jQuery

jQuery (1) – javascriptový framework, který (skoro) každý používá

Seriál, který tu chybí

Kdo chce začít používat jQuery, může zjistit, že toho tu k němu moc není, zejména ne uceleného a aktuálního. Je samozřejmě možnost vybrat si z nepřeberného množství materiálů v angličtině, ale našinec dá často raději přednost českým textům. Tento seriál má za cíl právě vyplnit tuto „díru" a poskytnout všem zájemcům ucelené informace o použití frameworku, včetně návodů, jak udělat to či ono. První část seriálu bude věnována základnímu jQuery, druhá část GUI rozšíření jQuery UI a třetí dotykově optimalizovanému jQuery Mobile.

Co je jQuery

Asi dnes už málokdo, kdo se zabývá tvorbou webů nebo dokonce vývojem webových aplikací, neví, o co se jedná. Webové aplikace pro klientskou stranu (běžící ve webovém prohlížeči) se nejčastěji píší v jazyce JavaScript. Ten je kvalitně podporován ve všech běžně používaných moderních prohlížečích a bývá také standardně zapnutý. Proto je jazykem první volby.

Jazyk sám o sobě ale samozřejmě nestačí. Je potřeba nějak interagovat s daty, která jsou prohlížečem zobrazována, načítat data ze serveru či je tam naopak odesílat. V prohlížečových implementacích JavaScriptu je k dispozici aplikační rozhraní, které to umožňuje. Lze přistupovat k objektovému modelu dokumentu (DOM), posílat požadavky na server, využívat různá specializovaná API prohlížečů atd.

Jsou tu ale dva problémy. Jednak je poskytované rozhraní jaksi „spartánské". Například triviální úkol spočívající v posunu nějaké vizuální komponenty po stránce vyžaduje zavolat několik funkcí, testovat návratové hodnoty apod. Pokud je aplikace podobnými operacemi doslova protkaná, nebude nikoho bavit je implementovat přímo, proto si dříve či později napíše funkce, které to zjednoduší.

Druhým problémem je různorodost prohlížečů. Přestože existují určité standardy, ne vždy se jich tvůrci prohlížečů drží. A co hlavně, ne všechno je standardy upraveno. Chcete-li například z JavaScriptu poslat požadavek na server, v závislosti na prohlížeči a jeho verzi se bude postup mírně lišit. I tady je tedy potřeba – má-li aplikace fungovat všude – udělat abstrakční vrstvu, která proměnlivé věci ponechá dole a ve vlastní aplikaci se budou funkce volat vždy stejně.

Ovšem proč si vyvíjet vlastní řešení, když už tu takové existuje? Je jím například právě jQuery. Zjednodušuje rutinní operace a obaluje prohlížečově závislé části API, takže psaní aplikací (i těch nejjednodušších) je mnohem snazší.

Framework jQuery je svobodný software šířený pod permisivní licencí typu MIT, takže není problém ho využívat ani v proprietárních aplikacích. Jeho kód je datově velmi úsporný a navíc lze případně využít různé CDN, takže mohou aplikace sdílet jednou načtený soubor umístěný mimo server vlastní aplikace (je to možnost, ne nutnost – takže o bezpečnost a soukromí není třeba mít obavy).

Jak jQuery funguje

O jQuery se často hovoří jako o „nečistě" navrženém frameworku, protože příliš nectí objektový model. To je samozřejmě pravda a mimo jiné i proto vznikla řada jiných javascriptových frameworků. Na druhou stranu je to právě použitý způsob práce, co přiblížilo jQuery mnoha vývojářům.

Syntaxe jQuery je jednoduchá a způsob psaní kódu úsporný, takže se „neupíšete k smrti". V mnoha případech lze využít řetězení metod, tedy psát ještě méně kódu a současně ho učinit přehlednější a názornější.

Znáte CSS a práci se selektory? To je přesně ono. V jQuery je na nich založeno téměř vše. Stručně řečeno: pomocí selektorů si vyberete množinu objektů v DOM a provedete s nimi nějakou operaci, například přesun ve stromě DOM, změnu CSS stylu, úpravu obsahu, vložení nových objektů apod.

Další významnou vlastností jQuery je intenzivní práce s obsluhou událostí, a to jak těch vzniklých v prohlížeči, tak i řady dalších (například vrácení různých HTTP kódů v odpovědi na požadavek). S tím souvisí již zmíněné prohlížečově nezávislé a navíc snadno použitelné posílání HTTP požadavků.

Nelze zapomenout ani na různé utility (pro různé konverze dat, práci s textem atd.) a na modularitu jQuery. Chybí-li vám nějaká funkcionalita, můžete ji snadno dodat pomocí pluginů. Existuje jich mnoho již hotových (pod různými licencemi), není ale nic těžkého si vytvořit vlastní.

Tvoříme lepší GUI, aneb jQuery UI

Základní jQuery je nízkoúrovňový framework, který toho umí hodně, ale neřeší vyšší vrstvy, tedy například uživatelské rozhraní. Pro vylepšení GUI je tu nadstavba nazvaná jQuery UI. Chcete umožnit posouvání nějakého prvku myší? Žádný problém, je to v podstatě věc jediného zavolání funkce. Podobně třeba implementace animací nebo přidání bohatších widgetů (kalendář, dialog a řada dalších).

Snazší vývoj také pro mobily

Nejnovějším členem rodiny je jQuery Mobile. Je to opět nadstavba jQuery a přidává funkcionalitu vhodnou pro mobilní telefony a tablety. Do té patří například speciální „dotykové" události, widgety, práce s tématy nebo responzivní rozložení prvků.

Základy práce

Příště už se podíváme na základy práce s jQuery – tedy jak vůbec začít (kde jQuery získat, jak zahrnout do dokumentu), jak používat selektory, jak volat objektové metody a jak zabránit případné kolizi s jinými frameworky.

jQuery (2) – získání, první kroky, selektory, řešení kolizí

Získání jQuery

Existuje několik možností, jak framework jQuery získat. Každá má své výhody a nevýhody.

Ruční stažení a umístění na server

V mnoha případech stačí navštívit web projektu a odtamtud si stáhnout buď nejnovější verzi, nebo nějakou, která splňuje určité požadavky. Vždy se jedná o jeden jediný soubor. Existují ovšem dvě varianty: komprimovaná (pro ostrý provoz) a nekomprimovaná (pro vývoj a ladění).

Použít nejnovější verzi či zůstat u nějaké starší? Pokud má aplikace fungovat i v Internet Exploreru verzí starších než 9 (tedy na Windows XP), je potřeba použít řadu 1. V moderních prohlížečích poběží verze řady 2 rychleji.

O aktualizace se musíte trvale starat ručně (případně si nějak automatizovat ve vlastní režii). Současně ale máte tento proces pod kontrolou, nemůže tedy přijít žádné překvapení.

Instalace pomocí nástroje Bower

Nástroj Bower umožňuje poměrně snadno a komplexně řešit správu webových aplikací, včetně vyhodnocování závislostí. Použijete-li ho pro svůj projekt, jen uvedete tento framework mezi závislosti a Bower se sám postará o instalaci a pozdější aktualizace.

Je jasné, že pro jednodušší případy je to „kanon na vrabce", protože jen příprava balíčku pro Bower může zabrat více času než ruční instalace jQuery po celou dobu existence webu. Místo Boweru můžete použít také podobný nástroj Composer nebo některé další.

Využití CDN

Asi úplně nejjednodušší je jQuery odnikud nezískávat – s tím, že se prostě přímo stáhne odněkud odjinud. To „odjinud" zde znamená nějakou síť pro doručování obsahu (*Content Delivery Network*, *CDN*), která disponuje servery rozmístěnými po světě a poskytující potřebný soubor.

Soubory jQuery lze získat z celé řady CDN, z těch nejpoužívanějších například jQuery CDN (běží na MaxCDN; tuto síť doporučují tvůrci jQuery), Google, Microsoft nebo jsDelivr. Místo lokálního souboru prostě v dokumentu použijete absolutní URL.

CDN má výhodu v tom, že není potřeba nic instalovat/aktualizovat, navíc pokud se využívá tentýž soubor (se stejným URL) ve více aplikacích, stačí ho přenést do prohlížeče jen jednou. Znamená to ale spoléhat se na cizí službu, kde nemáte obsah pod kontrolou (může být i změněn/podvržen) a do určité míry to také narušuje soukromí.

Vložení do dokumentu

Aby framework fungoval, je potřeba jeho soubor vložit do (X)HTML dokumentu, kde se bude používat, a to vždy před javascriptový soubor, kde se bude využívat. Zda to bude na začátku nebo na konci, není z funkčního hlediska až tak podstatné. Je ale obecně lepší vkládat javascriptové soubory na začátek, protože se tím urychlí jejich načtení (může probíhat paralelně s dokumentem).

Jakékoli operace by se měly dělat nejdříve po úplném načtení dokumentu (bude o tom ještě řeč). Při používání cizích skriptů je někdy lepší dát je na konec (to se ale jQuery netýká); případně lze využívat atributy async a defer, ale to už přesahuje rámec článku.

Typicky může začátek dokumentu vypadat například takto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script src="/js/jquery.js" type="text/javascript"></script>
<script src="/js/app.js" type="text/javascript"></script>
```

Tento kus kódu vloží do dokumentu soubor pro jQuery umístěný na serveru v adresáři js a ze stejného místa pak i aplikační javascriptový soubor. Pro variantu s CDN by to vypadalo takto:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script src="http://code.jquery.com/jquery-2.1.3.min.js" type="text/javascript"></script>
<script src="/js/app.js" type="text/javascript"></script>
```

Bude se vkládat komprimovaná verze 2.1.3. Vyjde-li aktualizace, stačí změnit URL.

Základy práce s jQuery, volání metod

Po vložení jQuery do dokumentu s ním lze začít pracovat. Asi nejlepší bude ukázat si to na jednoduchém příkladu:

```
$(document).ready(function() {
$('.clickable').click(function() {
alert('Klik');
}
});
```

Příklad ukazuje hned několik důležitých věcí. Především je to symbol dolaru ($), který symbolizuje objektovou třídu jQuery. Používá se, protože je to jeden znak a je dobře odlišitelný od všeho ostatního. Místo něj lze použít i přímo jQuery, ale kromě zvláštních případů (viz dále) je to zbytečné.

Dále je to vytvoření instance třídy jQuery pro nějaký objekt dokumentu, v tomto případě pro dokument samotný (document). V konstruktoru lze použít nejen přímo objekt, ale také selektor, jak je tomu zde v případě CSS třídy .clickable (o selektorech bude řeč později).

Práce s jQuery spočívá především ve volání objektových metod – tady je to vidět na metodách ready()a click(). V obou případech jsou to metody nastavující obsluhu událostí (událostem bude věnován samostatný díl seriálu). Metodou ready() se nastaví funkce volaná v okamžiku, kdy je daný objekt, tedy zde dokument, plně připraven fungovat. Právě toto je dobrá praxe při používání JavaScriptu nad dokumentem, protože zaručuje, že je kompletně načtený a zpracovaný prohlížečem.

Metoda click() nastavuje obsluhu kliknutí myší – v příkladu se po kliknutí zobrazí informativní zpráva.

V některých případech se volají metody nikoli na instanci třídy jQuery, nýbrž přímo na této třídě. To je třeba případem různých obecně použitelných utilit:

```
var s = $.trim($('#test').text());
```

Tento kus kódu získá text z dokumentového objektu vybraného podle CSS identifikátoru test a odstraní z jeho začátku a konce „bílé znaky" (mezery, konce řádků apod.).

Selektory a jejich použití

S jQuery se pracuje obvykle tak, že se nějak vyberou objekty dokumentu a zavolá se pro ně nějaká metoda, která něco dělá. Pro výběr lze použít buď přímo daný objekt (pokud je k dispozici), nebo častěji tzv. *selektory*. Ty známe především z CSS, kde se jimi určuje, na co se budou vztahovat definice stylů.

Tady je to velmi podobné jako v CSS, odlišností je jen málo. Opět nejlépe napoví příklad:

```
var jq = $('*');
var jq = $('a');
var jq = $('.abc');
var jq = $('#xyz');
var jq = $('#a, #b');
var jq = $('input:enabled');
```

Všechny řádky přiřadí proměnné jq instanci třídy jQuery, která bude aplikovat operace na množinu dokumentových objektů vybraných daným selektorem. První řádek vybere úplně všechno. Druhý vybere všechny elementy a, tedy hypertextové odkazy. Ve třetím se vybírají elementy CSS třídy abc, ve čtvrtém s CSS identifikátorem xyz.

Pátý řádek ukazuje vícenásobný výběr (položky seznamu se oddělují čárkou), zde opět s identifikátory. A konečně poslední využívá pseudotřídu enabled, tedy povolené (nezakázané) elementy input.

Při používání selektorů je potřeba si uvědomit několik věcí. Jednak ne všechny metody jQuery mohou pracovat s více objekty (pak se metoda použije jen na první z nich). Dále ne všechny selektory pro výběr téhož jsou stejně efektivní – některé jsou výrazně rychlejší než jiné, ty by se měly preferovat. A konečně by výběr měl být vždy co nejužší, protože při zbytečně širokém výběru se mohou do množiny dostat i nechtěné objekty a způsobit nesprávnou funkci, o výkonnostních dopadech nemluvě.

Pokud je například potřeba vybrat jen odstavce s třídou centered, je vhodnější použít selektor p.centered, než jen .centered, byť se třída jinde nepoužívá. Později totiž může být takové použití zavedeno (a bude mít logiku), což povede k provádění operací i nad objekty, kde se provádět nemají.

Příště se na selektory podíváme z praktičtější stránky, a to právě včetně různých záludností a úskalí.

Řešení kolizí

Občas je potřeba používat více frameworků najednou – například z historických důvodů nebo proto, že pro každý z nich je k dispozici nějaké hotové řešení, které je vhodné použít. A problém spočívá v tom, že se některé věci vyskytují ve více frameworcích a mohou vzájemně kolidovat.

Typicky se tento problém týká identifikátoru třídy jQuery, tedy znaku dolaru – používají ho i další frameworky, takže když se zkombinují, je výsledkem nepředvídatelné chování, často naprosté selhání aplikace. Jak z toho ven?

První možností je místo dolaru používat dlouhý název, tedy jQuery. To lze v podstatě vždycky, znamená to ale více psaní a nutnost upravovat existující kód. Mnohdy je to ale metoda volby.

Druhou možností je vrácení kontroly nad symbolem původnímu určení. Tady jsou dva příklady:

```
$.noConflict();

$.noConflict();
jQuery(document).ready(function($) {
  …
});
```

První příklad pouze uvolní dolar k původnímu použití, kdy už nelze dále používat pro účely jQuery. Ve druhém příkladu se navíc dolar nadefinuje lokálně (uvnitř funkce volané při připravenosti dokumentu), aniž by se to dotklo globální úrovně. Dalších možností je celá řada, najdete je v dokumentaci.

Selektory v praxi

To by bylo k úplným základům všechno. Příště se podíváme na různé praktické příklady použití selektorů, včetně možných úskalí, která se zde objevují.

jQuery (3) – selektory v praxi

Není selektor jako selektor

Selektory jsou velmi silný nástroj pro výběr objektů v objektovém modelu (DOM). Významnou pomocnou ruku poskytuje přímo webový prohlížeč, konkrétně nativní javascriptová metoda querySelectorAll() volaná na rodičovském objektu. Touto metodou lze vyselektovat objekty v míře, kterou daný prohlížeč podporuje, a to velmi rychle.

Ovšem jednak ne každý prohlížeč nabízí stejnou podporu (míra podpory specifikací CSS je v různých prohlížečích a jejich verzích poněkud odlišná), navíc některé užitečné selektory nejsou aktuálně k dispozici nikde.

Selektory nepodporované metodou querySelectorAll() řeší jQuery vlastní implementací (rozšířením), takže z pohledu aplikace všechno funguje, jak má. Ale rozdíl ve výkonu je značný. Je proto dobré se před použitím nějakého méně obvyklého selektoru přesvědčit, jak s ním bude jQuery v obvyklých případech nakládat.

Proto například nepoužívejte tyto selektory (v tomto i dalším příkladu budou uváděny vždy jako parametr konstruktoru jQuery, případně jako parametr metody volané na instanci):

```
$(':button')
$(':header')
$(':image')
$(':root')
```

Místo toho můžete použít následující, mnohem rychlejší ekvivalenty:

```
$('button, input[type="button"]')
$('h1, h2, h3, h4, h5, h6')
$('[type="image"]')
$('html')
```

První selektor vybírá tlačítko (element input nebo button), druhý nadpisy všech úrovní, třetí elementy typu „image“ (typicky to budou elementy input). Čtvrtý vybírá kořenový objekt celého dokumentu, což je v HTML vždy element html.

Některé selektory implementované v jQuery nemají svůj nativní ekvivalent. Pak je žádoucí provést „předfiltraci“, tedy použít například filtraci podle elementu. Jinak totiž selekce funguje tak, že se procházejí úplně všechny objekty (selektor '*') a u každého se rozhoduje, zda vyhovuje definovaným podmínkám. Opět pár příkladů zbytečně pomalých selektorů:

```
$('tr :even')
$(':animated')
```

A jejich vylepšené, rychlejší verze:

```
$('tr td:even')
$('div:animated')
```

U prvního případu je cílem použít selektor pro výběr sudých buněk tabulky. Selektor nejenže je pomalý, ale navíc spoléhá na to, že potomkem řádku tabulky (tr) bude vždy buňka tabulky (td). To ale není pravda, protože se používá také buňka záhlaví tabulky (th).

Druhý příklad tak trochu předbíhá o hodně dílů seriálu vpřed, až do jQuery UI. Proto prosím nyní nepřemýšlejte o tom, co ten animovaný objekt znamená. Budou se animovat jen a pouze elementy div, proto lze využít uvedený předvýběr. Celé to lze ještě dále urychlit, ale k tomu se vrátíme až za chvíli.

Dobrá praxe při používání selektorů

Pryč s univerzálním selektorem

Univerzální selektor ('*') je doslova žroutem výkonu. Znamená procházet celý strom DOM a pracovat se všemi objekty. Tento selektor se objevuje implicitně i tam, kde se použije nějaká pseudotřída. Málokdy je potřeba tento selektor použít – spíše by se dalo říct, že pokud k němu vše směřuje, je někde chyba.

Používejte identifikátory

Ve všech případech, kdy potřebujete vyselektovat jeden konkrétní objekt, je tou správnou cestou jeho identifikátor. Tedy například v HTML bude:

```
<img src="images/logo.png" alt="Logo" class="page-image" id="logo-image">
```

Potom se tento obrázek vyselektuje nejlépe takto:

```
$('#logo-image')
```

Identifikátor by měl být v rámci dokumentu vždy unikátní (požadavek specifikace HTML). Pokud není, je dokument „rozbitý“ a chování selektoru je pak nepředvídatelné.

Používejte dostatečně specifické selektory

Je lepší selektovat přímo jen ty objekty, kterých se operace týká. Nedostatečně specifické selektory mohou způsobit neočekávané chování, navíc mohou být pomalejší. Mějme třeba tyto dva příklady:

```
$('.external-link')
$('p a.external-link')
```

Cílem je udělat nějakou operaci s odkazem v textu, který patří do třídy „external-link". Ta se sice aktuálně nikde nevyskytuje, ale v budoucnu se vyskytovat může. Podobně lze působnost selektoru omezit na odstavce, protože pokud by se nějaký odkaz dané třídy „zatoulal" jinam, aplikovala by se operace i na něj, přestože to není cílem.

