte několik prvků <source>, iOS rozpozná jenom první z nich. Jelikož zařízení s iOS podporují pouze H.264+AAC+MP4, znamená to pro vás, že musíte vždy na prvním místě uvést soubor MP4. I tento problém je v iOS 4.0 vyřešen.

5.17 Problémy na zařízeních s Androidem

Android je operační systém Googlu, na kterém běží celá řada různých mobilních telefonů a přenosných zařízení. Verze Androidu starší než 2.3 měly s videem HTML5 několik problémů.

- Atribut type v prvcích <source> Android naprosto mátl. Ironií je, že jediným způsobem, jak je možné přimět Android rozpoznat zdroj videa, je atribut type úplně vypustit a zajistit, aby název vašeho video souboru H.264+AAC+MP4 končil příponou .mp4. Atribut type můžete použít u jiných zdrojů videa, protože H.264 je jediným formátem videa, který Android 2.2 podporuje. (Tento problém je vyřešen v Androidu 2.3.)

8 http://camendesign.com/code/video_for_everybody

- Atribut `controls` nebyl podporován. Jeho použití nevedlo k žádným nepříjemnostem, ale Android nezobrazoval žádné uživatelské rozhraní pro ovládání videa. Budete muset poskytnout své vlastní uživatelské rozhraní pro ovládání videa. Minimálně byste měli poskytnout skript, který začne video přehrávat, když na něj uživatel klikne. I tohle je v Androidu 2.3 vyřešeno.

5.18 Úplný příklad ze života

Tady je příklad videa, které používá tyto techniky. Rozšířil jsem kód z „Video for Everybody" tak, aby zahrnoval video ve formátu WebM. Zakódoval jsem jedno zdrojové video do tří formátů pomocí těchto příkazů:

```
## Theora/Vorbis/Ogg
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240
--output pr6.ogv pr6.dv

## H.264/AAC/MP4
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch" --vb 200
--width 320 --two-pass --turbo --optimize --input pr6.dv --output
pr6.mp4

## VP8/Vorbis/WebM
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16
-keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
-vcodec libvpx -b 204800 -s 320x240 -aspect 4:3 -an -f webm -y NUL
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16
-keyint_min 0 -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
-vcodec libvpx -b 204800 -s 320x240 -aspect 4:3 -acodec libvorbis
-ac 2 -y pr6.webm
```

Konečný kód používá prvek `<video>` pro video HTML5, vnořený prvek `<object>` pro náhradní řešení využívající Flash a malý skript pro zařízení s Androidem:

```
<video id="movie" width="320" height="240" preload controls>
    <source src="pr6.webm" type="video/webm; codecs=vp8,vorbis" />
    <source src="pr6.ogv" type="video/ogg; codecs=theora,vorbis" />
    <source src="pr6.mp4" />
    <object width="320" height="240" type="application/
    x-shockwave-flash"
```

```
        data="flowplayer-3.2.1.swf">
        <param name="movie" value="flowplayer-3.2.1.swf" />
        <param name="allowfullscreen" value="true" />
        <param name="flashvars" value="config={'clip': {'url':
        'http://wearehugh.com/dih5/pr6.mp4', 'autoPlay':false,
        'autoBuffering':true}}" />
        <p>Download video as <a href="pr6.mp4">MP4</a>, <a href="pr6.
        webm">WebM</a>, or <a href="pr6.ogv">Ogg</a>.</p>
        </object>
    </video>
    <script>
        var v = document.getElementById("movie");
        v.onclick = function() {
            if (v.paused) {
                v.play();
            } else {
                v.pause();
            }
        };
    </script>
```

Pomocí kombinace HTML5 a Flash byste měli být schopni se podívat na toto video téměř ve všech prohlížečích a téměř na všech zařízeních:

5.19 K dalšímu čtení

- HTML5: The <video> element[9]
- Video for Everybody[10]
- A gentle introduction to video encoding[11]
- Theora 1.1 is released — what you need to know[12]
- Configuring servers for Ogg media[13]
- Encoding with the x264 codec[14]
- Video type parameters[15]

9 https://html.spec.whatwg.org/multipage/embedded-content.html#video
10 http://camendesign.com/code/video_for_everybody
11 http://web.archive.org/web/20100527215322/http:/diveintomark.org/tag/give
12 https://hacks.mozilla.org/2009/09/theora-1-1-released/
13 https://developer.mozilla.org/en-US/docs/Web/HTTP/Configuring_servers_for_Ogg_media
14 http://www.mplayerhq.hu/DOCS/HTML/en/menc-feat-x264.html
15 https://wiki.whatwg.org/wiki/Video_type_parameters

- Everything you need to know about HTML5 audio and video[16]
- Making HTML5 video work on Android phones[17]. Ach jo.
- Internet Explorer 9 Guide for Developers: HTML5 video and audio elements[18]

Hotová rozhraní pro ovládání videa HTML5:

- VideoJS[19]
- MediaElement.js[20]
- Kaltura HTML5 Video & Media JavaScript Library[21]

16 https://dev.opera.com/articles/everything-you-need-to-know-html5-video-audio/
17 http://www.broken-links.com/2010/07/08/making-html5-video-work-on-android-phones/
18 https://msdn.microsoft.com/en-us/ie/ff468705.aspx#_HTML5_video_audio
19 http://www.videojs.com/
20 http://mediaelementjs.com/
21 http://html5video.org/wiki/HTML5_Video_Wiki

6. Jste tady (a všichni ostatní taky)

6. Jste tady (a všichni ostatní taky) – 161

6.1 Začínáme

Geolokace je umění zjistit, kde ve světě jste, a (případně) sdílet tuto informaci s lidmi, kterým důvěřujete. Zjistit, kde právě jste, jde více způsoby – z vaší IP adresy, z vašeho bezdrátového připojení, z toho, s jakým vysílačem komunikuje váš mobil, anebo pomocí speciálních zařízení GPS, jež vypočítávají zeměpisnou šířku a délku z informací, které jim posílají satelity.

PTEJTE SE PROFESORA ZNAČKY

Q: Geolokace zní děsivě. Můžu ji vypnout?

A: Je pochopitelné, že když mluvíme o sdílení fyzické polohy se vzdáleným webovým serverem, dělá nám starosti ochrana soukromí. Geolokační API výslovně říká: „Uživatelské agenty nesmějí odesílat webovým stránkám informace o poloze bez jednoznačného svolení uživatele." Jinými slovy: sdílení polohy je vždy volitelné. Pokud to dělat nechcete, nemusíte.

6.2 Geolokační API

Geolokační API umožňuje sdílet vaši polohu s webovými stránkami, kterým důvěřujete. Údaje o zeměpisné délce a šířce si přečte JavaScript na stránce, a ten je pak může poslat zpátky vzdálenému webovému serveru a dělat s nimi zábavné kousky, jako vyhledávat podniky v okolí nebo ukázat vaši polohu na mapě.

Jak můžete vidět z následujícího přehledu, geolokační API podporuje většina prohlížečů na stolních počítačích i mobilních zařízeních. Navíc mohou některé starší prohlížeče a zařízení získat podporu pomocí knihoven obalujících volání, jak uvidíme dále v této kapitole.

Podpora geolokačního API						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
9.0+	3.5+	5.0+	5.0+	10.6+	3.0+	2.0+

Kromě podpory standardního geolokačního API existuje celá řada API určených pro konkrétní zařízení na jiných mobilních platformách. O tom všem budu mluvit později v této kapitole.

6.3 Ukažte mi kód

Geolokační API se soustřeďuje kolem nové vlastnosti na globálním objektu `navigator`: `navigator.geolocation`.

Nejjednodušší použití geolokačního API vypadá takto:

```
function get_location() {
    navigator.geolocation.getCurrentPosition(show_map);
}
```

Toto použití neobsahuje detekci, odstraňování chyb ani žádné možnosti. Vaše webová aplikace by však aspoň první dvě z těchto tří věcí měla dělat. Pro detekci podpory geolokačního API můžete použít Modernizr:

```
function get_location() {
    if (Modernizr.geolocation) {
        navigator.geolocation.getCurrentPosition(show_map);
    } else {
        // geolokace není nativně podporována; co zkusit náhradní
        řešení?
    }
}
```

Je na vás, co budete dělat bez podpory geolokace. Za chvilku vám vysvětlím, jak funguje náhradní řešení v JavaScriptu, ale nejdřív chci říct něco o tom, co se děje během zavolání `getCurrentPosition()`. Jak už jsem zmínil na začátku této kapitoly, podpora geolokace je *volitelná*. To znamená, že vás prohlížeč nikdy nebude nutit k tomu, abyste vzdálenému serveru prozradili svou fyzickou polohu. V každém prohlížeči přitom uvidíte něco trochu jiného. Například v Mozille Firefoxu vyvolání funkce geolokačního API `getCurrentPosition()` způsobí, že v okně prohlížeče nahoře vyskočí informační panel. Informační panel vypadá takto:

Děje se tu hodně věcí. Vy, jako koncový uživatel,

- se dozvídáte, že chce webová stránka znát vaši polohu,
- se dozvídáte, *která* webová stránka chce znát vaši polohu,

- se můžete proklikat na stránku s nápovědou Mozilly „Prohlížení se znalostí polohy“, která vám vysvětlí, co se to vlastně děje (ve stručnosti: Google poskytuje polohu a ukládá vaše údaje v souladu se svými Zásadami ochrany soukromí u služeb určování polohy),
- se můžete rozhodnout sdílet svou polohu,
- se můžete rozhodnout svou polohu nesdílet,
- můžete sdělit prohlížeči, aby si zapamatoval vaši volbu (sdílet, nebo nesdílet), takže na této webové stránce už informační panel neuvidíte.

Navíc je informační panel

- nemodální, takže vám nezabrání přepnout do jiného okna nebo panelu prohlížeče,
- vázaný na konkrétní panel, takže zmizí, pokud přepnete do jiného okna nebo panelu prohlížeče, a znovu se objeví, když přepnete zpátky do původního panelu,
- bezpodmínečný, takže ho webová stránka nemůže žádným způsobem obejít,
- blokující, takže webová stránka nemá šanci žádným způsobem určit vaši polohu, zatímco čeká na vaši odpověď.

Právě jste viděli javascriptový kód, který způsobuje, že se objeví informační panel. Je to jednoduché vyvolání funkce, které využívá zpětně volanou funkci (kterou jsem pojmenoval `show_map`). Vyvolání `getCurrentPosition()` se vrátí okamžitě, ale to ještě neznamená, že máte přístup k poloze uživatele. Získání informací o poloze máte zaručeno, až když se octnete ve zpětně volané funkci. Zpětně volaná funkce vypadá takto:

```
function show_map(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    // zobrazme mapu nebo udělejme něco zajímavého!
}
```

Zpětně volaná funkce bude vyvolána s jediným parametrem – objektem s dvěma vlastnostmi: `coords` a `timestamp`. Timestamp – časové razítko – je datum a čas, kdy byla vypočtena vaše poloha. (Jelikož toto všechno probíhá asynchronně, nemůžete dopředu vědět, kdy to proběhne. Uplyne nějaká doba, než si uživatel prohlédne informační panel a odsouhlasí sdílení své polohy, než se zařízení s čipy GPS připojí k satelitu a tak dále.) Objekt `coords` má vlastnosti jako `latitude` a `longitude` – zeměpisná šířka a délka, které udávají fyzickou polohu uživatele na planetě.

Objekt `position`		
Vlastnost	**Typ**	**Poznámka**
`coords.latitude`	double	v desetinných stupních
`coords.longitude`	double	v desetinných stupních
`coords.altitude`	double nebo `null`	v metrech nad referenčním elipsoidem
`coords.accuracy`	double	v metrech
`coords.altitudeAccuracy`	double nebo `null`	v metrech
`coords.heading`	double nebo `null`	ve stupních ve směru hodinových ručiček od skutečného severu
`coords.speed`	double nebo `null`	v metrech za sekundu
`timestamp`	DOMTimeStamp	jako objekt `Date()`

Zaručeně se vrátí pouze tři z těchto vlastností (`coords.latitude`, `coords.longitude`, a `coords.accuracy`). Zbytek může mít hodnotu `null`, v závislosti na schopnostech vašeho zařízení a geolokačního serveru, s nímž komunikuje. Vlastnosti `heading` a `speed` (směr pohybu a rychlost) jsou dle možností vypočítány na základě předchozí polohy uživatele.

6.4 Ošetřování chyb

Geolokace je komplikovaná. Dost se toho může pokazit. Problém nutnosti získat souhlas uživatele jsem už zmínil. Pokud vaše webová aplikace požaduje uživatelovou polohu, ale uživatel vám ji nechce dát, jste nahraní. Uživatel vždycky vyhrává. Ale jak to vypadá v kódu? Vypadá to jako druhý argument ve funkci `getCurrentPosition()`: zpětně volaná funkce pro ošetřování chyb.

```
navigator.geolocation.getCurrentPosition(
    show_map, handle_error)
```

Když se něco pokazí, vaše zpětně volaná funkce pro ošetřování chyb bude zavolána pomocí objektu `PositionError`.

Objekt `PositionError`		
Vlastnost	**Typ**	**Poznámky**
`code`	short	výčtová hodnota
`message`	DOMString	není určeno koncovým uživatelům

Vlastnost `code` bude mít jednu z následujících hodnot:

- `PERMISSION_DENIED (1)`, jestliže uživatel klikne na tlačítko „Nesdílet“ nebo vám jiným způsobem odepře přístup k své poloze.
- `POSITION_UNAVAILABLE (2)`, jestliže není k dispozici připojení k síti nebo není možné kontaktovat satelity zjišťující polohu.
- `TIMEOUT (3)`, jestliže je k dispozici připojení k síti, ale vypočtení polohy uživatele trvá příliš dlouho. Co to znamená, „příliš dlouho“? Jak to určit, vám ukážu v dalším oddílu.

  Prohrávejte se ctí ↷

```
function handle_error(err) {
    if (err.code == 1) {
        // uživatel řekl ne!
    }
}
```

PTEJTE SE PROFESORA ZNAČKY

Q: Funguje geolokační API na Mezinárodní vesmírné stanici, na Měsíci nebo na jiných planetách?

A: Specifikace geolokace říká: „Referenční systém zeměpisných souřadnic, použitý v atributech tohoto prostředí, je Světový geodetický systém (2d) [WGS84]. Žádný jiný referenční systém není podporován.“ Mezinárodní vesmírná stanice obíhá Zemi, takže astronauti na stanici mohou popsat svou polohu pomocí zeměpisné šířky a délky a nadmořské výšky. Světový geodetický systém je ale geocentrický, takže nemůže být použit pro popis polohy na Měsíci nebo jiných planetách.

6.5 Výběr! Dejte mi na výběr!

Některá oblíbená mobilní zařízení – třeba iPhony a mobilní telefony s Androidem – podporují dvě metody zjišťování polohy. První metoda vyměřuje pozici na základě relativní vzdálenosti od různých vysílačů, které používá mobilní operátor. Tato metoda je rychlá a nevyžaduje žádný speciální hardware pro GPS, ale dává pouze hrubou představu o tom, kde jste. V závislosti na počtu vysílačů v oblasti se může tato hrubá představa rovnat pouhému jednomu bloku domů, ale také čtverci o straně jeden kilometr.

Druhá metoda využívá v zařízení speciální hardware pro GPS, který komunikuje se speciálními satelity pro zjišťování polohy obíhajícími Zemi. GPS obvykle dokáže určit polohu s přesností několika metrů. Nevýhodou je, že speciální čip GPS v zařízení potřebuje hodně energie, takže mobily a další univerzální mobilní zařízení čip obvykle vypínají, když ho zrovna nepotřebují. To znamená, že po spuštění musíte čekat, než čip naváže spojení se satelity GPS. Pokud jste někdy použili Google Mapy na iPhonu nebo jiném chytrém telefonu, viděli jste obě metody v akci. Nejdřív uvidíte velký kruh označující vaši přibližnou polohu (po nalezení nejbližšího vysílače), pak menší kruh (po vyměření s pomocí dalších vysílačů) a nakonec malou tečku udávající přesnou polohu (poskytnutou satelity GPS).

Důvodem pro používání obou metod je, že u některých webových aplikací nepotřebujete vysokou úroveň přesnosti. Pokud hledáte, co se hraje v okolních kinech, přibližná poloha vám nejspíš úplně stačí. Kin není zase tolik, ani v hustě obydlených městech, a stejně se asi budete chtít podívat na program více kin najednou. Na druhou stranu pokud v reálném čase někomu přesně popisujete cestu, opravdu potřebujete vědět, kde přesně uživatel je, abyste mu mohli říct „Za dvacet metrů odbočte doprava“ a podobně.

Funkce `getCurrentPosition()` má volitelný třetí argument, objekt `PositionOptions`. V objektu `PositionOptions` můžete nastavit tři vlastnosti. Všechny vlastnosti jsou volitelné. Je na vás, jestli nastavíte všechny, jen některé, nebo nenastavíte žádnou.

Objekt `PositionOptions`			
Vlastnost	**Typ**	**Výchozí**	**Poznámky**
`enableHighAccuracy`	booleovský	false	true může být pomalejší
`timeout`	long	(není)	v milisekundách
`maximumAge`	long	0	v milisekundách

Vlastnost `enableHighAccuracy` dělá přesně to, co byste podle názvu očekávali – umožňuje vysokou míru přesnosti určení polohy. Pokud je její hodnota `true` a zařízení je schopné ji podporovat, a zároveň uživatel souhlasí se sdílením své přesné polohy, zařízení se pokusí poskytnout ji. iPhony i mobily s Androidem vyžadují povolení pro určování polohy s nízkou a vysokou mírou přesnosti zvlášť, takže je možné, že volání funkce `getCurrentPosition()` s `enableHighAccuracy:true` selže, ale s `enableHighAccuracy:false` se podaří.

Vlastnost `timeout` určuje počet milisekund, které je webová aplikace ochotna čekat na určení polohy. Časovač se spustí až poté, co uživatel dá povolení k tomu, aby se zařízení vůbec pokusilo vypočítat jeho polohu. Čas neběží pro uživatele, ale pro síť.

Vlastnost `maximumAge` umožňuje zařízení odpovědět okamžitě a vrátit dříve uloženou polohu. Například si představme, že poprvé zavoláte `getCurrentPosition()`, uživatel souhlasí a funkce zpětně volaná při úspěchu je zavolána s polohou, která byla vypočtena přesně v 10:00 ráno. O minutu později, v 10:01 ráno, zavoláte `getCurrentPosition()` znovu s vlastností `maximumAge` nastavenou na `75000`.

```
navigator.geolocation.getCurrentPosition(
    success_callback, error_callback, {maximumAge: 75000});
```

Říkáte tím, že nemusíte nezbytně znát uživatelovu *aktuální* polohu. Bude vám stačit vědět, kde se uživatel nacházel před 75 sekundami (75 000 milisekundami). Zařízení ví, kde byl uživatel před 60 sekundami (60 000 milisekundami), protože vypočítalo jeho polohu ve chvíli, kdy jste poprvé zavolali `getCurrentPosition()`. Takže se neobtěžuje s tím, aby přepočítávalo, kde je uživatel teď. Prostě vrátí úplně stejnou informaci jako poprvé: stejnou zeměpisnou šířku a délku a stejné časové razítko (10:00 ráno).

Než se budete ptát na uživatelovu polohu, měli byste si rozmyslet, jak přesný údaj potřebujete, a podle toho nastavit `enableHighAccuracy`. Pokud potřebujete zjistit uživatelovu polohu víckrát než jednou, rozmyslete si, jak starou informaci ještě můžete použít, a podle toho nastavte `maximumAge`. Pokud potřebujete uživatelovu polohu zjišťovat *nepřetržitě*, funkce `getCurrentPosition()` se pro vás nehodí. Budete muset aktualizovat na `watchPosition()`.

Funkce `watchPosition()` má stejnou strukturu jako `getCurrentPosition()`. Používá dvě zpětně volané funkce – povinnou pro úspěch a volitelnou pro chybu – a může přibrat i volitelný objekt `PositionOptions`, který má ty vlastnosti, o kterých jste se právě dozvěděli. Rozdíl spočívá v tom, že se zpětně volaná funkce zavolá *pokaždé, když se uživatelova poloha změní*. Nemusíte jeho polohu aktivně zjišťovat. Zařízení určí optimální interval zjišťování a zavolá zpětně volanou funkci pokaždé, když zjistí, že se uživatelova poloha změnila. Můžete toho využít k aktualizaci viditelné značky na mapě, poskytnutí instrukcí, kam pokračovat dál, nebo k čemukoliv jinému chcete. Je to úplně na vás.

Funkce `watchPosition()` sama o sobě vrací číslo. To číslo byste si asi měli někam uložit. Pokud někdy budete chtít přestat sledovat změny uživatelovy polohy, můžete zavolat metodu `clearWatch()` a předat jí toto číslo, a zařízení pak přestane volat zpětně volanou funkci. Pokud jste někdy v JavaScriptu používali funkce `setInterval()` a `clearInterval()`, tohle funguje stejně.

6.6 A co IE?

Před verzí 9 (technicky 9.0RC1) Internet Explorer geolokační API W3C, které jsem právě popsal, nepodporoval. Ale nezoufejte! U starších verzí budete potřebovat náhradní javascriptové řešení. Není to totéž, jako geolokační API W3C, ale svůj účel to plní.

Když už jsme u těch starších platforem, měl bych zmínit, že hodně starších mobilních platforem má své vlastní geolokační API přizpůsobené konkrétním zařízením. BlackBerry, Nokia, Palm, a OMTP BONDI mají všechny své vlastní geolokační API. Samozřejmě každé z nich funguje jinak, a všechny zase fungují jinak než geolokační API W3C. Jééé!

6.7 Na pomoc přichází geoPosition.js

`geoPositon.js` je javascriptová knihovna licencovaná MIT, která zahlazuje rozdíly mezi geolokačním API W3C, geolokačními službami IP a API poskytovanými mobilními platformami. Abyste ji mohli použít, potřebujete dolů na stránku umístit jeden prvek `<script>`. (Technicky vzato ho můžete umístit kamkoliv, ale skripty v `<head>` způsobují, že se stránka načítá pomaleji. Tak je tam radši nedávejte.)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Ponořme se do HTML5</title>
</head>
<body>
    ...
    <script src="geoPosition.js"></script>    ← Do <head> to radši nedávejte
</body>
</html>
```

Teď jste připraveni používat jakékoliv nainstalované geolokační API.

```
if (geoPosition.init()) {
    geoPosition.getCurrentPosition(geoSuccess, geoError);
}
```

Podívejme se na to krok po kroku. Nejprve musíte explicitně zavolat funkci `init()`. Funkce `init()` vrací hodnotu `true`, jestliže je k dispozici podporované geolokační API.

```
if (geoPosition.init()) {
```

Vyvolání funkce `init()` samo o sobě ve skutečnosti nezjistí polohu. Pouze to potvrdí, že najít vaši polohu je možné. Pro skutečné vyhledání polohy musíte vyvolat funkci `getCurrentPosition()`.

```
geoPosition.getCurrentPosition(geoSuccess, geoError);
```

Po zavolání funkce `getCurrentPosition()` vás prohlížeč požádá o svolení najít a sdílet vaši polohu. Pokud má prohlížeč nativní podporu geolokačního API, zobrazí na vrchu stránky informační panel, který se zeptá, zda chcete dané webové stránce sdělit svou polohu.

Funkce `getCurrentPosition()` přijímá jako argumenty dvě zpětně volané funkce. Pokud funkce `getCurrentPosition()` uspěla při vyhledání vaší polohy – což znamená, že jste dali svolení a geolokační API se mohlo pustit do díla –, vyvolá funkci předanou v prvním argumentu. V tomto případě je funkce zpětného volání při úspěchu nazvaná `geoSuccess`.

```
geoPosition.getCurrentPosition(geoSuccess, geoError);
```

Zpětně volaná funkce při úspěchu má jeden argument, který obsahuje informace o poloze.

Zpětně volaná funkce pro úspěch ↷

```
function geoSuccess(p) {
    alert("Našli jsme vás v šířce " + p.coords.latitude +
    "' délce " + p.coords.longitude);
}
```

Pokud funkce `getCurrentPosition()` nedokáže zjistit vaši polohu – ať už proto, že jste odmítli dát svolení, nebo geolokační API z nějakého důvodu selhalo –, zavolá funkci předanou jako druhý argument. V tomto příkladu se zpětně volaná funkce při selhání jmenuje `geoError`.

```
geoPosition.getCurrentPosition(geoSuccess, geoError);
```

Zpětně volaná funkce při selhání nemá žádné argumenty.

zpětně volaná funkce při selhání ↷

```
function geoError() {
    alert("Nemohli jsme vás najít!");
}
```

`geoPosition.js` v současnosti nepodporuje funkci `watchPosition()`. Pokud potřebujete nepřetržité udávání polohy, musíte ji sami aktivně zjišťovat pomocí `getCurrentPosition()`.

6.8 Úplný případ ze života

Jak to funguje? Podívejme se na to. Při načtení stránka vyvolá `geoPosition.init()`, aby zjistila, jestli je k dispozici geolokace prostřednictvím některého z rozhraní, které podporuje `geoPosition.js`. Pokud ano, vytvoří odkaz, na který můžete kliknout pro zjištění své polohy. Kliknutí na odkaz vyvolá funkci `lookup_location()` ukázanou zde:

```
function lookup_location() {
    geoPosition.getCurrentPosition(show_map, show_map_error);
}
```

Pokud dáte svolení k vypátrání své polohy a zároveň toho bude služba skutečně schopná, vyvolá `geoPosition.js` první zpětně volanou funkci, `show_map()`, s jediným argumentem `loc`. Objekt má vlastnost `coords`, která obsahuje zeměpisnou šířku, délku a údaj o přesnosti. (Tento příklad údaj o přesnosti nevyužívá.) Zbytek funkce `show_map()` používá API Google Map pro vytvoření vložené mapy.

```
function show_map(loc) {
    $("#geo-wrapper").css({'width':'320px','height':'350px'});
    var map = new GMap2(document.getElementById("geo-wrapper"));
    var center = new GLatLng(loc.coords.latitude,
    loc.coords.longitude);
    map.setCenter(center, 14);
    map.addControl(new GSmallMapControl());
    map.addControl(new GMapTypeControl());
    map.addOverlay(new GMarker(center, {draggable: false, title:
    "Jste tady (více méně)"}));
}
```

Pokud `geoPosition.js` nedokáže určit vaši polohu, vyvolá druhou zpětně volanou funkci, `show_map_error()`.