Nepřehánějte to se specifičností selektorů

Nic se nemá přehánět, platí to i u specifičnosti selektorů. Ve snaze dodržet v maximální míře předchozí doporučení, může vzniknout i něco jako:

```
$('form#mail input.mail-form')
$('#mail input')
```

Situace je však taková, že do třídy „mail-form" patří všechny elementy input ve formuláři. Proto je uvedení třídy zbytečné. Ještě zbytečnější je uvedení elementu v kombinaci s identifikátorem, protože zcela vyhovuje samotný identifikátor a elementem se to nijak nevylepší.

Pozor však na situaci, kdy chceme vybrat sice podle identifikátoru, ale jen v případě, že objekt splňuje nějaké podmínky (což může být třída, pseudotřída, případně i element). Pak je samozřejmě kombinace identifikátoru a další podmínky zcela na místě.

```
$('#mail div input')
```

Toto je další ukázka zbytečně specifického selektoru. Většinou nemá žádný smysl přidávat do selektoru další úrovně (pokud se v DOM nacházejí), jen to zbytečně zpomaluje selekci a může to být zdrojem problémů při případných pozdějších změnách struktury.

Využijte metody

Tohle je opět trochu předběhnutí v seriálu. Instance jQuery mohou volat různé selekční metody, které vykonají stejnou práci jako selektory (obecně toho ale umějí více), ovšem často rychleji. Můžeme mít například selektory:

```
$('#mail input')
$('#mail > input')
$('#mail:parent')
```

Ty lze přepsat do následující podoby:

```
$('#mail').find('input')
$('#mail').children()
$('#mail').parent()
```

Metoda find() hledá v celém podstromě, metoda children() vybírá jen přímé potomky, metoda parent()vybírá rodičovský objekt. Všechny tři mohou mít (tady je to konkrétně použito v prvním případě) jako argument další selektor, který se použije pro výběr. V jQuery je celá řada dalších podobných užitečných metod, na které se také později podíváme. Jejich výhodou je nejen vyšší rychlost při správném použití, ale také větší flexibilita. Můžeme si totiž dovolit udělat něco jako toto:

```
var form = $('#mail');
var txt = form.find('input[type="text"]');
var subm = form.find('input[type="submit"]');
```

Je snad už na pohled zřejmé, jaké výhody zde přináší použití metod oproti tomu, aby se pokaždé používal kompletní selektor. První řádek vyselektuje objekt formuláře, další řádky už pak hledají v podstromě pod formulářem. Pokud se například změní identifikátor formuláře, stačí ho změnit na jednom místě, není už potřeba sahat jinam. Současně je taková selekce rychlejší.

```
$('li:odd')
$('li').filter(':odd')
```

Závěrečný příklad ukazuje rozseknutí selektoru na dvě části. Nová je tu metoda filter(), která filtruje pomocí dalšího selektoru. Výsledek je stejný, jako kdyby se použil kompletní selektor, ovšem podobně jako u výše uvedeného použití metody parent() je to rychlejší.

Pohyb ve stromě DOM pomocí jQuery

Poslední úsek článku, který už trochu „naťukl" některé metody, je v podstatě úvodem pro článek příští. Často se totiž potřebujeme různě pohybovat po stromě DOM a pracovat s jednotlivými objekty či jejich skupinami. Na to má jQuery jak již uvedené metody, tak i řadu dalších a podrobněji se na to podíváme příště.

jQuery (4) – pohyb ve stromě DOM

Pár slov o stromě DOM

Objektový model dokumentu (DOM) je v zásadě tvořen stromem, který má nějaký *kořen* a pod ním jsou různé větve zakončené *listy*, tedy objekty bez dalších potomků (nerodičovské objekty). Webový prohlížeč si při načítání stránky a zpracování zdrojového kódu (X)HTML vytváří tento strom, se kterým lze následně pracovat.

Tato „práce" spočívá jak v přístupu k jednotlivým uzlům (objektům), tak i v nakládání s celými větvemi, podstromy. Lze přidávat nové objekty, existující odebírat nebo přesouvat, případně kopírovat jinam – to vše právě i hromadně, po celých podstromech. V rámci stromu se lze i pohybovat, tedy vyjít od jednoho objektu a přecházet jinam.

Pohybujeme se ve stromě

Pohyb směrem dolů

Často je potřeba postupovat ve stromě do větší hloubky. Získáme přístup k nějakému objektu a potřebujeme pracovat s jeho potomky nebo potomky potomků atd. Něco už se objevilo v minulém článku, možností je ale mnohem více. Podívejme se na tento příklad:

```
var o = $('#abcd');
var cld1 = o.children();
var cld2 = o.children('p');
var sub = o.find('p');
```

První řádek slouží k získání instance jQuery pro nějaký objekt s identifikátorem „abcd". Z něj se na druhém řádku vychází pro získání instance pro všechny přímé potomky objektu (až na výjimky, viz dále). Třetí řádek vybere jen ty potomky, kteří mají element p (každého jistě napadne, že by zde šlo také vyjít z instance cld1a použít metodu filter()). A konečně poslední řádek prohledá celý podstrom na elementy p.

Pokud to lze, je vhodnější pro výběr přímých potomků používat vždy metodu children(), přestože někdy může dávat stejné výsledky jako find(). Lze tak předejít neočekávanému chování a je to také rychlejší. Pozor však na případy, kdy se z nějakého důvodu vloží mezi úrovně například „neutrální" element div, protože tím se pro children() změní situace, aniž by byl na stránce vidět nějaký rozdíl.

Další úskalí spočívá v tom, že children() nevrátí všechny objekty, nýbrž jen elementy. Textové a komentářové objekty (text neobalený žádným elementem a HTML komentáře) tato metoda ignoruje. Většinou je takové chování žádoucí, protože nás tyto objekty obvykle nezajímají. Pokud je však potřebujeme, musí se to udělat jinak (k příkladu si prosím přimyslete první řádek z toho předchozího – bude to tak i u dalších příkladů):

```
var all = o.contents();
```

Pozor, metoda contents() je bezparametrická, nelze použít žádný selektor jako filtr (pokud je potřeba, je to řešitelné následným zavoláním metody filter())!

Pokud je v instanci, pro niž se metody volají, více objektů, bude to fungovat tak, jako kdyby se to zavolalo pro každý objekt zvlášť a získané množiny objektů se sloučily do jedné – vrácená instance tedy bude obsahovat všechny. Lze tak vlastně procházet více podstromů najednou.

Pohyb směrem nahoru

Opačným případem je, že máme instanci jQuery pro nějaký objekt „dole" a potřebujeme se dostat směrem vzhůru:

```
var par1 = o.parent();
var par2 = o.parent('.aaa');
var par3 = o.parents('div');
var par4 = o.parentsUntil('body');
```

První případ je nejjednodušší, metoda vrací instanci jQuery pro rodičovský objekt. Ten je standardně jeden, pro objekt nejvyšší úrovně (document) není ale samozřejmě žádný. Také tady platí, že pokud je v instanci jQuery více objektů, rodič se najde pro každého a vloží se do cílové množiny; obrací se však pořadí (oproti původnímu) a odstraňují duplikáty.

Druhý řádek přidává filtr podle selektoru. Buď se tedy najde jeden rodič nebo žádný. Ve třetím řádku je metoda, která hledá všechny rodiče v cestě nahoru ke kořeni. Tady se typicky využívá filtr, protože nás nezajímají objekty všech typů, ale třeba právě uvedený div. Objekty se přidají do množiny v pořadí hledání, tedy opačně než v dokumentu.

Metoda parents() končí na elementu html, výše už nejde. Ke kořenovému objektu celého dokumentu (tedy document; je instancí javascriptové třídy HTMLDocument) se lze dostat pouze metodou parent()zavolanou na elementu html (i když je otázka, k čemu by to bylo dobré).

Na posledním řádku je pak metoda hledající všechny rodiče až po definovaného (který už do množiny zařazen není). Množinu lze omezit filtrem (druhý argument metody), jako „zarážku" pro hledání lze využít jak selektor (jak je tomu v příkladu), tak i instance elementu nebo objektu jQuery, což se často hodí.

```
var ul = o.closest('ul');
```

Tato metoda prohledává strom směrem vzhůru, počínaje aktuálním objektem, než nalezne objekt odpovídající selektoru (je to vlastně find(), ovšem nahoru). Metoda se hodí k nalezení nadřazeného objektu, u kterého nevíme, kde se přesně ve stromě nachází.

Pohyb na stejné úrovni

Ve stromě se lze pohybovat i v rámci stejné úrovně, tedy přistupovat k sourozeneckým objektům. Opět pár příkladů:

```
var sibs = o.siblings('div.foto');
var sib1 = o.siblings().first();
var sib2 = o.next('p');
var sib3 = o.siblings().addBack();
```

První řádek vybere všechny sourozence objektu, které jsou elementem div a mají třídu „foto". Potřebujeme-li pouze prvního sourozence, použijeme druhý řádek – metoda first() ostatní zahodí. Třetí příkaz vybere následujícího sourozence (podle pořadí, jak jdou v dokumentu za sebou), ale jen pokud má element p (nefunguje to tedy tak, že by se nejdřív použil selektorový filtr a teprve potom vybral objekt), jinak vrátí prázdnou množinu.

Čtvrtý řádek vybere všechny sourozence a přidá k nim i původní objekt – jinak řečeno, vybere všechny objekty na dané úrovni.

Když potřebujeme něco složitějšího

Někdy nestačí jen prostý pohyb ve stromě a potřebujeme nějaký složitější výběr objektů.

```
var ht = $('table').has('th');
```

Potřebujeme najít objekt, který má pod sebou objekty určitých vlastností. Například to může být tabulka, která obsahuje hlavičkové buňky, jako je tomu ve výše uvedeném příkladu. Poslouží zde metoda has(), které stačí dát selektor, případně instanci typu Element, a ta už zjistí, který objekt má někde pod sebou „ty správné" potomky.

```
var ht = $('table').has('th').not($('table').has('td'));
```

Tohle je trochu „divočejší" rozšíření předchozího příkladu. Vybere tabulky, které mají hlavičkové buňky, ale současně nemají žádné obyčejné. Metoda not() odstraňuje objekty z výběru buď na základě selektoru, nebo může využít i instanci jQuery s množinou objektů (případně lze použít i funkci, ale to teď ponecháme stranou).

Všechny zde popsané metody, plus ještě pár dalších, které jsou v jQuery k dispozici (a ke kterým se dostaneme někdy později), lze samozřejmě všelijak kombinovat. Nic by se ale nemělo přehánět, protože nadměrný pohyb ve stromě zbytečně spotřebovává výkon, nehledě na to, že se ve složitějších konstrukcích už nemusí nikdo vyznat.

A teď DOM upravíme...

Jak výběr pomocí selektoru, tak i pohyb ve stromě bývají obvykle jen přípravou na to, že se bude s objekty nebo celými podstromy něco dělat. Příště se podíváme na to, jak upravovat obsah objektů, přidávat nové, přemísťovat je a odstraňovat.

jQuery (5) – manipulace s objekty

Jak změnit obsah objektu

Změny obsahu objektů DOM patří mezi vůbec nejčastější činnosti prováděné v JavaScriptu. Ať už se změny provádějí na základě nějakých činností v prohlížeči nebo podle dat získaných ze serveru, v moderních webových aplikacích se mění objekty „jak na běžícím pásu".

Změna textu

Nejjednodušší je změna textového obsahu. Mějme například následující fragment HTML:

```
<body>
<h1>Nadpis<h1>
…
</body>
```

Chceme například změnit nadpis (tj. element h1) na nějaký konkrétní text. Pomocí selektoru tedy vybereme příslušný objekt a metodou text() mu nastavíme textový obsah:

```
$('h1').text('Nový nadpis');
```

Vybere-li selektor více objektů, nastaví se text všem. Změna se na zobrazené stránce projeví okamžitě. Metoda text(), podobně jako řada dalších podobných, neslouží jen ke změně, ale také k získání aktuální hodnoty:

```
var t = $('h1').text();
$('h1').text(t + ' nový');
```

Je zřejmé, co tento kód udělá – zjistí text v h1, připojí za něj další řetězec a ten opět nastaví. Pozor však na jednu věc: zatímco v nastavovacím použití metoda text() nastaví danou hodnotu všem objektům vybraných selektorem, při získávání textu se spojí veškerý textový obsah všech vybraných objektů včetně jejich podstromů. Každý prohlížeč však může odlišně naložit s „bílými znaky" (mezerami, konci řádků apod.), takže je lepší se vyhýbat situacím, kdy se získává text z více než jednoho objektu.

Metodu nelze použít u objektů pro vstup dat (input, textarea apod.). Bude o tom řeč později.

Změna HTML

Metoda text() pracuje s holým textem. Pokud se jí předá HTML kód, převede znaky lomených závorek na HTML entity, takže se zobrazí jako zdrojový kód místo toho, aby se interpretoval. Pokud potřebujeme, aby se interpretoval (což bývá dost často), je potřeba použít jinou metodu:

```
$('h1').html('<a href="odkaz.html">Nadpis</a>');
```

Tato metoda udělá to, co bychom asi očekávali – vloží do objektu HTML kód. Ten se samozřejmě zpracuje, jako kdyby byl přímo v dokumentu. To znamená, že se případně i vytvoří nové objekty v DOM. V daném případě se vytvoří objekt pro element a, tedy pro odkaz. Hned vzápětí už lze s novými objekty běžným způsobem pracovat.

Toto se velmi často využívá ve spojení s daty vyžádanými ze serveru (AJAX – *Asynchronous JavaScript And XML*; název je trochu zavádějící, ale zažitý). Na základě požadavku přijde fragment HTML a ten se vloží na „správné" místo v DOM, kde nahradí původní kód.

Podobně jako u metody text() lze i zde vkládat na více míst najednou. Pozor na použití identifikátorů – ty by měly být unikátní, proto pokud byste je použili v HTML kódu vkládaném na více míst, povede to k nepředvídatelnému a potenciálně problematickému chování.

Metoda html() umožňuje také získat současný obsah objektu. Ten nemusí přesně odpovídat tomu, co bylo ve zdrojovém kódu nebo co bylo do dokumentu později vloženo. Prohlížeče si totiž mohou DOM poněkud upravit. Na závadu funkce by to ale být nemělo.

Tuto metodu nelze použít v případech, kdy se pracuje s XML DOM. To není obvyklý případ, protože kvůli kompatibilitě se staršími prohlížeči, ale i kvůli vyšší toleranci k chybám, se XHTML zpracovává jako HTML a nikoli jako XML. Kdo by chtěl ale pracovat s XML, musí zvolit jiný postup, metoda html() je zde zapovězena.

Vkládání nových objektů

Vložit nové objekty lze již popsanou metodou html(). Jsou tu ale i jiné metody, umožňující provádět další „kouzla". Častým požadavkem je vložit něco před současný obsah objektu (tedy nikoli místo něj, jak dělá metoda html()) nebo za něj. To lze snadno zařídit:

```
$('body').prepend('<div id="hlavicka">Tady bude hlavička...</div>');
$('body').append('<div id="paticka">Powered by jQuery</div>');
```

Asi každý pozná, co uvedené řádky dělají. První vloží „hlavičku" úplně na začátek všeho uvnitř elementu body (tedy těla dokumentu), druhý vloží na samý konec těla „patičku". Je to velmi pohodlné a dá se říci, že i návykové.

Potřebujeme-li se při této činnosti „odpíchnout" od objektu na stejné úrovni (tj. od sourozence), není potřeba si nejdřív zjišťovat rodiče a pak na něm volat uvedené metody. Jde to jednodušeji:

```
$('#hlavicka').after('<p>Odstavec těsně za hlavičkou</p>');
$('#paticka').before('<p>Odstavec těsně před patičkou</p>');
```

Obě metody navazují na hlavičku a patičku z předchozího příkladu. Vloží vždy daný odstavec těsně za hlavičku, resp. před patičku. U těchto i předchozích metod lze samozřejmě vkládat na více míst najednou – stačí, aby selektor vybral vícečlennou množinu objektů (opět ale pozor na identifikátory).

Odstraňování objektů

Odstranit objekt z DOM je velmi jednoduché a jsou zde v podstatě tři možnosti. Tady je nejprve příklad:

```
$('table').empty();
$('tr').remove();
var data = $('tr').detach();
```

První řádek vymaže všechny objekty, které jsou pod jedním či více objekty v instanci jQuery.

Jinak řečeno, co bylo uvnitř všech tabulek, je nenávratně pryč (včetně textu). Druhý řádek funguje podobně, ovšem odstraňuje z DOM přímo vybrané objekty, nikoli potomky. Navíc umožňuje aplikovat ještě filtr (v příkladu není použit). A konečně poslední řádek sice odstraní objekty obdobně jako remove(), vloží je však do dané proměnné – neztratí se tedy a lze je v případě potřeby později opět vložit. I tato metoda dovoluje použití filtru.

Kopírování a přesun objektů

Někdy potřebujeme objekty kopírovat či přesouvat na jiná místa. Stačí si představit třeba dva seznamy, mezi kterými se přesouvají položky:

```
<ul id="seznam1">
<li>Položka 1</li>
...
</ul>
<ul id="seznam2">
</ul>
```

Na začátku jsou v prvním seznamu nějaké položky a druhý je prázdný. Chceme vzít první položku z prvního seznamu a vložit ji do druhého. Lze to udělat třeba takto:

```
$('#seznam2').append($('#seznam1 li:first-child'));
```

Metoda append() je jistě každému známa z předchozích odstavců. Ale co dělá tady a jak souvisí s přesunem? Ona totiž může mít jako parametr kromě jiného také instanci jQuery a v takovém případě přesune objekty z původního místa na nové. Nezkopíruje, přesune – což je ovšem přesně to, co chceme.

Pozor však na to, že pokud se takto použitá metoda append() zavolá na instanci jQuery obsahující více objektů, přesune se jen do prvního z nich a do dalších se nakopíruje (z původního místa však opět zmizí).

Obdobně lze použít i další metody určené pro vkládání obsahu, tedy prepend(), before() a after(). Existují i opačně pojaté metody, volané naopak na instanci jQuery objektů, které se mají vkládat. Jsou to metody appendTo(), prependTo(), insertBefore() a insertAfter(). Rozdíl je téměř jen syntaktický, i když některé věci u nich fungují mírně jinak a jdou tak trochu proti snadné použitelnosti jQuery (například můžete zkonstruovat instanci třídy jQuery obsahující HTML fragment, což může poněkud znepřehlednit kód, protože se takto obvykle nepostupuje).

Čistokrevné kopírování lze zajistit metodou clone(). Ta kopíruje vždy a ještě umožňuje definovat, jestli se budou kopírovat i data a události navázané na objekty (o tom bude řeč mnohem později, ale zaslouží si to teď zmínku).

```
$('#seznam2').append($('#seznam1 li:first-child').clone());
```

Toto je upravený příklad se dvěma seznamy. Rozdíl oproti příkladu výše je však v tom, že tentokrát se položka ze seznamu zkopíruje a bude tedy figurovat v obou seznamech.

Přesun ve dvou krocích lze zajistit pomocí již zmíněné metody detach() a následně některé z metod pro vložení. Význam to má pro ty případy, kdy je potřeba mezi oněma dvěma kroky provést nějaké operace, během kterých nesmí daný objekt (včetně svého podstromu) existovat v DOM.

„Potápění" a „vyzvedávání" objektů

Zvláštním případem je situace, kdy chceme přidat nebo odebrat nějakou úroveň DOM. Máme například toto:

```
<div id="inner">
<p>Text...</p>
</div>
```

Chceme přidat ještě jednu úroveň navíc, resp. naopak jednu odebrat. Dá se to samozřejmě udělat pomocí již dříve popsaných postupů pro přesun, ale je to zbytečné. Úroveň přidáme jednodušeji takto:

```
$('#inner').wrap('<div id="outer"></div>');
```

Výsledkem bude, že se zkrátka místo elementu s identifikátorem inner vloží nová úroveň a původní objekt se vloží pod ni. Pokud bude instance jQuery obsahovat více objektů, vloží se nová úroveň pro každý zvlášť. Pokud je to nežádoucí, je třeba místo toho použít metodu wrapAll(). Ještě existuje varianta wrapInner(), která vkládá novou úroveň pod dané objekty.

Obdobně se provede i „vytažení nahoru". Metoda odstraňuje rodiče, je tedy potřeba ji volat pro ty objekty, které je potřeba vyzvednout:

```
$('#inner p').unwrap();
$('#inner').children('p').unwrap();
```

Oba řádky dělají totéž, druhý je delší, ovšem poněkud rychlejší. To jen pro připomenutí.

To ještě není všechno

Končí tento díl seriálu, ale nikoli úpravy objektů v DOM. Příště se podíváme na atributy, které jsou nedílnou součástí HTML elementů a často významně ovlivňují to, jak bude dokument vypadat, jak bude fungovat a v neposlední řadě i to, jak v něm můžeme hledat objekty.

jQuery (6) – práce s atributy a vlastnostmi

Stručně o atributech a vlastnostech

Atributy zná každý, kdo někdy něco tvořil v (X)HTML. Pomocí atributů se specifikuje například identifikátor objektu, třídy, styly, jazyk a spousta dalších vlastností. Někdy se uvádějí explicitně (pokud je potřeba je ovlivnit), většina jich je ale implicitních, s výchozími nebo prázdnými hodnotami.

Z atributů vycházejí *vlastnosti* (*properties*). Zatímco atribut specifikuje výchozí hodnotu, vlastnost může někdy nabývat různých hodnot na základě nějakých událostí v objektu. Mějme například následující HTML kód:

```
<input type="checkbox" name="cb" value="abc" id="cb" checked="checked" title="Zaškrtávátko">
```

Definuje zaškrtávací políčko a to se zobrazí již zaškrtnuté. Pokud uživatel na políčko klikne, zůstane hodnota atributu beze změny (bude odpovídat řetězci „checked"), kdežto hodnota vlastnosti se změní z true na false.

Hodnoty atributů i vlastností lze v JavaScriptu (a tedy i v jQuery) nejen číst, ale také nastavovat. Někdy se tak činí univerzálními metodami, jindy je výhodnější použít speciální metody, které pracují třeba jen s určitou částí (to je třeba případ metody css(), na kterou přijde řeč později).

Pozor – starší verze jQuery (před 1.6) nebyly při práci s atributy a vlastnostmi konzistentní. Pro některé atributy se tehdy vracely hodnoty vlastností, čemuž pak odpovídal i kód, který s tím pracoval. Ve starším kódu tedy ještě můžete najít práci s atributy, která neodpovídá popisu uvedenému v tomto článku.

Kromě „nativních" vlastností (souvisejících s atributy definovanými ve specifikaci (X)HTML) lze pracovat i s prakticky libovolnými dalšími vlastnostmi objektů. Ty nejsou spojeny s atributy a nemají interní význam. Obdobně si lze definovat i vlastní atributy, ale to obecně není dobrý nápad, protože naruší validitu dokumentu.

Zjišťování hodnot atributů a vlastností

Pro výše uvedený kus HTML kódu lze napsat například takovýto javascriptový kód:

```
var chkattr = $('#cb').attr('checked');
var chkprop = $('#cb').prop('checked');
```

První řádek zjistí hodnotu atributu checked. Ta bude odpovídat řetězci, který byl nastaven, tedy „checked". Nebude se měnit při změně stavu zaškrtnutí; změní se však v případě přímé změny atributu (viz dále). Druhý řádek zjišťuje hodnotu vlastnosti checked. Ta bude true pro zaškrnuté políčko, resp. false pro nezaškrtnuté.

Není na tom nic složitého. Jsou však případy, kdy je vhodnější použít pro získání hodnoty vlastnosti specifickou metodu. Máme například tento HTML kód:

```
<input type="text" name="txt" value="původní text" id="txt">
```

Hodnotu atributu – tedy původní text obsažený v daném textovém poli – lze získat úplně stejně jako u toho zaškrtávátka:

```
var txt = $('#txt').attr('value');
```

Ovšem pro získání hodnoty je lepší použít val() a nikoli prop():

```
var txt = $('#txt').val();
```

Důvodů je hned několik. Jednak je val() v moderních prohlížečích rychlejší. Podstatnějším důvodem je jednotnost při práci s různými druhy vstupů, protože například pro získání textu z objektu textarea metodu prop() nelze použít. A konečně nejdůležitějším důvodem je konzistence chování nezávislá na prohlížeči.

Dalším podobným případem je třeba práce s CSS třídami. Pokud potřebujeme zjistit, zda má některý objekt určitou třídu, lze to udělat takto:

```
if ($('#hdr').hasClass('first')) {
...
}
```

Tento kód zjistí, zda má daný objekt přiřazenu určitou třídu (může jich mít víc), a pokud ano, něco se vykoná.

Pozor na to, odkud daná metoda získává hodnoty. Například metody attr(), prop() nebo val() berou vždycky první objekt z množiny v instanci jQuery (ostatní ignorují), metoda hasClass() vrátí true, pokud má danou třídu kterýkoli objekt. Je potřeba si vždy zjistit v dokumentaci, jak to v tom kterém případě je.

Pokud zvolíte metodu attr() nebo prop() pro neexistující atribut/vlastnost, vrátí hodnotu undefined.

Nastavování hodnot atributů a vlastností

Opakem zjišťování hodnot je jejich nastavování. Používají se obvykle totožné (stejně nazvané) metody jako při zjišťování hodnot, případně metody logicky konzistentní (viz dále).

Jak tedy například změnit stav zaškrtávátka? Je to jednoduché:

```
$('#cb').prop('checked', true);
$('#cb').prop('checked', !$('#cb').prop());
```

První řádek políčko zaškrtne, druhý řádek invertuje jeho stav (udělá vlastně totéž, jako když na něj uživatel klikne; neřešíme ale teď události, které nastávají). Lze to implementovat podstatně elegantněji, k tomu se ale dostaneme někdy později.

```
$('#cb').attr('checked', 'checked');
```

Je to k něčemu dobré? Ano, k nastavení výchozí hodnoty. Hlavně se ale metoda attr() používá u atributů, které nepodléhají změnám tohoto typu. Například takto lze změnit atribut title:

```
$('#cb').attr('title', 'Nový titulek');
```

Metodu val() lze použít pro nastavování, pokud se pracuje s objekty pro vstup dat, resp. je to ta správná cesta.

Na rozdíl od použití ke zjišťování hodnot, tady se metody prop(), attr() a val() aplikují na všechny objekty v instanci jQuery. To znamená, že lze nastavovat hodnotu více objektům najednou.

Co se týká CSS tříd, lze s nimi přímo manipulovat pomocí sady metod:

```
$('#cb').addClass('abcd').removeClass('efgh ijkl');
$('#cb').toggleClass('abcd');
$('#cb').toggleClass('abcd', true);
```

První řádek ukazuje použití hned dvou metod: addClass() třídu přidává, removeClass() dvě jiné třídy odebírá. Druhý řádek invertuje přítomnost třídy – tedy pokud třída není přítomna, přidá ji, jinak ji odebere. A konečně třetí řádek představuje jiné použití metody toggleClass(), protože ji lze použít místo zmíněných dvou metod. Hodnota true znamená přidání jedné či více tříd, false naopak odebrání.

Odebírání atributů a vlastností

V některých případech – a nebývá to moc často – je potřeba atribut nebo vlastnost z objektu odebrat. Zaškrtávací pole, které se táhne celým tímto článkem, má v HTML definovaný titulek a ten můžeme odebrat:

```
$('#cb').removeAttr('title');
$('#cb').removeProp('title');
```

První příkaz odebere příslušný atribut, druhý odebere vlastnost. Jaký je mezi nimi rozdíl? Velmi podstatný. Odebrání atributu způsobí, že atribut i související vlastnost zcela zmizí. Ovšem odebrání vlastnosti nastaví atribut na hodnotu undefined a ta se přenese i do vlastnosti, ovšem ve formě řetězce. Může to být divné, ale je to tak. Proto je v daném případě metodou volby removeAttr().

Popsané chování platí pro nativní vlastnosti, propojené s atributy. Pokud máme v objektu nějakou svoji speciální vlastnost, chování bude jiné:

```
$('#cb').prop('origtitle', $('#cb').attr('title'));
...
$('#cb').attr('title', $('#cb').prop('origtitle')).removeProp('origtitle');
```

Příklad uschová do vlastnosti origtitle původní hodnotu atributu title, ve druhé fázi ji zase opačným procesem obnoví a vlastnost origtitle odebere.

Práce s CSS

Speciálním případem práce s atributy a vlastnostmi je ovládání CSS. Často je vhodné to řešit přes třídy (přidávání a odebírání tříd popsané výše), někdy je ale vhodnější nebo dokonce nezbytné nastavovat styly přímo. Na to se podíváme příště.

jQuery (7) – práce s CSS

Přímá práce s kaskádovými styly

Obvykle je vhodné pracovat s kaskádovými styly (CSS) tak, že se vše nadefinuje samostatně pro třídu (pokud má styl platit pro více objektů) nebo pro identifikátor (pokud má platit pro jeden objekt). Identifikátor bývá napevno, takže i styl je neměnný, zatímco třídy lze měnit podle potřeby (viz minulý článek) a tak styly přepínat. Existují ale případy, kdy je vhodné, nebo dokonce nezbytné styly nastavovat přímo.

Typickým případem je pozicování. Pokud se má nějaký objekt pozicovat interaktivně nebo dokonce animovaně (plynulý pohyb), přes třídy to řešit nelze. Totéž se týká třeba plynulých změn barvy, velikosti apod. Nehledě na to, že potřebujeme vlastnosti stylů nejen nastavovat, ale také zjišťovat.

Jedna metoda zvládne (téměř) vše

Přímá práce s CSS je v zásadě velmi jednoduchá a pro drtivou většinu operací se používá jen jedna jediná metoda: css(). Mějme například následující fragment HTML kódu:

```
<h1 id="nadpis1" style="color: blue; font-weight: bold">Nadpis</h1>
```

Je to prostě jen obyčejný nadpis první úrovně, který je přímo atributem style (ale dalo by se to udělat i v samostatném souboru, například pro všechny elementy h1 nebo pro identifikátor; tady je to jen pro zjednodušení uděláno takto) obarven namodro a má nastavené tučné písmo. Pak můžeme s CSS pracovat například takto:

```
alert('Barva: ' + $('#nadpis1').css('color') + '\n' + 'Pozadí:' +
$('#nadpis1').css('background-color'));
$('#nadpis1').css('color', '#ffff00').css('font-weight', 'normal');
```

První řádek zobrazí, jakou barvu tento nadpis má a jak barevné je pozadí. Přestože byla barva nastavena jako „blue“, bude vrácena v jiném formátu (číselného charakteru). Jakém, to bohužel záleží na prohlížeči. Může to být například něco ve stylu rgb(0, 0, 255), ale zaručeno to není.

Pro práci s barvami v jQuery existuje plugin jQuery Color, k němuž se ve vzdálenější budoucnosti dostaneme. Bez něj lze také barvy uchovávat samostatně (v proměnných nebo vlastnostech objektů) a metodou css() je jen nastavovat.

Totéž se týká i barvy pozadí, která nebyla v HTML explicitně nastavena, takže se použije ta, která je definována někde jinde (např. v CSS souboru), případně výchozí barva. Výchozí bude většinou průhledná, takže se zobrazí „transparent“.

Druhý řádek nastavuje novou barvu (bude žlutá) a novou tučnost písma (bude normální, kdežto předtím bylo tučné). Jak si můžete všimnout, lze v tomto případě volání metody css() řetězit, protože vždy vrací opět odkaz na tentýž objekt jQuery.

Místo formátu typu background-color lze použít i backgroundColor jako v klasickém JavaScriptu. V uvedeném případě je to úplně jedno, jsou ale případy, kdy je na to potřeba dávat pozor – viz dále.

Využití objektu a pole jako argumentu

Pokud potřebujete nastavit, nebo zjistit více hodnot najednou, není nutné metodu css() volat na každou jednotlivou hodnotu. Jde to i jednodušeji. Tady je opět příklad:

```
$('#nadpis1').css({borderLeft : '1px solid silver', 'padding-left' :
'10px', 'background-color' : ''});
var a = ['margin-top', 'margin-bottom'];
var o = $('#nadpis1').css(a);
alert('Okraje: ' + o['margin-top'] + ', ' + o['margin-bottom']);
```

Příklad ukazuje už v prvním řádku hned několik zajímavých věcí. První je samotné použití objektu jako argumentu. Objekt je nadefinován přímo v místě volání. Jeho první položka je nazvána tím „javascriptovým" způsobem, což umožňuje vynechat uvozovky a přitom to jQuery správně zpracuje.

Druhá položka využívá klasický CSS zápis – tam jsou uvozovky nutné, protože jinak by to nevyhovovalo syntaxi JavaScriptu.

A konečně třetí položka ukazuje, jak zrušit nastavení konkrétní hodnoty – prázdný řetězec ji vrátí do výchozího stavu před jakýmkoliv nastavením, použije se tedy to, co je „zadrátováno" v prohlížeči (případně v uživatelském stylu, ale to už jde nad rámec seriálu). Takže když bylo například předtím obarveno pozadí nadpisu, použití prázdného řetězce vrátí zpět průhledné pozadí.

Další tři řádky příkladu ukazují zjištění určitých hodnot s využitím objektu. Nejprve se připraví pole názvu CSS parametrů, které se mají zjistit. Pole se předá metodě css() a ta vrátí objekt obsahující jako své položky požadované hodnoty (objekt obdobný jako ten, který se používá při nastavování v prvním řádku).

Pozor na to, že při použití názvů v CSS formátu (např. margin-top) je nezbytné s objektem pracovat pomocí syntaxe asociativního pole, kdežto při použití javascriptového formátu (marginTop) lze použít i klasickou objektovou notaci. Který z formátů budete používat, je na vás. Je ale dobré držet se v celém kódu jednoho, míchání obou formátů nadělá zbytečný zmatek. Názvy ovšem používejte podle specifikace CSS, převod do přesné podoby pro ten který prohlížeč (i při použití javascriptového formátu) zajistí jQuery.

Okraje, podobně jako jiné rozměry, budou vráceny v pixelech (včetně „px" na konci hodnoty), ať již byly předtím nastaveny v jakýchkoli jednotkách. Při nastavování lze uvést kterékoli povolené jednotky; pokud se vynechají, jQuery hodnoty zpracuje v pixelech.

Na co si ještě dát pozor

Speciální CSS parametry ovlivňující více věcí najednou (neatomické parametry), například margin (zahrnující v sobě margin-left, margin-right atd.) lze bez problémů používat při nastavování; při zjišťování hodnot je však třeba se jim vyhnout. Některé prohlížeče je úspěšně zpracují, obecně to ale neplatí.

Dále je potřeba si uvědomit, že lze metodu css() volat jen na objektech, které jsou součástí DOM. Nelze tedy vyjmout objekt z DOM zavoláním detach() a pak na něj použít metodu css().

Rozměry a pozice

Oblast rozměrů vizuálních objektů a jejich pozice je specifickou oblastí. Je to dáno tím, že se s nimi jednak velmi často pracuje, současně také metoda css() pracuje i s jednotkami, což je mnohdy zbytečné. A v neposlední řadě bývá výhodné nechat některé rutinní výpočty na jQuery a nemuset se s nimi trápit. Je toho trochu více, proto si to necháme na příště.

jQuery (8) – rozměry a pozice

Proč nepoužít metodu css()?

Mějme následující kus HTML kódu:

```
<div id="blok" style="width: 160px; height: 100px>
...
</div>
```

Každý jistě hned vidí, že to je blok o rozměrech 160 x 100 pixelů, v němž je nějaký obsah. Teď budeme potřebovat blok o něco rozšířit, přičemž nebudeme moci vycházet z původní šířky, nýbrž z té aktuální. Jednou z možností je použít již dobře známou metodu css():

```
$('#blok').css('width', (Number($('#blok').css('width').replace('px', '')) + 40) + 'px');
```

Takový kód je poměrně dlouhý, špatně srozumitelný a navíc výkonově neefektivní. Co totiž dělá? Získá hodnotu v pixelech, která je vyjádřena včetně jednotek a tedy jako řetězec. Ten se musí zbavit jednotek, převést na číslo, přičíst požadované rozšíření, převést zpět na řetězec a připojit jednotky (poslední dva kroky lze i vynechat, proběhnou automaticky).

Novější verze jQuery (od 1.6) to umožňují zapsat podstatně jednodušeji. Metoda css() přijímá „relativní hodnoty" a lze tedy příklad přepsat do této podoby:

```
$('#blok').css('width', '+=40');
```

To už je výrazně příjemnější, ale jen v případě, že se přidává pevně definovaná hodnota, kterou lze zapsat přímo do řetězce. Získáme-li však tuto hodnotu odjinud, řetězec by se musel „lepit" dohromady. Především je tu ale pořád problém s výkonem, protože se nelze obejít bez zpracování řetězce.

Metody pro práci s rozměry

Pokud potřebujeme s rozměry provádět nějaké výpočty, nejlepší bývá používat k tomu určené metody – tedy width() a height(). Slouží jak ke zjišťování rozměrů, tak k jejich nastavování. Pracuje se s čísly bez jednotek, rozměry jsou vyjádřeny v pixelech.

Výše uvedený příklad by se přepsal do této podoby:

```
var b = $('#blok');
b.width(b.width() + 40);
```

Při použití pro zjištění šířky se pracuje pouze s prvním objektem v instanci jQuery (pokud by jich tam bylo víc), při nastavování se nová šířka nastaví všem.

Při zjišťování šířky může metoda vrátit i desetinné číslo. Obvykle to není nijak na závadu, ale je dobré s tím počítat.

Při nastavování není hodnota omezena jen na číslo – lze použít i jednotky, resp. cokoli, co by se použilo při volání metody css() – například i řetězec „auto". V případě pixelů je to samozřejmě zbytečné.

Podobně jako se pro šířku používá metoda width(), pro výšku je tu height(). Pracuje se s ní úplně stejně, takže k ní není v podstatě co dodat. Snad jen to, že lze při nastavování rozměrů metody řetězit, protože vracejí příslušnou instanci jQuery:

```
$('#blok').width(300).height(200);
```

Příklad je zcela zřejmý, nastaví šířku výše popsaného bloku na 300 a výšku na 200 pixelů.

Pozor ještě na jednu významnou odlišnost oproti css(). Metody width() a height() vždy vracejí rozměry obsahu, tedy bez vnitřního odsazení (*padding*) a ohraničení (*border*). Naopak css('width'), resp. css('height') vrací rozměry podle aktuálního nastavení box-sizing, která při hodnotě border-box a padding-box znamená připočtení odsazení i ohraničení, resp. jen odsazení. Je třeba to mít na paměti.

Další rozměrové metody

Někdy potřebujeme velikost včetně vnitřního odsazení, případně i ohraničení (viz další informace výše). Pak využijeme další dvojice metod, které jQuery nabízí. Jsou to innerWidth() a innerHeight(), resp. outerWidth() a outerHeight(). První dvě připočtou k rozměrům i vnitřní odsazení (*padding*), druhé dvě navíc ještě ohraničení (*border*) a případně i vnější odsazení (*margin*).

Všechny z uvedených metod lze použít i k nastavování a platí pro ně stejná pravidla jako pro width()a height(). Ovlivňují vždy jen rozměry obsahu, do ostatních parametrů nezasahují (fungují, jako kdyby odečetly příslušné „okraje" a pak zavolaly metodu pro nastavení čisté velikosti).