```
function show_map_error() {
    $("#live-geolocation").html('Nedokázali jsme zjistit vaši
    polohu.');
}
```

6.9 K dalšímu čtení

- W3C geolocation API[1]
- BlackBerry geolocation API[2]
- Nokia geolocation API[3]
- Palm geolocation API[4]
- OMTP BONDI geolocation API[5]
- geoPosition.js, obalovací skript geolokačního API[6]
- Internet Explorer 9 Guide for Developers: Geolocation[7]

1 http://www.w3.org/TR/geolocation-API/
2 http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx
3 http://www.developer.nokia.com/Resources/Library/Web/
4 http://developer.palm.com/index.php?option=com_content&view=article&id=1673
5 http://bondi.omtp.org/1.0/apis/geolocation.html
6 https://github.com/estebanav/javascript-mobile-desktop-geolocation
7 https://msdn.microsoft.com/en-us/ie/ff468705.aspx

— 6. Jste tady (a všichni ostatní taky)

7. Minulost, současnost a budoucnost místního úložiště pro webové aplikace

7. Minulost, současnost a budoucnost místního úložiště pro webové aplikace – 175

7.1 Začínáme

Trvalé místní úložiště je jednou z oblastí, v níž měly nativní klientské aplikace výhodu oproti aplikacím webovým. Operační systém obvykle poskytuje nativním aplikacím abstrakční vrstvu pro ukládání a načítání speciálních dat aplikace, jako jsou nastavení nebo údaje o stavu běhu aplikace. Na různých platformách se tyto hodnoty ukládají do registru, do souborů INI nebo XML nebo někam jinam. Pokud nativní klientská aplikace vyžaduje místní úložiště k ukládání i jiných věcí než dvojic klíč-hodnota, můžete vložit vlastní databázi, přijít s vlastním formátem souborů nebo to vyřešit jedním z mnoha dalších způsobů.

Webové aplikace dříve nic z toho přepychu nemívaly. Soubory cookies byly vynalezeny na počátku epochy webu a samozřejmě se dají použít pro trvalé ukládání nevelkého množství dat lokálně. Mají však tři nevýhody, které mohou rozhodnout v jejich neprospěch:

- Cookies se přenášejí s každým požadavkem protokolu HTTP, a zbytečným opakovaným zasíláním stejných údajů zpomalují webovou aplikaci.
- Cookies se přenášejí s každým požadavkem protokolu HTTP, což znamená, že údaje nejsou šifrované (pokud celá webová aplikace neběží přes SSL).
- Velikost cookies je omezena na cca 4 KB, což stačí ke zpomalení aplikace (viz výše), ale nestačí k uspokojení všech potřeb.

Ve skutečnosti však potřebujeme:

- velké úložiště
- na straně klienta,
- které přežije obnovení stránky
- a nebude se posílat serveru.

Před HTML5 byly všechny pokusy o dosažení těchto cílů v různých ohledech neuspokojivé.

7.2 Stručná historie hacků pro místní úložiště před HTML5

Zpočátku byl pouze Internet Explorer – nebo aspoň Microsoft chtěl, aby si to svět myslel. Proto Microsoft, v rámci prvních velkých válek prohlížečů, vynalezl spoustu věcí a vložil je do svého prohlížeče, který měl všechny prohlížečové války ukončit – do Internet Exploreru. Jedna

z těchto inovací se jmenovala DHTML Behaviors (vzorce chování DHTML) a jeden z těchto vzorců se nazýval userData.

`userData` umožňuje webovým stránkám, aby ukládaly až 64 KB dat na doménu, a to v hierarchické struktuře založené na XML. (Důvěryhodné domény, jako jsou stránky v intranetu, můžou ukládat až desetinásobné množství. A 640 KB snad musí každému stačit.) IE nezobrazuje žádné dialogové okno pro svolení a neexistuje žádná možnost, jak zvýšit velikost úložiště.

V roce 2002 společnost Adobe přišla s novou funkcí ve Flash 6, která získala nešťastný a zavádějící název „Flash cookies". V rámci flashového prostředí se tato funkce správně nazývá „místní sdílené objekty". Stručně řečeno tato funkce umožňuje flashovým objektům, aby ukládaly až 100 KB dat na doménu. Brad Neuberg vyvinul raný prototyp Flash-to-Javascript můstku nazvaný AMASS (AJAX Massive Storage System), avšak omezovaly ho některé pro Flash specifické libůstky. V roce 2006 s příchodem ExternalInterface v Flash 8 se přístup k místním sdíleným objektům z Javascriptu podstatně usnadnil a zrychlil. Brad přepsal AMASS a integroval ho do oblíbené sady nástrojů Dojo pod názvem `dojox.storage`. Flash dává každé doméně 100KB úložiště „zadarmo". Kromě toho nabízí uživateli zvětšení prostoru pro ukládání dat o další řády (na 1 MB, 10 MB a tak dále).

V roce 2007 Google spustil Gears, open-source zásuvný modul prohlížeče, jehož cílem bylo poskytnout v prohlížečích ukládací kapacitu navíc. Gears poskytoval API k zabudované databázi SQL založené na SQLite. V roce 2010 Google přesunul své úsilí k tomu, aby zavedl všechny schopnosti Gears do webových standardů jako HTML5, což nakonec vedlo k ukončení Google Gears.

Mezitím Brad Neuberg a další pokračovali v úpravách `dojox.storage`, aby vytvořili jednotné prostředí pro všechny ty různé zásuvné moduly a API. V roce 2009 dokázal `dojox.storage` automaticky detekovat (a navíc poskytnout jednotné prostředí pro) Adobe Flash, Gears, Adobe AIR a raný prototyp úložiště HTML5, které bylo realizováno pouze ve starších verzích Firefoxu.

Když se na ta řešení podíváte, zjistíte následující: všechna z nich jsou buď specifická pro jeden konkrétní prohlížeč, nebo závislá na zásuvném modulu třetí strany. Přes hrdinskou snahu ty rozdíly nějak zahladit (v `dojox.storage`) všechny zveřejňují zásadně odlišná rozhraní, mají různá omezení ukládání a uživatel s nimi bude muset pracovat různými způsoby. A to je právě problém, který se HTML5 rozhodlo vyřešit: poskytnout standardizované API, realizované nativně a konzistentně v mnoha prohlížečích, které není závislé na zásuvných modulech třetí strany.

7.3 Představujeme úložiště HTML5

To, o čem zde budu mluvit jako o „úložišti HTML5“, je specifikace jménem Webové úložiště, která byla původně součástí vlastní specifikace HTML5, ale pak byla od ní z nudných politických důvodů odebrána do zvláštní specifikace. Někteří dodavatelé prohlížečů mu říkají také „místní úložiště“ nebo „úložiště DOM“. S tím pojmenováním je to ještě komplikovanější kvůli dalším příbuzným, podobně pojmenovaným nově se objevujícím standardům, o kterých budu mluvit později v této kapitole.

Co je vlastně úložiště HTML5? Jednoduše řečeno je to způsob, jak mohou webové stránky místně, v klientském webovém prohlížeči, ukládat dvojice klíč-hodnota. Stejně jako cookies se tato data uchovávají i poté, kdy z webové stránky odejdete, zavřete panel prohlížeče, ukončíte prohlížeč a podobně. Na rozdíl od cookies se tato data nikdy nepředávají vzdálenému webovému serveru (pokud je tam sami nepošlete manuálně). Na rozdíl od všech předchozích pokusů o poskytnutí trvalého místního úložiště je úložiště HTML5 realizováno ve webových prohlížečích nativně, takže je k dispozici i tehdy, když jsou zásuvné moduly třetí strany nedostupné.

Úložiště HTML5 podporují nejnovější verze v podstatě všech prohlížečů… dokonce i Internet Exploreru!

Podpora pro úložiště HTML5						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
8.0+	3.5+	4.0+	4.0+	10.5+	2.0+	2.0+

Z javascriptového kódu se dostanete do úložiště HTML5 prostřednictvím objektu `localStorage` na globálním objektu `window`. Než ho začnete používat, měli byste detekovat, zda ho daný prohlížeč podporuje[1].

zkontrolujte úložiště HTML5 ↷

```
function supports_html5_storage() {
    try {
        return 'localStorage' in window && window['localStorage'] !==
        null;
    } catch (e) {
        return false;
    }
}
```

1 http://diveintohtml5.info/detect.html

Místo toho, abyste tuto funkci museli psát sami, můžete pro detekci podpory úložiště HTML5 použít Modernizr.

```
if (Modernizr.localstorage) {
    // window.localStorage je k dispozici!
} else {
    // nativní podpora pro úložiště HTML5 není k dispozici :(
    // můžete zkusit dojox.storage nebo řešení třetí strany
}
```

7.4 Používání úložiště HTML5

Úložiště HTML5 se zakládá na dvojicích pojmenovaný klíč-hodnota. Když uložíte data na základě pojmenovaného klíče, můžete se pak pomocí stejného klíče k datům vrátit. Pojmenovaný klíč je řetězec. Data můžou být jakéhokoliv typu z těch, které podporuje JavaScript, včetně řetězců, booleovských hodnot, celých čísel nebo desetinných čísel. Ve skutečnosti se však data ukládají jako řetězce. Pokud ukládáte a načítáte cokoli jiného než řetězce, budete potřebovat funkce jako `parseInt()` nebo `parseFloat()`, abyste data, která jste získali, převedli na správný javascriptový datový typ.

```
interface Storage {
    getter any getItem(in DOMString key);
    setter creator void setItem(in DOMString key, in any data);
};
```

Volání `setItem()` s už existujícím pojmenovaným klíčem tiše přepíše předchozí hodnotu. Volání `getItem()` s neexistujícím klíčem nevyvolá výjimku, ale vrátí hodnotu `null`.

Stejně jako u ostatních javascriptových objektů se k objektu `localStorage` můžete chovat jako k asociativnímu poli. Místo použití metod `getItem()` a `setItem()` můžete jednoduše používat hranaté závorky. Například tento kousek kódu:

```
var foo = localStorage.getItem("bar");
// ...
localStorage.setItem("bar", foo);
```

…může být přepsán do syntaxe využívající hranaté závorky:

```
var foo = localStorage["bar"];
// ...
localStorage["bar"] = foo;
```

Existují také metody pro odstranění hodnoty daného pojmenovaného klíče a vymazání celého úložiště (tj. vymazání všech klíčů a hodnot najednou).

```
interface Storage {
    deleter void removeItem(in DOMString key);
    void clear();
};
```

Vyvolání `removeItem()` s neexistujícím klíčem neudělá nic.

Konečně existuje vlastnost pro zjištění celkového počtu hodnot v úložišti a pro procházení všech klíčů podle indexu (pro zjištění jména každého klíče).

```
interface Storage {
    readonly attribute unsigned long length;
    getter DOMString key(in unsigned long index);
};
```

Pokud vyvoláte `key()` s indexem, který není mezi 0 a `length`-1, funkce vrátí `null`.

7.5 Sledování změn v úložišti HTML5

Pokud chcete programově sledovat, kdy se úložiště změní, můžete odchytávat události `storage`. Událost `storage` je na objektu `window` spuštěna při každém volání `setItem()`, `removeItem()` nebo `clear()`, *když se skutečně něco změní*. Například pokud nastavíte položku na existující hodnotu nebo zavoláte `clear()`, ačkoli nejsou k dispozici žádné pojmenované klíče, událost `storage` se nespustí, protože se v úložišti ve skutečnosti nic nezměnilo.

Událost `storage` je podporována všude, kde je podporován objekt `localStorage`, včetně Internet Exploreru 8. IE 8 nepodporuje standard W3C `addEventListener` (i když to se v IE 9 konečně změní). Proto pro použití události `storage` potřebujete zjistit, které mechanismy událostí daný prohlížeč podporuje. (Pokud jste už to dělali dříve u jiných událostí, můžete přeskočit na konec tohoto oddílu. Odchycení události `storage` funguje stejně jako odchycení kterékoliv jiné události. Pokud dáváte pro obsluhu událostí přednost iQuery nebo jiné javascriptové knihovně, můžete to tak udělat i s událostí `storage`.)

```
if (window.addEventListener) {
    window.addEventListener("storage", handle_storage, false);
} else {
    window.attachEvent("onstorage", handle_storage);
};
```

Zpětně volaná funkce `handle_storage` bude vyvolána pomocí objektu `StorageEvent`, s výjimkou Internet Exploreru, kde je objekt události uložen ve `window.event`.

```
function handle_storage(e) {
    if (!e) { e = window.event; }
}
```

Proměnná `e` je zde objektem `StorageEvent`, který má následující užitečné vlastnosti.

Objekt StorageEvent		
Vlastnost	**Typ**	**Popis**
`key`	řetězec	pojmenovaný klíč, který byl přidán, odstraněn nebo upraven
`oldValue`	libovolný	předchozí hodnota (nyní přepsaná); nebo `null`, pokud šlo o přidání nové položky
`newValue`	libovolný	nová hodnota, nebo `null`, jestliže byla položka odstraněna
`url*`	řetězec	stránka, která vyvolala metodu, jež spustila tuto změnu
* Poznámka: vlastnost `url` se původně jmenovala `uri`. Některé prohlížeče byly vydány s touto vlastností, než se specifikace změnila. Pro maximální kompatibilitu byste měli zkontrolovat, zda existuje vlastnost `url`, a pokud ne, hledejte vlastnost `uri`.		

Událost `storage` je nezrušitelná. Neexistuje žádný způsob, jak v rámci zpětně volané funkce `handle_storage` zabránit provedení změny. Pro prohlížeč je to prostě způsob, jak vám sdělit: „Podívej se, právě se stalo tohle. Teď už s tím neuděláš; jenom jsem ti to chtěl říct."

7.6 Omezení v současných verzích prohlížečů

Když jsem mluvil o historii hacků pro místní úložiště, které využívaly zásuvné moduly třetích stran, upozorňoval jsem na omezení každé z těchto technik, například omezení co do velikosti úložného prostoru. Uvědomil jsem si ovšem, že jsem neříkal nic o omezeních nově standardizovaného úložiště HTML5. Nejdřív vám řeknu odpovědi a pak je vysvětlím. Odpovědi, v pořadí podle důležitosti, zní: „5 megabajtů," „`QUOTA_EXCEEDED_ERR`" a „ne".

„5 megabajtů" je velikost úložného prostoru, který dostane každé místo původu (origin) ve výchozím nastavení. Tato velikost je ve všech prohlížečích překvapivě konzistentní, i když ve specifikacích úložišť HTML5 je formulovaná jako pouhý návrh. Mějte na paměti, že ukládáte řetězce, nikoliv data v původním formátu. Pokud ukládáte hodně celých nebo desetinných čísel,

rozdíl v reprezentaci se může docela nasčítat. Každá číslice v desetinném čísle se ukládá jako znak, nikoliv v obvyklé podobě jako číslo s pohyblivou řádovou čárkou.

„QUOTA_EXCEEDED_ERR“ je výjimka, která se vyvolá, pokud překročíte úložný limit 5 megabajtů. „Ne“ je odpověď na otázku, která je nasnadě: „Můžu požádat uživatele o další úložný prostor?“ V okamžiku psaní této knihy (únor 2011) nepodporuje žádný z prohlížečů žádný mechanismus, pomocí kterého by autoři stránek mohli požádat o více úložného prostoru. Některé prohlížeče (jako Opera) umožňují uživateli spravovat úložné limity jednotlivých stránek, ale jedná se o akci, která je iniciovaná čistě uživatelem, nikoliv o něco, co byste vy jako vývojář mohli zabudovat do své webové aplikace.

7.7 Úložiště HTML5 v akci

Podívejme se na úložiště HTML5 v akci. Vzpomínáte si na hru halma, kterou jsme vytvořili v kapitole o plátně? S tou hrou je malý problém: pokud uprostřed hry zavřete okno prohlížeče, rozehraná partie se vynuluje. Ale s úložištěm HTML5 můžete hru místně ukládat v rámci samotného prohlížeče. Tady je ukázka naživo[2]. Udělejte pár tahů, zavřete panel prohlížeče a pak ho znovu otevřete. Pokud váš prohlížeč podporuje úložiště HTML5, ukázková stránka by si měla jako kouzlem zapamatovat, jak si přesně ve hře stojíte, včetně počtu tahů, které jste udělali, polohy každého hracího kamenu na desce, a dokonce i zda byl zrovna vybrán nějaký kámen.

Jak to funguje? Pokaždé, když ve hře dojde ke změně, vyvoláme tuto funkci:

```
function saveGameState() {
    if (!supportsLocalStorage()) { return false; }
    localStorage["halma.game.in.progress"] = gGameInProgress;
    for (var i = 0; i < kNumPieces; i++) {
        localStorage["halma.piece." + i + ".row"] = gPieces[i].row;
        localStorage["halma.piece." + i + ".column"] = gPieces[i].
        column;
    }
    localStorage["halma.selectedpiece"] = gSelectedPieceIndex;
    localStorage["halma.selectedpiecehasmoved"] = gSelectedPiece-
    HasMoved;
    localStorage["halma.movecount"] = gMoveCount;
    return true;
}
```

2 http://diveintohtml5.info/examples/localstorage-halma.html

Jak můžete vidět, funkce používá objekt `localStorage`, který uloží informaci, zda je rozehraná nějaká hra (`gGameInProgress`, booleovská hodnota). Pokud ano, funkce projde kameny (`gPieces`, javascriptový `Array`) a uloží číslo řádku a sloupce každého kamenu. Pak uloží další stav hry, včetně toho, který kámen je zrovna vybrán (`gSelectedPieceIndex`, celé číslo), zda je tento kámen uprostřed potenciálně dlouhé série přeskoků (`gSelectedPieceHasMoved`, booleovská hodnota) a celkový počet dosud provedených tahů (`gMoveCount`, celé číslo).

Při načtení stránky se místo automatického vyvolání funkce `newGame()`, která by tyto proměnné nastavila na výchozí hodnoty, vyvolá funkce `resumeGame()`. S použitím úložiště HTML5 funkce `resumeGame()` zkontroluje, zda je stav rozehrané hry místně uložen. Pokud ano, obnoví tyto hodnoty pomocí objektu `localStorage`.

```
function resumeGame() {
    if (!supportsLocalStorage()) { return false; }
    gGameInProgress = (localStorage["halma.game.in.progress"] ==
    "true");
    if (!gGameInProgress) { return false; }
    gPieces = new Array(kNumPieces);
    for (var i = 0; i < kNumPieces; i++) {
        var row = parseInt(localStorage["halma.piece." + i + ".row"]);
        var column = parseInt(localStorage["halma.piece." + i
        + ".column"]);
        gPieces[i] = new Cell(row, column);
    }
    gNumPieces = kNumPieces;
    gSelectedPieceIndex = parseInt(localStorage
    ["halma.selectedpiece"]);
    gSelectedPieceHasMoved = localStorage
    ["halma.selectedpiecehasmoved"] == "true";
    gMoveCount = parseInt(localStorage["halma.movecount"]);
    drawBoard();
    return true;
}
```

Nejdůležitější částí této funkce je varování, které jsem už v této kapitole zmínil, ale radši ho zopakuju: *Data se ukládají jako řetězce. Pokud ukládáte něco jiného než řetězec, budete to muset při načítání sami převést.* Například příznak toho, zda existuje rozehraná partie (`gGameInProgress`), je booleovská hodnota. Ve funkci `saveGameState()` jsme ji prostě uložili a neřešili jsme datový typ:

```
localStorage["halma.game.in.progress"] = gGameInProgress;
```

Ale ve funkci `resumeGame()` se musíme k hodnotě, kterou jsme dostali z místního úložiště, chovat jako k řetězci a ručně nastavit správné booleovské hodnoty sami:

```
gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

Podobně počet tahů se ukládá v `gMoveCount` jako celé číslo. Ve funkci `saveGameState()` jsme ho pouze uložili:

```
localStorage["halma.movecount"] = gMoveCount;
```

Ovšem ve funkci `resumeGame()` musíme převést hodnotu na celé číslo, k čemuž použijeme javascriptovou funkci `parseInt()`:

```
gMoveCount = parseInt(localStorage["halma.movecount"]);
```

7.8 Víc než jen dvojice klíč-hodnota: soupeřící vize

Zatímco minulost je poseta hacky a náhradními řešeními, současný stav úložiště HTML5 vypadá překvapivě růžově. Nové API bylo standardizováno a realizováno ve všech nejpoužívanějších prohlížečích, platformách a zařízeních. Jako webový vývojář něco takového asi nevidíte každý den, že? Ale je tu ve hře víc než jen „5 megabajtů dvojic pojmenovaný klíč-hodnota". Budoucnost trvalého místního úložiště je… jak bych to řekl… zkrátka, existují zde soupeřící vize.

Jednou z těchto vizí je zkratka, kterou už pravděpodobně znáte: SQL. V roce 2007 Google spustil Gears, open-source zásuvný modul pro více prohlížečů, který obsahoval zabudovanou databázi založenou na SQLite. Tento raný prototyp později ovlivnil vznik specifikace Webové databáze SQL. Webová databáze SQL (dříve známá pod názvem „WebDB") poskytuje tenký obal kolem databáze SQL, což vám umožní dělat v JavaSriptu věci, jako je tahle:

kód, který skutečně funguje ve 4 prohlížečích ↷

```
openDatabase('documents', '1.0', 'Local document storage',
5*1024*1024, function (db) {
    db.changeVersion('', '1.0', function (t) {
        t.executeSql('CREATE TABLE docids (id, name)');
    }, error);
});
```

Jak můžete vidět, většina akce spočívá v řetězci, který předáte metodě `executeSql`. Tímto řetězcem může být jakýkoliv podporovaný příkaz SQL včetně `SELECT`, `UPDATE`, `INSERT` a `DELETE`. Je to stejné jako programování databází, jenom se to dělá v JavaScriptu. To je radosti!

Specifikace Webové databáze SQL byla realizována čtyřmi prohlížeči a platformami.

Podpora Webové databáze SQL						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
·	·	4.0+	4.0+	10.5+	3.0+	2.0+

Samozřejmě pokud jste v životě použili více než jednu databázi, víte, že je „SQL“ spíše marketingový pojem než pevný standard. (Někteří lidé by to řekli i o HTML5, ale toho si nevšímejte.) Ano, existuje sice specifikace SQL (jmenuje se SQL-92), ale na světě není žádný databázový server, který by se podřizoval jedině a pouze této specifikaci. Existuje SQL od Oracle, SQL od Microsoftu, MySQL, PostgreSQL a SQLite. A každý z těchto produktů postupně přidává do svého SQL nové prvky, takže ani když řeknete „SQL od SQLite“, tak nemusí být úplně jasné, o čem je řeč. Musíte říct „ta verze SQL, která byla vydána v SQLite verze X.Y.Z.“

Což nás přivádí k následujícímu prohlášení, které se v současnosti nachází na začátku specifikace Webové databáze SQL:

> Tato specifikace se dostala do slepé uličky: všichni zainteresovaní realizátoři používají stejný backend SQL (Sqlite), ale abychom na cestě standardizace pokročili dopředu, potřebujeme několik nezávislých implementací. Dokud nebude mít zájem o implementaci této specifikace další realizátor, popis dialektu SQL prostě odkazuje na Sqlite, což u standardu není přijatelné.

Na tomto pozadí představím další soupeřící vizi pokročilého a trvalého místního úložiště pro webové aplikace: API indexované databáze, dříve známé jako „WebSimpleDB“ a nyní láskyplně přezdívané „IndexedDB“.

API indexované databáze představuje něco, čemu se říká „úložiště objektů“ (object store). Úložiště objektů toho má hodně společného s databází SQL. Jsou zde „databáze“ se „záznamy“ a každý záznam má nastavený počet „polí“. Každé pole má specifický datový typ, který je určen při vytvoření databáze. Můžete vybrat podskupinu záznamů a pak je projít pomocí „kurzoru“. Změny v úložišti objektů se odehrávají v rámci „transakcí“.

Pokud jste někdy programovali databáze SQL, tyto pojmy vám asi zní povědomě. Hlavním rozdílem je, že úložiště objektů nemá žádný strukturovaný jazyk pro příkazy. Nevytváříte pří-

kazy jako „`SELECT * from USERS where ACTIVE = 'Y'`“. Místo toho používáte metody, které poskytuje úložiště objektů, abyste otevřeli kurzor na databázi nazvané „`USERS`“, projeli záznamy, vyřadili záznamy neaktivních uživatelů a použili přístupové metody k získání hodnot jednotlivých polí ve zbývajících záznamech. An early walk-through of IndexedDB (Raný průvodce IndexedDB)[3] je dobrým návodem, který představuje, jak IndexedDB funguje, a porovnává IndexedDB a Webovou databázi SQL.

V době, kdy jsem psal tuto knihu, byla IndexedDB implementovaná pouze v betaverzi Firefox 4. (Mozilla navíc prohlásila, že Webovou databázi SQL nikdy implementovat nebude.) Google prohlásil, že uvažuje o podpoře IndexedDB pro Chromium a Google Chrome. A dokonce i Microsoft se dal slyšet, že je IndexedDB „skvělým řešením pro web.“

Ptáte se, co můžete vy, jako webový vývojář, dělat s IndexedDB? V tuto chvíli s výjimkou pár technických demoaplikací v podstatě nic. Za rok? Možná leccos. Podívejte se do oddílu „K dalšímu čtení“ na odkazy na několik dobrých návodů jak začít.