Metody pro práci s pozicemi

Podobná situace jako u rozměrů je také u pozic. Je to ale trochu složitější – může nás totiž zajímat hned vícero druhů pozicování. Běžně (v CSS) se pozice počítají vzhledem k tzv. obsahujícímu bloku, jehož určení nemusí být úplně jednoduché. Proto se mnohdy hodí pracovat raději s pozicováním v rámci celého dokumentu.

Pozice vůči obsahujícímu bloku

Toto je pozice, jakou známe z CSS (vlastnosti left, right, top a bottom). Počítá se vůči obsahujícímu bloku, což je – zjednodušeně řečeno – nejbližší nadřazený blok, který má vlastnost position nastavenou na absolute, relative nebo fixed. Pokud takový blok na cestě vzhůru není, počítá se pozice vůči celému dokumentu (v případě pozicování fixed vůči obrazovce, resp. u stránkovaného tisku vůči stránce).

Je to tedy podobné, jako když zjišťujeme rozměry – místo metody css(), která by zde šla použít, použijeme speciální metodu určenou jen pro pozice: position(). Tady je však jiné to, že se pozice nezjišťuje po jednotlivých složkách, nýbrž celá v jediném volání. Návratovou hodnotou je objekt obsahující složky lefta top.

```
var pos = $('#blok').position();
alert('Pozice: ' + pos.left + ', ' + pos.top);
```

Příklad zjistí aktuální pozici bloku a vypíše ji do notifikačního dialogu. Čísla neobsahují jednotky, jsou v pixelech a opět mohou být i desetinná. Přepsáno pro metodu css() by to vypadalo nějak takto:

```
var l = $('#blok').css('left').replace('px', '');
var t = $('#blok').css('top').replace('px', '');
alert('Pozice: ' + l + ', ' + t);
```

Je to samozřejmě pomalejší a těžkopádnější, na druhou stranu lze získat i pozice z opačných stran (tj. righta bottom), což metoda position() neumožňuje. Tato metoda neumožňuje ani nastavování pozice. Potřebujeme-li ji nastavit vůči obsahujícímu bloku, nezbude než sáhnout po metodě css():

```
var pos = $('#blok').position();
$('#blok').css('left', pos + 40);
```

Netřeba snad připomínat, že pokud by se tento kód použil přímo na objekt z HTML fragmentu na začátku článku, nic by se nestalo – výchozí pozicování je totiž „statické" a s objektem proto nelze hýbat. Muselo by se, ať už jakýmkoli způsobem, nejprve změnit například na absolute nebo relative.

Pozice vůči dokumentu

Tohle nejspíš využijeme častěji. Pozice vůči dokumentu je nezávislá na tom, přes jaké bloky k objektu vede cesta. Máme k dispozici metodu offset(), jež vrací úplně stejný objekt jako position(), jen samozřejmě často s jinými hodnotami. Příklad pro zjištění pozice tedy můžeme přepsat:

```
var pos = $('#blok').offset();
alert('Pozice: ' + pos.left + ', ' + pos.top);
```

Změní se jen název metody, jinak je vše stejné. Skutečně „zábavné" to ale začíná být až v okamžiku, kdy tutéž metodu použijeme pro nastavení pozice:

```
var pos = $('#blok').offset();
pos.left += 40;
$('#blok').offset(pos);
```

Jak vidíte, objekt se souřadnicemi se použije i pro nastavení nové pozice. Tady ale navíc proběhne „kouzlo", které posune i blok, u kterého to „nejde", tedy třeba ten z výše uvedeného HTML kódu. Pokud je totiž blok pozicován staticky, metoda to automaticky přehodí na relativní pozicování. Pak už samozřejmě pozici změnit lze.

Funkce jako parametr

Mnohé z dosud popsaných metod (a řada dalších) přijímá jako parametr nejen nějakou konkrétní hodnotu, ale také funkci. K čemu je to dobré a jak se s tím pracuje, se podíváme v příštím dílu seriálu.

jQuery (9) – funkce jako parametr

Stručně o funkcích

Každý, kdo někdy něco napsal v některém z běžných programovacích jazyků, zná funkce jako něco, co se někde definuje (případně ještě samostatně deklaruje, jako třeba v hlavičkových souborech u jazyka C) a někde jinde zavolá, často na více místech. Hlavním smyslem funkce je odstranit opakování téhož kódu všude, kde je potřeba – stačí ho napsat jen jednou.

V objektovém programování se hovoří o metodách. Ty jsou spjaty s objekty (jejich instancemi nebo třídami), jinak se na ně lze ale dívat jako na funkce. Ještě o nich bude řeč.

Když se v JavaScriptu definuje funkce, vytvoří se tím zvláštní objekt – instance třídy Function. To znamená, že lze nadefinovanou funkci předávat jako hodnotu identifikovanou svým názvem. Dokonce lze funkci nadefinovat i přímo v místě, kde se s ní dál pracuje („anonymní funkce"). Toto všechno poskytuje řadu možností, které výrazně usnadňují práci.

Funkce jako parametr

Tím, že je každá funkce instancí objektu, můžeme ji jako jakoukoli jinou objektovou instanci (třeba Stringnebo RegExp) někam předat. „Někam" znamená samozřejmě do funkce či metody jako parametr (argument). Uvnitř dané funkce se předaná funkce volá podle potřeby.

K čemu je to dobré? Například k tomu, aby se funkce zavolala pro každý z nějakých zpracovávaných objektů a něco s ním udělala. V jQuery je takovým typickým příkladem metoda filter(). Ta slouží k vyfiltrování některých objektů z původní množiny. Dá se použít selektor, ale například také právě funkce. Tady je příklad:

```
function myFilter(index, element) {
    if (index == 0)
        return false;

    return element.accessKey == '0';
}

var a = $('#menu a').filter(myFilter);
```

Příklad přiřadí k dalšímu použití do proměnné a instanci jQuery s vyfiltrovanými objekty odkazů. Nejprve si definujeme funkci myFilter(), která bude zajišťovat filtraci. V tomto případě funguje tak, že první odkaz vždy zahodí a z dalších ponechá ty, které mají přístupovou klávesu „0". Následně se takto definovaná funkce předá metodě filter(), která zajistí její zavolání pro každý objekt v původní množině získané selektorem.

Každý si může zkusit funkci přepsat do elegantnější, byť méně názorné podoby. Místo dvou kroků lze vše řešit v jednom, v rámci výrazu poskytujícího návratovou hodnotu funkce.

Anonymní funkce

Většinou bývá lepší si funkci definovat „klasicky", tedy pojmenovanou, jako je tomu výše. Existují ovšem případy, že je to zbytečné – funkce se používá jen na jediném místě a proto není na závadu ji definovat až přímo v místě použití.

```
$('textarea').val(function(index, value) {
return 'Přeložte: ' + value;
});
```

Máte například formulář s textovými oblastmi, kde chcete uživatele vyzvat, aby texty přeložil. Tak se ke každému jednoduše přidá příslušná instrukce. Metoda val() před nastavením nové hodnoty každému objektu získá tuto hodnotu zavoláním anonymní funkce, tedy definované přímo v okamžiku volání.

Pro podobné jednoduché případy je to výborná věc, která brání zaplevelení globálního jmenného prostoru názvy drobných, jednorázově použitých funkcí. Máte-li v projektu takových funkcí stovky, při pojmenování každé by ani s našeptávačem ve vývojovém prostředí nemuselo být snadné se v tom vyznat.

Metoda jako parametr

Úplně stejně jako globální funkci lze použít i metodu – a to jak metodu definovanou přímo v konkrétní instanci, tak definovanou ve třídě (prototypu). Předá se pak identifikací instance a metody (tečkovou notací), případně přes proměnnou, pokud to má smysl. Tady je stručný příklad:

```
var o = {
test: function() { … }
};

obj.proc(o.test);
```

Přímo v instanci o se definuje metoda test(), která něco dělá (není teď podstatné co). Metodě proc()instance nějakého jiného objektu obj se nadefinovaná metoda předá a ta s ní pak nějak nakládá.

Hlavní rozdíl metody oproti funkci je v tom, že má metoda přístup k datům (položkám) objektu. A teď přijde to důležité – v jQuery se obvykle funkce předaná jako parametr stane metodou nějakého objektu a tím získá přístup k nějakým dalším datům, které nejsou funkci předávána přes její parametry.

Když se vrátíme k té anonymní funkci z příkladu výše – tam metoda val() udělá vlastně to, že se z předané funkce stane dočasně metoda instance objektové třídy Element, čímž umožní k této instanci přistupovat, a to obvyklým způsobem, tedy přes this.

Příklad můžeme tedy rozšířit třeba takto:

```
$('textarea').val(function(index, value) {
return 'Přeložte (' + this.name + '): ' + value;
});
```

Funkce, tedy vlastně už metoda, si do objektu sáhne pro jeho název (atribut name) a ten přidá do instrukce k přeložení. Budete-li tedy v jQuery předávat funkci jako parametr, vždycky si předem v dokumentaci zjistěte, k jakému objektu získáte přístup a co z něj lze využít. Lze si tím často dost usnadnit práci.

Obsluha událostí

Funkce předávané jako parametr se nejčastěji používají v jedné poměrně široké oblasti jQuery, a to je obsluha událostí. Mnoho věcí se totiž ve webové aplikaci odehrává asynchronně a proto je zcela nezbytné implementovat reakce na různé události, které iniciuje uživatel, webový prohlížeč nebo i samotná aplikace. Na to, jak to řešit, se podíváme příště.

jQuery (10) – obsluha událostí

Události a jejich obsluha

Jedním ze základních paradigmat při vývoji počítačových programů všeho druhu je už několik desetiletí obsluha na událostí – hovoří se o *programování řízeném událostmi* (event-driven programming). Nastane-li nějaká událost, která je pro program významná, v reakci na ni se vykoná určitý kód.

Události mohou vznikat jak v programu, tak velmi často mimo něj. Nejčastějšími událostmi u webových aplikací je nějaká akce uživatele (třeba najetí kliknutí, stisk klávesy atd.), načtení dokumentu, dokončení nějakého efektu a podobně.

V řadě případů mají události implementovánu ve webovém prohlížeči nějakou výchozí reakci. Pokud si implementujeme svoji, lze ponechat i tu výchozí nebo ji nahradit. Některé události tzv. probublávají hierarchií DOM, takže pokud se na ně nereaguje na nižší úrovni, lze na ně reagovat výše.

Reakce na události v jQuery

V jQuery se využívají v zásadě dva způsoby, jak nastavovat reakce na události. Jednak jsou to speciální metody určené přímo pro tento účel, druhým způsobem je přiřazení do položek v objektu, který se předává nějaké metodě. Nyní se podíváme na první způsob, používaný například pro reakci na akce uživatele.

Metody on() a off()

Pro širokou škálu událostí lze použít metodu on() pro nastavení reakce, resp. off() pro její odebrání. Tady je příklad:

```
function showImageAlt() {
alert($(this).attr('alt'));
}

$('img').on('click',  showImageAlt);
```

Funkce showImageAlt() slouží k zobrazení alternativního textu obrázku, na který uživatel klikne. Obsluha kliknutí se přiřadí zavoláním metody on(). Obslužná funkce se vnitřně stane metodou instance třídy Element pro daný obrázek (kde událost nastala), takže lze s instancí pracovat přes this. Zrušení reakce je podobné:

```
$('img').off('click', showImageAlt);
```

Tento způsob volání se hodí pro případy, kdy je potřeba odebrat jen tuto konkrétní reakci (lze jich totiž nastavit libovolný počet). Pro globálnější odebírání lze někdy použít i jednodušší volání:

```
$('img').off('click');
$('img').off();
```

První řádek odebere všechny reakce na daný typ události (zde kliknutí), druhý odebere všechno, co bylo předtím nastaveno.

Dříve se pro nastavování a rušení reakcí na události používaly i další metody – bind()/unbind(), live() a delegate(). Od verze 1.7 ale není příliš důvod je používat, on() a off() totiž nabízejí vše pohromadě s jednotným rozhraním.

Pomocí metod on() a off() lze pracovat s mnoha různými událostmi – jejich přehled najdete v dokumentaci. Konkrétní škála se liší podle typu objektu v DOM, například událost submit (odeslání) může vzniknout jen ve formuláři, zatímco keyup (uvolnění klávesy) může nastat v libovolném vizuálním elementu. S metodou on()lze dělat i další „kouzla":

```
$('#abc').on('click.myApp', 'img', showImageAlt);
```
Příklad ukazuje hned dvě věci. Jednak je to vlastní jmenný prostor myApp. Při pozdějším volání off() lze pak použít celý název události, což způsobí odebrání jen té obsluhy, která je nastavena pro tento jmenný prostor; ostatních se to nedotkne:
```
$('#abc').off('click.myApp');
```
Druhá část se týká delegace obsluhy. Obsluha se nenastavuje pro obrázek (img), nýbrž pro nějaký nadřazený element a pak se filtruje. Kliknutí z obrázku probublá do daného nadřazeného elementu a použije se obslužná funkce nastavená pro tento element. Je to užitečné v případech, kdy se příslušný podstrom DOM mění (zde například přibývají a ubývají obrázky) a při přidání každého objektu by se jinak musela obsluha explicitně nastavit, což by bylo otravné. Tady se to dělat nemusí.

K záležitostem okolo probublávání událostí a delegace obsluhy se vrátíme příště. Je to relativně komplikovaná oblast a zaslouží si dostatečný prostor k vysvětlení.

Metoda one()

Výše uvedené metody nastavují obsluhu natrvalo, tedy do jejího odebrání, případně do zániku příslušného objektu DOM. Někdy je ale potřeba, aby se na událost reagovalo jen jednou. K tomu slouží metoda one(), pracuje se s ní jako s on().
```
$('#menu').one('mouseenter', function() {
    alert('Toto je menu.');
});
```
Uvedený příklad zajistí, že když se poprvé najede kurzorem myši na prvek s identifikátorem „menu", vyskočí informace o tom, k čemu prvek slouží. Při dalším najetí už obsluha události nebude aktivní a nic se tedy nestane. V příkladu vidíte využití anonymní funkce – toto řešení se v takových případech používá poměrně často (viz minulý díl).

Zjednodušené metody

Kromě uvedených univerzálních metod poskytuje jQuery celou řadu metod zjednodušených, určených pro konkrétní události. Znamenají poněkud méně psaní a hodí se pro případy, kdy není třeba využívat rozšířených vlastností univerzálních metod. Například příslušná část prvního příkladu s klikáním na obrázky by se dala přepsat takto:
```
$('img').click(showImageAlt);
```
Podobně existují například metody mouseenter(), change(), submit(), resize() a mnoho dalších. Zajímavé postavení zaujímá metoda ready(), která se společně s metodou click() objevila hned v první příkladu seriálu (ve druhém dílu). Ta se používá pro reakci na úplné dokončení přípravy nějakého objektu, nejčastěji celého dokumentu. Právě u dokumentu má největší význam, protože zajistí spuštění inicializačního kódu v okamžiku, kdy je celý dokument připraven k použití:
```
$(document).ready(function() {
    …
});
```
Nevýhodou zjednodušených metod je, že nemají své „odebírací" verze. Pro odebírání obsluhy se tedy musí volat off(). Většinou se ale zjednodušené metody používají tam, kde se obsluha jednou nastaví a pak už neodebírá.

Vyvolání událostí

Pro ladicí účely i běžné použití je často potřeba nějakou událost vyvolat, i když obvykle nastává z jiného podnětu. Typickým případem je třeba automatický upload souboru hned po jeho výběru, bez nutnosti klikat na další tlačítko. Můžeme mít například následující formulář:
```
<form id="fileform" action="upload.php" method="post" enctype="multipart/form-data">
    <input type="file" id="file" name="file" accept="image/*">
</form>
```
A potřebujeme, aby hned poté, co uživatel vybere soubor k odeslání, se data rovnou přenesla, bez nutnosti někam klikat (ani tu žádné odesílací tlačítko není). Jsou tedy potřeba dvě věci (v nejjednodušším případě; pro „blbuvzdornou" aplikaci by bylo potřeba toho udělat trochu víc) – zareagovat na událost výběru souboru a formulář odeslat.
```
$('#file').change(function() {
    $('#fileform').submit();
});
```
Jak si můžete všimnout, formulář lze odeslat stejnou metodou, jakou se mu nastavuje obsluha této události. To platí obecně – pokud se zjednodušené metodě nedají žádné parametry, místo nastavení obsluhy události danou událost vyvolá. Další možností je použít metodu trigger(), která umožňuje i předat obslužné funkci další parametry, což ale tady není potřeba.

Probublávání, delegace a další „věda" okolo událostí

Tímto úvodem není samozřejmě problematika událostí a jejich obsluhy rozhodně vyčerpána. Vrátíme se k ní hned příště, protože si některá témata zaslouží podívat se na ně podrobněji.

jQuery (11) – další práce s událostmi

Probublávání událostí, potlačení další obsluhy

Většina událostí, které vznikají někde ve stromě DOM, „probublávají" (též se to označuje jako „propagace") směrem vzhůru. Pokud se například klikne na obrázek v odstavci, lze událost obsloužit nejdřív v obrázku. Pak událost probublá do odstavce a lze ji obsloužit tam. Takhle může probublat až do kořene (objektu document).

Existují události, které ve všech nebo v některých prohlížečích neprobublávají. Ovšem jQuery řeší prohlížečově specifické věci svými prostředky a tak například událost change bublá vzhůru i v prohlížeči Internet Explorer (verze 8 a nižší), i když by se tak normálně nestalo.

Většinou je žádoucí, aby události takto probublávaly, někdy se to ale nehodí. Proto lze tento proces zastavit. Mějme třeba zmíněný obrázek v odstavci:
```
<p>
<img src="obrazek.jpg" alt="Obrázek" id="img">
</p>
```
Budeme obsluhovat kliknutí a nechceme, aby událost probublala do odstavce:
```
$('#img').click(function(e) {
    e.stopPropagation();
    ...
}
```
Všimněte si, že se tu využívá instance objektu události (Event), který může posloužit jak pro volání metod, tak pro získávání zajímavých informací (viz dále). Tady se mu zavolá metoda stopPropagation(), která probublávání zastaví. Lze ji zavolat i pro události, které probublaly z níže umístěných objektů ve stromě.

Uvedená metoda však nezastaví zavolání dalších obslužných metod navázaných přímo na daném objektu. Pokud by to bylo nutné, lze použít metodu stopImmediatePropagation(). Není to však dobrá praxe – takový kód se špatně ladí, pokud něco nefunguje.

Podobné je to s výchozí obsluhou (definovanou „natvrdo" v prohlížeči). Tu lze sice zakázat metodou preventDefault(), ovšem je potřeba to používat s opatrností, protože ne všechny výchozí obsluhy lze potlačit, odlišnosti jsou i mezi prohlížeči.

Ještě je tu jedna zajímavá možnost, jak ovlivnit obsluhu událostí:

```
$('#img').click(function() {
...
return false;
}
```

Takové řešení se chová, jako kdyby se zavolaly metody stopPropagation() a preventDefault(). Událost tedy nebublá výše, ani se pro ni nezavolá výchozí obsluha. Explicitně nastavené obslužné funkce se však zavolají. Tohle se v praxi používá relativně často (byť i zde je samozřejmě na místě opatrnost), typicky právě pro událost kliknutí – většinou u odkazů, kde měníme jejich výchozí chování na nějaké vlastní.

Místo metody vracející vždycky false lze jako obslužnou funkci předat přímo hodnotu false. Sémanticky je to úplně stejné, jen se zjednoduší zápis.

Delegovaná obsluha událostí

Delegované obsluze byl v minulém dílu věnován příklad. Při delegaci se využívá právě výše popsaný efekt probublávání, takže z toho vyplývají i další souvislosti. Například pokud se někde nastaví přímá obsluha a v ní se zakáže probublávání, přestane fungovat delegovaná obsluha.

Dále může delegovaná obsluha klást vyšší požadavky na zjišťování informací. Přes klíčové slovo this se totiž dostaneme nikoli k objektu, na kterém událost nastala (jako je to u přímé obsluhy), nýbrž k tomu, kde se obsluhuje. Potřebujeme-li se dostat k objektu přímo spjatém s událostí jako takovou, musí se na to jít jinak.

```
$('p').on('click', 'img', function(e) {
var o = e.target;
...
});
```

Poslouží k tomu již zmíněný objekt typu Event, který poskytuje kromě jiného položku target. Tam se nachází objekt typu Element (podobně jako v this), se kterým lze dále pracovat – ať už přímo nebo vytvořením instance jQuery. U přímo obsluhovaných událostí tam bude samozřejmě totéž jako přes this.

Objekt události obsahuje ještě dvě další podobné položky. Jednak je to currentTarget – to je element, kde se aktuálně událost obsluhuje (tedy totéž jako this). Zajímavější je ale delegateTarget, protože umožňuje už dopředu určit, kde se bude událost delegovaně obsluhovat – a třeba tuto obsluhu zrušit. Pro nedelegovanou obsluhu mají obě položky (resp. všechny tři) samozřejmě stejnou hodnotu.

Objekt události, předávání dat

Již zmíněný objekt třídy Event toho obsahuje mnohem víc. Mezi nejzajímavější položky patří tyto:

- type – typ události (řetězec použitý pro nastavení v on(), například click)
- pageX, pageY – pozice kurzoru myši relativně k dokumentu
- result – výsledek předchozí volání obsluhy
- which – stisknutá klávesa nebo tlačítko myši
- data – data předaná při nastavování obsluhy, viz dále

Existují i další položky. V závislosti na konkrétní události a dalších okolnostech se liší množina položek, jejichž hodnoty jsou definovány (nedefinované jsou samozřejmě typu undefined). Například result není definována, pokud nebyla volána nějaká obsluha vracející konkrétní hodnotu.

```
$('body').on('click', 'img', function(e) {
alert('x=' + e.pageX + ', y=' + e.pageY);
});
```

Příklad ukazuje jednoduché využití dat z objektu události kliknutí na obrázek. Událost se obsluhuje v delegovaném režimu, což umožňuje zpracovávat i kliknutí na všechny později přidané obrázky. Po kliknutí vyskočí zpráva se souřadnicemi (v rámci dokumentu), kam se kliklo.

Jak předávat data do obslužné funkce

Zvlášť zajímavé je ale využití položky data, přes kterou lze do obslužné funkce přenést libovolná data „zvenčí". Ne vždy si totiž vystačíme s tím, co lze získat přímo z objektu události nebo elementu, případně by to bylo zbytečně komplikované. Můžeme proto udělat něco jako toto:

```
var o = { x: 100, y: 200 };
$('body').on('click', 'img', o, function(e) {
alert('x=' + (e.pageX + e.data.x) + ', y=' + (e.pageY + e.data.y));
});
```

Příklad je stejný jako ten výše, předává se ovšem ještě „offset", o který se mají souřadnice přepočítat. Obslužná funkce je dostane přes objekt události. Význam by to mělo hlavně u neanonymní funkce (protože tam se využije variabilita tohoto řešení), ale pro snazší pochopení je příklad opět s funkcí anonymní. Při nastavení obsluhy tedy volíme data (může to být třeba i pouhé číslo nebo řetězec), která se předají do obsluhy.

Předávání dat při explicitním vyvolání události

Data lze předávat také při vyvolání události, například metodou trigger() – viz minulý díl. Tady to bude fungovat trochu jinak, data se předají přes parametry obslužné funkce. Ta by se nastavila například takto:

```
$('img').on('mouseenter', function(e, debug) {
if (typeof debug != 'undefined' && debug) {
alert('Simulace najetí myší');
}
...
});
```

Tato obsluha je implementovaná tak, že umožňuje předat informaci o tom, zda jde o ladicí režim (pak se uživateli zobrazí zpráva). V obslužné funkci tedy přibyl parametr debug. Pokud je definován a má hodnotu true, je obsluha volána v ladicím režimu. Událost by se pak vyvolala následovně:

```
$('img#test').trigger('mouseenter', true);
```

Událost vyvolaná s hodnotou false nebo skutečným najetí myší by v ladicím režimu nebyla. Parametrů může být předáno více – u obslužné funkce se prostě přidají na konec, ve volání trigger() se ale musí vložit do pole.

Požadavky na server a nastavování obsluhy v objektu

Na zbývající věci okolo obsluhy událostí se podíváme příště. Bude to nastavování obsluhy přes položky objektu. Bude nejlepší si to ukázat na komunikaci se serverem, kdy lze nastavit obsluhu široké palety událostí.

jQuery (12) – komunikace se serverem

AJAX, jeho schopnosti a komplikace

Jedním z pojmů často používaných v souvislosti s webovými aplikacemi je AJAX. Zkratka sice znamená *Asynchronous JavaScript and XML*, ale v praxi se často ani nepoužívá asynchronně, ani se nepřenáší XML. Běžně jím tedy označujeme obecně komunikaci javascriptové aplikace se serverem. Umožňuje získávat data za běhu aplikaci, bez nutnosti znovu načítat stránku.

Základní aplikační rozhraní se mezi různými prohlížeči liší. Tady samozřejmě nastupuje jQuery, kdy stačí používat ve všech prohlížečích stejné metody. Dále může probíhat pouze jediný požadavek na server, při pokusu odeslat další během vyřizování předchozího musí ten nový čekat.

Nelze také zapomenout na bezpečnostní omezení – většina požadavků podléhá tzv. same-origin policy, tedy omezení přístupu jen na stejné schéma (protokol), server a port. Existují metody, jak toto omezení obejít, ale obecně platí a nelze libovolně sahat na cizí zdroje.

A konečně pozor na „svody" použít synchronní komunikaci, i když tu tato možnost je a může být někdy užitečná. Je to sice implementačně jednodušší, ale aplikace během vyřizování požadavku čeká na data ze serveru a nemůže tedy obsluhovat události. Tedy uživatel se pak může „uklikat" a ono to nic nedělá.

Změna HTML ve stránce

Relativně jednoduchou, ale přitom velmi užitečnou operací je změna kusu HTML kódu (fragmentu, snippetu) ve stránce. Výhoda je v tom, že není nutno nějak specificky řešit implementaci změn ve stránce na základě dat ze serveru. Prostě se pošle požadavek, ze serveru přijdou data a ta se do stránky zapracují.

Některé webové frameworky, například Nette, umožňují vyřešit tuto operaci na serverové straně velmi jednoduše. Framework podle hlavičky (X-Requested-With: XMLHttpRequest) sám pozná, kdy byl požadavek odeslán z JavaScriptu a pošle jen příslušný kus stránky.

Požadavky se posílají typicky na základě nějakého podnětu uživatele (kliknutí, stisk klávesy…) či jiné události, například periody časovače. Obecně na tom příliš nezáleží, jen je potřeba dát pozor na to, aby se požadavky na server neposílaly zbytečně často.

```
var reqRun = false;

function sendRequest() {
    if (reqRun)
        return;

    reqRun = true;

    $.get('content.php', null, function(data) {
        $('#content').html(data);
    }, 'html')
    .always(function() {
        reqRun = false;
    });
}
```

Ve stránce máme box s identifikátorem content, jehož obsah se bude měnit podle dat ze serveru. V javascriptovém kódu deklarujeme proměnnou reqRun, která brání opětovnému odeslání požadavku, dokud není dokončen. Uvnitř funkce sendRequest(), v níž je veškerý kód pro odesílání požadavku, se nejprve proměnná otestuje (pokud je nastavena, končí se) a pak se nastaví na true.

Dalším krokem je vlastní odeslání požadavku. Tady se používá jednoduchá metoda get(), určená pro požadavky s HTTP metodou GET. Prvním parametrem je zde URL, druhý je prázdný (tam mohou být parametry požadavku, textově i formou objektu, což je docela příjemné, protože se jQuery postará o poskládání do URL), ve třetím je přímo tělo funkce pro obsluhu úspěšného obsloužení a čtvrtým je typ dat, který je ze serveru očekáván.

Metoda get() vrací objekt (typu jqXHR), který lze využít pro volání metod, jimiž se nastavuje obsluha dalších událostí. Zde se volá always(), čímž se nastaví obsluha události úplného dokončení požadavku, ať už byl úspěšný či nikoli. V tomto případě se jen nastaví proměnná reqRun na false, ale může tam být v podstatě cokoli podle potřeby.

Takto by šlo nastavit i funkci pro obsluhu úspěšného dokončení požadavku – předala by se jako parametr do metody done(). Funkce pro obsluhu selhání požadavku (ať již selhal z jakéhokoli důvodu) by se nastavila zavoláním fail().

Více možností požadavků na server

Uvedený příklad ukazuje jednoduché využití požadavků AJAX. Někdy ale potřebujeme více ovlivnit, jak bude požadavek vypadat a jak přesně proběhne. Pak je lepší si předem připravit objekt a do něj uložit potřebná data. Už i ten předchozí jednoduchý by šel přepsat do této podoby (jen výkonná část, zbytek je stejný):

```
var req = {
    url: 'content.php',
    dataType: 'html'
};

$get(req)
.done: function(data) {
    $('#content').html(data);
}
.always(function() {
    reqRun = false;
});
```

Zde můžete vidět jak přesun parametrů do samostatného objektu, tak nastavení obslužné funkce pro úspěšný požadavek až po vytvoření objektu požadavku. Úplně stejně by ale šla nastavit do položky success v objektu.

Přístup s objektem umožňuje ještě podstatně větší flexibilitu. Získáme ji zejména s metodou ajax(), kterou lze použít i pro jiné typy požadavků než GET. Hodí se to zejména v případech, kdy využíváme striktní přístup CRUD a využívají se tedy čtyři HTTP metody.
Jiné metody než GET a POST nemusí fungovat ve všech prohlížečích, proto se mnohdy využívá zjednodušené řešení, kdy GET slouží pro požadavky ke čtení a POST pro všechny požadavky nějak měnící data.

```
var req = {
method: 'POST',
url: 'save.php',
dataType: 'json',
contentType: 'multipart/form-data',
timeout: 3000,
data: {...},
success: function() {...},
statusCode: {
404: function() {
alert('Obslužný skript nebyl nalezen, kontaktujte správce.');
}
}
};

$.ajax(req);
```

Příklad ukazuje část možností parametrizace požadavku a obsluhy souvisejících událostí. Jako HTTP metoda se použije POST, jako datový typ je očekáván JSON (jQuery data zvaliduje a při úspěchu zkonvertuje, při neplatných datech nastane chybová událost), data se odesílají převedená na typ multipart/form-dataa časový limit je 3000 ms.
Do položky data se vloží objekt (je převeden na zmíněný „formulářový" formát), při úspěchu se volá příslušná obslužná funkce (zde vložená přímo do objektu). Zajímavá je položka statusCode, která umožňuje reagovat na konkrétní stavové kódy HTTP. Zde se reaguje na kód 404, tedy nenalezení požadovaného objektu. Funkce k obsluze událostí se nastavují do položek objektu.

Zjednodušující metody

Kromě již zmíněné zjednodušující metody get() jsou tu i další podobné. Například požadavky typu POST lze posílat metodou post(). K získání dat určitého typu dat jsou to specializovanější metody – konkrétně getJSON() a getScript(). Obě fungují jako get() s nastavením příslušného typu.
Pokud je tím typem „script" (bez ohledu na to, která metoda se volá), je načtený skript následně spuštěn.
Úplnou lahůdkou je pak metoda load(), která zajišťuje načtení kusu HTML a jeho použití ve stránce (tedy to, co bylo v prvním příkladu). Volá se přímo na objektu v DOM, jehož obsah se má změnit:

```
$('#content').load('content.php');
```

Velmi jednoduché, že? Pozor ale na to, že metoda nevkládá přímo to, co se načetlo – místo toho najde v načteném kódu příslušný identifikátor a nahradí jen obsah daného elementu. Není samozřejmě také nijak řešeno opakované zavolání v případě, že požadavek ještě probíhá.

Animace a efekty

To by bylo okolo událostí i požadavků na server zatím vše, i když se k nim ještě později vrátíme. Příští díl bude zaměřen na podporu jQuery pro některé vizuální efekty a animace.

jQuery (13) – animace, efekty

Opravdu živé webové stránky

Dochází-li na stránce k nějaké změně, může tato změna nastat skokem, nebo může být plynule animována. Stačí si třeba představit, že se má něco na stránce skrýt – může to najednou „zhasnout" nebo mizet postupně. Druhý způsob vypadá zajímavěji, efektněji. Podobně třeba změna rozměrů nebo pozice (určitě znáte třeba překryvné reklamní prvky, které po kliknutí na zavírací tlačítko odjedou ze stránky pryč).
S klasickým JavaScriptem by to bylo poměrně obtížně realizovatelné. Znamenalo by to aktivovat periodický časovač, který by například každou padesátinu sekundy zajistit přepočet hodnot a jejich nastavení. Bylo by potřeba napsat relativně mnoho kódu. To ale s jQuery není nutné, framework totiž obsahuje přímou podporu pro jednoduchou implementaci animací a souvisejících efektů. Často stačí napsat jen jednoduchý příkaz, který vykoná vše potřebné.

Obrázku, zmiz!

Jak zajistit zmizení obrázku? Jednoduše – *opacita* (neprůhlednost) obrázku se bude postupně snižovat, až obrázek zcela zmizí. S jQuery je to velmi snadné:

```
$('#image').fadeOut();
$('#image').fadeOut(2000);
$('#image').fadeOut(1000, 'linear');
```

Všechny tři příkazy dělají víceméně totéž, jen s drobnými odchylkami. První zajistí zmizení obrázku za výchozí čas, což je 400 milisekund. Ve druhém se použije explicitní doba mizení, tedy 2000 ms. A ve třetím obrázek zmizí za 1000 ms, ovšem lineárně – výchozí je totiž způsob animace (tzv. easing) nazvaný „swing", kdy je rozjezd i dojezd pomalejší.
Další způsoby animace lze přidat pomocí pluginů, bude o tom řeč v některém z pozdějších dílů seriálu.
Metodu fadeOut() lze ještě dále parametrizovat – předá se jí objekt s hodnotami, které chceme nastavit:

```
var opts = {
duration: 200,
easing: 'linear',
complete: function() {
alert('Hotovo!');
}
};

$('#image').fadeOut(opts);
```

Obrázek zmizí lineárně za 200 ms. Po dokončení animace ze zavolá definovaná funkce, tedy v tomto případě ta anonymní, volající zobrazení zprávy pro uživatele. Tam by šla přiřadit také další animace, která by následovala, to lze ale zajistit i jednodušeji:

```
$('#image').fadeOut(2000).fadeIn(3000);
$('#image').fadeOut(2000).delay(5000).fadeIn(3000);
```

Animace lze totiž řetězit za sebou – po skončení jedné bude pokračovat další. V tomto případě na prvním řádku obrázek nejdřív zmizí a pak se hned zase objeví (pomaleji, než zmizel). Druhý řádek přidává prodlevu, která proběhne po zmizení obrázku, než se začne opět objevovat.

Chcete, aby obrázek jen „polozmizel" (tedy zůstal částečně zobrazen, transparentní)? I to lze zajistit:

```
$('#image').fadeTo(200, 0.3);
```

Obrázek se zde animuje do opacity 30 %, takže zůstane částečně vidět. Jde to samozřejmě i opačně, tedy směrem k vyšší opacitě.

Metody fadeOut() a fadeIn() jsou z hlediska volání prakticky identické, každá má řadu možností zavolání. V porovnání s nimi je metoda fadeTo() relativně omezená, například jí nejde předat objekt s parametry.

Ještě je k dispozici metoda fadeToggle(), která skrývá nebo zobrazuje prvek podle toho, v jakém stavu se nacházel (zda byl zobrazený nebo skrytý).

„Složitější" mizení

Prvky mohou mizet nebo se naopak objevovat ještě trochu jiným způsobem, kombinujícím v jedné animaci dva efekty – změnu opacity a velikosti. Není to ale o nic složitější:

```
$('#image').hide(300);
$('#image').show(2000);
$('#image').hide(400).delay(1000).show(200);
```

Obrázek zmizí za 300 ms tak, že se bude zmenšovat a současně průhlednět – až zmizí docela. Příkaz na druhém řádku ho naopak pozvolna zobrazí. Třetí příkaz kombinuje zmizení, prodlevu a opětovné zobrazení. Pozor však na toto:

```
$('#image').hide();
$('#image').show();
```

Na rozdíl od metod fadeOut() a fadeIn() tu výchozí hodnota není 400 ms, nýbrž 0. Prvek tedy zmizí nebo se zobrazí okamžitě. Animace tedy v podstatě neproběhne, vykoná se jediná operace.

Zarolování a vyrolování

Určitě znáte ten efekt, kdy se po kliknutí například na záhlaví nějakého boxu na stránce obsah tohoto boxu zaroluje (a zůstane jen to záhlaví), resp. se naopak vyroluje. I na to jsou v jQuery připraveny metody:

```
$('.boxhdr').click(function() {
$(this).parent().child('.boxcont').slideToggle();
});
```

Příklad vychází z toho, že všechna záhlaví zarolovatelných boxů mají třídu boxhdr a obsah boxů boxcont(přičemž oba prvky jsou potomky boxu jako takového). Pak stačí nastavit reakci na kliknutí pro všechna záhlaví tak, že se pro záhlaví zjistí obsah boxu a ten se buď zaroluje nebo vyroluje, podle aktuálního stavu – to zajistí metoda slideToggle().

Pro samotné zarolování lze použít slideUp(), pro vyrolování slideDown(). Také tady je k dispozici celá škála možností jako u fadeIn() a fadeOut().

Další možnosti animací

V příštím dílu seriálu se podíváme na další věci, které lze pomocí animací dělat a také jak lze ovlivnit samotný proces animace (snímkovou frekvenci a další vlastnosti).

jQuery (14) – animace a jejich nastavení

Animujte (skoro) cokoliv

Kromě specializovaných animačních metod, které už znáte z minulého dílu, je v jQuery k dispozici také obecná metoda animate(). Ta umožňuje animovat téměř libovolné vlastnosti nastavitelné pomocí CSS, a to i více vlastností najednou. Vždy se ve volání metody předá cílový stav, kterého má být dosaženo na konci animace. Tady je příklad:

```
var props = {
  left: 500,
  top: 200
  };

$('#box').animate(props, 2000);
```

Do objektu v proměnné props se nastaví cílová poloha, do které má být daný box posouván (z aktuální polohy). Metodě animate() lze předat další parametry, v tomto případě je to doba trvání animace (2000 ms), může to být styl animace (*easing*), případně funkce volaná po skončení. Další možností je předat metodě druhý objekt, kde toho lze nastavit ještě mnohem víc – viz minulý díl.

Ne každou vlastnost popsaná pomocí CSS lze animovat. Základní jQuery umí jen jednoduché (skalární) číselné hodnoty, například pozice nebo rozměry. Naopak barvy animovat nelze, protože pro jejich popis je potřeba více hodnot (jsou to vektorové parametry). Pro animaci barev však existuje plugin jQuery.Color, který v případě potřeby podporu zajistí.

Animační fronta

Všechny dosavadní příklady měly jedno společného – jakmile animace začala, už se do ní nesahalo a doběhla přesně tak, jak bylo předem nadefinováno. Jenže to mnohdy není úplně to, co bychom si přáli. Zvlášť u složitějších, déle trvajících animací bývá žádoucí mít možnost do procesu zasáhnout a buď animaci zastavit nebo změnit její podobu.

Animace se obvykle spouštějí pomocí front, ve kterých čekají ty efekty, na které přijde řada později. Běžně se u objektu používá jen jediná fronta (nazvaná fx), ale obecně jich může být víc. Fronty lze použít i k jiným činnostem než k animacím, byť se právě pro animace používají asi nejčastěji.