7.9 K dalšímu čtení

Úložiště HTML5:

- HTML5 Storage[4] – specifikace
- Introduction to DOM Storage[5] na MSDN
- Web Storage: easier, more powerful client-side data storage[6] v Opera Developer Community
- DOM Storage[7] v Mozilla Developer Center (Poznámka: většina této stránky se věnuje pilotní implementaci objektu `globalStorage`, nestandardního předchůdce `localStorage`. Mozilla přidala podporu pro standardní prostředí `localStorage` ve verzi Firefoxu 3.5.)
- Unlock local storage for mobile Web applications with HTML5[8], tutoriál pro IBM DeveloperWorks

Raná práce Brada Neuberga aj. (před HTML5):

- Internet Explorer Has Native Support for Persistence?!?![9] (o objektu `userData` v IE)

3 http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/
4 https://w3c.github.io/webstorage/
5 https://msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx
6 https://dev.opera.com/articles/web-storage/
7 https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage
8 http://www.ibm.com/developerworks/xml/library/x-html5mobile2/
9 http://codinginparadise.org/weblog/2005/08/ajax-internet-explorer-has-native.html

- Dojo Storage[10], část většího tutoriálu pro (nyní zaniklou) knihovnu Dojo Offline
- dojox.storage.manager API reference[11]
- dojox.storage[12] repozitář Subversion

Webová databáze SQL:
- Web SQL Database[13] – specifikace
- Introducing Web SQL Databases[14]
- Web Database demonstration[15]
- persistence.js[16], „asynchronní javascriptový ORM" vytvořený na základě Webové databáze SQL a Gears

IndexedDB:
- Indexed Database API[17] – specifikace
- Beyond HTML5: Database APIs and the Road to IndexedDB[18]
- Firefox 4: An early walk-through of IndexedDB[19]

10 http://docs.google.com/View?docid=dhkhksk4_8gdp9gr#dojo_storage
11 http://dojotoolkit.org/api/jsdoc/HEAD/dojox.storage.manager
12 http://svn.dojotoolkit.org/src/dojox/trunk/storage/
13 http://dev.w3.org/html5/webdatabase/
14 http://html5doctor.com/introducing-web-sql-databases/
15 http://html5demos.com/database
16 http://zef.me/2774/persistence-js-an-asynchronous-javascript-orm-for-html5gears
17 http://w3c.github.io/IndexedDB/
18 https://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/
19 https://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/

8. Hoďme to offline

8. Hoďme to offline – 189

8.1 Začínáme

Co je nepřipojená (offline) webová aplikace? Na první pohled se to jeví jako protimluv. Webové stránky přece stahujeme a pak zobrazujeme. Stahování předpokládá síťové připojení. Jak můžete stahovat, když nemáte připojení? Samozřejmě, že nemůžete. Ale když jste připojeni, stahovat můžete – což umožňuje běh nepřipojených aplikací v HTML5.

Zjednodušeně řečeno, nepřipojená webová aplikace je seznamem webových adres – HTML, CSS, JavaScript, obrázků nebo jakýchkoliv jiných zdrojů. Domovská stránka nepřipojené webové aplikace odkazuje na tento seznam nazývaný manifestový soubor, jímž je textový soubor umístěný na webovém serveru. Webový prohlížeč s podporou nepřipojených aplikací HTML5 načte seznam URL z manifestového souboru, stáhne zdrojové soubory, uloží je místně do mezipaměti a bude je automaticky aktualizovat v případě změn. Jakmile se pokusíte dostat do webové aplikace bez internetového připojení, prohlížeč se automaticky přepne na místní kopie těchto souborů.

Zbytek práce je na vás, webovém vývojáři. V DOMu máte příznak určující, zda jste připojen nebo nepřipojen, a události, které se spustí při změně stavu připojení (jednou nejste připojen, za chvíli už ano, nebo naopak). A to je tak všechno. Pokud vaše aplikace vytváří data nebo ukládá stav, musíte sami zařídit místní ukládání těchto dat, když nejste připojen, a jejich synchronizaci se vzdáleným serverem při opětovném připojení. Jinými slovy, HTML5 je schopno přenést vaši webovou aplikaci offline. Co si počnete, až se tam ocitnete, je zcela na vás.

Podpora nepřipojených aplikací						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
·	3.5+	4.0+	5.0+	10.6+	2.1+	2.0+

8.2 Cache manifest

Nepřipojená webová aplikace se soustřeďuje kolem mezipaměťového manifestového souboru (cache manifest). Co je manifestový soubor? To je seznam všech zdrojů, k nimž může webová aplikace potřebovat přístup, když nebude připojena k síti. Abyste připravili proces stahování těchto zdrojů a jejich ukládání do mezipaměti, musíte odkázat na manifestový soubor pomocí atributu `manifest` v prvku `<html>`.

```
<!DOCTYPE html>
<html manifest="/cache.manifest">
<body>
```

```
...
</body>
</html>
```

Soubor cache manifest může být umístěn kdekoliv na webovém serveru, ale musí být předáván s typem obsahu `text/cache-manifest`. Pokud provozujete webový server Apache, možná bude stačit do souboru `.htaccess` v kořenovém adresáři webu přidat příkaz `AddType`:

```
AddType text/cache-manifest .manifest
```

Zajistěte, aby název souboru cache manifest končil na `.manifest`. Jestliže používáte jiný webový server nebo jinou konfiguraci Apache, ověřte si podle příručky k serveru, jak je řízena hlavička `Content-Type`.

PTEJTE SE PROFESORA ZNAČKY

Q: Moje webová aplikace má více než jednu stránku. Bude potřeba mít atribut `manifest` na každé stránce, nebo ho stačí dát jen na tu domovskou?

A: Každá stránka webové aplikace potřebuje atribut `manifest` odkazující na soubor cache manifest pro celou aplikaci.

Takže každá stránka HTML odkazuje na soubor cache manifest a soubor cache manifest se předává s příslušnou hlavičkou `Content-Type`. Ale co se dává *do* manifestového souboru? Tady to začíná být zajímavé.

První řádek každého souboru cache manifest vypadá takto:

```
CACHE MANIFEST
```

Pod nim se každý manifestový soubor dělí na tři části: „explicitní“ oddíl, „záložní“ oddíl a oddíl „výjimek pro online“. Každý oddíl má řádek s hlavičkou. Pokud nemá manifestový soubor žádnou hlavičku oddílu, má se za to, že všechny uvedené zdroje se nacházejí v „explicitním“ oddíle. Tu terminologii neřešte, nebo vám praskne hlava.

Toto je funkční manifestový soubor. Obsahuje odkazy na tři zdroje: soubor CSS, soubor JavaScript a obrázek JPEG.

```
CACHE MANIFEST
/clock.css
```

```
/clock.js
/clock-face.jpg
```

Tento soubor cache manifest nemá hlavičku oddílu, proto se všechny uvedené zdroje automaticky nacházejí v „explicitním“ oddíle. Zdroje v „explicitním“ oddíle budou staženy a uloženy do lokální mezipaměti a při absenci síťového připojení budou použity místo svých protějšků na síti. To znamená, že při načtení souboru cache manifest prohlížeč stáhne soubory `clock.css`, `clock.js` a `clock-face.jpg` z kořenového adresáře webového serveru. Pak můžete odpojit síťový kabel a aktualizovat stránku: všechny tyto zdroje budou dostupné i bez připojení.

PTEJTE SE PROFESORA ZNAČKY

Q: Musím zahrnout do seznamu cache manifest i stránky HTML?

A: Ano i ne. Pokud celou vaši webovou aplikaci tvoří jediná stránka, stačí, když bude odkazovat na cache manifest pomocí atributu `manifest`. Když otevřete stránku HTML obsahující atribut `manifest`, taková stránka je již považována za součást webové aplikace, proto ji nemusíte zmiňovat v samotném manifestovém souboru. Pokud však má webová aplikace více stránek HTML, musíte je všechny uvést v manifestovém souboru, jinak se prohlížeč nedoví, že je potřeba stáhnout a uložit do mezipaměti i další stránky.

SÍŤOVÝ ODDÍL

Vezměme si o něco složitější příklad. Dejme tomu, že chcete, aby vaše aplikace „hodiny“ sledovala návštěvníky pomocí skriptu tracking.cgi, který se dynamicky načítá z atributu `<img src>`. Uložení do mezipaměti by zničilo účel sledování, proto by aplikace nikdy neměla být uložena a zpřístupněna bez připojení. Uděláte to takto:

```
CACHE MANIFEST
NETWORK:
/tracking.cgi
CACHE:
/clock.css
/clock.js
/clock-face.jpg
```

Tento soubor cache manifest obsahuje hlavičky oddílů. Řádek označený `NETWORK:` je začátkem oddílu „výjimek pro online“. Zdroje uvedené v tomto oddílu se nikdy neukládají do mezipaměti a nejsou dostupné bez připojení. (Při pokusu je načíst bez připojení se zobrazí chyba.) Řádek

označený `CACHE:` je začátek „explicitního“ oddílu. Zbytek souboru je stejný jako v předchozím příkladu. Každý ze tří uvedených zdrojů bude uložen a zpřístupněn bez síťového připojení.

ZÁLOŽNÍ ODDÍL

V souboru cache manifest existuje ještě jeden oddíl, záložní. V záložním oddílu můžete nastavit náhrady za zdroje na síti, které se z nějakého důvodu neuložily do mezipaměti nebo jejichž ukládání selhalo. Specifikace HTML5 nabízí chytrý příklad využití záložního oddílu:

```
CACHE MANIFEST
FALLBACK:
/ /offline.html
NETWORK:
*
```

Jak to funguje? Představte si portál, který obsahuje miliony stránek, jako například Wikipedie. Celou Wikipedii si stáhnout nemůžete a ani nechcete. Ale předpokládejme, že byste chtěli nějakou její část zpřístupnit bez připojení. Ale jak byste rozhodli, které stránky uložit do mezipaměti? Co říkáte na tohle: každá stránka, kterou kdy na hypotetické Wikipedii s podporou prohlížení bez připojení navštívíte, se stáhne a uloží do mezipaměti. To by zahrnulo všechny encyklopedické články, které jste navštívili, všechny diskusní stránky (na kterých můžete diskutovat o jednotlivých encyklopedických článcích) a všechny stránky pro úpravy (na kterých můžete encyklopedické články měnit).

A tohle přesně dělá tento cache manifest. Představme si, že by každá stránka HTML (článek, diskusní stránka, stránka pro úpravy, stránka historie úprav) na Wikipedii odkazovala na tento soubor cache manifest. Když navštívíte stránku, která odkazuje na cache manifest, váš prohlížeč řekne „Hele, tahle stránka je částí nepřipojené webové aplikace, znám ji?“ Pokud prohlížeč tento konkrétní soubor cache manifest ještě nestahoval, vytvoří novou „appcache“ (zkratka pro „application cache“ – „aplikační mezipaměť“) pro režim bez připojení, stáhne si všechny zdroje uvedené v cache manifestu a pak přidá aktuálně prohlíženou stránkou do aplikační mezipaměti. Pokud váš prohlížeč daný cache manifest zná, prostě přidá aktuálně prohlíženou stránku do existující aplikační mezipaměti. V každém případě stránka, kterou jste právě navštívili, skončí v aplikační mezipaměti. To je důležité. Znamená to, že můžete mít webovou aplikaci pracující bez připojení, které „líně“ přidává stránky spolu s tím, jak je navštěvujete. Nemusíte v cache manifestu vypsat úplně všechny stránky HTML.

Teď se podívejme na záložní oddíl. Záložní oddíl tohoto cache manifestu má pouze jeden řádek. První část řádku (před mezerou) není URL. Ve skutečnosti je to vzor URL. Jeden znak (/) bude odpovídat jakékoliv stránce na vašem webu, nejen domovské. Když zkoušíte navštívit nějakou stránku, když nejste připojeni, váš prohlížeč ji bude hledat v aplikační mezipaměti. Když

tam prohlížeč stránku najde (protože jste ji navštívili, zatímco jste byli připojeni, a stránka se tam v tu chvíli implicitně přidala), zobrazí kopii stránky z mezipaměti. Když prohlížeč stránku v aplikační mezipaměti nenajde, nezobrazí chybové hlášení, ale stránku `/offline.html`, jak to určuje druhá půlka toho řádku v záložním oddílu.

A konečně se podívejme blíže na síťový oddíl. Síťový oddíl v tomto cache manifestu má také jeden řádek, který obsahuje jediný znak (*). Tento znak má pro síťový oddíl zvláštní význam. Říká se mu „žolíkový příznak výjimek pro online". Pod tímto zajímavým názvem se skrývá fakt, že cokoliv, co není v aplikační mezipaměti, se může stáhnout z původní webové adresy, pokud máte internetové připojení. To je důležité pro „otevřené" webové aplikace pracující bez připojení. Znamená to, že zatímco si *online* prohlížíte naši hypotetickou Wikipedii s podporou prohlížení bez připojení, prohlížeč bude načítat obrázky, videa a další vložené zdroje normálně, i když se nacházejí v jiné doméně. (To je obvyklé u velkých webů, ačkoli nejsou částí nepřipojené webové aplikace. Stránky HTML jsou vytvářeny a předávány místně, zatímco obrázky a videa se předávají z CDN v jiné doméně.) Bez tohoto žolíkového příznaku by se naše hypotetická Wikipedie fungující bez připojení při návštěvě s připojením chovala zvláštně – konkrétně řečeno by nenačítala žádné externě uložené obrázky ani videa!

Je tento příklad úplný? Není. Wikipedii netvoří jen soubory HTML. Používá na každé stránce také soubory CSS, JavaScript a obrázky. Každý z těchto zdrojů by musel být v oddílu `CACHE:` manifestového souboru výslovně uveden, aby se stránky správně zobrazovaly a pracovaly bez připojení. Ale smyslem záložního oddílu je, že můžete mít „otevřené" nepřipojené webové aplikace, které se neomezují pouze na zdroje výslovně uvedené v manifestovém souboru.

8.3 Tok událostí

Zatím jsem mluvil o nepřipojených webových aplikacích, cache manifestu a aplikační mezipaměti („appcache") pro stav bez připojení neurčitými, napůl magickými slovy. Věci se stahují, prohlížeče dělají rozhodnutí, a všechno to prostě nějak funguje. Ale co si tu budeme povídat: mluvíme tu o vytváření webů, a tam nic nefunguje jen tak „nějak".

Nejdřív si řekněme něco o toku událostí, konkrétně událostí v DOMu. Kdykoliv váš prohlížeč navštíví stránku odkazující na cache manifest, spustí sérii událostí na objektu `window.applicationCache`. Vím, že to vypadá komplikovaně, ale tohle je ta nejjednodušší verze, co mě napadla, která nevynechává žádné podstatné informace.

- Jakmile si prohlížeč všimne atributu `manifest` na prvku `<html>`, spustí událost `checking`. (Všechny zde popsané události se spouštějí na objektu `window.applicationCache`.) Událost `checking` se spouští vždy, bez ohledu na to, zda už jste předtím navštívili tuto stránku nebo jinou, která odkazuje na stejný cache manifest.

- Pokud prohlížeč daný cache manifest nikdy neviděl…
    - Spustí událost `downloading` a pak začne stahovat zdroje uvedené v cache manifestu.
    - Zatímco se zdroje stahují, prohlížeč bude pravidelně spouštět události `progress`, které obsahují informace o tom, kolik souborů už se stáhlo a kolik čeká ve frontě na stažení.
    - Poté, co se všechny zdroje uvedené v cache manifestu úspěšně stáhnou, prohlížeč spustí poslední událost: `cached`. To je pro vás signálem, že je webová aplikace kompletně uložena do mezipaměti a připravena na použití bez připojení. A to je všechno, máte hotovo.
- Pokud jste ovšem tuto stránku nebo jinou stránku odkazující na stejný cache manifest už navštívili, váš prohlížeč už tento cache manifest zná. Možná už má v aplikační mezipaměti nějaké zdroje. Možná tam má dokonce celou fungující nepřipojenou webovou aplikaci. Teď vyvstává otázka, jestli se cache manifest změnil od chvíle, co ho váš prohlížeč naposledy kontroloval.
    - Je-li odpověď ne, cache manifest se nezměnil, prohlížeč okamžitě spustí událost `noupdate`. A to je vše, máte hotovo.
    - Je-li odpověď ano, cache manifest se změnil, prohlížeč spustí událost `downloading` a začne znovu stahovat všechny zdroje uvedené v cache manifestu.
    - Zatímco se stahují, prohlížeč bude pravidelně spouštět události `progress`, které obsahují informace o tom, kolik souborů už se stáhlo a kolik čeká ve frontě na stažení.
    - Poté, co se všechny zdroje uvedené v cache manifestu úspěšně znovu stáhnou, prohlížeč spustí poslední událost `updateready`. To je pro vás signálem, že je nová verze webové aplikace kompletně uložena do mezipaměti a připravena na použití bez připojení. *Nová verze se ovšem ještě nepoužívá.* Abyste vyměnili starou verzi za novou, aniž byste nutili uživatele stránku znovu načíst, můžete ručně vyvolat funkci `window.applicationCache.swapCache()`.
- Pokud se v průběhu celého tohoto procesu něco zásadním způsobem pokazí, prohlížeč spustí událost `error` a zastaví se. Tady je zdaleka nevyčerpávající seznam věcí, které se mohou pokazit:
    - Cache manifest vrátil chybu HTTP 404 (Stránka nenalezena) nebo 410 (Trvale smazáno).

- Cache manifest byl nalezen a nezměnil se, ale stránku HTML odkazující na manifest se nepovedlo správně stáhnout.
- Cache manifest se změnil v průběhu aktualizace.
- Cache manifest byl nalezen a změnil se, ale prohlížeč nedokázal stáhnout jeden z v něm uvedených zdrojů.

8.4 Umění odstraňování chyb, aneb „Zabijte mě někdo!"

Chtěl bych tu upozornit na dvě důležité věci. První z nich jste si právě přečetli, ale vsadím se, že vám to ještě úplně nedošlo, takže opakuju: Když se byť jen jeden ze zdrojů uvedený ve vašem souboru cache manifest správně nestáhne, celý proces ukládání vaší nepřipojené webové aplikace do mezipaměti selže. Prohlížeč spustí událost `error`, ale z toho se nedozvíte, v čem vlastně spočívá problém. Kvůli tomu může být odstraňování chyb u nepřipojených webových aplikací ještě větší otrava než obvykle.

Druhým důležitým bodem je něco, co technicky vzato není chyba, ale než zjistíte, co se děje, vypadá to v prohlížeči jako závažný problém. Souvisí to s tím, jak přesně váš prohlížeč kontroluje, zda se cache manifest změnil. Je to třífázový proces. Tato část je nudná, ale důležitá, tak prosím dávejte pozor.

- Pomocí normální sémantiky HTTP prohlížeč zkontroluje, zda není cache manifest zastaralý. Jako u všech ostatních souborů, které se předávají přes HTTP, váš webový server obvykle v hlavičce odpovědi HTTP uvede metainformace o souboru. Některé z těchto hlaviček HTTP (`Expires` a `Cache-Control`) říkají prohlížeči, zda může uložit soubor do mezipaměti, aniž by se ptal serveru na změnu. Tento způsob ukládání do mezipaměti nemá s nepřipojenými webovými aplikacemi nic společného. Děje se to snad u každé stránky HTML, šablony stylů, skriptu, obrázku nebo jiného zdroje na webu.
- Pokud je cache manifest zastaralý (podle jeho HTTP hlaviček), prohlížeč se zeptá serveru, zda existuje nová verze, a pokud ano, prohlížeč ji stáhne. To se stane tak, že prohlížeč vyšle požadavek HTTP, který bude obsahovat datum poslední změny cache manifestu, které váš webový server uvedl v hlavičce odpovědi HTTP, když jste naposledy stahovali soubor manifestu. Pokud webový server zjistí, že se od toho data soubor manifestu nezměnil, jednoduše vrátí status `304` (`Not Modified`). Opakuju, že nic z toho není specifické pouze pro nepřipojené webové aplikace. Děje se to v podstatě u všech zdrojů na webu.
- Pokud webový server dojde k názoru, že se od daného data soubor manifestu změnil, vrátí status HTTP `200` (`OK`), následovaný obsahem nového souboru spolu s novými hlavičkami

`Cache-Control` a novým datem poslední změny, takže příště se kroky 1 a 2 řádně provedou. (Výhodou HTTP je, že webové servery vždycky myslí na budoucnost. Když už vám webový server nutně musí poslat nějaký soubor, dělá vše pro to, aby zajistil, že ho nebude muset zbytečně posílat dvakrát.) Jakmile se stáhne nový soubor cache manifest, prohlížeč porovná jeho obsah s kopií, kterou si stáhl posledně. Pokud je obsah stejný, prohlížeč nebude znovu stahovat žádný ze souborů uvedených v manifestu.

U každého z těchto kroků můžete při vyvíjení a testování nepřipojené webové aplikace narazit. Například nahrajete verzi souboru cache manifest a za 10 minut zjistíte, že musíte přidat další zdroj. Žádný problém, že? Prostě přidáme nový řádek a nahrajeme znovu. Ouha! Stane se totiž tohle: aktualizujete stránku, prohlížeč si všimne atributu `manifest`, spustí událost `checking` a pak... nic. Váš prohlížeč zarytě trvá na tom, že se soubor cache manifest nezměnil. Proč? Protože výchozí nastavení vašeho webového serveru je pravděpodobně takové, že říká prohlížečům, aby statické soubory ukládaly do mezipaměti na několik hodin (prostřednictvím sémantiky HTTP, pomocí hlaviček `Cache-Control`). To znamená, že se váš prohlížeč nikdy nedostane dále než na první krok toho třífázového procesu. Jistě, webový server ví, že se soubor změnil, ale prohlížeč se vůbec nedostane k tomu, aby se na to serveru zeptal. Proč? Protože když prohlížeč naposledy stahoval cache manifest, webový server mu řekl, aby zdroj uložil do mezipaměti na několik hodin (prostřednictvím HTTP sémantiky, pomocí hlaviček `Cache-Control`). A teď, o 10 minut později, prohlížeč přesně tohle dělá.

Aby bylo jasno, nejedná se o chybu, ale o funkci. Všechno funguje přesně tak, jak má. Kdyby webové servery neměly jak prohlížečům (a proxy serverům) sdělit, aby ukládaly věci do mezipaměti, celá síť by se přes noc zhroutila. Ale to vás v situaci, kdy jste strávili několik hodin bádáním nad tím, proč si prohlížeč nevšímá vašeho aktualizovaného cache manifestu, příliš neutěší. (A co je ještě lepší: když si počkáte dostatečně dlouho, záhadně to začne fungovat, protože mezipaměť HTTP zastarala – přesně jak má. Zabijte mě někdo, prosím!)

Proto byste měli určitě udělat tohle: změňte nastavení svého webového serveru, aby sémantika HTTP neukládala cache manifest do mezipaměti. Pokud váš webový server běží na Apache, měly by to zařídit tyhle dva řádky v souboru `.htaccess`:

```
ExpiresActive On
ExpiresDefault "access"
```

Tím se vypne ukládání do mezipaměti pro všechny soubory v tomto adresáři a podadresářích. V produkčním prostředí to ale asi nechcete, takže byste to měli specifikovat příkazem `<Files>`, aby se zákaz ukládání do mezipaměti vztahoval pouze na soubor cache manifest, nebo vytvořte podadresář, který bude obsahovat pouze soubor `.htaccess` a cache manifest. Detaily nastavení se jako vždy server od serveru liší, takže si ověřte podle příručky k serveru, jak je řízeno ukládání hlaviček HTTP do mezipaměti.

I když zamezíte ukládání souboru cache manifest do mezipaměti, může se vám občas přihodit, že změníte některý ze zdrojů v aplikační mezipaměti, ale na serveru stále bude na stejné URL. V tom případě vás dostane krok 2. Pokud se soubor cache manifest nezmění, prohlížeč si nevšimne, že se změnil jeden ze zdrojů, které byly dříve uloženy do mezipaměti. Podívejte se na tento případ:

```
CACHE MANIFEST
# rev 42
clock.js
clock.css
```

Pokud změníte soubor `clock.css` a znovu ho nahrajete, žádné změny neuvidíte, protože sám soubor cache manifest se nezměnil. Pokaždé, když změníte některý ze zdrojů v nepřipojené webové aplikaci, musíte změnit i soubor cache manifest. Může to být třeba i jen změna jednoho znaku. Nejjednodušší způsob, jak toho dosáhnout, je podle mě přidat řádek s komentářem a psát do něj číslo revize. Když změníte číslo revize v komentáři, webový server vrátí nově změněný soubor cache manifest, prohlížeč si všimne, že se obsah souboru změnil, a zahájí proces nového stahování všech zdrojů uvedených v manifestu.

```
CACHE MANIFEST
# rev 43
clock.js
clock.css
```

8.5 Vytvořme si nepřipojenou aplikaci

Pamatujete si na hru halma, kterou jsme představili v kapitole o plátně a pak vylepšili možností ukládání rozehrané hry pomocí trvalého místního úložiště? Zkusme ji teď upravit, aby šla hrát bez připojení.

Na to potřebujeme manifest, ve kterém budou uvedeny všechny pro hru potřebné zdroje. Takže máme tady hlavní HTML stránku, jeden javascriptový soubor, který obsahuje veškerý kód hry a… to je všechno. Nejsou tu žádné obrázky, protože veškeré vykreslování se děje programově pomocí API plátna. Všechny potřebné styly CSS jsou obsaženy v prvku `<style>` na vrchu HTML stránky. Takže náš cache manifest vypadá takhle:

```
CACHE MANIFEST
halma.html
../halma-localstorage.js
```

Něco o cestách. V adresáři `examples/` jsem vytvořil podadresář `offline/`, ve kterém je umístěn tento soubor cache manifest. Protože naše stránka HTML potřebuje jedno drobné doplnění, aby mohla fungovat i bez připojení (k tomu se dostaneme za chvilku), vytvořil jsem zvláštní kopii souboru HTML, která se také nachází v podadresáři `offline/`. Ale protože javascriptový kód se od přidání podpory místního úložiště nemění, používám stejný soubor `.js` jako ten v nadřazeném adresáři (`examples/`). Dohromady vypadají ty soubory takhle:

```
/examples/localstorage-halma.html
/examples/halma-localstorage.js
/examples/offline/halma.manifest
/examples/offline/halma.html
```

V souboru cache manifest (`/examples/offline/halma.manifest`) chceme odkazovat na dva soubory. Prvním je offline verze souboru HTML (`/examples/offline/halma.html`). Protože jsou tyto dva soubory ve stejném adresáři, je v manifestovém souboru tento soubor uvedený bez prefixu cesty. Druhým je javascriptový soubor, který se nachází v nadřazeném adresáři (`/examples/halma-localstorage.js`). Ten je uvedený v manifestovém souboru pomocí zápisu pro relativní adresu: `../halma-localstorage.js`. Podobně byste relativní adresu mohli použít v atributu `<img src>`. A jak uvidíte v dalším příkladu, můžete použít i absolutní cesty (které začínají v kořenovém adresáři aktuální domény) nebo i absolutní adresy (které odkazují na zdroje v jiných doménách).