Zastavení animace

Vezměme nyní příklad z minulého dílu, kde se po kliknutí na záhlaví zaroluje nebo vyroluje box. Pokud někdo klikne ještě před doběhnutím animace, má smůlu a musí počkat, než animace doběhne. To lze ale snadno změnit:

```
$('.boxhdr').click(function() {
$(this).parent().child('.boxcont').stop().slideToggle();
});
```

Všimněte si drobné změny – přidání metody stop(). Ta zastaví probíhající animaci, což umožní okamžité rozběhnutí animace opačným směrem. V tomto případě probíhá jen jediná animační operace, takže není potřeba nijak manipulovat s frontou. Pokud bychom měli operací více, bylo by žádoucí frontu vyprázdnit (hodnotou true v parametru metody stop()):

```
$(this).parent().child('.boxcont').stop(true).slideToggle().delay(100);
```

Celé to lze ještě upravit tak, že se animace dokončí skokem – přímo se nastaví hodnota, jako kdyby animace doběhla do konce. Tady je opět příklad:

```
$('#image').show(5000);
```

```
$('#image').click(function() {
$(this).stop(false, true);
});
```

Animovat se bude objevování obrázku. Jakmile na něj uživatel klikne, animace se zastaví a obrázek se naplno zviditelní. Výše uvedené příklady pracují všechny s výchozí frontou fx;název případné jiné fronty by se uvedl jako první parametr volání.

Přidávání a odebírání efektů, vymazání fronty

Efekty (animační operace) nebo i jiné funkce lze za běhu animace přidávat a odebírat. Přidáváme-li na konec fronty běžnou animační funkci, lze to udělat tak, jako kdybychom začínali animovat od začátku. Jiné je to ale u funkcí, které nejsou přímo animační, tam to za běhu tak jednoduše nejde.

```
$('#image').queue(function() {
alert('Upozornění');
$(this).dequeue();
});
```

Kód přidá na konec fronty k vykonání funkci, která zobrazí upozornění. Po provedení kódu je nutné zavolat metodu dequeue(), která zajistí odebrání z fronty a spuštění další funkce v řadě.

Metodou queue() lze provádět i složitější manipulace s frontou. Lze získat pole všech funkcí ve frontě, upravit ho podle potřeby a zase vrátit do fronty (nebo nahradit úplně jiným polem):

```
var q = $('body').queue();
q.shift();
$('body').queue(q);
```

Tento kód odebere z fronty první funkci v pořadí (která čeká na provedení; nemá vliv na právě probíhající animaci). Chceme-li frontu úplně vymazat, lze to udělat buď metodou stop() s parametrem true (viz výše); to ale zastaví i probíhající animaci. Nebo můžeme frontu vyprázdnit samostatně:

```
$('body').clearQueue();
```

Použití vlastní fronty

Běžně není důvod používat na témže objektu DOM více front než jednu. Proto si většinou vystačíme s výchozí frontou fx. Někdy ale můžeme potřebovat, aby se animace nebo jiná posloupnost operací spustila nezávisle na tom, co už běží. Pak se další fronta hodí.

Při použití vlastní fronty je potřeba používat u animačních metod vždy tu verzi, která přijímá objekt s vlastnostmi animace. Může to tedy vypadat třeba takto:

```
var opts = {
duration: 5000,
queue: 'myqueue'
};

$('#image').show(opts);
```

Tím se zajistí vložení animace (postupného zobrazení obrázku) do fronty myqueue. Ovšem pozor – na rozdíl od výchozí fronty se animace nespustí automaticky, je potřeba explicitně odebrat první prvek a spustit jeho funkci (viz výše):

```
$('#image').dequeue('myqueue');
```

Při operacích s frontou se pak musí samozřejmě používat její název, například vymazání fronty by vypadalo takto:

```
$('#image').clearQueue('myqueue');
```

Nastavování animací

Upřímně řečeno, moc se toho nastavovat nedá. Jsou jen dva nastavitelné parametry – vypnutí animací a frekvence animačních kroků. Vypnutí se hodí třeba v případě, že se stránka otevře na nějakém hodně pomalém zařízení a to je „nestíhá" zobrazovat. Někoho mohou animace také rušit. Stačí pak například vytvořit malé tlačítko:

```
<span id="no-anim">Vypnout animace</span>
```

Na tlačítko se pak „pověsí" vypínací kód:

```
$('#no-anim').click(function() {
$.fx.off = true;
});
```

Vrácením hodnoty na false se animace opět zapnou. Hodnotu lze samozřejmě i číst a tedy využít k zobrazení stavu, resp. volbě vhodného popisku tlačítka i příští hodnoty. Vypnutí animací neznamená, že by neproběhly vůbec – jen se prostě přímo nastaví cílový stav.

Druhým nastavitelným parametrem je frekvence kroků, resp. jejich perioda. Výchozí hodnota je 13 ms, což odpovídá kmitočtu 77 Hz. Zkracovat tuto periodu příliš nemá smysl, protože snímková frekvence běžných LCD monitorů bývá 60 Hz. Naopak může mít smysl ji přiměřeně prodloužit, sníží se tím zátěž procesoru a na přenosných zařízeních to tedy i poněkud prodlouží výdrž na baterii (to se ale projeví jen tam, kde animace trvá relativně dlouho; pro animace rozbalování se pro prakticky neprojeví).

Následující kód nastaví periodu 40 ms:

```
$.fx.interval = 40;
```

S delší periodou může animace působit méně hladce, je to ale lepší řešení, než když procesor výchozí periodu nestíhá a animace se zasekává.

Co zbývá v jQuery

Příští díl bude poslední o „holém" jQuery (pak už bude následovat jQuery UI). Proto do něj spadlo to ostatní užitečné, co v jQuery ještě je – například připojování dat k objektům nebo odložené zpracování.

jQuery (15) – připojování dat k objektům, odložené zpracování

Proč připojovat data k objektům

Mnohdy je potřeba v souvislosti s objekty DOM pracovat s nějakými dalšími daty. Ty si lze samozřejmě uložit do nějaké globální proměnné, ale to pak znamená, že v globálním jmenném prostoru přibývají identifikátory. Také si lze globálně vytvořit objekt, jehož položky budou data různých objektů DOM.

Ale proč to všechno, když jQuery nabízí elegantnější a jednodušší mechanismus? K objektům DOM lze připojovat data a pak s nimi dále pracovat. Rozhraní má podobu klasického úložiště typu klíč–hodnota.

Data objektů DOM

Nastavování a získávání dat

Je to velmi jednoduché – prostě se na instanci jQuery zavolá metoda, která data připojí:

```
$('#search-form').data('timeout', 2000);
$('#search-form').data('timeouts', [ 2000, 1000, 800 ]);
```

První řádek připojí k objektu s daným identifikátorem data s klíčem „timeout" a jednoduchou číselnou hodnotou. Druhý řádek přiřadí pole hodnot; podobně by se dal použít i objekt. Pokud se již nastavená hodnotu pro určitý klíč nastaví znovu, data se přepíší. Lze nastavit i více dat najednou:

```
var obj = {
timeout: 2000,
msg: 'Vypršel časový limit.'
};

$('#search-form').data(obj);
```

Zafunguje to tak, jako kdyby se hodnoty nastavily postupně. Případné dříve nastavené hodnoty se přepíší, ostatní se nově přidají. Lze tedy používat jak již zmíněné rozhraní typu klíč–hodnota, tak i běžný datový objekt, viz dále.

Pokud je v instanci jQuery více objektů DOM, nastaví se data každému z nich jednotlivě, data tedy nejsou sdílena.

Potřebujeme-li nastavenou hodnotu později získat, použije se tatáž metoda:

```
var to = $('#search-form').data('timeout');
```

Při zavolání bez parametrů metoda vrátí kompletní objekt:

```
var obj = $('#search-form').data();
```

S takto získaným objektem lze dále pracovat, například změnit hodnoty některých položek a znovu ho nastavit (vrácený objekt je kopií dat, není to reference na živý objekt, kde by se změna položek okamžitě projevila).

Je-li v instanci jQuery více objektů DOM, metoda v obou případech vrátí data prvního z nich.

K dispozici je ještě jedno rozhraní pro nastavování a zjišťování dat:

```
$.data(document.body, 'timeout', 1000);
```

Metoda se volá přímo na třídě jQuery a místo selektoru se používá instance typu Element. V tomto případě nemůže nastat situace, že by se operace týkala více objektů DOM najednou.

Test existence, smazání dat

Potřebujeme-li otestovat, zda nějaká data u objektu DOM existují, je to snadné:

```
if ($('#menu').hasData('effect')) {
…
}
```

U testu platí totéž jako u jejich získávání, tedy v případě více objektů DOM ve výběru se test týká jen prvního z nich. Obdobně můžeme data také odstranit, pokud už nejsou potřeba:

```
$('#menu').removeData('effect');
$('#menu').removeData([ 'effect', 'timeout' ]);
$('#menu').removeData();
```

První příkaz odstraní hodnotu uloženou pod klíčem „effect" (ze všech objektů DOM, které selektor vybere), druhý hodnoty pro všechny klíče v daném poli, poslední pak odstraní všechno (s určitými výjimkami, viz dále). Také tady je k dispozici i druhé rozhraní, kdy se metoda volá na třídě jQuery.

Atributy HTML5, obsluha událostí

Pozor na to, že jQuery využívá tento mechanismu také pro své vnitřní potřeby. Jednak do připojených dat uloží atributy (data-*) daného elementu, což může někoho překvapit, když si uložené hodnoty vytáhne a uvidí tam „cosi navíc". Tyto hodnoty navíc nelze odstranit voláním removeData(), takže i pokud se tato metoda zavolá bez parametrů, mohou v objektu zbýt data.

Detekuje-li jQuery, že jsou data v atributu uložena jako validní JSON, zpracuje je a převede na javascriptovou objektovou strukturu. V ostatních případech zůstávají v původní, textové podobě.

Dalším překvapením může být, že si jQuery k objektům připojuje svá data pro obsluhu událostí. Pokud si tedy nastavíte obsluhu například pomocí metody bind() nebo některé ze zkratek (třeba click()), najdete data v objektu vráceném metodou data()zavolanou bez parametrů.

Odložené zpracování

Pokud si vzpomenete na animační frontu z minulého dílu, budete zhruba v obraze, co znamená odložené zpracování. Zkrátka se připraví a (dříve či později) spustí nějaká činnost, která může chvíli trvat. Pomocí front lze zařídit hodně, ale někdy je potřeba ještě víc, například sledovat průběh nějaké operace.

Už poměrně dávno (ve verzi 1.5) proto přibyl do jQuery objekt Deferred. Ten nabízí širokou škálu možností pro odložené zpracování, navíc ho jQuery používá i interně, například pro síťovou komunikaci (metoda ajax()a další).

Příprava a spuštění odloženého zpracování

Příprava odloženého zpracování začíná získáním instance třídy Deferred z tovární metody třídy jQuery:

```
var d = $.Deferred();
```

Dále je potřeba nastavit obsluhu událostí. To lze učinit přímo na instanci objektu Deferred, pokud k tomu má ale dojít někde jinde, je lepší použít instanci třídy Promise. Ta poskytuje jen metody k nastavení obsluhy událostí a nikoli ostatní metody objektu Deferred, jejichž zavolání by zasáhlo do probíhajícího zpracování.

```
var p = d.promise();
p.done(function() {
alert('Úspěšně hotovo');
})
.fail(fuction() {
alert('Došlo k chybě');
});
```

Výše uvedený kód nastaví reakce na úspěšné dokončení zpracování a na chybu. Lze nastavit i více obslužných funkcí najednou, stačí místo jedné funkce předat jejich pole. Tímto je základní příprava skončena a zbývá napsat výkonnou funkci, která zajistí vlastní odložené zpracování.

```
function mywork(d) {
…
if (ok) {
d.resolve();
} else {
d.reject();
```

```
}
}
```
Funkce (může to být samozřejmě i metoda objektu) vykoná svou práci a podle výsledku zavolá na objektu Deferred buď resolve() při úspěchu nebo reject() při chybě. V příkladu je vše bez parametrů, ale lze předávat i parametry.
```
setTimeout(mywork, 1000);
```
Jeden z obvyklých způsobů spouštění odloženého zpracování je časovač (viz výše), ale obecně lze používat i jiné cesty (třeba reakci na činnost uživatele). Za běhu lze zvenčí zjišťovat stav zpracování (bude buď „pending", pokud ještě běží, nebo „resolved" či „rejected" při úspěchu, resp. chybě):
```
alert('Stav zpracování: ' + p.state());
```
Sledování průběhu zpracování

Tento mechanismus umožňuje kromě jiného i sledovat průběh, například posouvat nějaký ukazatel, zobrazovat čísla apod. Je potřeba nastavit reakci podobně, jako se to dělá pro dokončení procesu:
```
p.progress(function(val) {
$('#complete').text(val);
});
```
Při vlastním zpracování se pak provádí notifikace o jeho průběhu:
```
for (var i = 1; i <= 100; i++) {
...
d.notify(i);
}
```
V příkladu je zpracování v cyklu, které informuje po každém jeho proběhnutí. Na závěr se samozřejmě zavolá metoda podle výsledku zpracování.

Mechanismus odloženého zpracování umí ještě různé další lahůdky, například řetězení a filtraci. Informace najdete v dokumentaci nebo podrobném tutoriálu.

jQuery UI

To by bylo o základním jQuery zatím vše, i když se k němu budeme ještě vracet. Příští díl seriálu začne novou část, která bude věnována nadstavbě: jQuery UI. Ta je zaměřena vysloveně na uživatelské rozhraní a je zajímavá přinejmenším stejně jako jádro jQuery.

jQuery (16) – úvod do jQuery UI

Co jQuery (ne)umí

Framework jQuery poskytuje poměrně širokou škálu funkcí, které podstatně zjednodušují vývoj javascriptových webových aplikací. Některé věci byste tam však hledali marně. To se týká zejména různých ovládacích komponent a jejich fungování. Filosofie je totiž taková, že v základním frameworku ani být nemají – jsou soustředěny do nadstavby nazvané jQuery UI. Pokud pokročilejší práci s UI nepotřebujete, nemusíte jQuery UI využívat, vystačíte si se samotným jQuery. V případě potřeby pak můžete kdykoli přidat jQuery UI a použít z něj, co je potřeba. Styl práce je samozřejmě stejný jako u jQuery (tedy především využití selektorů).

Kde a jak jQuery UI získat

V první řadě je potřeba mít (nebo využít z CDN) základní jQuery. Současná řada verzí jQuery UI, tedy 1.11, vyžaduje jQuery verze 1.6 nebo vyšší. Totéž se týká i dvou starších, nicméně stále podporovaných řad 1.10 a 1.9.

Dalším krokem je stažení jQuery UI. Na rozdíl od jQuery je jQuery UI potřeba vždycky stáhnout, není k dispozici prostřednictvím CDN (nebo být eventuálně může, ale to nemusí vždy vyhovět potřebám aplikace). Je tomu tak proto, že vždy obsahuje výchozí grafické téma (případně nemusí, ale pak nelze využívat řadu funkcí) a těch může být nespočet. Lze si vybírat z připravených témat nebo si sestavit vlastní.

Navíc pokud nepotřebujete celý balík, můžete si připravit takový, kde budou jen potřebné funkce. K tomu všemu slouží Download Builder. Vyberete si verzi, zaškrtnete požadované skupiny funkcí nebo přímo jednotlivé funkce a vyberete si téma (viz dále). Po stisku tlačítka **Download** si můžete stáhnout hotový balíček, který obsahuje vše potřebné – jak javascriptové soubory, tak i soubory kaskádových stylů a v nich použité obrázky.

Pokud není důvod držet balíček co nejmenší, můžete nechat zaškrtnuté všechno (výchozí stav). Zejména ve fázi vývoje je to nejlepší cesta, pro produkční nasazení se pak může případně použít ořezaná verze, ve které budou jen použité komponenty. Balíček potom rozbalíte k souborům dané webové aplikace. Co se týká volby tématu, k dispozici je široká škála různých připravených témat. Prohlédnout si je můžete v ThemeRolleru, který jinak slouží také pro návrh témat vlastních.

Na kartě **Gallery** vidíte připravená témata – jako výchozí je při stahování nastaveno **UI lightness**, ale někdo může dát přednost tlumenějším či tmavším barvám nebo si barevnost přizpůsobit své aplikaci.

Použití v HTML kódu

Do HTML kódu je potřeba přidat (samozřejmě kromě odkazu na jQuery) odkaz jak na javascriptový soubor jQuery UI, tak i na soubor CSS. Pro vývoj použijte „plnohodnotné" verze (s čitelným kódem), pro produkci verze „minifikované" (mají v názvu „min"). Vývojový kód by tedy vypadal například takto:
```
<link rel="stylesheet" type="text/css" href="jquery-ui.css">
<script type="application/javascript" src="jquery.js"></script>
<script type="application/javascript" src="jquery-ui.js"></script>
```
Pro produkční kód by stačilo před příponu názvu jednotlivých souborů přidat „min", bude to tedy například jquery-ui.min.js. Nemáte-li samostatný soubor s jQuery, můžete použít ten, který je automaticky přibalován do balíčku. Příslušný řádek by pak (při adresářích ponechaných beze změn) vypadal takto:
```
<script type="application/javascript" src="external/jquery/jquery.js"></script>
```
Co všechno jQuery UI obsahuje

V jQuery jsou funkce různých druhů, při tvorbě balíčku ke stažení jsou rozděleny do těchto skupin:

- **UI Core** – Obsahuje základní funkcionalitu využívanou v různých widgetech, efektech a dalších funkcích.
- **Interactions** – Interakce uživatele a aplikace. Můžete například velmi snadno učinit grafické prvky přemístitelné myší nebo umožnit měnit myší jejich velikost.
- **Widgets** – Grafické prvky (widgety), například tlačítko, dialog, posuvník nebo kalendář.
- **Effects** – Efekty všeho druhu, například blednutí, odrážení, pulsování apod.

V dalších dílech seriálu se postupně podíváme na to, jak se s jednotlivými funkcionalitami pracuje. Jako první přijdou na řadu uživatelské interakce.

jQuery (17) – přesun objektů myší v jQuery UI

Posouvání objektů myší

Představme si například nějaký nástrojový panel, který chceme mít na stránce tam, kde nepřekáží a kde je naopak co nejvíc při ruce. Vhodné místo se ale může měnit podle toho, co člověk zrovna dělá. Proto není nic jednoduššího, než si panel posunout myší.

Existují na to v zásadě dvě cesty. Lze si vše potřebné naimplementovat ručně, nebo využít prostředků jQuery UI a přidat jen zcela triviální kus kódu. Můžeme mít třeba takovýto kus HTML kódu reprezentující panel s tlačítky:

```
<div id="toolbar">
<button id="button1"></button>
<button id="button2"></button>
...
</div>
```

Udělat panel přetahovatelný lze s jQuery UI velmi snadno:

```
$('#toolbar').draggable();
```

To je opravdu všechno, pokud není třeba nic speciálního. Od zavolání této metody lze s panelem hýbat myší.

Metoda draggable() ovšem přijímá parametry, s jejichž pomocí lze dělat další „kouzla“. Chcete třeba povolit pouze svislý posun a během něj nechat panel částečně zprůhlednět? Žádný problém:

```
var param = {
axis: 'y',
opacity: 0.5
};
$('#toolbar').draggable(param);
```

Panel se může také třeba přichytávat k jiným objektům (s nastavitelnou tolerancí):

```
var param = {
snap: true,
snapTolerance: 10
};
$('#toolbar').draggable(param);
```

Podobných vlastností je k dispozici celá řada, bližší informace najdete v dokumentaci.