Do souboru HTML teď musíme přidat atribut `manifest`, který odkazuje na soubor cache manifest.

```
<!doctype html>
<html lang="en" manifest="halma.manifest">
```

A to je vše! Až prohlížeč umožňující práci bez připojení poprvé načte stránku HTML s možností prohlížení bez připojení, stáhne soubor cache manifest, na který se odkazuje, a začne stahovat všechny uvedené zdroje a ukládat je do aplikační mezipaměti. Od té chvíli se algoritmus nepřipojené aplikace spustí pokaždé, když stránku znovu navštívíte. Můžete hrát halmu bez připojení, a protože se stav rozehraných partií ukládá do místního úložiště, můžete odcházet a zase se vracet tak často, jak se vám líbí.

8.6 K dalšímu čtení

Standardy:
- Offline web applications[1] ve specifikaci HTML5

Dokumentace od dodavatelů prohlížečů:
- Offline resources in Firefox[2]
- HTML5 offline application cache[3], část Safari client-side storage and offline applications programming guide[4]

Tutoriály a dema:
- Gmail for mobile HTML5 series: using appcache to launch offline - part 1[5]
- Gmail for mobile HTML5 series: using appcache to launch offline - part 2[6]
- Gmail for mobile HTML5 series: using appcache to launch offline - part 3[7]
- Debugging HTML5 offline application cache[8]
- an HTML5 offline image editor and uploader application[9]
- 20 Things I Learned About Browsers and the Web[10], pokročilé demo použití aplikační mezipaměti a dalších technik HTML5

Nástroje mezipaměti nepřipojených aplikací HTML5:
- Cache Manifest Validator[11], online validační služba
- Manifesto[12], validační bookmarklet

1 https://html.spec.whatwg.org/multipage/browsers.html#offline
2 https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache
3 https://developer.apple.com/library/safari/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/OfflineApplicationCache/OfflineApplicationCache.html
4 https://developer.apple.com/library/safari/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Introduction/Introduction.html
5 http://googlecode.blogspot.cz/2009/04/gmail-for-mobile-html5-series-using.html
6 http://googlecode.blogspot.cz/2009/05/gmail-for-mobile-html5-series-part-2.html
7 http://googlecode.blogspot.cz/2009/05/gmail-for-mobile-html5-series-part-3.html
8 https://jonathanstark.com/blog/debugging-html-5-offline-application-cache
9 https://hacks.mozilla.org/2010/02/an-html5-offline-image-editor-and-uploader-application/
10 http://www.20thingsilearned.com/cs-CZ
11 http://manifest-validator.com/
12 http://manifesto.ericdelabar.com/

— 8. Hoďme to offline

9. Forma šílenství

9. Forma šílenství – 203

9.1 Začínáme

Webové formuláře umí dělat každý, že jo? Uděláte `<form>`, přidáte pár prvků `<input type="text">`, možná taky `<input type="password">`, zakončete to tlačítkem `<input type="submit">` a máte hotovo.

Jenže to není zdaleka všechno. HTML5 definuje víc než desítku nových vstupních typů, které můžete ve formulářích používat. A když říkám „používat", tak tím myslím, že je můžete začít používat hned teď – bez všelijakých záplat, hacků a náhradních řešení. Neradujte se ale předčasně – nechci tím říct, že všechny ty super nové funkce podporuje každý prohlížeč. Kdepak, to opravdu ne. V moderních prohlížečích budou vaše formuláře nepřekonatelné. Ve starších prohlížečích budou stále fungovat, i když už to nebude tak slavné. Tím chci říct, že všechny tyto nové funkce se v každém prohlížeči zhoršují elegantně. A to i v IE6.

9.2 Zástupný text

Podpora zástupného textu						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
10.0+	4.0+	4.0+	4.0+	11.0+	4.0+	2.1+

První zlepšení, které HTML5 do webových formulářů přináší, je schopnost ve vstupním poli nastavit zástupný text. Zástupný text se zobrazuje uvnitř vstupního pole, dokud je pole prázdné. Když kliknete (nebo přepnete tabulátorem) do vstupního pole a začnete psát, zástupný text zmizí.

Zástupný text už jste pravděpodobně viděli. Například Mozilla Firefox používá v adresním řádku zástupný text ve znění „Hledat nebo vložit adresu".

Když kliknete (nebo přepnete tabulátorem) na adresní řádek, zástupný text zmizí:

Zástupný text do webových formulářů můžete umístit takto:

```
<form>
    <input name="q" placeholder="Přejít na adresu">
    <input type="submit" value="Search">
</form>
```

Prohlížeče, které atribut `placeholder` nepodporují, ho budou jednoduše ignorovat. Žádný problém. Podívejte se, zda váš prohlížeč podporuje zástupný text[1].

PTEJTE SE PROFESORA ZNAČKY

Q: Můžu v atributu `placeholder` použít kód HTML? Chci vložit obrázek nebo možná změnit barvy.

A: Atribut `placeholder` může obsahovat pouze text, nikoliv kód HTML. K dispozici jsou ale rozšíření CSS, která vám v některých prohlížečích umožní měnit styl zástupného textu.

9.3 Autofokus polí

Podpora autofokusu						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
10.0+	4.0+	4.0+	3.0+	10.0+	·	3.0+

Webové stránky mohou používat JavaScript, aby automaticky nastavily fokus prvnímu vstupnímu poli webového formuláře. Například domovská stránka Google.com má autofokus na vstupním poli, takže můžete napsat hledaná klíčová slova bez toho, abyste museli umístit kurzor do vyhledávacího pole. Tohle je sice pro většinu lidí pohodlné, ale pro pokročilé uživatele nebo lidi se speciálními potřebami to může být otrava. Pokud zmáčknete mezerník s tím, že chcete stránku posouvat, stránka se vám posouvat nebude, protože fokus už je ve vstupním poli formuláře. (Místo posouvání tak napíšete do pole mezeru.) Pokud dáte fokus na jiné vstupní pole, zatímco se stránka ještě načítá, skript pro autofokus stránky vám „pomůže“ tím, že po úplném načtení přemístí fokus zpátky do původního vstupního pole, což naruší váš rytmus a způsobí, že začnete psát do špatného místa.

1 http://diveintohtml5.info/examples/input-placeholder.html

Protože autofokus funguje pomocí JavaScriptu, může být obtížné vypořádat se se všemi těmito mezními případy, a pro lidi, kteří nechtějí, aby webová stránka fokus „kradla“, moc východisek není.

Pro vyřešení tohoto problému HTML5 přichází s atributem `autofocus` pro webové formuláře. Atribut `autofocus` dělá přesně to, co říká: hned jak se stránka načte, přesouvá fokus do určitého vstupního pole. Ale protože se jedná pouze o součást značkovacího jazyka a nikoliv o skript, jeho chování bude na všech webových stránkách konzistentní. Dodavatelé prohlížečů (nebo autoři rozšíření) také můžou uživatelům nabídnout způsob, jak autofokus vypnout.

Autofokus nastavíme formulářovému poli následovně:

```
<form>
    <input name="q" autofocus>
    <input type="submit" value="Search">
</form>
```

Prohlížeče, které atribut `autofocus` nepodporují, ho budou jednoduše ignorovat. Podívejte se, zda váš prohlížeč podporuje autofokus[2].

Cože? Že chcete, aby váš autofokus fungoval ve všech prohlížečích, nejen v těch fešáckých, co podporují HTML5? Můžete zůstat u svého současného skriptu pro autofokus. Stačí provést dvě drobné změny:

- Přidejte do kódu HTML atribut `autofocus`.
- Zjistěte, zda prohlížeč podporuje atribut `autofocus`, a skript pro autofokus spusťte pouze v případě, že pro autofokus neexistuje nativní podpora.

  Náhradní řešení pro autofokus ↷

```
<form name="f">
    <input id="q" autofocus>
    <script>
        if (!("autofocus" in document.createElement("input"))) {
            document.getElementById("q").focus();
        }
    </script>
    <input type="submit" value="Go">
</form>
...
```

2 http://diveintohtml5.info/examples/input-autofocus.html

Podívejte se na příklad náhradního řešení pro `autofocus`[3].

NASTAVTE AUTOFOKUS CO NEJDŘÍVE

Hodně stránek s nastavením fokusu čeká na spuštění `window.onload`. Ale událost `window.onload` se nespustí, dokud se nenačtou všechny obrázky. Pokud má vaše stránka hodně obrázků, takto naivní skript by mohl teoreticky vrátit fokus do pole ve chvíli, kdy už uživatel začal pracovat s jinou částí stránky. Proto pokročilí uživatelé autofokusové skripty nemají rádi.

V příkladu v předchozím oddílu byl autofokusový skript umístěn hned za formulářovým polem, ke kterému odkazuje. To je ideální řešení, ale strkat kus javascriptového kódu doprostřed stránky vám možná přijde nevzhledné. (Anebo, což bývá prozaičtější důvod, vaše backendové systémy možná nejsou tak flexibilní.) Pokud nemůžete vložit skript doprostřed stránky, měli byste nastavit fokus během uživatelské události, jako je `$(document).ready()` v jQuery, místo `window.onload`.

Autofokus s náhradním řešením v jQuery ↷

```
<head>
<script src="jquery.min.js"></script>
<script>
    $(document).ready(function() {
        if (!("autofocus" in document.createElement("input"))) {
            $("#q").focus();
        }
    });
</script>
</head>
<body>
<form name="f">
    <input id="q" autofocus>
    <input type="submit" value="Go">
</form>
```

Podívejte se na příklad použití `autofocus` s náhradním řešení v jQuery[4].

jQuery spustí událost `ready` hned, jak bude k dispozici DOM stránky – to znamená, že čeká, až se stránka načte, ale nečeká na načtení všech obrázků. To není ideální přístup – pokud je stránka nezvykle velká nebo internetové připojení nezvykle pomalé, uživatel může i tak začít

3 http://diveintohtml5.info/examples/input-autofocus-with-fallback.html
4 http://diveintohtml5.info/examples/input-autofocus-with-fallback-document-ready.html

pracovat se stránkou dřív, než se spustí skript pro nastavení fokusu. Ale je to pořád mnohem lepší než čekat na spuštění události `window.onload`.

Pokud chcete a můžete do kódu stránky přidat jeden příkaz skriptu, můžeme najít kompromis, který je méně nevzhledný než první možnost a lepší než druhá. Můžete použít události v jQuery pro definování vlastní události, kterou pojmenujete třeba `autofocus_ready`. Pak můžete tuto událost ručně spustit hned, jakmile je připraveno formulářové pole, které má dostat fokus. *Tímto chci poděkovat E. M. Sternbergovi za to, že mě tuto techniku naučil.*

Autofokus s náhradním řešením využívajícím vlastní událost ↷

```
<head>
<script src="jquery.min.js"></script>
<script>
    $(document).bind('autofocus_ready', function() {
        if (!("autofocus" in document.createElement("input"))) {
            $("#q").focus();
        }
    });
</script>
</head>
<body>
<form name="f">
    <input id="q" autofocus>
    <script>$(document).trigger('autofocus_ready');</script>
    <input type="submit" value="Go">
</form>
```

Podívejte se na příklad použití `autofocus` s náhradním řešením využívajícím vlastní událost[5].

Toto řešení je stejně optimální jako první přístup: nastaví fokus do formulářového pole ihned, jak to bude technicky možné, zatímco se text stránky ještě načítá. Ale zároveň přesouvá většinu logiky vaší aplikace (fokus formulářového pole) z těla stránky do jejího záhlaví. Tento příklad pracuje s jQuery, ale vlastní události můžete vytvářet i jinde. Podobné možnosti nabízejí i další javascriptové knihovny, např. YUI a Dojo.

Když to shrneme:

- Správné nastavení fokusu je důležité.

5 http://diveintohtml5.info/examples/input-autofocus-with-fallback-custom-event.html

- Pokud je to možné, nechte to prohlížeč udělat pomocí nastavení atributu `autofocus` na formulářové pole, na kterém chcete fokus.
- Pokud píšete kód náhradního řešení pro starší prohlížeče, detekujte podporu atributu `autofocus`, abyste zajistili, že se toto náhradní řešení spustí pouze ve starších prohlížečích.
- Nastavte fokus hned, jak to půjde. V kódu vložte skript pro fokus hned za formulářové pole. Pokud se vám to nelíbí, použijte javascriptovou knihovnu, která podporuje vlastní události, a spusťte vlastní událost hned po kódu pro formulářové pole. Pokud to není možné, použijte něco jako událost `$(document).ready()` v jQuery.
- V žádném případě nečekejte s nastavením fokusu až na `window.onload`.

9.4 E-mailové adresy

Po více než desetiletí se webové formuláře skládaly z polí jen několika typů. Mezi ty nejběžnější patřily:

Typ pole	Kód HTML	Poznámky
zaškrtávací políčko	`<input type="checkbox">`	může být zaškrtnuto nebo odškrtnuto
přepínač	`<input type="radio">`	může se shlukovat spolu s dalšími vstupními poli
pole pro heslo	`<input type="password">`	místo znaků, které píšete, zobrazuje tečky
rozbalovací seznamy	`<select><option>…`	
výběr souborů	`<input type="file">`	otevře dialogové okno „otevřít soubor“
tlačítko pro odeslání	`<input type="submit">`	
prostý text	`<input type="text">`	atribut `type` je možno vynechat

Všechny tyto vstupní typy fungují i v HTML5. Pokud „aktualizujete na HTML5“ (například změnou DOCTYPE), nemusíte na svých webových formulářích nic měnit. Buďme rádi za zpětnou kompatibilitu!

HTML5 ovšem definuje 13 nových typů polí a z důvodů, které pochopíte za chvíli, nemáte proč je nezačít používat.

První z těchto nových vstupních typů je pro e-mailové adresy. Vypadá takhle:

```
<form>
    <input type="email">
    <input type="submit" value="Go">
</form>
```

Chtěl jsem napsat větu, která by začínala „v prohlížečích, které nepodporují `type="email"`…“, ale zarazil jsem se. Proč? Protože si nejsem jistý, co by vlastně mělo znamenat, že nějaký prohlížeč nepodporuje `type="email"`. Všechny prohlížeče `type="email"` „podporují“. Možná s ním nedělají nic zvláštního (pár příkladů zvláštního zacházení uvidíte za chvíli), ale prohlížeče, které `type="email"` nerozpoznají, s ním budou nakládat jako s `type="text"` a zobrazí ho jako pole s prostým textem.

Nemůžu dostatečně zdůraznit, jak je to důležité. Na webu najdete miliony formulářů, které po vás vyžadují zadání e-mailové adresy, a všechny používají `<input type="text">`. Uvidíte textové pole, napíšete do něj svou e-mailovou adresu a je to. Pak se objevuje HTML5, které definuje `type="email"`. Mají s tím prohlížeče problém? Nemají. Každý prohlížeč na planetě se k neznámému atributu `type` chová jako k `type="text"` – dokonce i IE6. Takže můžete své webové formuláře aktualizovat na `type="email"` klidně hned teď.

Co to znamená, že prohlížeč podporuje `type="email"`? Můžete to znamenat spoustu věcí. Specifikace HTML5 pro nové vstupní typy nevyžaduje žádné konkrétní uživatelské rozhraní. Většina prohlížečů jako Safari, Chrome, Opera a Firefox ho prostě zobrazí jako textové pole – úplně stejně jako `type="text"` –, takže vaši uživatelé nepoznají rozdíl (dokud se nepokusí vyplněný formulář odeslat).

A pak je tady iPhone.

iPhone nemá fyzickou klávesnici. Veškeré psaní se provádí poklepem na virtuální klávesnici na obrazovce, která se objevuje, jen když je to potřeba, jako například když dáte fokus do formulářového pole na webové stránce. Apple v prohlížeči pro iPhone vymyslel chytrou věc. Díky ní prohlížeč rozpoznává některé z nových vstupních typů HTML5 a *dynamicky mění klávesnici na obrazovce*, aby ji optimalizoval pro konkrétní vstupní typ.

Například e-mailové adresy jsou text, že? To sice ano, ale je to zvláštní typ textu. Například v podstatě všechny e-mailové adresy budou obsahovat znak `@` a aspoň jednu tečku (.), ale těžko budou obsahovat mezery. Takže když používáte iPhone a dáte fokus do pole s prvkem `<input`

`type="email">`, vyskočí vám na obrazovce klávesnice, která bude obsahovat mezerník menší než obvykle, a navíc speciální klávesy pro `@` a tečku.

Vyzkoušejte si `type="email"` sami[6].

Pro shrnutí: okamžité převedení všech formulářových polí pro e-mailové adresy na `type="email"` nepřináší žádné nevýhody. V podstatě nikdo si toho nevšimne, s výjimkou uživatelů iPhonů, kteří si toho ale nejspíš nevšimnou taky. Ale ti, kdo si toho všimnou, se tiše usmějí a poděkují vám, že jste jim práci na webu zase o něco usnadnili.

9.5 Webové adresy

Webové adresy – kterým tvůrci standardů říkají URL, kromě pár pedantů, kteří jim říkají URI – jsou dalším typem zvláštního textu. Syntax webové adresy omezují příslušné internetové standardy. Pokud váš někdo požádá o to, abyste do formuláře zadali webovou adresu, bude očekávat něco ve stylu `http://www.google.com/`, a nikoliv „Dlouhá 31“. Ve webových adresách se běžně používají lomítka – dokonce i domovská stránka Googlu má tři. Běžné jsou i tečky, ale mezery jsou zakázané. A každá webová adresa obsahuje doménovou koncovku jako „.com“ nebo „.org“.

← Hle... (slavnostní fanfáru, prosím)... `<input type="url">`. Na iPhonu vypadá takhle.

Vyzkoušejte si `type="type"` sami[7].

iPhone upravil virtuální klávesnici stejně jako u e-mailových adres, ale tentokrát ji optimalizoval pro adresy webové. Mezerník úplně nahradily tři virtuální klávesy: tečka, lomítko a tlačítko „.com“. (Tlačítko „.com“ můžete přidržet pro výběr z dalších běžných koncovek jako „.org“ nebo „.net“.)

6 http://diveintohtml5.info/examples/input-type-email.html
7 http://diveintohtml5.info/examples/input-type-url.html

Většina moderních prohlížečů zobrazuje `type="url"` jednoduše jako obyčejné textové pole, takže si uživatelé rozdílu ani nevšimnou, dokud neodešlou formulář. Prohlížeče bez podpory HTML5 se budou k `type="url"` chovat úplně stejně jako k `type="text"`, takže vám použití nového vstupního typu pro webové adresy nepřinese žádné nevýhody.

9.6 Čísla jako číselníky

Další jsou na řadě čísla. Požadování čísla je složitější než požadování e-mailové nebo webové adresy. Čísla jsou totiž mnohem komplikovanější, než si asi myslíte. Rychle: zvolte si číslo. -1? Ne, myslel jsem číslo mezi 1 a 10. 7½? Ale ne, ne zlomek. π? No tohle od vás bylo iracionální. Chci tím říct, že málokdy požadujete „jakékoliv číslo". Pravděpodobnější je, že budete požadovat číslo z určitého rozmezí. A možná i v rámci tohoto rozmezí vás budou zajímat pouze některá čísla – třeba celá čísla a ne zlomky nebo desetinná čísla, nebo něco bizarnějšího, jako třeba jen čísla dělitelná 10. HTML5 vám v tom pomůže.

Vyberte si (téměř) libovolné číslo ↷

```
<input type="number"
     min="0"
     max="10"
     step="2"
     value="6">
```

Podívejme se na jednotlivé atributy. (Pokud chcete, můžete u toho sledovat příklad z praxe[8].)

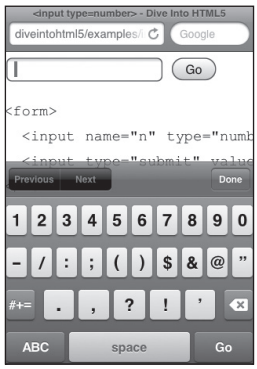
- `type="number"` znamená, že se jedná o číselné pole.
- `min="0"` určuje minimální přijatelnou hodnotu pole.
- `max="10"` určuje maximální přijatelnou hodnotu.
- `step="2"` v kombinaci s hodnotou `min` definuje přijatelná čísla v daném rozmezí: `0, 2, 4` a tak dále, až do hodnoty `max`.
- `value="6"` je výchozí hodnota. Měla by vám být povědomá. Je to stejné jméno atributu, které jste vždy používali pro určení hodnot formulářových polí. (Zmiňuju to proto, abych zdůraznil, že HTML5 staví na předchozích verzích HTML. Nemusíte se znovu učit, jak dělat něco, co už děláte.)

8 http://diveintohtml5.info/examples/input-type-number-min-max-step.html

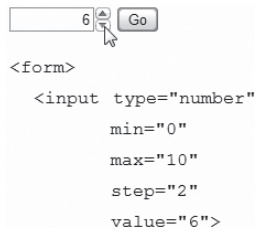
Takto vypadá kód číselného pole. Myslete na to, že všechny tyto atributy jsou volitelné. Pokud chcete určit minimální hodnotu, ale ne maximální, můžete použít atribut `min` bez atributu `max`. Výchozí hodnota kroku je 1 a atribut `step` můžete vynechat, pokud nepotřebujete jinou hodnotu kroku. Pokud nechcete žádnou výchozí hodnotu, může atribut `value` tvořit prázdný řetězec nebo může být vynechán úplně.

Ale tím HTML5 nekončí. Za stejně výhodnou cenu – tj. zadarmo – dostanete i tyhle užitečné javascriptové metody:

- `input.stepUp(n)` zvýší hodnotu pole o n.
- `input.stepDown(n)` sníží hodnotu pole o n.
- `input.valueAsNumber` vrátí aktuální hodnotu jako číslo s plovoucí desetinnou čárkou. (Vlastnost `input.value` je vždycky řetězec.)



Máte problém si to představit? Vytvoření konkrétního rozhraní pro zadávání čísel je na prohlížeči a různí dodavatelé prohlížečů realizovali podporu různými způsoby. Na iPhonu, kde je zadávání z principu obtížné, prohlížeč opět optimalizuje virtuální klávesnici pro zadávání čísel.



Ve verzi Opery pro stolní počítače je stejné pole `type="number"` zobrazováno jako číselník s malými šipkami nahoru a dolů, které po kliknutí mění hodnotu.

Opera respektuje atributy `min`, `max` a `step`, takže vždycky skončíte s přijatelnou číselnou hodnotou. Pokud nastavíte hodnotu na maximum, šipka v seznamu zšedne a nebude už možné na ni kliknout.

Jako u všech vstupních typů, o kterých jsem v této kapitole zatím mluvil, se prohlížeče, které nepodporují `type="number"`, k němu budou chovat jako k `type="text"`. V poli se zobrazí výchozí hodnota (protože je uložená v atributu), ale ostatní atributy jako `min` a `max` budou ignorovány. Můžete je implementovat sami, anebo můžete použít javascriptový framework, který číselníky už obsahuje. Pouze nejdřív zkontrolujte nativní podporu HTML5, a to takhle:

```
if (!Modernizr.inputtypes.number) {
    // nativní podpora pro type=number fields není k dispozici
    // můžete zkusit Dojo nebo jiný javascriptový rám
}
```

9.7 Čísla jako posuvníky

Číselníky nejsou jediným způsobem jak zadávat čísla. Asi už jste viděli „posuvníky“, které vypadaly takto:

Vyzkoušejte si `type="range"` sami[9].

Posuvníky teď můžete ve svých webových formulářích používat také. Kód se velice podobá číselníkům:

Je to skoro totéž ↷

```
<input type="range"
    min="0"
    max="10"
    step="2"
    value="6">
```

Všechny použitelné atributy jsou stejné jako u `type="number"`: `min`, `max`, `step`, `value` a znamenají úplně totéž. Jediný rozdíl spočívá v uživatelském rozhraní. Místo pole, do kterého se píše, by prohlížeče měly zobrazit `type="range"` jako posuvník. Safari, Chrome, Opera, Internet Explorer 10 a iPhone s iOS 5 to všechny zvládají. (iPhone se starší verzí iOS bohužel místo posuvníku zobrazí obyčejné textové pole. Dokonce ani neoptimalizuje klávesnici na obrazovce pro zadávání čísel.) Všechny ostatní prohlížeče se k poli chovají jednoduše jako k `type="text"`, takže není důvod, proč byste ho nemohli hned začít používat.

9 http://diveintohtml5.info/examples/input-type-range.html

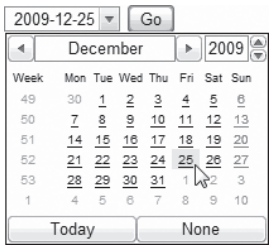
9.8 Výběr data

HTML 4 neobsahovalo vstupní prvek pro výběr data. Javascriptové frameworky (Dojo, jQuery UI, YUI, Closure Library) se to rozhodly napravit, ale každé z těchto řešení pochopitelně vyžaduje „koupit si podíl“ v daném frameworku.

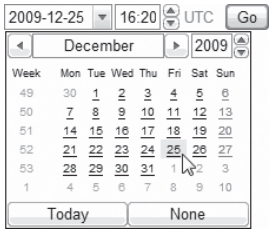
HTML5 konečně přichází se způsobem, jak používat nativní nástroj pro výběr data bez toho, abyste si museli sami psát skript. Těch způsobů je dokonce šest: datum, měsíc, týden, čas, datum + čas a datum + čas + časové pásmo.

Podpora je zatím… vzácná.

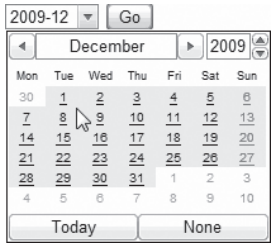
Podpora výběru data							
Vstupní typ	**IE**	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Jiné prohlížeče**
type="date"	·	·	5.0+	20.0+	9.0+	5.0+	·
type="datetime"	·	·	5.0+	·	9.0+	5.0+	·
type="datetime-local"	·	·	5.0+	·	9.0+	5.0+	·
type="month"	·	·	5.0+	·	9.0+	5.0+	·
type="week"	·	·	5.0+	·	9.0+	5.0+	·
type="time"	·	·	5.0+	20.0+	9.0+	5.0+	·



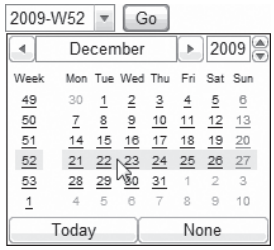
← Opera zobrazuje `<input type="date">` takto



← Pokud chcete kromě data zobrazovat i čas, Opera podporuje i `<input type="datetime">`



← Pokud potřebujete jenom měsíc + rok (například datum konce platnosti kreditní karty), Opera dokáže zobrazit `<input type="month">`:



← Méně obvyklé, ale také možné je zvolit určitý týden roku pomocí `<input type="week">`:

A konečně si můžete zvolit čas pomocí `<input type="time">`: 

Je pravděpodobné, že nakonec budou tyto vstupní typy podporovat i další prohlížeče. Ale stejně jako `type="email"` a další vstupní typy se tato formulářová pole v prohlížečích, které nerozeznávají `type="date"` a další varianty, zobrazí jako obyčejná textová pole. Pokud chcete, můžete `<input type="date">` a jeho kamarády používat, potěšit tím uživatele Opery a iOS a čekat, až se přidají i ostatní prohlížeče. Praktičtější pak je použít `<input type="date">`, zjistit, zda prohlížeč nabízí nativní podporu výběru data, a použít některé z náhradních skriptovaných řešení (Dojo, jQuery UI, YUI, Closure Library nebo další).

Výběr data pomocí náhradního řešení ↷

```
<form>
    <input type="date">
</form>
...
<script>
    var i = document.createElement("input");
    i.setAttribute("type", "date");
    if (i.type == "text") {
        // Nativní podpora pro výběr data není k dispozici :(
        // Pro její vytvoření použijte Dojo/jQueryUI/YUI/Closure
        // a poté dynamicky nahraďte prvek <input>.
    }
</script>
```

9.9 Vyhledávací pole

Tohle nám dá trochu zabrat. Myšlenka je to jednoduchá, ale její realizace je potřeba trochu vysvětlit. Mluvím o…

Vyhledávání. A to nejen o vyhledávání na Googlu nebo Yahoo (i když o tom taky). Představte si vyhledávací pole na libovolné stránce. Amazon má vyhledávací pole. Newegg ho má taky. Stejně jako většina blogů. Jaký mají kód? `<input type="text">`, jako všechna ostatní textová pole na webu. Pojďme s tím něco udělat.

```
<form>
 <input name="q" type="search">        ← Vyhledávací pole nové éry
 <input type="submit" value="Find">
</form>
```


```
<form>
  <input type="search">
  <input type="submit" v
</form>
```

Vyzkoušejte `<input type="search">` ve svém prohlížeči. V některých prohlížečích nenajdete žádný rozdíl mezi novým vstupním typem a obyčejným textovým polem. Ale pokud používáte Safari na Mac OS X, bude to vypadat takhle:


```
<form>
  <input type="search">
  <input type="submit" v
</form>
```

Vidíte ten rozdíl? Vstupní pole má zaoblené rohy! Ano, chápu, máte potíže potlačit vzrušení. Ale počkejte, to není všechno! Když začnete do pole `type="search"` psát, Safari na pravou stranu pole vloží malé tlačítko „x“. Když na něj kliknete, obsah pole se vymaže. (Google Chrome, jehož technologie má se Safari hodně společného, to umí taky.) Obě tyto drobné vychytávky jsou tu proto, aby se vstupní typ podobal nativním vyhledávacím polím v iTunes a dalších klientských aplikacích pro Mac OS X.

Apple.com kdysi používal `<input type="search">` pro vyhledávací pole na své stránce, aby více připomínala Mac. Ale není to omezeno pouze na Mac. Je to jen kód, který libovolný prohlížeč na libovolné platformě může zobrazit podle zvyklostí dané platformy. Stejně jako je tomu u dalších nových vstupních typů, prohlížeče, které `type="search"` nerozpoznávají, se k němu budou chovat jako k `type="text"`. Není tedy žádný důvod, proč byste nemohli začít používat `type="search"` pro vyhledávací pole ještě dnes.

PROFESOR ZNAČKA ŘÍKÁ

Starší verze Safari ve výchozím nastavení nepoužívají na pole `<input type="search">` ani nejzákladnější kaskádové styly. Pokud chcete donutit Safari k tomu, aby se k vašemu

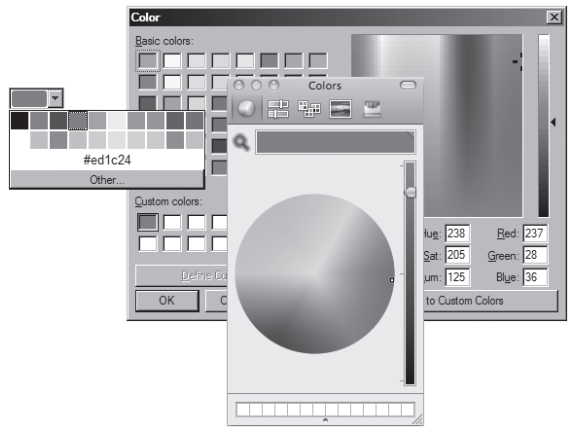
vyhledávacímu poli chovalo jako k normálnímu textovému poli (abyste mohli použít své vlastní kaskádové styly), přidejte do šablony stylů toto pravidlo:

```
input[type="search"] {
    -webkit-appearance: textfield;
}
```

Díky Johnu Leinovi za to, že mě tento trik naučil.

9.10 Výběr barev

HTML5 také definuje vstupní typ `<input type="color">`, který vám umožní vybrat si barvu a vrátí ji jako hexadecimální kód. ~~Žádný prohlížeč ho ale ještě nepodporuje, což je škoda, protože nástroj pro výběr barev v Mac OS se mi vždycky líbil~~. Dobrá zpráva pro všechny! Opera 11 nyní podporuje `type=color`. Na Macu a ve Windows se zobrazí nativní nástroj platformy pro výběr barvy. V Linuxu se zobrazí základní výběr barev. Na všech platformách tento vstupní prvek vrátí šestimístnou hexadecimální barvu v RGB, vhodnou pro rámování nebo k použití všude, kde se přijímají barvy CSS.



Vyzkoušejte si `type="color"` sami[10].

Díky Patricku Laukovi a Chrisu Millsovi za poskytnutí licence na tento obrázek, abych ho mohl použít v této knížce. Měli byste si přečíst jejich článek o nových funkcích formulářů v Opeře 11[11].

10 http://diveintohtml5.info/examples/input-type-color.html
11 https://dev.opera.com/articles/view/new-form-features-in-html5/

9.11 Validace formulářů

Podpora validace formulářů						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
10.0+	4.0+	5.0+	10.0+	9.0+	·	·

V této kapitole jsem mluvil o nových vstupních typech a nových funkcích, jako je autofokus formulářových polí, ale nezmínil jsem něco, co je na formulářích HTML5 možná nejzajímavější: automatická validace při vkládání. Vezměte si běžný problém zadávání e-mailové adresy do webového formuláře. Asi máte nějakou validaci na straně klienta v JavaScriptu, následovanou validací na straně serveru v PHP nebo Pythonu, nebo nějakém jiném skriptovacím jazyce. HTML5 nemůže nikdy nehradit validaci na straně serveru, ale jednoho dne může nahradit validaci na straně klienta.

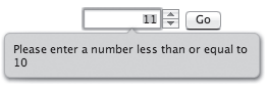
S validací e-mailových adres v JavaScriptu jsou dva velké problémy:

- Překvapivý počet vašich návštěvníků (tak kolem 10 %) nebude mít JavaScript zapnutý.
- Popletete to.

Myslím to vážně, popletete to. Rozhodnout, jestli je náhodný řetězec znaků platná e-mailová adresa, je neuvěřitelně komplikované. A čím víc se tím zabýváte, tím je to komplikovanější. Říkal jsem už, že je to opravdu, ale opravdu komplikované? Nebylo by jednodušší nechat všechnu tu dřinu na prohlížeči?

← Většina moderních prohlížečů validuje `type="email"`

Tento snímek obrazovky je z Opery 11, i když tato funkce byla k dispozici už od verze 9. Podobnou funkci nabízejí i Firefox 4 a Chrome 10. Jediné, co je třeba udělat, je nastavit atribut `type` na `"email"`. Když se uživatel pokusí odeslat formulář s polem `<input type="email">`, prohlížeč automaticky nabídne validaci e-mailu podle RFC, i když je skriptování vypnuté.



HTML5 nabízí také validaci webových adres vkládaných do polí `<input type="url">` a čísel v polích `<input type="number">`. Validace čísel dokonce bere ohled na atributy `min` a `max`, takže když zadáte příliš velké číslo, prohlížeč vám nedovolí formulář odeslat.

Pro aktivaci validace formulářů HTML5 není potřeba žádný kód, protože je ve výchozím nastavení zapnuta. Pro její vypnutí použijte atribut `novalidate`.

Nevaliduj mě, kámo ↷

```
<form novalidate>
    <input type="email" id="addr">
    <input type="submit" value="Subscribe">
</form>
```

9.12 Povinná pole

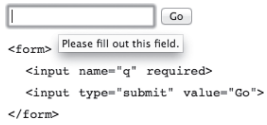
Podpora `<input required>`						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
10.0+	4.0+	·	10.0+	9.0+	·	·

Validace formulářů HTML5 se neomezuje jen na typ jednotlivých polí. Můžete také určit, že jsou některá z polí povinná. Povinná pole musejí před odesláním formuláře obsahovat nějakou hodnotu.

Kód povinných polí je maximálně jednoduchý ↷

```
<form>
    <input id="q" required>
    <input type="submit" value="Search">
</form>
```

Vyzkoušejte `<input required>` ve svém prohlížeči[12]. Prohlížeče můžou k povinným polím dodávat další informace. Například když podržíte kurzor nad požadovaným polem, objeví se v Mozille Firefoxu toto vyskakovací okno:



Navíc pokud se pokusíte formulář odeslat bez vyplnění povinné hodnoty, ve Firefoxu vyskočí informační panel, který vám oznámí, že je pole povinné a nemůže zůstat nevyplněné.

12 http://diveintohtml5.info/examples/input-required.html

9.13 K dalšímu čtení

Specifikace a standardy:
- typy `<input>`[13]
- atribut `<input placeholder>`[14]
- atribut `<input autofocus>`[15]
- atribut `<form novalidate>`[16]
- atribut `<input required>`[17]

Javascriptové knihovny:
- Modernizr[18], detekční knihovna HTML5

Užitečné články:
- Forward Thinking Form Validation[19]
- New form features in Opera 11[20]
- Mozilla Developer Center: Forms in HTML[21]
- HTML5 Form Validation[22]
- Internet Explorer 10 Guide for Developers: HTML5 Forms[23]

13 https://html.spec.whatwg.org/multipage/forms.html#states-of-the-type-attribute
14 https://html.spec.whatwg.org/multipage/forms.html#common-input-element-attributes
15 https://html.spec.whatwg.org/multipage/forms.html#association-of-controls-and-forms
16 https://html.spec.whatwg.org/multipage/forms.html#association-of-controls-and-forms
17 https://html.spec.whatwg.org/multipage/forms.html#common-input-element-attributes
18 http://modernizr.com/
19 http://alistapart.com/article/forward-thinking-form-validation
20 https://dev.opera.com/articles/view/new-form-features-in-html5/
21 https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms_in_HTML
22 https://blog.mozilla.org/webdev/2011/03/14/html5-form-validation-on-sumo/
23 https://msdn.microsoft.com/library/hh673546

10. „Distribuovaná“, „rozšiřitelnost“ a další působivá slova

10. „Distribuovaná“, „rozšiřitelnost“ a další působivá slova – 223

10.1 Začínáme

HTML5 obsahuje víc než 100 prvků. Některé z nich jsou čistě sémantické, zatímco jiné jsou pouhými kontejnery pro skriptovaná API. V průběhu historie HTML se tvůrci standardů dohadovali, které prvky zahrnout do jazyka HTML. Mělo by HTML obsahovat prvek `<figure>`? Prvek `<person>`? A co `<rant>` (česky stížnost, tiráda)? Dělají se rozhodnutí, píší se specifikace, implementují realizace a web se pomalu, ale jistě posouvá vpřed.

Je samozřejmé, že se HTML nezavděčí každému. To se nepovede žádnému standardu. Některé nápady se do něj prostě nedostanou. Například v HTML5 nemáme žádný prvek `<person>` (a ani prvek `<rant>`!) Nic vám nezabrání v tom, abyste na své stránce použili prvek `<person>`, ale nebude se validovat, nebude konzistentně fungovat v různých prohlížečích a může se ocitnout v konfliktu s budoucí specifikací HTML, pokud ho budeme chtít přidat později.

Dobrá, tak když vytváření vlastních prvků není řešení, co má tedy autor webu se sémantickými sklony dělat? Byly zde pokusy rozšířit předchozí verze HTML. Nejoblíbenější metodou jsou mikroformáty, které používají atributy `class` a `rel` v HTML. Další možností je RDFa (systém popisu zdrojů v atributech), který byl původně navržen pro použití v XHTML, ale pak byl přidán i do HTML.

Mikroformáty i RDFa mají své výhody a nevýhody. Ač každá zcela jiným způsobem, obě metody usilují o stejný cíl: rozšířit webové stránky o sémantiku, která není součástí základního jazyka HTML. Nechci z této kapitoly udělat hádku o formáty. (Na to bych rozhodně potřeboval prvek `<rant>`!) Místo toho se chci zaměřit na třetí možnost, která byla vyvinuta na základě poučení získaného z mikroformátů a RDFa a navržena tak, aby se dala integrovat do samotného HTML5: mikrodata.

10.2 Co jsou mikrodata?

Každé slovo v následující větě je důležité, proto dávejte pozor.

PROFESOR ZNAČKA ŘÍKÁ

Mikrodata přidávají do DOMu z uživatelských slovníků dvojice název-hodnota omezené na určitý obor platnosti.

Ale co to znamená? Začněme od prostředka. Mikrodata se soustředí kolem uživatelských slovníků. Představte si „sadu všech prvků HTML5“ jako jeden slovník. Tento slovník obsahuje

prvky pro reprezentaci sekce nebo článku, ale neobsahuje prvky pro reprezentaci osoby nebo události. Pokud chcete na webové stránce reprezentovat osobu, musíte definovat svůj vlastní slovník. Mikrodata vám to umožňují. Definovat slovník mikrodat a začít vkládat uživatelské vlastnosti na stránku může každý.

Další věc, kterou potřebujete o mikrodatech vědět, je, že pracují s dvojicemi název-hodnota. Každý slovník mikrodat definuje sadu pojmenovaných vlastností. Například slovník Osoba může definovat vlastnosti jako `name` nebo `photo`. Pro vložení specifické mikrodatové vlastnosti na stránku musíte napsat na určitém místě název vlastnosti. Pravidla extrahování hodnot mikrodatových vlastností závisí na tom, kde deklarujete název vlastnosti (více se o tom dozvíte v další podkapitole.)

Kromě pojmenovaných vlastností mikrodata také hodně těží z konceptu „obor platnosti“ (scoping). Nejjednodušší způsob, jak si udělat obrázek o oboru platnosti mikrodat, je představit si přirozený vztah prvků v DOMu jako rodičů a potomků. Prvek `<html>` obvykle obsahuje dva potomky, `<head>` a `<body>`. Prvek `<body>` obvykle obsahuje vícero potomků, z nichž každý má své vlastní potomky-prvky. Vaše stránka například může obsahovat prvek `<h1>` uvnitř prvku `<hgroup>` uvnitř prvku `<header>` uvnitř prvku `<body>`. Datová tabulka může obsahovat `<td>` uvnitř `<tr>` uvnitř `<table>` (uvnitř `<body>`). Mikrodata využívají hierarchickou strukturu DOMu, aby vyjádřily, že „všechny vlastnosti v rámci *tohoto* prvku jsou převzaty z *tohoto* slovníku.“ To vám umožňuje používat na jedné stránce více než jeden slovník mikrodat. Můžete dokonce nořit slovníky mikrodat do jiných slovníků, a to tím, že opět využijete přirozenou strukturu DOMu. (Během této kapitoly vám ukážu několik příkladů vnořených slovníků.)

Už jsem zmínil DOM, ale chci vám k němu povědět něco víc. Mikrodata přidávají sémantiku k datům, *která se už na vaší webové stránce zobrazují*. Mikrodata nejsou navržena jako samostatný datový formát. Jsou doplňkem HTML. Jak uvidíte v další podkapitole, mikrodata fungují nejlíp, když používáte správně HTML, ale slovník HTML vám nestačí. Mikrodata jsou skvělá pro vylepšování sémantiky dat obsažených v DOMu. Pokud data, ke kterým přidáváte sémantiku, nejsou v DOMu, měli byste se zastavit a zamyslet se, zda jsou mikrodata pro vás tím správným řešením.

Dává vám teď ta věta větší smysl? „Mikrodata přidávají do DOMu z uživatelských slovníků dvojice název-hodnota omezené na určitý obor platnosti.“ Doufám, že ano. Podívejme se, jak to funguje v praxi.

10.3 Datový model pro mikrodata

Vytvořit si svůj slovník mikrodat je snadné. Nejdříve potřebujete jmenný prostor, což je prostě URL. URL jmenného prostoru může klidně odkazovat na funkční webovou stránku, ale není

to úplně nezbytné. Řekněme, že chci vytvořit slovník mikrodat, který popisuje osobu. Jestliže jsem majitelem domény `data-vocabulary.org`, použiju jako jmenný prostor pro svůj slovník mikrodat URL `http://data-vocabulary.org/Person`. To je snadný způsob, jak vytvořit globálně jedinečný identifikátor: vybrat si URL v doméně, kterou ovládáte.

V tomto slovníku definuji několik pojmenovaných vlastností. Začněme třemi základními:

- `name` (vaše celé jméno),
- `photo` (odkaz na vaši fotku),
- `url` (odkaz na stránku spojenou s vámi, jako je blog nebo profil na Googlu).

Některé z těchto vlastností jsou URL, jiné prostý text. Každá z nich se dá přirozeně použít v kódu, a to i bez ohledu na mikrodata, slovníky a podobně. Představte si, že máte profilovou stránku nebo stránku „O mně“. Vaše jméno je pravděpodobně nadpis, jako prvek `<h1>`. Vaše fotka je pravděpodobně prvek `<img>`, protože chcete, aby ji lidi viděli. A všechny URL spojené s vaším profilem jsou asi hypertextové odkazy, protože chcete, aby na ně lidi mohli klikat. Pro účely tohoto příkladu také celý profil vložíme do prvku `<section>`, aby se oddělil od zbytku obsahu stránky. Takže:

↶ Všechno se to točí kolem mě

```
<section>
    <h1>Mark Pilgrim</h1>
    <p><img src="http://www.example.com/photo.jpg" alt="[jak se
    směju]"></p>
    <p><a href="http://diveintomark.org/">weblog</a></p>
</section>
```

Datovým modelem pro mikrodata jsou dvojice název-hodnota. Název mikrodatové vlastnosti (v tomto příkladu jsou to `name`, `photo` nebo `url`) je vždy deklarován v prvku HTML. Odpovídající hodnota vlastnosti se pak přebírá z DOMu prvku. U většiny prvků HTML je hodnotou vlastnosti jednoduše textový obsah daného prvku. Ale existuje pár výjimek.

Odkud se vzaly hodnoty mikrodatových vlastností?

Prvek	Hodnota
`<meta>`	atribut `content`
`<audio>`	atribut `src`
`<embed>`	
`<iframe>`	
`<img>`	
`<source>`	
`<video>`	
`<a>`	atribut `href`
`<area>`	
`<link>`	
`<object>`	atribut `data`
`<time>`	atribut `datetime`
všechny ostatní prvky	textový obsah

„Přidání mikrodat“ na stránku spočívá v přidání pár atributů k prvkům HTML, které už máte. Nejdříve musíte vždy deklarovat používaný slovník mikrodat, a to přidáním atributu `itemtype`. Druhou věcí, kterou musíte udělat, je deklarovat obor platnosti slovníku pomocí atributu `itemscope`. V tomto příkladu se všechna data, kterým chceme přidat sémantiku, nacházejí v prvku `<section>`, takže atributy `itemtype` a `itemscope` deklarujeme v prvku `<section>`.

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

Vaše jméno je prvním kouskem dat v prvku `<section>` a je vloženo v prvku `<h1>`. Prvek `<h1>` se v datovém modelu pro mikrodata HTML5 nezpracovává žádným zvláštním způsobem, spadá tedy do skupiny „všechny ostatní prvky“, takže hodnotou mikrodatové vlastnosti je jednoduše textový obsah daného prvku. (Stejně dobře by to fungovalo, kdyby bylo vaše jméno vloženo v prvku `<p>`, `<div>` nebo `<span>`.)

```
<h1 itemprop="name">Mark Pilgrim</h1>
```

Normálním jazykem se tím říká: „Tady je vlastnost `name` ze slovníku `http://data-vocabulary.org/Person` a její hodnotou je `Mark Pilgrim`.“

Následuje vlastnost `photo`. Měla by to být URL. Podle datového modelu pro mikrodata HTML5 je „hodnotou“ prvku `<img>` jeho atribut `src`. No a URL vaší profilové fotky už v atributu `<img src>` je. Jediné, co potřebujete udělat, je deklarovat, že prvek `<img>` je onou vlastností `photo`.

```
<p><img itemprop="photo"
    src="http://www.example.com/photo.jpg"
    alt="[jak se směju]"></p>
```

Normálním jazykem se tím říká: „Tady je vlastnost `photo` ze slovníku `http://data-vocabulary.org/Person` a její hodnotou je `http://www.example.com/photo.jpg`.“

A konečně vlastnost `url` je také URL. Podle datového modelu pro mikrodata HTML5 je „hodnotou“ prvku `<a>` jeho atribut `href`. Opět se to krásně zapadá do kódu, který už máte. Je potřeba už jen oznámit, že prvek `<a>` je tou vlastností `url`:

```
<a itemprop="url" href="http://diveintomark.org/">weblog</a>
```

Normálním jazykem se tím říká: „Tady je vlastnost `url` ze slovníku `http://data-vocabulary.org/Person` a její hodnotou je `http://diveintomark.org/`.“

Pokud váš kód vypadá trochu jinak, samozřejmě to není problém. Mikrodatové vlastnosti a hodnoty můžete přidat do jakéhokoliv kódu HTML, i do těch děsivých stránek z 20. století, které jsou formátovány pomocí tabulek a u kterých litujete, že jste kdy souhlasili, že je budete spravovat. I když tento typ kódu nedoporučuji, je stále běžný a vy do něj můžete přidat mikrodata.

↶ Proboha vás prosím, nedělejte to

```
<TABLE>
    <TR><TD>Jméno<TD>Mark Pilgrim
    <TR><TD>Odkaz<TD>
    <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
</TABLE>
```

Pro stanovení vlastnosti `name` přidejte atribut `itemprop` do buňky tabulky, která obsahuje jméno. Buňky tabulky se při přidělování hodnoty neřídí žádnými zvláštními pravidly, takže dostanou výchozí hodnotu „mikrodatovou vlastností je textový obsah.“

```
<TR><TD>Name<TD itemprop="name">Mark Pilgrim
```

Přidání vlastnosti `url` vypadá o něco složitěji. Tento kód nepoužívá prvek `<a>` správně. Místo toho, aby dal cíl odkazu do atributu `href`, nemá v něm nic užitečného a v atributu `onclick` používá JavaScript pro vyvolání funkce (není ukázaná), která extrahuje adresu URL a přechází na ni. Abychom si dokreslili dobu, představme si ještě, že ta funkce otvírá odkaz ve vyskakovacím okénku bez posuvných lišt. No nebyla s Internetem v minulém století legrace?

I tak to můžete proměnit v mikrodatovou vlastnost, jen musíte být trochu kreativní. Přímé použití prvku `<a>` nepřichází v úvahu. Cíl odkazu není v atributu `href` a neexistuje žádný způsob, jak obejít pravidlo „v prvku `<a>` hledejte hodnotu mikrodatové vlastnosti v atributu `href`“. Ale *můžete* celý ten zmatek obalit jiným prvkem, který použijete k přidání mikrodatové vlastnosti `url`.

↶ Tohle získáte podkopáváním HTML

```
<TABLE itemscope itemtype="http://data-vocabulary.org/Person">
    <TR><TD>Jméno<TD>Mark Pilgrim
    <TR><TD>Odkaz<TD>
        <span itemprop="url">
            <A href=# onclick=goExternalLink()>http://diveintomark.
            org/</A>
        </span>
</TABLE>
```

Protože prvek `<span>` není zpracováván žádným zvláštním způsobem, je použito výchozí pravidlo „mikrodatovou vlastností je textový obsah.“ „Textový obsah“ neznamená „veškerý kód uvnitř tohoto prvku“ (jak by tomu bylo třeba u vlastnosti DOM `innerHTML`). Znamená to „pouze text, madam.“ V tomto případě, tedy u `http://diveintomark.org/`, textový obsah prvku `<a>` uvnitř prvku `<span>`.

Abychom to shrnuli: mikrodatové vlastnosti můžete přidat k libovolnému kódu. Pokud používáte HTML správně, budou se vám mikrodata přidávat snadněji, než když máte v kódu guláš, ale udělat to jde vždy.

10.4 Označování osob

Mimochodem, příklady pro začátek v předchozím oddíle nebyly úplně vymyšlené. Opravdu existuje slovník mikrodat pro označování informací o lidech a opravdu je to tak snadné. Podívejme se na to zblízka.

Nejjednodušším způsobem, jak integrovat mikrodata do osobní webové stránky, je dát je na stránku „O mně.“ Máte stránku „O mně“, že ano? Pokud ne, můžete sledovat, jak budu rozšiřovat tuto vzorovou stránku „O mně“[1] o další sémantiku. Konečný výsledek najdete zde: person-plus-microdata.html[2].