Zjišťování a změny parametrů

Hodnoty nastavených parametrů lze podle potřeby zjišťovat a měnit. Chceme-li například zjistit, jak je nastavena tolerance přichytávání (viz výše), stačí zavolat:

```
var tol = $('#toolbar').draggable('option', 'snapTolerance');
```

Obdobně se nastavená hodnota také změní:

```
$('#toolbar').draggable('option', 'snapTolerance', 5);
```

Můžeme získat také všechny parametry najednou, budou uloženy do objektu, jehož položky budou odpovídat jednotlivým parametrům.

```
var opts = $('#toolbar').draggable('option');
var tol = opts.snapTolerance;
```

Uvedený kód dělá totéž jako ten výše, tedy zjišťuje nastavenou toleranci přichytávání. Tento způsob má ale logiku až při práci s více parametry najednou.

Vypnutí a zrušení posuvnosti

Někdy můžeme potřebovat nastavenou posuvnost vypnout nebo zcela zrušit. Jaký je mezi oběma operacemi rozdíl? Vypnutí jen zruší možnost posouvat objekt myší, všechny parametry však zůstanou nastaveny a uplatní se po případném opětovném zapnutí:

```
$('#toolbar').draggable('disable');
```

Obdobně se posuvnost zase povolí:

```
$('#toolbar').draggable('enable');
```

Úplné zrušení posuvnosti oproti tomu znamená, že se nastavení parametrů vymaže a vše se uvede do původního stavu, jako kdyby objekt nikdy posuvný nebyl.

```
$('#toolbar').draggable('destroy');
```

Události související s posouváním

Při posouvání objektů nastávají různé události a může být užitečné na ně nějak reagovat. Typickým příkladem může být uložení nové pozice přemísťovaného objektu (ať už do prohlížeče nebo na server).

```
var param = {
stop: function() {
localStorage.setItem('toolbar', JSON.stringify($(this).offset()));
}
};
$('#toolbar').draggable(param);
```

Kód v příkladu nastaví obsluhu události ukončení posunu, tedy uvolnění tlačítka myši po přesunu. Obslužná funkce uloží do lokálního úložiště (v prohlížeči) pozici panelu, kterou lze při obnovení stránky nebo jiných situacích zpět načíst a použít. Pozici lze získat jak z objektu (použito v příkladu), tak i z druhého parametru předávaného do obslužné funkce.

Podobně lze reagovat na začátek přesouvání, na samotný posun (než proběhne; událost nastává během pohybu, aniž by byl objekt „upuštěn“) a na nastavení posouvatelnosti. Obslužnou funkci lze nastavovat i po počátečním zapnutí posouvatelnosti:

```
$('#toolbar').on('dragstop', function() {});
```

Uvedený kód nastaví prázdnou funkci, obdobně by se nastavila i funkce, která něco dělá. Využití metody on()má výhodu v univerzálnosti použití, podrobnosti najdete v desátém dílu seriálu. Nabízí se samozřejmě také metoda one(), pokud je potřeba zavolat obsluhu pouze jednou. Příslušná událost se vždy jmenuje drag*, kde je místo hvězdičky název události podle dokumentace jQuery UI (tedy pro stop je to dragstop apod.).

Drag & drop

S přetahováním objektů souvisí také jejich pouštění. Někdy potřebujeme větší interaktivitu než jen ponechání objektu na novém místě. Pak se do akce zapojí také objekt na místě, kde přetahovaný objekt upustíme. Příště se tedy podíváme na to, co tento druh interakce umožňuje a jak ho využívat.

jQuery (18) – drag&drop v jQuery UI

Stručně o drag&drop

Technika „drag&drop", tedy česky „táhni a pusť", se v grafických uživatelských rozhraních vyskytuje už velmi dlouho a uživatelé si tedy zvykli přetahovat myší různé objekty na jiné objekty, které pak s přetaženými a upuštěnými objekty nějak nakládají. Například při přetažení ikony souboru v souborovém správci na ikonu adresáře se soubor přesune či zkopíruje do daného adresáře.

Podobně, jako jsme zvyklí používat tuto techniku u běžných desktopových aplikací, ji už delší dobu celkem běžně využíváme i na webu (nebo dokonce v kombinaci obojího). Zrovna zmíněný správce souborů může být realizován i webově a tam se právě tento přístup výborně uplatní, podobných případů je ale mnoho.

První fází drag&drop je prosté přetahování objektu, druhou pak jeho upuštění. V ten okamžik je cílový objekt uvědoměn o tom, že se mu předávají data a o jaká data jde. Může se tedy rozhodnout, zda je vůbec přijme, a pokud ano, jak s nimi naloží.

Drag&drop v jQuery UI

Framework jQuery UI implementuje tuto techniku pomocí kombinace přetahování (viz minulý díl) a pouštění (realizovaného speciálním widgetem). Objekt, který má být způsobilý k tomu, aby se na něj něco upustilo, je potřeba připravit jako widget Droppable. To se zařídí podobně jako u objektu určeného k přetahování – Draggable.

```
<div id="draggable"></div>
<div id="droppable"></div>
```

Tento kus HTML vytvoří ve stránce jeden element, který půjde přemísťovat myší, a druhý, do kterého půjde ten první upustit. Tohle samo o sobě ještě samozřejmě neudělá nic. Je potřeba to teprve „oživit":

```
$('#draggable').draggable();
$('#droppable').droppable();
```

Reakce na upuštění

Výše uvedený kód je zcela základní a jen zprovozní fungování drag&drop. Nezajistí nic dalšího, sice tedy půjde objekt přetáhnout a pustit, nic dalšího se ale nestane. Je totiž potřeba implementovat reakci na upuštění. Druhý řádek tedy přepíšeme do složitější podoby:

```
$('#droppable').droppable({
    drop: function() {
    alert('Upuštěno!');
    }
});
```

Reakce na událost drop v tomto případě spočívá pouze v oznámení, že bylo něco upuštěno. Obvykle chceme ale něco zajímavější, například vložit přesouvaný objekt do toho, na který se upustí. Samozřejmě není problém to realizovat:

```
$('#droppable').droppable({
    drop: function(event, ui) {
    $(this).append(ui.draggable);
    }
});
```

Filtrace upustitelných objektů

Nyní cílový objekt přijme cokoli přetahovatelného, co na něj upustíme. Jenže to nemusí být vždy správné. Jak například zařídit, aby cílový objekt přijímal jen odstavce textu? Jednoduše:

```
$('#droppable').droppable({
    accept: 'p',
    drop: ...
    });
```

Funkce přiřazená jako drop bude stejná jako v předchozím případě, nový je tu pouze parametr accept. Do něj lze vložit selektor (zde „p", tedy odstavec) nebo funkci, která vrátí true v případě, je je objekt akceptován.

Zvýraznění cílového objektu

Dalším požadavkem může být, aby se cílový objekt nějak zvýraznil a tedy uživatel věděl, kam může přetahovaný objekt pustit. Lze například použít parametr activeClass:

```
$('#droppable').droppable({
activeClass: 'text-droppable'
    ...
});
```

Parametrem se určí třída, která se přidá v okamžiku, kdy je tažen nějaký objekt, který sem lze pustit. Třídě se pak nastaví nějaké vizuální vlastnosti, třeba barevné pozadí, rámeček nebo něco takového.

Všechny widgety Droppable získávají automaticky třídu ui-droppable, kterou lze také využít k tomuto účelu, není však selektivní. Nastavení této třídy lze zakázat nastavením parametru addClassesna hodnotu false.

Má-li se cílový objekt zvýraznit až v okamžiku, kdy se něco akceptovaného táhne přes něj, využije se k nastavení třídy parametr hoverClass. Podobně je k dispozici také možnost reagovat na události, které s tím souvisejí.

```
$('#droppable').droppable({
    over: function() {
    $(this).fadeTo(500, 0.5);
    },
    out: function() {
    $(this).fadeTo(500, 1);
    }
    …
});
```

Událost over nastane při vstupu přetahovaného objektu nad cílový – zde to spustí postupné zprůhledňování (až na 50 % za 500 ms). Naopak událost out nastane při opuštění cílového objektu a zde se na ni reaguje opětovným zneprůhledněním.

Další vlastnosti Droppable

Ještě jedna důležitá věc si zaslouží zmínku – tzv. *hladovost*. Pokud lze objekt upustit na nějaký objekt, který má pod sebou další potomky, na které lze něco upustit, bude upuštění znamenat, že událost nastane v celé hierarchii a to nemusí být vždy žádoucí. Pak lze parametr greedy nastavit na true, tím nejnižší objekt v hierarchii „sežere" upuštěný objekt a ten už se výše do hierarchie nedostane.

Vlastnosti widgetu Droppable lze měnit i později, podobně jako je tomu u Draggable (nebo zjišťovat jejich hodnoty). Protože k tomu dost informací najdete v minulém dílu, bude zde stačit krátký příklad na ukázku:

```
$('#toolbar').droppable('option', 'greedy', false);
```
Tímto příkazem se nastaví již zmíněný parametr greedy na hodnotu false. Minulý díl obsahuje také informace o tom, jak widget deaktivovat, opětovně aktivovat a úplně odstranit. Tady je to úplně stejné.

Změna rozměrů pomocí myši

Příští díl bude věnován dalšímu typu uživatelské interakce, a to změně rozměrů různých prvků pomocí myši.

jQuery (19) – změna rozměrů pomocí myši v jQuery UI

Zvětším toto a zmenším tamto...

S tím, jak se i rozsáhlejší aplikace přesouvají na web, roste samozřejmě také potřeba přizpůsobovat si prostředí aplikací. Jedním z takových přizpůsobování je změna rozměrů různých prvků. Uživatelsky nejpřirozenější je měnit rozměry myší – prostě se uchopí některá hrana prvku nebo jeho roh a popotáhne se podle potřeby.

Podporu pro tento druh interakcí poskytuje v jQuery UI widget Resizable. Se změnami rozměrů je to podobné jako s posouváním – snadno se aktivují a lze je ovládat podle potřeby. Začněme tím úplně nejjednodušším, prostým umožněním měnit velikost obrázku. HTML bude vypadat takto:

```
<img src="image.jpg" alt="Obrázek" id="img">
```
Základní javascriptový kód, který umožní změny rozměrů, je velmi jednoduchý:
```
$('#img').resizable();
```
Tento kód použije výchozí nastavení. Rozměry lze měnit prakticky libovolně, ve všech směrech (všechny hrany a rohy), bez omezení poměru stran, od téměř nulové velikosti až prakticky bez horního limitu.

Složitější změny rozměrů

To ale není vždy to „pravé ořechové", mnohdy chceme ovlivnit, jak se bude změna rozměrů chovat.

Nastavení mezí

Často se hodí nastavit meze „od – do", protože příliš malý nebo velký prvek by nepřinesl nic pozitivního, naopak by komplikoval další práci. Lze omezit šířku i výšku, nastavit minimum i maximum.
```
var props = {
minWidth: 50,
maxWidth: 500,
minHeight: 50,
maxHeight: 300
};

$('#img').resizable(props);
```
Výše uvedený kód nastaví všechny meze, a to prostřednictvím objektu, který se předá metodě připravující widget. Kteroukoli mez lze vynechat (pokud není potřebná) nebo naopak změnit či přidat později:
```
$('#img').resizable("option", "maxHeight", 200);
```
Další možností je omezit maximální velikost hranicemi nějakého jiného objektu. Možností je více, tady jsou dva příklady:
```
$('#img').resizable({ containment: 'parent' });
$('#img').resizable({ containment: '#desktop' });
```
První příkaz omezí rozměry přímo nadřazeným objektem (rodičem). Podobně lze využít také řetězec „document". Druhý příkaz využije selektor, který v tomto případě vybere objekt podle identifikátoru, lze ale použít prakticky cokoli. Místo selektoru lze předat jako parametr také instanci třídy Element.

Práce s poměrem stran

Zrovna u obrázků bývá výhodné implementovat změnu rozměrů tak, aby byl dodržen určitý poměr stran (aby se obrázek nedeformoval). Tady jQuery UI umožňuje využít dva mechanismy – dodržení původního poměru nebo explicitně definovaného poměru.
```
$('#img').resizable({ aspectRatio: true });
$('#img').resizable({ aspectRatio: 16/9 });
```
Smysl je z kódu snadno pochopitelný. První řádek zajistí dodržování původního poměru stran, druhý řádek pak bude udržovat poměr 16:9.

Tolerance, krok a zpoždění

Další sada parametrů umožňuje ovlivnit, jak se bude reagovat na akce uživatele. Ten může mít někdy v úmyslu udělat něco jiného, než by se stalo ve výchozím nastavení. Například může chtít jen na něco kliknout, místo aby změnil velikost. Nebo může chtít snadno nastavit nějaké prvky stejně velké, což se mu běžně nemusí podařit.

První dva problémy řeší tolerance a zpoždění. Pokud nedojde ke změně rozměru o určený počet pixelů nebo změna nebude trvat určitou dobu (stanovenou v milisekundách), bude se ignorovat.
```
$('#img').resizable({ distance: 20,  delay: 100 });
```
Pokud nebude rozměr změněn aspoň o 20 pixelů a současně změna nebude trvat nejméně 100 ms, velikost se nezmění. Druhý problém, tedy snazší nastavení nějakého konkrétního rozměru, lze řešit nastavením kroku (mřížky):
```
$('#img').resizable({ grid: [5, 5] });
```
V tomto příkladu bude krok v obou osách 5 pixelů. Hodnota se zadává jako dvourozměrné pole.

Změna rozměrů dalších objektů

Někdy se může hodit, aby se s přímo upravovaným objektem měnila i velikost nějakého dalšího (nebo více objektů). Je to jednoduché:
```
$('#phototmpl').resizable({ alsoResize: 'img.photo' });
```
Kód zajistí, že při změně daného obrázku se změní také velikost všech obrázků třídy „photo". Místo selektoru lze použít také instanci tříd jQuery nebo Element.

Reakce na události

Podobně jako jiné interakce, také změna rozměrů generuje různé události, na které lze reagovat. Tento příklad ukazuje hned tři události najednou:
```
var props = {
start: function(e, ui) {
$('#status-op').text('Změna velikosti');
},
stop: function(e, ui) {
$('#status-op').text('');
},
resize: function(e, ui) {
```

```
$('#status-size').text(ui.size.width + ', ' + ui.size.height);
}
};

$('#img').resizable(props);
```

Při začátku změny rozměru se do operačního stavového pole nastaví, že probíhá změna velikosti, při ukončení se text zase odstraní. Během samotné změny se do velikostního stavového pole zapisují aktuální rozměry.

Widget Resizable toho umí ještě víc, například lze ovlivňovat animaci změny velikosti, upravovat zobrazení grafických prvků uvědomujících uživatele o možnosti změnit velikost objektu, vypnout/zapnout změnu velikosti apod. Podrobnosti najdete v dokumentaci.

Změna pořadí

V příštím dílu se podíváme na interakci, která opět souvisí s přetahováním objektů myší: na změnu pořadí prvků v seznamu nebo mřížce.

jQuery (20) – změna pořadí prvků myší v jQuery UI

Jak se neuklikat

Seznamy nebo mřížky s různými položkami se v aplikacích vyskytují velmi často. Mohou být uspořádávány automaticky nebo ručně. Ve starších aplikacích se ruční změna pořadí často řešila tlačítky. Takové řešení je ale v rozporu s požadavky na moderní uživatelské rozhraní, nelze uživatele nutit k tomu, aby se „uklikal".

Přirozenější je uchopit položku myší a přetáhnout ji tam, kde by nově měla být. V rozšířené podobě to znamená i přesouvání celých skupin položek, souvislých i nesouvislých (z předem vybraných položek). Změnit si pořadí touto cestou je pohodlné a rychlé.

Implementace pomocí jQuery UI

Framework jQuery UI má připravenu podporu takového přesouvání prvků v seznamech a mřížkách. Její použití je jednoduché a současně flexibilní. Základem je widget Sortable, který disponuje celou škálou parametrů umožňujících měnit chování přesouvacích operací.

Widget se nastaví tomu objektu, který bezprostředně (tedy jako své potomky) obsahuje položky, s nimiž chceme pracovat. Toto znamená, že pokud se mají řadit řádky tabulky, je potřeba nastavit widget elementu tbody (případně thead nebo tfoot, pokud se to týká záhlaví, resp. zápatí), nikoli elementu table.

```
$('tbody.sortable').sortable();
```

Příklad ukazuje nejjednodušší použití widgetu. V tomto případě se nastaví právě tělu tabulky a umožňuje řadit její řádky.

Parametrizace přesunu položek

Často potřebujeme nějak ovlivnit to, jak bude ruční řazení probíhat. Například můžeme změnit citlivost (viz díl o přesunu objektů myší):

```
var opts = {
distance: 3,
delay: 100
};

$('tbody.sortable').sortable(opts);
```

Příklad udělá totéž, jako ten první – tedy zapne ruční řazení. V tomto případě ale také zvýší toleranci na 3 pixely a dobu reakce na 100 ms (obě nastavení mají za cíl zabránit nechtěné změně pořadí při intenzivní práci s myší).

```
var opts = {
opacity: 0.5,
grid: [10, 10]
};

$('tbody.sortable').sortable(opts);
```

Tento příklad nastaví 50% průhlednost přesouvaných položek a dovolí pohyb jen v definovaných krocích (tedy zde 10 pixelů).

Řazení v mřížce

Podobně jako položky v seznamu (například řádky tabulky) lze řadit i položky v mřížce (například buňky tabulky). Tady neplatí pravidlo o bezprostředních potomcích, protože potřebujeme řadit položky o dvě úrovně níže.

```
var opts = {
items: '> tr > td'
};

$('tbody.sortable').sortable(opts);
```

Parametr items změní chování widgetu tak, že se nyní pracuje s jinými objekty, než jsou bezprostřední potomci. Hýbat jde tedy s buňkami, a to po celé tabulce. Občas ale potřebujeme, aby byl přesun možný jen v řádcích nebo sloupcích. I to lze snadno zajistit:

```
var opts = {
items: '> tr > td',
axis: 'x'
};

$('tbody.sortable').sortable(opts);
```

Teď už lze buňky přesouvat jen v řádcích. Použije-li se „y", bude se přesouvat ve sloupcích.

Události při přesunu

Všechny výše uvedené příklad předpokládají, že není potřeba na změnu pořadí položek nějak bezprostředně reagovat a například uložení změny pořadí se provede někdy později (třeba po stisku ukládacího tlačítka). Často má ale aplikace reagovat na přeuspořádání položek hned.

```
var opts = {
update: function(e, ui) {
…
}
};
```

```
$('tbody.sortable').sortable(opts);
```

Událost update nastane v okamžiku, kdy změna pořadí skončí a došlo ke změně v DOM (tedy opravdu nastala změna, nešlo jen o „jalové" hýbání s položkou). V obsluze lze například odeslat na server požadavek na uložení nového pořadí položek.

Je k dispozici i řada dalších událostí – například start (začátek změny řazení), stop (konec, bez ohledu na to, jestli nastala změna v DOM) nebo out (položka opustila seznam). Události out, over, remove a receive souvisí se změnou mezi různými seznamy. Ty musejí být vzájemně propojeny pomocí connectWith.

```
var opts = {
remove: function() {
...
},
receive: function() {
...
}
};

$('tbody.sortable tr').sortable(opts);
$('#row1').sortable('option', 'connectWith', '#row2');
$('#row2').sortable('option', 'connectWith', '#row1');
```

Je-li položka definitivně přesunuta do jiného seznamu, nastanou události remove (pro původní seznam) a receive (pro nový). Nedokončené přesuny (trvající) generují události out a over.