Podívejme se nejdřív na obyčejný kód předtím, než se k němu přidají mikrodatové vlastnosti:

```
<section>
    <img width="204" height="250"
           src="http://diveintohtml5.info/examples/2000_05_mark.jpg"
           alt="[Mark Pilgrim, okolo r. 2000]">

    <h1>Kontaktní údaje</h1>
    <dl>
        <dt>Jméno</dt>
        <dd>Mark Pilgrim</dd>

        <dt>Pracovní pozice</dt>
        <dd>Developer advocate pro Google, Inc.</dd>

        <dt>Adresa</dt>
        <dd>
            100 Main Street<br>
            Anytown, PA 19999<br>
            USA
        </dd>
    </dl>
    <h1>Moje digitální stopy</h1>
    <ul>
        <li><a href="http://diveintomark.org/">weblog</a></li>
        <li><a href="http://www.google.com/profiles/pilgrim">profil
        na Google</a></li>
        <li><a href="http://www.reddit.com/user/MarkPilgrim">profil
        na Reddit.com</a></li>
        <li><a href="http://www.twitter.com/diveintomark">Twitter
        </a></li>
    </ul>
</section>
```

1 http://diveintohtml5.info/examples/person.html
2 http://diveintohtml5.info/examples/person-plus-microdata.html

První věcí, kterou musíte vždy udělat, je deklarovat používaný slovník a obor platnosti vlastností, které chcete přidat. To můžete udělat přidáním atributů `itemtype` a `itemscope` nejzevnějšímu prvku, jenž obsahuje ostatní prvky, které obsahují samotné údaje. V tomto případě je to prvek `<section>`.

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

Nyní můžete začít definovat mikrodatové vlastnosti ze slovníku `http://data-vocabulary.org/Person`. Ale jaké jsou to vlastnosti? Jejich seznam můžete najít, když v prohlížeči přejdete na data-vocabulary.org/Person. Specifikace mikrodat to sice nevyžaduje, ale řekl bych, že se jedná o „nejlepší zvyklost“. Pokud chcete, aby vývojáři váš slovník mikrodat skutečně používali, musíte ho zdokumentovat. A kam nejlépe tuto dokumentaci umístit než na samotnou URL se slovníkem?

Slovník Osoba	
Vlastnost	**Popis**
`Name`	Jméno
`nickname`	Přezdívka
`Photo`	Odkaz na fotografii
`Title`	Název pracovní pozice (např. „Finanční manažer“)
`Role`	Náplň práce (např. „Účetní“)
`url`	Odkaz na webovou stránku, např. domovskou stránku osoby
`affiliation`	Jméno organizace, se kterou je osoba spojena (např. zaměstnavatel)
`friend`	Identifikuje sociální vztah mezi popisovanou osobou a další osobou
`contact`	Identifikuje sociální vztah mezi popisovanou osobou a další osobou
`acquaintance`	Identifikuje sociální vztah mezi popisovanou osobou a další osobou
`address`	Adresa osoby. Může obsahovat podvlastnosti `street-address`, `locality`, `region`, `postal-code` a `country-name`.

První věcí na této vzorové stránce „O mně“ je moje fotka. Přirozeně ji v kódu představuje prvek `<img>`. Vše, co potřebujeme pro deklaraci toho, že tento prvek `<img>` je můj profilový obrázek, je přidat prvku `<img>` atribut `itemprop="photo"`.

```
<img itemprop="photo" width="204" height="250"
    src="http://diveintohtml5.info/examples/2000_05_mark.jpg"
    alt="[Mark Pilgrim, okolo r. 2000]">
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

Kde je hodnota mikrodatové vlastnosti? Je už obsažená v atributu `src`. Pokud si vzpomínáte na datový model pro mikrodata HTML5, „hodnotou“ prvku `<img>` je jeho atribut `src`. Každý prvek `<img>` má atribut `src` – jinak by se obrázek správně nezobrazoval – a atribut `src` je vždy URL. Vidíte? Když používáte správně HTML, mikrodata jsou snadná.

Prvek `<img>` navíc není na stránce sám. Je to potomek prvku `<section>`, který jsme právě deklarovali atributem `itemscope`. Mikrodata používají vztah rodič-potomek prvků stránky, aby definovala obor platnosti mikrodatových vlastností. Normálním jazykem tím říkáme: „Tento prvek `<section>` reprezentuje osobu. Všechny mikrodatové vlastnosti, které můžete najít u potomků prvku `<section>`, jsou vlastnosti té osoby.“ Jestli vám to pomůže, můžete si představit, že prvek `<section>` obsahuje podmět věty. Atribut `itemprop` představuje větný přísudek, něco jako „se zobrazuje na“. Hodnota mikrodatové vlastnosti pak představuje předmět věty.

Tato osoba [explicitní, z `<section itemscope itemtype="...">`]

se zobrazuje na [explicitní, z `<img itemprop="photo">`]

`http://diveintohtml5.info/examples/2000_05_mark.jpg` [implicitní, z atributu `<img src>`]

Podmět je třeba definovat pouze jednou vložením atributů `itemscope` a `itemtype` do nejzevnějšího prvku `<section>`. Sloveso je definováno vložením atributu `itemprop="photo"` do prvku `<img>`. Předmět věty nepotřebuje žádný zvláštní kódový zápis, protože datový model pro mikrodata HTML5 říká, že hodnotou vlastnosti prvku `<img>` je jeho atribut `src`.

Když přejdeme k další části kódu, uvidíme hlavičku `<h1>` a začátek seznamu `<dl>`. Ani `<h1>`, ani `<dl>` nepotřebují zápis pomocí mikrodat. Ne každý kousek HTML se musí stát mikrodatovou vlastností. V mikrodatech jsou důležité vlastnosti samotné, nikoliv kód HTML nebo hlavičky, které vlastnosti obklopují. `<h1>` není vlastnost – je to pouze hlavička. Stejně tak tag `<dt>`, který říká „Jméno“, není vlastnost, ale pouhý popisek.

Nuda ↷

Nuda → `<h1>Kontaktní údaje</h1>`
`<dl>`
`<dt>Jméno</dt>`
`<dd>Mark Pilgrim</dd>`

Kde se tedy nacházejí opravdové informace? Ty jsou v prvku `<dd>`, a proto musíme atribut `itemprop` vložit právě tam. O jakou se jedná vlastnost? O vlastnost `name`. Kde je hodnota vlastnosti? Je to text uvnitř prvku `<dd>`. Musí to být speciálně označeno? Datový model pro mikrodata HTML5 říká, že prvky `<dd>` se nezpracovávají žádným zvláštním způsobem, takže hodnotou vlastnosti je prostě text uvnitř prvku.

↶ to je moje jméno, neobnoste mi ho

```
<dd itemprop="name">Mark Pilgrim</dd>
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

Co jsme to právě řekli v normálním jazyce? „Tato osoba se jmenuje Mark Pilgrim.“ Tak dobrá, jdeme dál.

Dvě další vlastnosti jsou o něco složitější. Takto vypadá kód před vložením mikrodat:

```
<dt>Pracovní pozice</dt>
<dd>Developer advocate pro Google, Inc.</dd>
```

Když se podíváte na definici slovníku Osoba, text „Developer advocate pro Google Inc.“ se ve skutečnosti skládá ze *dvou* vlastností: `title` („Developer advocate“) a `affiliation` („Google, Inc.“) Jak to můžete vyjádřit v mikrodatech? Stručná odpověď zní, že nijak. Mikrodata nedokážou rozdělit souvislý text na samostatné vlastnosti. Nemůžete říct: „prvních 18 znaků tohoto textu je jedna mikrodatová vlastnost a dalších 12 znaků vlastnost jiná“.

Ale není vše ztraceno. Představte si, že byste na text „Developer advocate“ chtěli použít jiné písmo než na text „Google, Inc.“ To by v CSS taky nešlo. Co byste udělali? Nejdřív byste museli zabalit různé kousky textu do pomocných prvků, např. `<span>`, a pak na každý prvek `<span>` použít jiné pravidlo CSS.

Tato technika je užitečná i pro mikrodata. V našem příkladu máme dvě různé informace: `title` a `affiliation`. Pokud každou z nich obalíme do pomocného prvku `<span>`, můžeme deklarovat, že je každý `<span>` zvláštní mikrodatová vlastnost.

```
<dt>Pracovní pozice</dt>
<dd><span itemprop="title">Developer advocate</span> pro
    <span itemprop="affiliation">Google, Inc.<span></dd>
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

A je to! „Pracovní pozice této osoby se jmenuje ‚Developer advocate‘. Tato osoba je zaměstnancem Google, Inc.“ Dvě věty, dvě mikrodatové vlastnosti. Trochu více kódu, ale výsledek stojí za to.

Tato technika se hodí i pro označení poštovních adres. Slovník Osoba definuje vlastnost `address`, která je sama o sobě položkou mikrodat. To znamená, že adresa má svůj vlastní slovník (`http://data-vocabulary.org/Address`) a definuje své vlastní vlastnosti. Slovník Adresa definuje 5 vlastností: `street-address`, `locality`, `region`, `postal-code` a `country-name`.

Jste-li programátor, asi znáte zápis pomocí teček pro definování objektů a jejich vlastností. Představte si ten vztah následovně:

- `Person`
- `Person.address`
- `Person.address.street-address`
- `Person.address.locality`
- `Person.address.region`
- `Person.address.postal-code`
- `Person.address.country-name`

V tomto příkladu je celá poštovní adresa obsažena v jediném prvku `<dd>`. (Znovu podotýkám, že prvek `<dt>` je pouze popisek, takže v přidávání sémantiky pomocí mikrodat nehraje žádnou roli.) Zápis vlastnosti `address` je snadný. Prostě přidejte atribut `itemprop` do prvku `<dd>`.

```
<dt>Poštovní adresa</dt>
<dd itemprop="address">
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

Uvědomte si ale, že vlastnost `address` sama o sobě je položkou mikrodat. To znamená, že musíme přidat také atributy `itemscope` a `itemtype`.

```
<dt>Poštovní adresa</dt>
<dd itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

Tohle už jsme všechno viděli, ovšem jen u položek nejvyšší úrovně. Prvek `<section>` definuje atributy `itemtype` a `itemscope`; a všechny prvky uvnitř prvku `<section>`, které definují mikrodatové vlastnosti, jsou „v oboru platnosti“ daného slovníku. Ale to je poprvé, co se setkáváme s *vnořenými* obory platnosti – ty definují nový `itemtype` a `itemscope` (na prvku `<dd>`) uvnitř již existujícího (na prvku `<section>`). Tento vnořený obor platnosti funguje úplně stejně jako DOM. Prvek `<dd>` má určitý počet potomků a všechny patří do oboru platnosti slovníku definovaného na prvku `<dd>`. Jakmile se prvek `<dd>` uzavře odpovídajícím tagem `</dd>`, obor platnosti se vrátí do slovníku definovaného v rodičovském prvku (v tomto případě `<section>`).

Vlastnosti Adresy mají stejný problém, s jakým jsme se setkali u vlastností `title` a `affiliation`. Jedná se o jeden dlouhý souvislý text, ale my ho chceme rozdělit do pěti různých mikrodatových vlastností. Řešení je stejné: zabalit každý jednotlivý kousek informace do pomocného prvku `<span>` a pak deklarovat mikrodatové vlastnosti v každém prvku `<span>`.

```
    <dd itemprop="address" itemscope
            itemtype="http://data-vocabulary.org/Address">
        <span itemprop="street-address">100 Main Street</span><br>
        <span itemprop="locality">Anytown</span>,
        <span itemprop="region">PA</span>
        <span itemprop="postal-code">19999</span>
        <span itemprop="country-name">USA</span>
    </dd>
</dl>
```

[Sledujte postup s námi! Předtím: person.html, potom: person-plus-microdata.html]

V normálním jazyce: „Tato osoba má poštovní adresu. Část poštovní adresy obsahující adresu ulice je ‚100 Main Street‘. Jméno obce je ‚Anytown‘. Státem je ‚PA‘. Poštovní směrovací číslo je ‚19999‘. Jméno země je ‚USA‘.“ Je to hračka.

PTEJTE SE PROFESORA ZNAČKY

Q: Je tento formát poštovní adresy specifický pro USA?

A: Není. Vlastnosti slovníku Adresa jsou dostatečně obecné na to, aby se s nimi dala pospat většina poštovních adres na světě. Ne všechny adresy budou mít u každé vlastnosti hodnotu, ale to je v pořádku. Některé adresy budou potřebovat na jednu vlastnost více „řádků“, ale to je taky v pořádku. Například pokud vaše poštovní adresa obsahuje adresu ulice a číslo bytu, obojí půjde do podvlastnosti `street-address`.

```
<p itemprop=”address” itemscope
       itemtype=”http://data-vocabulary.org/Address”>
    <span itemprop=”street-address”>
        100 Main Street
        Suite 415
    </span>
    ...
</p>
```

Na vzorové stránce „O mně“ se nachází ještě něco: seznam URL. Slovník Osoba má na to vlastnost nazvanou `url`. Vlastností `url` může být v podstatě cokoliv. (Tedy musí to být URL, ale to vás asi napadlo.) Chci tím říct, že je vlastnost `url` definovaná volně. Tuto vlastnost může tvořit jakákoliv URL, kterou chcete spojit s Osobou: odkaz na blog, fotogalerii nebo profil na jiné stránce typu Facebook nebo Twitter.

Další důležitou věcí je, že jedna osoba může mít více vlastností `url`. Technicky vzato se každá vlastnost může objevit více než jednou, ale doposud jsme toho nevyužili. Například můžete mít dvě vlastnosti `photo`, z nichž každá bude odkazovat na URL jiné fotografie. Tady chci uvést čtyři různé URL: svůj blog, svou profilovou stránku na Googlu, svůj uživatelský profil na Reddit a svůj účet na Twitteru. V HTML je to seznam odkazů: čtyři prvky `<a>`, každý ve svém vlastním prvku `<li>`. V mikrodatech dostává každý prvek `<a>` atribut `itemprop="url"`.

```
<h1>Moje digitální stopy</h1>
<ul>
    <li><a href="http://diveintomark.org/"
        itemprop="url">weblog</a></li>
    <li><a href="http://www.google.com/profiles/pilgrim"
        itemprop="url">profil na Googlu</a></li>
    <li><a href="http://www.reddit.com/user/MarkPilgrim"
        itemprop="url">profil na Reddit.com</a></li>
```

```
        <li><a href="http://www.twitter.com/diveintomark"
            itemprop="url">Twitter</a></li>
    </ul>
```

Podle datového modelu pro mikrodata HTML5 se prvky `<a>` zpracovávají zvláštním způsobem. Hodnotou mikrodatové vlastnosti je atribut `href`, nikoliv textový obsah potomka. Texty odkazů se při zpracovávání mikrodat ignorují. Proto to v normálním jazyce znamená: „Tato osoba má URL `http://diveintomark.org/`. Tato osoba má další URL `http://www.google.com/profiles/pilgrim`. Tato osoba má další URL `http://www.reddit.com/user/MarkPilgrim`. Tato osoba má další URL `http://www.twitter.com/diveintomark`.“

PŘEDSTAVUJEME RICH SNIPPETS OD GOOGLU

Na chvíli bych se chtěl zastavit a zeptat se: „Proč to vlastně děláme?“ Přidáváme sémantiku jenom proto, abychom přidali sémantiku? Nechápejte mě špatně – se špičatými závorkami si jako každý webový fanda hraju rád. Ale proč mikrodata? Proč se s nimi obtěžovat?

Existují dva hlavní typy aplikací, které konzumují HTML, potažmo mikrodata HTML5:

- webové prohlížeče,
- vyhledávače.

Pro prohlížeče definuje HTML5 sadu API DOMu pro extrahování mikrodatových položek, vlastností a hodnot vlastností z webové stránky. V okamžiku psaní této knihy (únor 2011) tyto API žádný prohlížeč nepodporuje. Ani jeden. Takže jsme v koncích, aspoň dokud to prohlížeče nedoženou a neimplementují API na straně klienta.

Druhým hlavním konzumentem HTML jsou vyhledávače. Co můžou vyhledavače dělat s mikrodatovými vlastnostmi týkajícími se osoby? Představte si následující: místo jednoduchého zobrazení názvu stránky a výňatku z textu by vyhledavač mohl začlenit některé z těch strukturovaných informací a zobrazit je. Celé jméno, název pracovní pozice, zaměstnavatele, adresu, možná i miniaturu profilové fotky. Přitáhlo by to vaši pozornost? Moji ano.

Google podporuje mikrodata jako součást programu Rich Snippets (strukturované úryvky). Když webový prolézací modul Googlu analyzuje vaši stránku a najde na ní mikrodatové vlastnosti, které vyhovují slovníku `http://data-vocabulary.org/Person`, zpracuje tyto vlastnosti a uloží je spolu se zbytkem dat stránky. Google dokonce poskytuje užitečný nástroj, díky kterému můžete spatřit, jakým způsobem Google „vidí“ vaše mikrodatové vlastnosti[3]. Když

3 https://developers.google.com/structured-data/testing-tool/

ho použijete na testování naší vzorové stránky „O mně“ obsahující mikrodata, dostanete tento výstup:

```
Item
    Type: http://data-vocabulary.org/person
    photo = http://diveintohtml5.info/examples/2000_05_mark.jpg
    name = Mark Pilgrim
    title = Developer advocate
    affiliation = Google, Inc.
    address = Item( 1 )
    url = http://diveintomark.org/
    url = http://www.google.com/profiles/pilgrim
    url = http://www.reddit.com/user/MarkPilgrim
    url = http://www.twitter.com/diveintomark

Item 1
    Type: http://data-vocabulary.org/address
    street-address = 100 Main Street
    locality = Anytown
    region = PA
    postal-code = 19999
    country-name = USA
```

Najdete tam všechno: vlastnost photo z atributu `<img src>`, všechny čtyři URL ze seznamu atributů `<a href>`, dokonce objekt adresy (uveden jako „Item 1“) a všech pět jeho podvlastností.

Jak Google použije všechny tyto informace? Přijde na to. Neexistují žádná přísně stanovená pravidla ohledně toho, jak se mají zobrazovat mikrodatové vlastnosti, které vlastnosti se mají zobrazit a zda se vůbec mají zobrazit. Pokud bude někdo hledat „Mark Pilgrim“ a Google určí, že se tato stránka „O mně“ umístí mezi výsledky, a rozhodne, že mikrodatové vlastnosti, které na stránce původně našel, stojí za zobrazení, může výsledek vyhledávání vypadat nějak takhle:

> O Marku Pilgrimovi
> Anytown PA - Developer advocate - Google, Inc.
> Tady se zobrazí fragment stránky.
> Tady se zobrazí fragment stránky.
> diveintohtml5.info/examples/person-plus-microdata.html - Archiv - Podobné

První řádek „O Marku Pilgrimovi“ je ve skutečnosti názvem stránky, uvedeným v prvku `<title>`. To není moc zajímavé; Google to dělá u každé stránky. Ale druhý řádek je plný informací převzatých přímo z mikrodatových popisů, které jsme přidali na stránku. „Anytown PA“ bylo

součástí poštovní adresy, zapsané pomocí slovníku `http://data-vocabulary.org/Address`. „Developer advocate“ a „Google, Inc.“ byly dvě vlastnosti (`title` a `affiliation`) ze slovníku `http://data-vocabulary.org/Person`.

Je to docela bomba. Na to, abyste mohli upravovat, jak se bude vaše stránka zobrazovat ve výsledcích hledání, nemusíte být velká korporace a uzavírat speciální dohody s dodavateli vyhledávačů. Stačí najít si deset minut a přidat na stránku pár atributů HTML, které přidají popisky k datům, jež byste zveřejnili tak jako tak.

PTEJTE SE PROFESORA ZNAČKY

Q: Udělal jsem všechno, co jste říkal, ale výsledek hledání mé stránky v Googlu vypadá stále stejně. Jak to?

A: „Google nezaručuje, že bude kód jakékoliv stránky použit ve výsledcích hledání.“ Ale i když se Google rozhodne vaše mikrodatové popisky nevyužít, jiný vyhledávač to třeba udělá. Stejně jako zbytek HTML5 jsou mikrodata otevřeným standardem, který může implementovat každý. Vaším úkolem je poskytnout tolik dat, kolik můžete. A na zbytku světa potom je se rozhodnout, co s nimi. Možná budete překvapeni!

10.5 Označování organizací

Mikrodata nejsou omezena na jediný slovník. Stránky „O mně“ jsou fajn, ale tu budete mít pravděpodobně jenom jednu. Chcete více? Podívejme se, jak označit organizace a firmy.

Toto je vzor stránky se záznamy o firmě[4]. Podívejme se na originální kód HTML bez mikrodat.

```
<article>
    <h1>Google, Inc.</h1>
    <p>
        1600 Amphitheatre Parkway<br>
        Mountain View, CA 94043<br>
        USA
    </p>
    <p>650-253-0000</p>
    <p><a href="http://www.google.com/">Google.com</a></p>
</article>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

4 http://diveintohtml5.info/examples/organization.html

Stručné a sympatické. Všechny informace o organizaci se nacházejí v prvku `<article>`, u něhož začneme.

```
<article itemscope itemtype="http://data-vocabulary.org/
Organization">
```

Stejně jako při označování osob musíte atributy `itemscope` a `itemtype` nastavit v nejzevnějším prvku. V tomto případě je takovým prvkem `<article>`. Atribut `itemtype` deklaruje používaný mikrodatový slovník (v tomto případě `http://data-vocabulary.org/Organization`) a atribut `itemscope` deklaruje, že se všechny vlastnosti nastavené pro prvky-potomky vztahují k tomuto slovníku.

Jak tedy vypadá slovník Organizace? Je jednoduchý a jasný – a z nějaké části už by vám mohl připadat povědomý.

Slovník Organizace	
Vlastnost	**Popis**
`name`	Název organizace (například „Initech“)
`url`	Odkaz na domovskou stránku organizace
`address`	Adresa organizace. Může mít podvlastnosti `street-address`, `locality`, `region`, `postal-code`, a `country-name`.
`tel`	Telefonní číslo organizace
`geo`	Uvádí zeměpisné souřadnice adresy. Vždy má dvě podvlastnosti, `latitude` a `longitude`.

První úsek kódu uvnitř nejzevnějšího prvku `<article>` je `<h1>`. Tento prvek `<h1>` obsahuje název firmy, proto přidáme atribut `itemprop="name"` přímo do něj.

```
<h1 itemprop="name">Google, Inc.</h1>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Podle datového modelu pro mikrodata HTML5 se prvky `<h1>` nezpracovávají žádným zvláštním způsobem. Hodnotu mikrodatové vlastnosti tvoří jednoduše textový obsah prvku `<h1>`. Normálním jazykem se tím říká: „Název této organizace je ‚Google, Inc.‘“

Následuje adresa sídla. Označování adresy organizace se provádí stejně jako označování adresy osoby. Nejprve přidejte atribut `itemprop="address"` do nejzevnějšího prvku adresy sídla

(v tomto případě do prvku <p>). To bude říkat, že toto je vlastnost address organizace. Co ovšem s vlastnostmi samotné adresy? Abychom označili, že i položka adresa má svoje vlastnosti, potřebujeme také definovat atributy itemtype a itemscope.

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Nakonec musíme zabalit každý z údajů do pomocného prvku , abychom každému z nich přiřadili patřičný název mikrodatové vlastnosti (street-address, locality, region, postal-code, a country-name).

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
    <span itemprop="street-address">1600 Amphitheatre Parkway
    </span><br>
    <span itemprop="locality">Mountain View</span>,
    <span itemprop="region">CA</span>
    <span itemprop="postal-code">94043</span><br>
    <span itemprop="country-name">USA</span>
</p>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Normálním jazykem se tím říká: „Tato organizace má adresu. Část adresy popisující ulici je ‚1600 Amphitheatre Parkway‘. Obec se jmenuje ‚Mountain View‘. Stát je ‚CA‘. Poštovní kód je ‚94043‘. Název země je ‚USA‘.“

Následuje telefonní číslo organizace. Telefonní čísla jsou proslulá svou záludností a jejich syntax závisí na konkrétní zemi. (A když chcete zavolat do jiné země, je to ještě horší.) V tomto příkladu máme telefonní číslo ve Spojených státech ve formátu vhodném pro volání odkudkoliv v rámci Spojených států.

```
<p itemprop="tel">650-253-0000</p>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

(Jo, pokud jste si nevšimli, slovník Adresa přestal platit po tom, co se uzavřel prvek <p>. Od teď definujeme vlastnosti opět ze slovníku Organizace.)

Pokud chcete uvést více telefonních čísel – třeba jeden pro tuzemské a jeden pro mezinárodní zákazníky – můžete to udělat. Mikrodatové vlastnosti se mohou opakovat. Hlavně zajistěte, aby se každé číslo nacházelo ve vlastním prvku HTML, odděleně od případného popisku, který mu přiřadíte.

```
<p>
    Zákazníci z USA: <span itemprop="tel">650-253-0000</span><br>
    Zákazníci z Velké Británie: <span itemprop="tel">00 + 1*
    + 6502530000</span>
</p>
```

Podle datového modelu pro mikrodata HTML5 se ani prvek `<p1>`, ani prvek `<span>` nezpracovává žádným zvláštním způsobem. Hodnotu mikrodatové vlastnosti `tel` tvoří jednoduše textový obsah. Mikrodatový slovník Organizace se nesnaží dále dělit jednotlivé části telefonního čísla. Celá vlastnost `tel` je textem ve volné formě: pokud chcete, můžete dát kód země do závorek nebo použít pro oddělení číselných skupin mezery místo spojovníků. Případná analýza telefonního čísla je pak zcela na konzumentovi mikrodat (vyhledávači nebo prohlížeči).

Dále tu máme ještě jednu povědomou vlastnost, `url`. Způsobem shodným s přiřazováním URL osobě můžete přiřadit URL organizaci. Může to být domovská stránka firmy, stránka s kontaktními údaji, prezentací zboží nebo čímkoliv jiným. Jestliže to je URL, k níž má organizace nějaký vztah, označte ji atributem `itemprop="url"`.

```
<p><a itemprop="url" href="http://www.google.com/">Google.com</a></p>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Podle datového modelu pro mikrodata HTML5 se prvek `<a>` zpracovává zvláštním způsobem. Hodnotu mikrodatové vlastnosti tvoří hodnota atributu `href`, a ne text odkazu. Normálním jazykem se tím říká: „Tato organizace má přiřazenu webovou adresu `http://www.google.com/`.“ Mikrodata nevypovídají nic o důvodech přiřazení, ani neobsahují text odkazu „Google.com“.

Nakonec chci promluvit o geolokaci. Ne, ne o geolokačním API od W3C, ale o tom, jak pomocí mikrodat označit fyzickou adresu organizace.

Až do teď se všechny příklady soustředily na označování *viditelných* údajů. To znamená, že když název organizace máte v prvku `<h1>`, přidáte do prvku `<h1>` atribut `itemprop`, abyste deklarovali, že tento (viditelný) nadpis je skutečně názvem organizace. Nebo do prvku `<img>`, který odkazuje na fotografii, přidáte atribut `itemprop`, abyste deklarovali, že tento (viditelný) obrázek je fotografií osoby.

V tomto příkladu je tomu s geolokací jinak. Není tu žádný viditelný text, který by udával přesnou zeměpisnou šířku a délku organizace (až na čtyři desetinná místa!). Příklad organization.html (bez mikrodat) neobsahuje vlastně žádné informace o souřadnicích. Je v něm odkaz na Mapy Google, ale zeměpisné souřadnice neobsahuje ani URL tohoto odkazu. (Obsahuje však podobnou informaci ve speciálním formátu Google.) Ani kdybychom měli k dispozici mapovou službu, která by používala zeměpisné souřadnice jako parametry URL, mikrodata by stejně neuměla rozpoznat jednotlivé části URL. Není možné například deklarovat, že první parametr dotazu v URL je šířka, druhý délka a ostatní parametry dotazu jsou nepodstatné.

Pro řešení takovýchto mezních případů poskytuje HTML5 možnost přidat *neviditelné* údaje. Tato technika by měla být použita pouze jako poslední možnost – pokud existuje nějaký způsob zobrazit pro vás důležitá data, udělejte to. Neviditelné údaje pro strojové čtení obvykle rychle „zastarávají“. To znamená, že po nějaké době někdo aktualizuje viditelný text, ale zapomene na neviditelné údaje. Stává se to častěji, než si myslíte, a vám se to určitě stane také.

Jsou ovšem případy, kdy se bez neviditelných údajů neobejdeme. Třeba váš šéf *fakt* chce strojově čitelné geolokační informace, ale nechce přitom zaneřádit rozhraní dvojicemi nesrozumitelných šesticiferných čísel. Jedinou volbou jsou neviditelné údaje. Jedinou světlou stránkou snad je, že neviditelné údaje můžete umístit hned za viditelným textem, který popisují, což může pro toho, kdo bude v budoucnu aktualizovat viditelný text, posloužit jako připomenutí, aby neviditelné údaje aktualizoval také.

V tomto příkladu vytvoříme pomocný prvek `<span>` uvnitř prvku `<article>`, kde se nacházejí všechny ostatní vlastnosti organizace, a dovnitř prvku `<span>` přidáme neviditelné geolokační údaje.

```
    <span itemprop="geo" itemscope
        itemtype="http://data-vocabulary.org/Geo">
        <meta itemprop="latitude" content="37.4149" />
        <meta itemprop="longitude" content="-122.078" />


    </span>
</article>
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Geolokační informace jsou definovány vlastním slovníkem, podobně jako adresa osoby nebo organizace, a proto potřebuje tento prvek `<span>` tři atributy:

- `itemprop="geo"` říká, že tento prvek reprezentuje vlastnost `geo` dané organizace

- `itemtype="http://data-vocabulary.org/Geo"` říká, kterým z mikrodatových slovníků se vlastnosti tohoto prvku řídí

- `itemscope` říká, že tento prvek je obalujícím prvkem pro položku mikrodat s vlastním slovníkem (uvedeným v atributu `itemtype`). Všechny vlastnosti uvnitř tohoto prvku jsou vlastnostmi slovníku `http://data-vocabulary.org/Geo`, a nikoli `http://data-vocabulary.org/Organization`, do něhož je vnořen.

Další důležitá otázka zní: „Jak přidat neviditelné údaje?“ Pomocí prvku `<meta>`. V předchozích verzích HTML se prvek `<meta>` mohl použít pouze uvnitř tagu `<head>`. V HTML5 ho můžete použit kdekoliv, a přesně to uděláme.

```
<meta itemprop="latitude" content="37.4149" />
```

[Sledujte postup s námi! Předtím: organization.html, potom: organization-plus-microdata.html]

Podle datového modelu pro mikrodata HTML5 se prvek `<meta>` zpracovává zvláštním způsobem. Hodnotu mikrodatové vlastnosti tvoří atribut `content`. Jelikož se tento atribut nikdy neukáže na stránce, máme skvělou příležitost ke vložení neomezeného množství neviditelných údajů. S velkou mocí ovšem přichází velká zodpovědnost. Vaší zodpovědností je v tomto případě zajistit synchronizaci neviditelných údajů s viditelným obsahem stránky.

Google Rich Snippets nemá přímou podporu slovníku Organizace, takže vám nemůžu ukázat žádné pěkné vzorové výsledky hledání. Ale organizace jsou hojně zastoupeny ve dvou dalších ukázkách – událostech a recenzích –, které Google Rich Snippets podporuje.

10.6 Označování událostí

Pořád se něco děje. A něco z toho se děje v předem určený čas. Nebylo by skvělé umět oznámit vyhledávačům, kdy přesně se má něco stát? I k tomu se hodí špičaté závorky.
Pojďme se nejprve podívat na vzorový program mých veřejných vystoupení[5].

```
<article>
    <h1>Google Developer Day 2009</h1>
    <img width="300" height="200"
        src="http://diveintohtml5.info/examples/gdd-2009-prague-
        pilgrim.jpg"
        alt="[Mark Pilgrim na pódiu]">
    <p>
```

5 http://diveintohtml5.info/examples/event.html

```
        Akce Google Developer Day jsou příležitostí dozvědět se
        o vývojářských produktech Google od techniků, kteří je
        vytvořili. Tato jednodenní konference nabízí semináře
        a „konzultační hodiny” o webových technologiích jako Mapy
        Google, OpenSocial, Android, AJAX API, Chrome a Google Web
        Toolkit.
    </p>
    <p>
        <time datetime="2009-11-06T08:30+01:00">6. listopadu 2009,
        8:30</time>
            &ndash;
        <time datetime="2009-11-06T20:30+01:00">20:30</time>
    </p>
    <p>
        Kongresové centrum<br>
        5. května 65<br>
        140 21 Praha 4<br>
        Česká republika
    </p>
    <p><a href="http://code.google.com/intl/cs/events/
    developerday/2009/home.html">Domovská stránka GDD/Praha</a></p>
</article>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Všechny informace o akci se nacházejí uvnitř prvku `<article>`, proto také tam musíme umístit atributy `itemtype` a `itemscope`.

```
<article itemscope itemtype="http://data-vocabulary.org/Event">
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Adresa slovníku Událost je `http://data-vocabulary.org/Event`; stránka náhodou také obsahuje hezkou tabulku s popisem vlastností slovníku. Které vlastnosti to jsou?

Slovník Událost	
Vlastnost	**Popis**
`summary`	Název události
`url`	Odkaz na stránku s informacemi o události

`location`	Adresa nebo místo konání události. Volitelně může být popsána vnořenými slovníky Organizace nebo Adresa.
`description`	Popis události
`startDate`	Datum a čas zahájení události ve formátu ISO pro zápis data a času
`endDate`	Datum a čas ukončení události ve formátu ISO pro zápis data a času
`duration`	Délka trvání události ve formátu ISO pro zápis trvání
`eventType`	Kategorie události (například „Koncert“ nebo „Přednáška“). Jedná se o řetězec ve volné formě, ne o výčtový atribut
`geo`	Udává zeměpisné souřadnice místa. Vždy obsahuje dvě podvlastnosti, `latitude` a `longitude`
`photo`	Odkaz na fotografii nebo obrázek spojený s událostí

Název události se nachází v prvku `<h1>`. Podle datového modelu pro mikrodata HTML5 se prvky `<h1>` nezpracovávají žádným zvláštním způsobem. Hodnotu mikrodatové vlastnosti tvoří jednoduše textový obsah prvku `<h1>`. Je potřeba už jen přidat atribut `itemprop`, a tím deklarovat, že daný prvek `<h1>` obsahuje název události.

```
<h1 itemprop="summary">Google Developer Day 2009</h1>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Normálním jazykem se tím říká: „Název této události je Google Developer Day 2009.“

Tento záznam o události má fotografii, kterou můžete označit vlastností `photo`. Jak asi tušíte, fotografie je už vložena pomocí prvku `<img>`. Stejně jako vlastnost `photo` ve slovníku Osoba je `photo` Události URL. Jelikož datový model pro mikrodata HTML5 tvrdí, že hodnota vlastnosti prvku `<img>` je jeho atribut `src`, zbývá jen přidat do prvku `<img>` atribut `itemprop`.

```
<img itemprop="photo" width="300" height="200"
    src="http://diveintohtml5.info/examples/gdd-2009-prague-pilgrim.
    jpg"
    alt="[Mark Pilgrim na pódiu]">
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Normálním jazykem se tím říká: „Fotografie této události se nachází na `http://diveintohtml5.info/examples/gdd-2009-prague-pilgrim.jpg`.“

Následuje delší popis události, jímž je odstavec textu ve volné formě.

```
<p itemprop="description">Akce Google Developer Day jsou
příležitostí dozvědět se o vývojářských produktech Google
od techniků, kteří je vytvořili. Tato jednodenní konference
nabízí semináře a „konzultační hodiny” o webových technologiích
jako Mapy Google, OpenSocial, Android, AJAX API, Chrome a Google
Web Toolkit.</p>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Další část je něčím novým. Události obvykle probíhají v jistých dnech a začínají v konkrétní čas. Datum a čas by se v HTML5 měly značit pomocí prvku `<time>`, což už zde děláme. Otázkou proto je, jak přidat mikrodatové vlastnosti do prvků `<time>`. Když se podíváme na datový model pro mikrodata HTML5, uvidíme, že prvek `<time>` se zpracovává zvláštním způsobem. Hodnotu mikrodatové vlastnosti prvku `<time>` tvoří hodnota atributu `datetime`. A hle, vlastnosti `startDate` a `endDate` ve slovníku Událost se zapisují ve formátu ISO, stejně jako vlastnost `datetime` prvku `<time>`. Znovu vidíme, jak se sémantika základního HTML a sémantika uživatelských mikrodatových slovníků nádherně doplňují. Označit data dat zahájení a ukončení pomocí mikrodat je jednoduché:

- Především správně použijte HTML (prvek `<time>` pro označení dat a časů)
- Přidejte jediný atribut `itemprop`

```
<p>
    <time itemprop="startDate" datetime="2009-11-06T08:30+01:00">
    6. listopadu 2009, 8:30</time>
        &ndash;
    <time itemprop="endDate" datetime="2009-11-06T20:30+01:00">
    20:30</time>

</p>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Normálním jazykem se tím říká: „Tato událost začíná 6. listopadu 2009 v 8:30 a trvá do 6. listopadu 2009 do 20:30 (pražského místního času, GMT+1).“

Následuje vlastnost `location`. Pravidla slovníku Událost uvádí, že může být buď součástí slovníku Organizace, nebo slovníku Adresa. V našem případě událost proběhne v místě určeném

pro konference, pražském Kongresovém centru. Když s ním budeme zacházet jako s organizací, budeme moct vložit jak název místa, tak jeho adresu.

Nejprve deklarujme, že prvek `<p>` obsahující adresu je vlastností události `location` a že je také svou vlastní mikrodatovou položkou, která se řídí slovníkem `http://data-vocabulary.org/Organization`.

```
<p itemprop="location" itemscope
    itemtype="http://data-vocabulary.org/Organization">
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Dále označíme název organizace tím, že ho umístíme do pomocného prvku `<span>` a přidáme mu atribut `itemprop`.

```
<span itemprop="name">Kongresové centrum</span><br>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Vzhledem k pravidlům oboru platnosti mikrodat atribut `itemprop="name"` definuje vlastnost ze slovníku Organizace, nikoli ze slovníku Událost. Prvek `<p>` definoval začátek oboru platnosti vlastností organizace a ještě nebyl uzavřen tagem `</p>`. Všechny mikrodatové vlastnosti, které tady definujeme, jsou ze slovníku aktuálního oboru platnosti. Vnořené slovníky se podobají zásobníku. Ještě jsme neodebrali položku ze zásobníku, takže se pořád bavíme o vlastnostech slovníku Organizace.

Nyní do zásobníku přidáme třetí slovník, Adresu dané Organizace pro tuto Událost.

```
<span itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Chceme zase označit každý kousek adresy jako zvláštní mikrodatovou vlastnost, takže potřebujeme hromadu pomocných prvků `<span>`, do kterých šoupneme atributy `itemprop`. (Pokud to nestačíte sledovat, vraťte se a přečtěte si o označování adresy Osoby a označování adresy Organizace.)

```
<span itemprop="street-address">5. května 65</span><br>
<span itemprop="postal-code">140 21</span>
<span itemprop="locality">Praha 4</span><br>
```

```
<span itemprop="country-name">Česká republika</span>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Tím jsme vyčerpali vlastnosti Adresy a můžeme uzavřít prvek `<span>`, který zahájil obor platnosti Adresa, a odebrat položku ze zásobníku.

```
</span>
```

Organizace nemá žádné další vlastnosti, proto uzavřeme prvek `<p>`, který zahájil obor platnosti Organizace, a znovu odebereme položku ze zásobníku.

```
</p>
```

Teď se vracíme ke stanovení vlastností Události. Další vlastností je `geo` a ta představuje místo konání Události. Tato vlastnost používá tentýž slovník Geo, který jsme použili k označení umístění Organizace v předchozí podkapitole. Potřebujeme prvek `<span>`, který by vystupoval jako kontejner – ten dostane atributy `itemtype` a `itemscope`. Dovnitř tohoto prvku `<span>` musíme přidat dva prvky `<meta>`: jeden pro vlastnost `latitude` a druhý pro vlastnost `longitude`.

```
<span itemprop="geo" itemscope itemtype="http://data-vocabulary.org/
Geo">
    <meta itemprop="latitude" content="50.047893" />
    <meta itemprop="longitude" content="14.4491" />
</span>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Také jsme uzavřeli prvek `<span>` obsahující vlastnosti Geo, takže dále zase definujeme vlastnosti slovníku Událost. Poslední vlastností je `url`, kterou už znáte. Přiřazení URL Události funguje stejně jako přiřazení URL Osobě a přiřazení URL Organizaci. Pokud používáte HTML správně (odkazy označujete tagem `<a href>`), pro deklarování odkazu jakožto mikrodatové vlastnosti `url` stačí přidat atribut `itemprop`.

```
<p>
    <a itemprop="url"
        href="http://code.google.com/intl/cs/events/
        developerday/2009/home.html">
        Domovská stránka GDD/Praha
    </a>
```

```
    </p>
</article>
```

[Sledujte postup s námi! Předtím: event.html, potom: event-plus-microdata.html]

Vzorová stránka s událostí obsahuje také druhou akci, můj proslov na konferenci ConFoo v Montréalu. Pro stručnost ji nebudu rozepisovat po řádcích – je v podstatě stejná jako ta v Praze: položka Událost s vnořenými položkami Geo a Adresa. Zmiňuji ji, jenom abych zdůraznil, že na jedné stránce může být několik událostí označených pomocí mikrodat.

NÁVRAT GOOGLE RICH SNIPPETS

Podle testovacího nástroje Google Rich Snippets vytáhnou webové prolézací moduly Googlu z vzorové stránky se záznamy o událostech tuto informaci:

```
Item
    Type: http://data-vocabulary.org/Event
    summary = Google Developer Day 2009
    eventType = conference
    photo = http://diveintohtml5.info/examples/
    gdd-2009-prague-pilgrim.jpg
    description = Akce Google Developer Day jsou příležitostí
    dozvědět se o vývojářských produktech Google od techniků, kteří
    je vytvořili. Tato jednodenní konference nabízí semináře
    a „konzultační hodiny” o webových technologiích jako Mapy Goo...
    startDate = 2009-11-06T08:30+01:00
    endDate = 2009-11-06T20:30+01:00
    location = Item(__1)
    geo = Item(__3)
    url = http://code.google.com/intl/cs/events/developerday/2009/home.
    html

Item
    Id: __1
    Type: http://data-vocabulary.org/Organization
    name = Kongresové centrum
    address = Item(__2)

Item
    Id: __2
    Type: http://data-vocabulary.org/Address
```

```
    street-address = 5. května 65
    postal-code = 140 21
    locality = Praha 4
    country-name = Česká republika

Item
    Id: __3
    Type: http://data-vocabulary.org/Geo
    latitude = 50.047893
    longitude = 14.4491
```

Jak vidíte, jsou tam všechny informace, které jsme přidali pomocí mikrodat. Vlastnostem, které jsou zvláštními položkami mikrodat, jsou přiřazeny vnitřní identifikátory (`Item(__1)`, `Item(__2)` atd.). Není to součástí mikrodatové specifikace, ale způsob zpřehledňování výstupu, který používá testovací nástroj Google, aby ukázal seskupování vnořených položek a jejich vlastnosti.

Takto by mohl Google zobrazit tuto vzorovou stránku ve výsledcích hledání. (Znovu upozorňuji, že to je jenom příklad. Google může kdykoliv změnit formát zobrazování výsledků hledání a není zaručeno, že vůbec vezme v potaz mikrodatový kód. Omlouvám se za opakování, ale musím to říkat kvůli právníkům.)

Kalendář událostí Marka Pilgrima
Tady se zobrazí fragment stránky.
Tady se zobrazí fragment stránky.

Google Developer Day 2009	pá 6. 11.	Kongresové centrum, Praha 4, Česká republika
ConFoo.ca 2010	st 10. 3.	Hilton Montreal Bonaventure, Montréal, Québec, Canada

diveintohtml5.info/examples/event-plus-microdata.html - Archiv - Podobné

Po nadpisu stránky a automaticky vygenerovaném fragmentu textu použije Google kód s mikrodaty, který jsme přidali na stránku, aby zobrazil tabulku s událostmi. Všimněte si formátu data: „pá 6. 11.“ Tento řetězec se nikde v kódu HTML ani v mikrodatech neobjevuje. Použili jsme dva řetězce ve formátu ISO: `2009-11-06T08:30+01:00` a `2009-11-06T20:30+01:00`. Google se podíval na tato dvě data, pochopil, že se jedná o tentýž den, a rozhodl se zobrazit jediné datum v čitelnějším formátu.

Teď se podívejte na adresy míst konání. Google zobrazil pouze název místa, město a zemi, a ne přesnou poštovní adresu. Bylo to možné díky tomu, že jsme rozdělili adresu do pěti podvlastností – `name`, `street-address`, `region`, `locality`, a `country-name` – a zapsali každou část adresy jako samostatnou mikrodatovou vlastnost. Google toho využívá k zobrazení zkrácené adresy. Jiní konzumenti stejných mikrodat mohou zvolit k zobrazení jiné části a jiný způsob.

Neexistuje dobrý nebo špatný způsob zobrazení, ale je na vás poskytnout co nejvíce údajů co nejpřesnějším způsobem. Na zbytku světa je, aby si je přebral.

10.7 Označování recenzí

Tady je další příklad, co lze na webu (a snad i ve výsledcích vyhledávání) vylepšit pomocí kódu: recenze podniků a produktů.

Toto je krátká recenze, kterou jsem napsal pro svou oblíbenou pizzerii poblíž domova. (Tato restaurace mimochodem opravdu existuje. Pokud někdy budete v Apexu v Severní Karolíně, opravdu vám ji doporučuju.) Podívejme se na původní kód[6]:

```
<article>
    <h1>Pizzerie u Anny</h1>
    <p>★★★★☆ (4 hvězdičky z 5)</p>
    <p>Pizzerie v newyorském stylu přímo v historickém centru
    Apexu</p>
    <p>
Jídlo je špičkové. Atmosféra je pro „pizzerii za rohem” ideální.
Uvnitř je trochu těsno – pokud máte nadváhu, budete mít problém
dostat se na své místo a pak z něj ven, i prodírat se mezi stoly.
Dříve dávali jako předkrm zdarma česnekové rohlíčky; teď dávají
jenom obyčejný chleba a za něco lepšího si musíte připlatit.
Celkově je to tam ale super.
    </p>
    <p>
        100 North Salem Street<br>
        Apex, NC 27502<br>
        USA
    </p>
    <p>— recenzi napsal Mark Pilgrim, naposledy aktualizováno
    31. března 2010</p>
</article>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Tato recenze je obsažená v prvku `<article>`, a proto dáme atributy `itemtype` a `itemscope` právě tam. Jmenným prostorem URL pro tento slovník je `http://data-vocabulary.org/Review`.

6 http://diveintohtml5.info/examples/review.html

```
<article itemscope itemtype="http://data-vocabulary.org/Review">
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Jaké vlastnosti jsou k dispozici ve slovníku Recenze? Jsem rád, že se na to ptáte.

Slovník Recenze	
Vlastnost	**Popis**
`itemreviewed`	Název recenzované položky. Může to být produkt, služba, podnik, atd.
`rating`	Číselné kvalitativní hodnocení položky na stupnici od 1 do 5. Pro použití nestandardní stupnice to může být i vnořený slovník `http://data-vocabulary.org/Rating`.
`reviewer`	Jméno autora recenze
`dtreviewed`	Datum, kdy byla položka recenzována, ve formátu data ISO
`summary`	Stručné shrnutí recenze
`description`	Tělo recenze

První vlastnost je jednoduchá: `itemreviewed` je prostý text, který je zde obsažen v prvku `<h1>`, takže atribut `itemprop` byste měli dát právě tam.

```
<h1 itemprop="itemreviewed">Pizzerie u Anny</h1>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Vlastní hodnocení teď přeskočím a vrátím se k němu na konci.

Další dvě hodnoty jsou taky jasné. Vlastnost `summary` je krátkým popisem toho, co recenzujete, a vlastnost `description` tvoří tělo recenze.

```
<p itemprop="summary">Pizzerie v newyorském stylu přímo
v historickém centru Apexu</p>
<p itemprop="description">
    Jídlo je špičkové. Atmosféra je pro „pizzerii za rohem“
    ideální. Uvnitř je trochu těsno – pokud máte nadváhu, budete
    mít problém dostat se na své místo a pak z něj ven, i prodírat
    se mezi stoly. Dříve dávali jako předkrm zdarma česnekové
    rohlíčky; teď dávají jenom obyčejný chleba a za něco lepšího
    si musíte připlatit. Celkově je to tam ale super.
```

```
</p>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Vlastnosti `location` a `geo` by pro nás neměly být nic nového. (Pokud se zde teprve rozhlížíte, podívejte se na označování adresy Osoby, označování adresy Organizace a označování geolokačních informací dříve v této kapitole.)

```
<p itemprop="location" itemscope
       itemtype="http://data-vocabulary.org/Address">
    <span itemprop="street-address">100 North Salem Street
    </span><br>
    <span itemprop="locality">Apex</span>,
    <span itemprop="region">NC</span>
    <span itemprop="postal-code">27502</span><br>
    <span itemprop="country-name">USA</span>
</p>
<span itemprop="geo" itemscope
       itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="35.730796" />
    <meta itemprop="longitude" content="-78.851426" />
</span>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Poslední řádek představuje nám už známý problém: obsahuje dvě různé informace v jednom prvku. Jméno recenzenta je `Mark Pilgrim` a datum recenze je `31. března 2010`. Jak tyto dvě různé vlastnosti zapíšeme do kódu? Zabalíme každou z nich do vlastního prvku a pak do každého prvku umístíme atribut `itemprop`. Datum v tomto příkladu by stejně mělo používat prvek `<time>`, což nám poskytne přirozený háček, na který zavěsíme náš atribut `itemprop`. A jméno recenzenta můžeme jednoduše zabalit do pomocného prvku `<span>`.

```
    <p>— <span itemprop="reviewer">Mark Pilgrim</span>, naposledy
    aktualizováno
        <time itemprop="dtreviewed" datetime="2010-03-31">
        31. března 2010
        </time>
    </p>
</article>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Tak dobrá, můžeme přejít k hodnocení. To představuje nejobtížnější část označování recenze. Výchozí nastavení slovníku Recenze používá stupnici od 1 do 5, kde 1 je „hrozné“ a 5 „skvělé“. Pokud chcete použít jinou stupnici, určitě můžete. Ale nejdřív si řekneme něco k té výchozí.

```
<p>★★★★☆ (<span itemprop="rating">4</span> hvězdičky z 5)</p>
```

[Sledujte postup s námi! Předtím: review.html, potom: review-plus-microdata.html]

Pokud používáte výchozí stupnici od 1 do 5, jediná vlastnost, kterou musíte do kódu zapsat, je hodnocení samotné (v tomto případě 4). Ale co když chcete použít jinou stupnici? To udělat můžete, jenom musíte deklarovat meze té stupnice. Pokud budete například chtít použít stupnici od 0 do 10, stále byste deklarovali vlastnost `itemprop="rating"`, ale místo přímého uvedení hodnoty hodnocení byste použili vnořený slovník `http://data-vocabulary.org/Rating` pro deklaraci nejhorší a nejlepší hodnoty vaší uživatelské stupnice a skutečné hodnoty hodnocení na této stupnici.

```
<p itemprop="rating" itemscope
   itemtype="http://data-vocabulary.org/Rating">
   ★★★★★★★★★☆
   (<span itemprop="value">9</span> na stupnici od
   <span itemprop="worst">0</span> do
   <span itemprop="best">10</span>)
</p>
```

Normálním jazykem se tu říká: „Produkt, který recenzuji, má hodnocení 9 na stupnici od 0 do 10.“

Říkal jsem už, že mikrodata v recenzi mohou ovlivnit výpis výsledků vyhledávání? Ano, mohou. Tady jsou „syrová data“, která nástroj Google Rich Snippets vytáhl z mé recenze obohacené o mikrodata:

```
Item
    Type: http://data-vocabulary.org/Review
    itemreviewed = Pizzerie u Anny
    rating = 4
    summary = Pizzerie v newyorském stylu přímo v historickém centru
    Apexu description = Jídlo je špičkové. Atmosféra je pro „pizzerii
    za rohem” ideální…
    address = Item(__1)
    geo = Item(__2)
    reviewer = Mark Pilgrim
```

```
    dtreviewed = 2010-03-31

Item
    Id: __1
    Type: http://data-vocabulary.org/Organization
    street-address = 100 North Salem Street
    locality = Apex
    region = NC
    postal-code = 27502
    country-name = USA

Item
    Id: __2
    Type: http://data-vocabulary.org/Geo
    latitude = 35.730796
    longitude = -78.851426
```

A takhle (když si odmyslíme vrtochy Googlu, fázi měsíce a tak dále) by moje recenze mohla vypadat ve výsledcích vyhledávání:

> Pizzerie u Anny: recenze
> ★★★★☆ Autor recenze: Mark Pilgrim – 31. března, 2010
> Tady se zobrazí fragment stránky.
> Tady se zobrazí fragment stránky.
> diveintohtml5.info/examples/review-plus-microdata.html - Archiv - Podobné

Špičaté závorky na mě zas takový dojem nedělají, ale musím uznat, že je tohle docela cool.

10.8 K dalšímu čtení

Zdroje o mikrodatech:

- Live microdata playground[7]
- HTML5 microdata specification[8]

Zdroje o Google Rich Snippets:

- About rich snippets and structured data[9]
- Marking up contact and social networking information[10]

7 http://foolip.org/microdatajs/live/
8 https://html.spec.whatwg.org/multipage/semantics.html#links
9 https://developers.google.com/structured-data/
10 https://developers.google.com/structured-data/rich-snippets/

- Businesses & organizations[11]
- Events[12]
- Reviews[13]
- Review ratings[14]
- Google Rich Snippets Testing Tool[15]
- Google Rich Snippets Tips and Tricks[16]

11 https://developers.google.com/structured-data/customize/contact-points
12 https://developers.google.com/structured-data/rich-snippets/events
13 https://developers.google.com/structured-data/rich-snippets/reviews
14 https://developers.google.com/structured-data/rich-snippets/reviews
15 https://developers.google.com/structured-data/testing-tool/
16 http://knol.google.com/k/google-rich-snippets-tips-and-tricks

11. Upravujeme historii pro zábavu a zisk

11.1 Začínáme

Adresní řádek v prohlížeči je možná nejspecializovanějším běžně používaným prvkem uživatelského rozhraní na světě. Adresy URL najdeme na billboardech, na stěnách vagonů, dokonce i v pouličních graffiti. Spolu s tlačítkem zpět – zdaleka nejdůležitějším tlačítkem v prohlížeči – získáváte účinný způsob, jak procházet dopředu i dozadu celou tou spoustou propojených zdrojů, která se nazývá web.

API historie HTML5 je standardizovaným způsobem, jak upravit historii prohlížeče skriptem. Část tohoto API (navigování v historii) byla k dispozici už v předchozích verzích HTML. Novinkami v HTML5 jsou možnost přidávat do historie prohlížeče položky, viditelně měnit URL v adresním řádku prohlížeče (bez spuštění obnovy stránky) a událost, která se spustí, když jsou tyto položky odstraněny z historie tak, že uživatel zmáčkne tlačítko zpět. To znamená, že URL v adresním řádku prohlížeče může i nadále fungovat jako jedinečný identifikátor aktuálního zdroje, a to i v hodně skriptovaných aplikacích, které ani neprovádějí úplné obnovení stránky.

11.2 Proč

Proč byste měli ručně upravovat adresní řádek prohlížeče? Vždyť přece pomocí jednoduchého odkazu se můžeme dostat na novou URL – tímto způsobem web fungoval 20 let. A bude tak fungovat i nadále. Toto API nechce web nijak podkopávat, právě naopak. V posledních letech weboví vývojáři objevili nové, zajímavé způsoby jak web podkopávat, aniž by si pomáhali nově vznikajícími standardy. API historie HTML5 je ve skutečnosti navrženo tak, aby zajistilo, že URL budou nadále užitečné i v hodně skriptovaných aplikacích.

Vraťme se k základním principům – co vlastně URL dělá? Identifikuje jedinečný zdroj. Můžete na něj přímo odkázat, můžete ho uložit do záložek, můžou ho do svých databází zařadit vyhledávače, můžete ho zkopírovat, vložit a odeslat e-mailem někomu dalšímu, kdo na něj může kliknout a vidět stejný zdroj, který jste původně viděli vy. To jsou všechno vynikající vlastnosti. URL jsou důležité.

Takže chceme, aby jedinečné zdroje měly jedinečné URL. Ovšem zároveň měly prohlížeče odjakživa jedno zásadní omezení: pokud změníte URL, a to i pomocí skriptu, cesta tam a zpět ke vzdálenému webovému serveru a úplná obnova stránky. To stojí čas a zdroje, a pokud přecházíte na stránku hodně podobnou té aktuální, vypadá to jako obzvláštní plýtvání. Na nové stránce se stahuje úplně všechno, i části, které jsou úplně stejné jako na stránce aktuální. Neexistuje způsob jak prohlížeči říct, aby změnil URL, ale stáhl jenom polovinu stránky.

API historie HTML5 vám to umožní. Místo spuštění úplné obnovy stránky můžete použít skript, který v podstatě stáhne jenom polovinu stránky. Tento trik je náročný na provedení a vyžaduje od vás určité úsilí. Sledujete pozorně?

Řekněme, že máme dvě stránky, stránku A a stránku B. Tyto dvě stránky jsou z 90 % totožné; pouze 10 % obsahu se liší. Uživatel jde na stránku A a pak se zkusí přesunout na stránku B. Ovšem vy místo spuštění úplné obnovy stránky tento přesun přerušíte a ručně provedete následující kroky:

- *Načtěte těch 10 % stránky B*, které se liší od stránky A (nejspíš pomocí `XMLHttpRequest`). To bude vyžadovat určité změny vaší webové aplikace na straně serveru. Budete muset napsat kód, který bude vracet jenom těch 10 % stránky B, které se liší od stránky A. Může to být skrytá URL nebo parametr příkazu, které by koncový uživatel normálně neviděl.

- *Prohoďte změněný obsah* (použijte `innerHTML` nebo jiné metody v DOMu). Možná budete muset vynulovat ovladače událostí u prvků v prohozeném obsahu.

- *Aktualizujte adresní řádek v prohlížeči* na URL stránky B za použití speciální metody z API historie HTML5, kterou vám ukážu za chvíli.

Na konci tohoto triku (pokud jste ho provedli správně), dostanete v prohlížeči DOM totožný se stránkou B, stejný, jako kdybyste přešli na stránku B přímo. V adresním řádku prohlížeče se objeví URL totožná se stránkou B, stejná, jako kdybyste přešli na stránku B přímo. Vy jste ovšem na stránku B nikdy nepřešli a nikdy jste neprovedli úplnou obnovu stránky – a to je ten trik. Ale protože „poskládaná“ stránka vypadá stejně a má stejnou URL jako stránka B, uživatel si nikdy nevšimne rozdílu (a neocení práci, kterou jste si dali s tím, že jste mu usnadnili prohlížení).

11.3 Jak

API historie HTML5 tvoří pouze několik metod na objektu `window.history` a jedna událost na objektu `window`. Tyto metody a událost můžete použít pro detekci podpory API historie. Podporu zatím poskytují jen nejnovější verze několika prohlížečů, což tyto techniky řadí do tábora „progresivních vylepšení“.

Podpora history.pushState						
IE	**Firefox**	**Safari**	**Chrome**	**Opera**	**iPhone**	**Android**
·	4.0+	5.0+	8.0+	11.50+	4.2.1+	·

„Ponořte se do psů[1]“ je jednoduchý, ale netriviální příklad používání API historie HTML5. Je vytvořen podle běžného vzoru: jako dlouhý článek s přidruženou vloženou fotogalerií. V podporovaném prohlížeči kliknutí na odkazy Další a Předchozí ve fotogalerii aktualizuje fotografii a *zároveň aktualizuje URL v adresním řádku prohlížeče*, aniž by se spustila úplná obnova stránky. V nepodporovaných prohlížečích – nebo i podporovaných prohlížečích, ve kterých si ale uživatel vypnul skriptování – odkazy jednoduše fungují jako klasické odkazy, které vás dovedou na novou stránku s úplnou obnovou.

Pojďme na „Ponořte se do psů“ a podívejme se, jak funguje. Toto je kód jedné fotografie:

Nabídka ↷

```
<aside id="gallery">
    <p class="photonav">
        <a id="photonext" href="casey.html">Další &gt;</a>
        <a id="photoprev" href="adagio.html">&lt; Předchozí</a>
    </p>
    <figure id="photo">
        <img id="photoimg" src="gallery/1972-fer-500.jpg"
            alt="Fer" width="500" height="375">
        <figcaption>Fer, 1972</figcaption>
    </figure>
</aside>
```

Neděje se tu nic zvláštního. Fotka samotná je prvek `<img>` uvnitř `<figure>`, odkazy tvoří klasické prvky `<a>` a všechno je to zabaleno v `<aside>`. To, že se jedná o klasické odkazy, které skutečně fungují, je důležité. Veškerý kód, který následuje, je v detekčním skriptu. Pokud uživatel používá nepodporovaný prohlížeč, nic z našeho elegantního kódu API historie se nespustí. A samozřejmě existují také uživatelé, kteří mají skriptování úplně vypnuto.

Hlavní ovládací funkce předá každý z těchto odkazů funkci `addClicker()`, která provede nastavení uživatelského ovladače `click`.

```
function setupHistoryClicks() {
    addClicker(document.getElementById("photonext"));
    addClicker(document.getElementById("photoprev"));
}
```

Následuje funkce `addClicker()`. Přebírá prvek `<a>` a přidává ovladač `click`. A právě u ovladače `click` to začíná být zajímavé.

1 http://diveintohtml5.info/examples/history/fer.html

```
function addClicker(link) {
    link.addEventListener("click", function(e) {
        swapPhoto(link.href);
        history.pushState(null, null, link.href);
        e.preventDefault();
    }, false);                                    ← Zajímavé
}
```

Funkce `swapPhoto()` provede první dva kroky našeho triku o třech krocích. První polovina funkce `swapPhoto()` si vezme část URL navigačního odkazu samotného – `casey.html`, `adagio.html`, atd. – a vytvoří URL odkazující na skrytou stránku, která neobsahuje nic než kód vyžadovaný další fotkou.

```
function swapPhoto(href) {
    var req = new XMLHttpRequest();
    req.open("GET",
        "http://diveintohtml5.info/examples/history/gallery/" +
        href.split("/").pop(),
        false);
    req.send(null);
```

Tady je ukázka kódu, který vrací stránka `http://diveintohtml5.info/examples/history/gallery/casey.html`. (Můžete si to ověřit u sebe v prohlížeči, když tuto URL navštívíte přímo.)

```
<p class="photonav">
    <a id="photonext" href="brandy.html">Další &gt;</a>
    <a id="photoprev" href="fer.html">&lt; Předchozí</a>
</p>
<figure id="photo">
    <img id="photoimg" src="gallery/1984-casey-500.jpg"
        alt="Casey" width="500" height="375">
    <figcaption>Casey, 1984</figcaption>
</figure>
```

Přijde vám to povědomé? Mělo by. Je to stejný základní kód, který původní stránka použila pro zobrazení první fotky.

Druhá polovina funkce `swapPhoto()` provádí druhý krok našeho triku o třech krocích: vkládá nově stažený kód na aktuální stránku. Vzpomeňte si, že figure (kontejner pro obrázek), fotka i popisek jsou obaleny v `<aside>`. Takže vkládání nového kódu pro fotku je záležitostí jednoho

řádku – nastavení vlastnosti `innerHTML` prvku `<aside>` na vlastnost `responseText` vrácenou `XMLHttpRequest`.

```
        if (req.status == 200) {
            document.getElementById("gallery").innerHTML =
            req.responseText;
            setupHistoryClicks();
            return true;
        }
        return false;
    }
```

(Také si všimněte zavolání `setupHistoryClicks()`. To je nezbytné pro resetování uživatelských ovladačů události `click` na nově vložených navigačních odkazech. Nastavení `innerHTML` vymaže všechny stopy starých odkazů a jejich ovladačů událostí.)

Nyní se vraťme k funkci `addClicker()`. Po úspěšném prohození fotek zbývá v našem triku o třech krocích poslední krok: nastavení URL v adresním řádku prohlížeče bez obnovy stránky.

Změna ↷

```
history.pushState(null, null, link.href);
```

Funkce `history.pushState()` přebírá tři parametry:

- `state` může být jakákoliv datová struktura JSON. Předává se zpátky ovladači události `popstate`, o kterém se dozvíte více za chvíli. V této ukázce nemusíme sledovat stav, takže jsem ho nechal jako `null`.

- `title` může být jakýkoliv řetězec. Tento parametr v současné době většina hlavních prohlížečů nepoužívá. Pokud chcete nastavit název stránky, měli byste ho uložit v argumentu `state` a nastavit ho manuálně zpětně volanou funkcí `popstate`.

- `url` může být, nepřekvapivě, jakákoliv URL. Toto je URL, kterou chcete zobrazit v adresním řádku prohlížeče.

Vyvolání `history.pushState` okamžitě změní URL v adresním řádku prohlížeče. Je už tedy trik u konce? Ne úplně. Ještě se musíme zmínit o tom, co se stane, když uživatel zmáčkne veledůležité tlačítko zpět.

Obvykle když uživatel přejde na novou stránku (s úplnou obnovou stránky), prohlížeč zařadí novou URL do historie prohlížení a stáhne a zobrazí novou stránku. Když uživatel zmáčkne tlačítko zpět, prohlížeč z historie odebere jednu stránku a znovu zobrazí stránku předchozí. Ale co se stane teď, když jste zkratovali tuto navigaci, abyste se vyhnuli úplnému obnovení stránky? „Přechod dopředu" na novou URL už jste předstírali; teď musíte také předstírat „přechod zpět" na předchozí URL. A klíčem k předstírání „přechodu zpět" je událost `popstate`.

Prestiž ↷

```
window.addEventListener("popstate", function(e) {
    swapPhoto(location.pathname);
});
```

Po použití funkce `history.pushState()` pro zařazení falešné URL do historie prohlížení se stane to, že když uživatel zmáčkne tlačítko zpět, prohlížeč spustí událost `popstate` na objektu `window`. To je vaše šance trik dokončit jednou provždy. Nestačí totiž jen nechat něco zmizet – musíte to nechat znovu se objevit.

V této ukázce spočívá „znovuobjevení" v pouhém prohození fotky za původní, čehož docílíme vyvoláním `swapPhoto()` se současným umístěním. Než se zpětně vyvolá `popstate`, URL viditelná v adresním řádku prohlížeče se změní na předchozí. Globální vlastnost `location` se rovněž aktualizuje na předchozí URL.

Abyste si to lépe představili, projdeme si celý trik znovu krok za krokem od začátku do konce:

- Uživatel načte `http://diveintohtml5.info/examples/history/fer.html`, uvidí povídání a fotku Fer.
- Uživatel klikne na odkaz s popiskem „Další", což je prvek `<a>`, jehož vlastností `href` je `http://diveintohtml5.info/examples/history/casey.html`.
- Místo přechodu na `http://diveintohtml5.info/examples/history/casey.html` s úplným obnovením stránky odchytne uživatelský ovladač `click` na prvku `<a>` kliknutí a spustí svůj vlastní kód.
- Uživatelský ovladač `click` vyvolá funkci `swapPhoto()`, která vytvoří objekt `XMLHttpRequest`, aby synchronně stáhnul úryvek HTML umístěný na `http://diveintohtml5.info/examples/history/`**`gallery`**`/casey.html`.
- Funkce `swapPhoto()` nastaví vlastnost `innerHTML` v prvku obalujícím fotogalerii (prvek `<aside>`), čímž vymění okomentovanou fotku Fer za okomentovanou fotku Casey.

- Konečně náš uživatelský ovladač `click` vyvolá funkci `history.pushState()`, aby ručně změnil URL v adresním řádku prohlížeče na `http://diveintohtml5.info/examples/history/casey.html`.

- Uživatel v prohlížeči klikne na tlačítko zpět.

- Prohlížeč si všimne, že byla do historie prohlížení ručně zařazena URL (pomocí funkce `history.pushState()`). Místo toho, aby přešel na předchozí URL a znovu načetl celou stránku, prostě aktualizuje adresní řádek na předchozí URL (`http://diveintohtml5.info/examples/history/fer.html`) a spustí událost `popstate`.

- Náš uživatelský ovladač `popstate` znovu vyvolá funkci `swapPhoto()`, tentokrát s předchozí URL, která se už v tu chvíli zobrazuje v adresním řádku prohlížeče.

- Opět s použitím `XMLHttpRequest` stáhne funkce `swapPhoto()` úryvek HTML umístěný na `http://diveintohtml5.info/examples/history/gallery/fer.html` a nastaví vlastnost `innerHTML` prvku `<aside>` obalujícího volání, čímž vymění okomentovanou fotku Casey za okomentovanou fotku Fer.

Trik je dokonalý. Všechny viditelné důkazy (obsah stránky a URL v adresním řádku) uživateli naznačují, že přešel o jednu stránku vpřed a pak o jednu stránku zpět. Ale nedošlo k žádné úplné obnově stránky – všechno to byl pečlivě provedený trik.

11.4 K dalšímu čtení

- Session history and navigation[2] ve standardu návrhu HTML5
- Manipulating the browser history[3] v Mozilla Developer Center
- Simple history API demo[4]
- 20 Things I Learned About Browsers and the Web[5], pokročilá ukázka, která využívá API historie a další techniky HTML5
- Using HTML5 today[6] popisuje, jak Facebook využívá API historie
- The Tree Slider[7] popisuje, jak Github využívá API historie
- History.js[8], meta-API pro úpravu historie v novějších i starších prohlížečích

2 https://html.spec.whatwg.org/multipage/browsers.html#history
3 https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Manipulating_the_browser_history
4 http://html5demos.com/history
5 http://www.20thingsilearned.com/cs-CZ
6 https://www.facebook.com/note.php?note_id=438532093919
7 https://github.com/blog/760-the-tree-slider
8 https://github.com/balupton/History.js/

Dodatek A: Vyčerpávající téměř abecední návod k detekování čehokoliv

(Jste zmateni? Jako terminologický úvod si přečtěte Detekce prvků HTML5. Chcete raději vyčerpávající knihovnu? Zkuste Modernizr.)

`<audio>`

```
• return !!document.createElement('audio').canPlayType;
```

`<audio>` ve formátu MP3

```
• var a = document.createElement('audio');
• return  !!(a.canPlayType  &&  a.canPlayType('audio/mpeg;').replace(/no/,
  ''));
```

`<audio>` ve formátu Vorbis

```
• var a = document.createElement('audio');
• return !!(a.canPlayType && a.canPlayType('audio/ogg;
  codecs="vorbis"').replace(/no/, ''));
```

`<audio>` ve formátu WAV

```
• var a = document.createElement('audio');
• return !!(a.canPlayType && a.canPlayType('audio/wav; codecs="1"').
  replace(/no/, ''));
```

`<audio>` ve formátu AAC

```
• var a = document.createElement('audio');
• return !!(a.canPlayType && a.canPlayType('audio/mp4;
  codecs="mp4a.40.2"').replace(/no/, ''));
```

`<canvas>`

```
• return !!document.createElement('canvas').getContext;
```

`<canvas>` – API textu na plátně

```
• var c = document.createElement('canvas');
• return c.getContext && typeof c.getContext('2d').fillText == 'functi-
  on';
```

`<command>`

```
• return 'type' in document.createElement('command');
```

`<datalist>`

```
• return 'options' in document.createElement('datalist');
```

```
<details>
  • return 'open' in document.createElement('details');
```

<form> – validace omezení

```
  • return 'noValidate' in document.createElement('form');

<iframe sandbox>
  • return 'sandbox' in document.createElement('iframe');

<iframe srcdoc>
  • return 'srcdoc' in document.createElement('iframe');

<input autofocus>
    return 'autofocus' in document.createElement('input');

<input placeholder>
  • return 'placeholder' in document.createElement('input');

<textarea placeholder>
  • return 'placeholder' in document.createElement('textarea');

<input type="color">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'color');
  • return i.type !== 'text';

<input type="email">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'email');
  • return i.type !== 'text';

<input type="number">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'number');
  • return i.type !== 'text';

<input type="range">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'range');
  • return i.type !== 'text';
```

```
<input type="search">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'search');
  • return i.type !== 'text';

<input type="tel">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'tel');
  • return i.type !== 'text';

<input type="url">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'url');
  • return i.type !== 'text';

<input type="date">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'date');
  • return i.type !== 'text';

<input type="time">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'time');
  • return i.type !== 'text';

<input type="datetime">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'datetime');
  • return i.type !== 'text';

<input type="datetime-local">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'datetime-local');
  • return i.type !== 'text';

<input type="month">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'month');
  • return i.type !== 'text';
```

```
<input type="week">
  • var i = document.createElement('input');
  • i.setAttribute('type', 'week');
  • return i.type !== 'text';

<meter>
  • return 'value' in document.createElement('meter');

<output>
  • return 'value' in document.createElement('output');

<progress>
  • return 'value' in document.createElement('progress');

<time>
  • return 'datetime' in document.createElement('time');

<video>
  • return !!document.createElement('video').canPlayType;
```

<video> titulky

```
  • return 'src' in document.createElement('track');

<video poster>
  • return 'poster' in document.createElement('video');
```

<video> ve formátu WebM

```
  • var v = document.createElement('video');
  • return !!(v.canPlayType && v.canPlayType('video/webm; codecs="vp8,
    vorbis"').replace(/no/, ''));
```

<video> ve formátu H.264

```
  • var v = document.createElement('video');
  • return !!(v.canPlayType && v.canPlayType('video/mp4;
    codecs="avc1.42E01E, mp4a.40.2"').replace(/no/, ''));
```

<video> ve formátu Theora

```
  • var v = document.createElement('video');
  • return !!(v.canPlayType && v.canPlayType('video/ogg;
    codecs="theora"').replace(/no/, ''));
```

contentEditable

- `return 'isContentEditable' in document.createElement('span');`

Posílání zpráv mezi okny

- `return !!window.postMessage;`

Přetažení

- `return 'draggable' in document.createElement('span');`

Souborové API

- `return typeof FileReader != 'undefined';`

Geolokace

- `return !!navigator.geolocation;`

Historie

- `return !!(window.history && window.history.pushState);`

Místní úložiště

- ```
  try {
      return 'localStorage' in window && window['localStorage'] !==
      null;
  } catch(e) {
      return false;
  }
  ```

Mikrodata

- `return !!document.getItems;`

Offline webové aplikace

- `return !!window.applicationCache;`

Události posílané ze serveru

- `return typeof EventSource !== 'undefined';`

Uložení relace

- ```
  try {
      return 'sessionStorage' in window && window['sessionStorage'] !==
      null;
  } catch(e) {
      return false;
  }
  ```

SVG
- `return !!(document.createElementNS && document.createElementNS('http://www.w3.org/2000/svg', 'svg').createSVGRect);`

SVG v text/html
- `var e = document.createElement('div');`
- `e.innerHTML = '<svg></svg>';`
- `return !!(window.SVGSVGElement && e.firstChild instanceof window.SVGSVGElement);`

Undo
- `return typeof UndoManager !== 'undefined';`

Indexovaná databáze
- `return !!window.indexedDB;`

Webové sockety
- `return !!window.WebSocket;`

Webová databáze SQL
- `return !!window.openDatabase;`

Web Workers
- `return !!window.Worker;`

Widgety: nacházím se v jednom z nich?
- `return typeof widget !== 'undefined';`

XMLHttpRequest: cross-domain požadavky
- `return "withCredentials" in new XMLHttpRequest;`

XMLHttpRequest: odeslat jako formulářová data
- `return !!window.FormData;`

XMLHttpRequest: události průběhu nahrávání
- `return "upload" in new XMLHttpRequest;`

K dalšímu čtení

Specifikace a standardy:

- HTML5[1]
- Geolokace[2]
- Událostí posílané ze serveru[3]
- Indexovaná databáze[4]
- UndoManager[5]
- Webové sockety[6]
- Webová databáze SQL[7]
- Webové úložiště[8]
- Web Workers[9]
- Widgety[10]
- XMLHttpRequest[11]

Knihovny JavaScript:

- Modernizr[12], detekční knihovna HTML5

1 https://html.spec.whatwg.org/multipage/
2 http://www.w3.org/TR/geolocation-API/
3 https://w3c.github.io/eventsource/
4 http://w3c.github.io/IndexedDB/
5 http://rniwa.com/editing/undomanager.html
6 https://w3c.github.io/websockets/
7 http://dev.w3.org/html5/webdatabase/
8 https://w3c.github.io/webstorage/
9 https://html.spec.whatwg.org/multipage/workers.html
10 http://www.w3.org/TR/widgets/
11 https://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html
12 http://modernizr.com/

— Dodatek A: Vyčerpávající téměř abecední návod k detekování čehokoliv

Mark Pilgrim
PONOŘME SE DO HTML5

Přeloženo z anglického originálu knihy *Dive into HTML5*.
1. vydání, 2010
Vydáno nakladatelstvím O'Reilly Media (oreilly.com), USA.

Vydavatel:
CZ.NIC, z. s. p. o.
Milešovská 5, 130 00 Praha 3
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2014
Kniha vyšla jako 10. publikace v Edici CZ.NIC.

ISBN 978-80-905802-6-8 (tištěná verze, PDF)
ISBN 978-80-88168-07-2 (ve formátu EPUB)
ISBN 978-80-88168-08-9 (ve formátu MOBI)

CZ.NIC | SPRÁVCE DOMÉNY CZ

Nejnovější verze značkovacího jazyka HTML výrazně mění způsob vývoje webových aplikací. Pokud stále nic nevíte o nových funkcích HTML5, nyní je čas to zjistit. Tato kniha poskytuje ucelený pohled na HTML5 a jeho prvky a atributy. Dozvíte se mimo jiné, jak pomocí HTML5 vyvinout aplikace, které běží, i když je uživatel zrovna bez připojení, nebo jak zobrazit video přímo v prohlížeči, aniž byste se museli spoléhat na zásuvné moduly.

O autorovi Mark Pilgrim se nesmazatelně zapsal do povědomí pythonovské komunity už svou knihou „Dive Into Python“, ve které originálním a nezapomenutelným způsobem přiblížil čtenářům osobitý styl programování v tomto jazyce, aby se o několik let později připomenul ještě výrazněji s knihou „Dive Into Python 3“, která je stejně originálním a zábavným způsobem věnována jeho nejnovější verzi. S podobným nadšením se však zabývá i dalšími tématy: jeho nejnovější kniha „Dive Into HTML5“ je čtivým úvodem do problematiky posledního hitu na poli předávání informací na Internetu – standardu HTML5.

O edici Edice CZ.NIC je jedním z osvětových projektů správce české domény nejvyšší úrovně. Cílem tohoto projektu je vydávat odborné, ale i populární publikace spojené s Internetem a jeho technologiemi. Kromě tištěných verzí vychází v této edici současně i elektronická podoba knih. Ty je možné najít na stránkách knihy.nic.cz.

knihy.nic.cz

ISBN 978-80-905802-6-8

9 788090 580268 >