Změna chování operace

Běžně se předpokládá, že bude widget Sortable pouze měnit pořadí. Ale jsou případy, kdy tomu má být jinak. Speciální chování se docílí použitím určitého pomocného objektu (*helper*). Již připraven je helper pro klonování (místo přesunu původní položky se pracuje s její kopií).

```
var opts = {
helper: 'clone'
};

$('tbody.sortable').sortable(opts);
```

Lze ovšem použít také vlastní funkci, která rozhodne, se kterou položkou se bude manipulovat. Může tak reagovat třeba na stisk kláves nebo si s daty naložit úplně podle svého.

K dispozici je ještě řada dalších parametrů, ovlivňujících například kontejner pro pohyb položek, typ a chování kurzoru, pohyb *viewportu* (obrazovky) při přesunu za hranu viditelné oblasti nebo určení *handlu* (prvku, za který lze položku uchopit k přesunu). Podrobnosti najdete v dokumentaci k jQuery UI.

Kde je manipulace se skupinami?

Tuto otázku může oprávněně položit každý, kdo si nahoře přečetl, že lze manipulovat s celými skupinami položek. Opravdu s nimi manipulovat lze, k tomu ale nejdřív potřebujeme položky vybrat. Na to se podíváme v příštím dílu.

jQuery (21) – výběr položek v jQuery UI

Jednoduchý výběr položek

Jistě víte, jak vypadají hromadné operace, například přesun více souborů, v desktopových grafických prostředích. Položky k provedení určité operace se vyberou pomocí myši s případnou pomocí z klávesnice. Proč by webové aplikace neměly mít stejný komfort?

Takový komfort zajistit lze, a to použitím widgetu Selectable v jQuery UI. Umí dva způsoby výběru: klikáním na položky (s klávesou Ctrl pro větší nesouvislý výběr) i výběrovým boxem (tažením kurzoru myši od jednoho rohu k protějšímu se vybírají položky).

Selectable bohužel neumí výběr souvislé posloupnosti přidržením klávesy Shift ani výběr pomocí klávesnice. Později se ale podíváme, jak tam tyto funkce přidat.

Práce s widgetem Selectable

Widget Selectable se používá velmi podobně jako widgety z předchozích dílů seriálu. Vytváří se z objektu v DOM, jehož potomky chceme vybírat pro další operace. Například může být následující kód HTML (představuje okno se soubory a složkami, aby to bylo podobné jako na desktopu):

```
<ul id="file-window">
<li class="parent">..</li>
<li class="folder">Složka</li>
<li class="file">Soubor 1</li>
<li class="file">Soubor 2</li>
</ul>
```

Potom se soubory a složky učiní vybíratelné tímto základním kódem (jenž vytvoří widget Selectable):

```
$('#file-window').selectable();
```

Tento kód ale samozřejmě nedělá nic jiného, než že umožní položky vybírat. K další práci je potřeba udělat trochu víc.

Vybrané a vybírané položky

Úplně nejdůležitější je zjišťování, které položky byly vybrány. To jde udělat kdykoliv, a to pomocí třídy ui-selected. Co by tak asi mohl dělat následující kód?

```
$(document).keyup(function(e) {
if (e.which == 127) {
$(this).children('.ui-selected').remove();
}
});
```

Pokud tipujete, že po stisku klávesy Del smaže vybrané položky, tipujete správně. Podobně lze třídě nastavit vhodný grafický styl, aby bylo zjevné, že jde o vybrané položky. Obdobně existuje třída ui-selecting, kterou získají objekty, u nichž zrovna probíhá výběr (dostaly se do výběrového boxu). CSS pak může vypadat třeba takto:

```
.ui-selecting {
background-color: silver;
}

.ui-selected {
```

```
background-color: navy;
}
```
Vybírané položky budou mít světle šedé („stříbrné“) pozadí, vybrané položky pozadí vyplněné námořnickou modří. Existuje ještě třída ui-selectee, viz dále.

Filtrace

Protože se widget vytváří z rodiče těch objektů, z nichž chceme učinit vybíratelné položky, narazíme někdy na problém, že lze vybrat i něco, co by nemělo. Ve výše uvedeném příkladu je to položka „..“, která označuje přechod do nadřazené složky/adresáře a rozhodně nemá jít vybrat. Co s tím? Použít filtr!

```
var opts = {
filter: '.folder, .file'
};
$('#file-window').selectable(opts);
```
Widget se vytvoří s tím, že půjdou vybírat jen objekty s uvedenými třídami („..“ žádnou z těchto tříd nemá). Po vytvoření widgetu bude vybíratelným objektům nastavena již zmíněná třída ui-selectee; to se může hodit například pro grafické zvýraznění.

Úprava chování

Výběrový box standardně vybírá vše, čeho se dotkne. Pokud to není žádoucí a chcete vybírat jen co, co je celé uvnitř boxu, není problém to zajistit (výchozí hodnota je „touch“, tedy dotyk):

```
var opts = {
tolerance: 'fit',
delay: 100,
distance: 20
};
$('#file-window').selectable(opts);
```
Příklad ukazuje i další dva parametry ovlivňující, jak bude výběr fungovat. Objevily se už v minulých dílech a zde je lze použít k ochraně proti nechtěnému výběru.

Události

Widget Selectable generuje události při vytvoření, při začátku a konci procesu vybírání, při a po vybrání, resp. odvybrání. K čemu je obsluha událostí dobrá? Třeba u souborů můžeme sčítat a někde zobrazovat celkovou velikost vybraných souborů, přičemž přepočet se provede buď po změně výběru nebo už během vybírání (podle toho, co je vhodnější).

```
var opts = {
stop: function() {
...
}
};
$('#file-window').selectable(opts);
```
Do obslužné rutiny události stop (která nastává, když proces výběru skončí) by se napsal kód, který posčítá velikosti vybraných souborů (reprezentovaných objekty třídy ui-selected). Pokud bychom potřebovali přepočet po každém jednotlivém souboru, obsloužily by se události selected a unselected, v případě potřeby reagovat už na změny výběrového boxu pak události selecting a unselecting.

Výběr a přetažení

Příště dáme dohromady výběr objektů a jejich přetažení jinam. Paralela s prací se soubory v oknech na desktopu je zde více než zřejmá.

jQuery (22) – výběr a přetahování v jQuery UI

Výběr a přetahování objektů

Snad každý zná přetahování vybraných objektů na jiné místo, například souborů mezi různými adresáři. Něco takového můžeme celkem oprávněně chtít také ve webové aplikaci. Problém ale je, že jQuery UI k tomu neposkytuje přímé prostředky, tak jako je poskytuje pro přetahování jednotlivých objektů. Je potřeba část práce udělat ručně.

Existují pluginy, které to usnadňují, ale k těm se dostaneme později. Teď je důležité, o co je vlastně potřeba se postarat. Jsou to v zásadě tři věci:

- vykreslování přesouvaných objektů na správných místech,
- ponechání pozice objektů po skončení přetahování,
- zpracování upuštění objektů.

Implementace

Celé to bude fungovat vlastně tak, že základem bude obyčejné přetahování jednoho objektu, které se bude „promítat“ do ostatních objektů. S výhodou k tomu lze využít událostí, které v procesu nastávají. Pokud bude přetahován nevybraný objekt, všechno se bude chovat „jako obvykle“. Při přetahování některého z vybraných objektů však bude chování jiné.

Asi bude nejlepší začít příkladem – nejdřív jen pro obyčejné přetahování, bez reakce na upuštění někam (předpokládejme, že přetahovatelnými objekty budou všechny elementy div uvnitř kontejneru s identifikátorem cont):

```
function updateMultidragOffsets(jq) {
var oldpos = jq.data('oldpos');
if (typeof oldpos != 'undefined') {
var pos = jq.offset();
var diff = { left: pos.left - oldpos.left, top: pos.top - oldpos.top };
var sel = $('.multidrag');
for (var i = 0; i < sel.length; i++) {
var o = $(sel.get(i));
var op = o.data('oldpos');
o.offset({ left: op.left + diff.left, top: op.top + diff.top });
}
}
}

$(document).ready(function() {
var opts = {
```

```
start: function() {
if ($(this).hasClass('ui-selected')) {
$('.ui-selected').not(this).addClass('multidrag');
$('.ui-selected').each(function() {
$(this).data('oldpos', $(this).offset());
});
}
},
drag: function() {
updateMultidragOffsets($(this));
}
stop: function() {
updateMultidragOffsets($(this));
$('.multidrag').removeClass('multidrag').removeData('oldpos');
$(this).removeData('oldpos');
},
};

$('#cont div').draggable(opts);
$('#cont').selectable();

});
```

Vezměme to od konce. Metody selectable() a draggable() připravují widgety pro výběr, resp. přetahování. Pro přetahování je potřeba připravit reakce na již zmíněné události. Na událost start (začátek tažení) se reaguje tím, že se vybraným objektům (kromě aktuálního) nastaví třída multidrag ke snazší identifikaci a všem vybraným objektům se uloží původní pozice pro výpočet pozice nové.

Pokud se přesouvá nevybraný objekt, nestane se nic, jak je z kódu zřejmé.

Při události drag se volá funkce aktualizující pozice objektů (viz dále), při události stop se aktualizuje pozice (což je nutné, protože při probíhajícím přetahování události obecně nenastávají tak často, aby se pozice aktualizovala vždy přesně), odebere se třída multidrag a odstraní uložené pozice.

Funkce updateMultidragOffsets() vypočítává rozdíl v pozici přímo přetahovaného objektu a podle něj vypočítá pozice všech ostatních objektů (přetahovaných prostřednictvím výběru). Pokud se přetahuje jen nevybraný objekt, funkce nedělá nic.

Reakce na upuštění objektů

Ještě zbývá vyřešit reakci na upuštění přetahovaných vybraných objektů někam. Klíčem je samozřejmě reakce na upuštění přímo přetahovaného objektu, ty ostatní vybrané se s ním „svezou“. Tady je krátký příklad, jak to může vypadat:

```
opts = {
drop: function(e, ui) {
ui.draggable.removeClass('ui-selected');
$('.multidrag').removeClass('ui-selected');
$(this).append(ui.draggable.detach());
$(this).append($('.multidrag').detach());
}
};

$('#target').droppable(opts);
```

Objekt s identifikátorem target je cílem, kam se budou přetahované objekty pouštět. V tomto případě bude reakcí na upuštění to, že se objekty přesunou do cíle. Je potřeba je „odvybrat“ (protože už jsou jinde a pracujeme s nimi tedy jinak) a následně provést jejich přesun v DOM.

Zde se počítá s tím, že cílový objekt ty přetahované vždy přijme. Pokud by tomu tak nebylo (byl by použit filtr), mohli bychom požadovat třeba návrat objektů na původní místo. Pak by se přepsala obsluha události stoptak, že se by v případě neakceptace nastavily zpět původní pozice, případně by to šlo také animovat.

Konkrétní widgety

Tímto dílem seriálu končí pasáž zabývající se interakcí s uživatelem a příště už se podíváme na konkrétní widgety, které jQuery UI poskytuje. Jako první přijdou na řadu tlačítka různého druhu.

jQuery (23) – tlačítka jQuery UI

Tlačítka v HTML a jejich limity

Každý zná tlačítka ve formulářích na webových stránkách. Tato tlačítka, ať už k odesílání nebo dalším akcím, jsou už mnoho let součástí jazyka HTML a bez nich by formuláře prakticky nešly řešit. Původně bylo k dispozici tlačítko jako forma elementu input, později přibyl ještě univerzálnější element button.

Tlačítko v prohlížečích Firefox a Chromium (distribuce Linux Mint)

Oba zmíněné elementy lze do určité míry stylovat, používat místo základní podoby vlastní obrázky, ovlivňovat jejich chování (například je aktivovat/deaktivovat) a zachytávat z nich události. Jenže výchozí vzhled těchto tlačítek se liší podle prohlížeče, jeho verze, operačního systému atd. Proto nezbývá, než si tlačítka upravit. Dělat to od základu by ale byla zbytečná práce navíc, protože v jQuery UI je připraven widget Button, který tyto věci řeší.

Widget Button

Button je widget jQuery UI, který v první řadě aplikuje na tlačítkový element jednotný styl použitého grafického tématu. Tlačítka mohou být klasická, tedy vracející se po stisku do výchozího stavu, ale i zaškrtávátka nebo přepínače.

Tlačítko realizované pomocí widgetu Button může nést text, jednu či dvě ikonu nebo kombinaci obojího. Widget tam může figurovat na stránce například jako klasické formulářové tlačítko, tlačítko na panelu nástrojů nebo součást pásu karet (*ribbonu*).

Tlačítka realizovaná elementem input (konkrétně button, submit, reset) mohou být pouze textová. Potřebujete-li na nich mít ikony, můžete to obejít tím, že použijete element button a funkce pro odeslání, resp. vymazání formuláře zajistíte JavaScriptem.

Vytvoření widgetu a práce s ním

Pokud potřebujeme vytvořit úplně obyčejné tlačítko, může HTML kód vypadat přibližně takto:

```
<button id="tlacitko">Tlačítko</button>
```

Instanci widgetu Button z něj potom vyrobíme jednoduchým kusem JavaScriptu:

```
$('#tlacitko').button();
```

Toto tlačítko bude samozřejmě textové (text bude „Tlačítko“). Na jeho stisk můžeme reagovat obvyklým způsobem:

```
$('#tlacitko').click(function() {
    ...
});
```

Vzhled tlačítka bude odpovídat tématu, lze si ho ale samozřejmě upravit podle svého (viz dále).

Tlačítko s widgetem Button (vypadá ve všech prohlížečích stejně)

Použití ikon

Chceme-li mít na tlačítku také ikonu, přidáme ji do parametrů při vytváření widgetu:

```
var params = {
    icons: {
        primary: 'ui-icon-copy'
    }
};
$('#tlacitko').button(params);
```

S ikonami se pracuje tak, že se do parametrů nedává URL obrázku, nýbrž CSS třída odpovídající příslušné ikoně. K dispozici je mnoho ikon pro běžné účely. Například v příkladu se použije ikona operace kopírování. Pomocí CSS vlastnosti background-image a dalších si lze připravit vlastní třídy pro ikony.

Tlačítko s ikonou

Uvedený příklad využívá pouze jednu ikonu – tu primární, která se standardně zobrazí na levé straně tlačítka. Můžeme použít i druhou ikonu (sekundární), zobrazenou na pravé straně:

```
var params = {
    icons: {
        primary: 'ui-icon-copy',
        secondary: 'ui-icon-star'
    }
};
```

Zaškrtávátka

Pro tyto ovládací prvky je potřeba využít element input. Aby vše fungovalo správně, musí být popisek (element label) propojen s prvkem prostřednictvím identifikátoru a nesmí být jeho kontejnerem. HTML kód by tedy vypadal například takto:

```
<input type="checkbox" id="persistent-login" name="persistent"> <label for="persistent-login">trvalé přihlášení</label>
```

Toto obyčejné zaškrtávátko se snadno převede na widget Button:

```
$('#persistent-login').button();
```

zaskrtavatko.png, Zaškrtnuté zaškrtávátko (má vzhled běžného tlačítka; lze to změnit přes CSS)

Pozor na to, když se stav zaškrtávátka změní z programu (tedy neudělá to uživatel). Pak je potřeba říct widgetu, aby se překreslil, protože to neudělá automaticky:

```
$('#persistent-login').button('refresh');
```

Přepínače

U přepínače je to velmi podobné jako u zaškrtávátka. Často ale potřebujeme jednotlivé volby vizuálně seskupit – to už si ale nevystačíme s widgetem Button a musíme si vzít na pomoc další widget: Buttonset. Ten je určen pro sadu tlačítek (libovolných, nejen přepínačových) a zajistí i vytvoření widgetů pro jednotlivá tlačítka, takže ušetří práci.

```
<fieldset id="color-sel">
    <legend>Barva</legend>
    <input type="radio" id="color-red"  name="color" checked="checked"> <label for="color-red">červená</label>
    <input type="radio" id="color-green"  name="color"> <label for="color-green">zelená</label>
</fieldset>
```

Element fieldset tu zafunguje jednak jako kontejner pro zpracování widgetem, současně má ale význam i pro přístupnost (umožňuje definovat legendu). Na toto nyní aplikujeme widget Buttonset:

```
$('#color-sel').buttonset({ items: 'input[type=radio]' });
```

Parametr items říká, které elementy se mají začlenit do skupiny a vytvořit pro ně widgety. V tomto případě to budou elementy input typu radio.

Přepínač ve výchozím stavu

Podobně jako to bylo u interakčních widgetů, také tady lze parametry měnit i za provozu pomocí metody option. Další informace najdete v dokumentaci.

Další widgety

V příštím dílu se podíváme na ukazatele průběhu a posuvníky. Ty už ve standardním HTML k dispozici nejsou, ale přitom je často potřebujeme používat.

jQuery (24) – ukazatele průběhu a posuvníky v jQuery UI

Ukazatel průběhu čili progressbar

Uživatelé bývají netrpěliví a jakmile nějaká činnost trvá déle, mohou mít pochybnosti, zda aplikace stále ještě dělá, co dělat má (a například nezůstala někde viset). Proto se používají ukazatele průběhu. Mohou ukazovat činnost o předem určené délce

(například zpracování určitého počtu prvků), ale i činnost bez známé délky (kupříkladu přenos dat ze serveru, kdy se neví, kolik jich bude).

Jakékoli činnosti, u nichž se předpokládá doba trvání nad 1 sekundu, je dobré indikovat ukazatelem průběhu (případně jinak, ale to není předmětem článku). K dobré praxi patří, aby měl uživatel možnost činnost přerušit a současně aby nemohl udělat jinou akci, která by do probíhající činnosti nevhodně zasáhla.

Jak správná funkce indikace průběhu, tak možnost akci přerušit, vyžaduje využívat asynchronnost komunikace se serverem. Pokud se komunikuje synchronně, webový prohlížeč s vykonáváním dalšího javascriptového kódu čeká, což může trvat i velmi dlouho.

Indikace činnosti o známé délce

K implementaci posuvníku slouží v jQuery UI widget Progressbar. Jako podkladový element může sloužit například div:

```
<div id="progressbar"></div>
```

Widget se pak vytvoří velmi jednoduše:

```
$('#progressbar').progressbar();
```

Takto vznikne ukazatel, jehož maximální hodnota je 100 (minimální je vždy nula), takže pokud máme hodnotu převedenou na procenta, není potřeba maximum nastavovat. Lze ale zobrazovat i přímo hodnotu, pokud se maximum nastaví:

```
$('#progressbar').progressbar({ max: 360 });
```

Pak už zbývá jen zajistit aktualizaci hodnoty, jakmile dojde k vykonání příslušné části činnosti:

```
var pb = $('#progressbar');
for (var i = 1; i <= 360; i++) {
...
pb.progressbar('option', 'value', i);
}
```

Výsledek může vypadat například takto:

Ukazatel průběhu (ukazuje 30 %)

Vzhled ukazatele průběhu lze snadno měnit – ukazatel jako takový má CSS třídu ui-progressbar, vyplněná část ui-progressbar-value. První z nich lze použít například k nastavení velikosti, druhou třeba pro změnu barvy v závislosti na okolnostech.

Indikace činnosti o neznámé délce

V případě předem neznámé délky činnosti postupujeme velmi podobně jako výše, jako hodnota se však (již při vytvoření, případně později) nastaví false:

```
$('#progressbar').progressbar({ value: false });
```

Následně se už tato hodnota obvykle neaktualizuje. Pokud se však délka činnosti stane známou, lze začít hodnotu nastavovat a indikátor se chová jako pro předem známou délku. Možný je samozřejmě i opačný postup (třeba když jednotlivá operace trvá příliš dlouho).

Ukazatel indikující průběh činnosti o neznámé délce

U ukazatele pro neznámou délku činnosti lze využít CSS třídu ui-progressbar-overlay, typicky se tam nastaví animovaný obrázek (pokud nechcete použít výchozí).

Posuvník

Posuvník (*slider*) je ovládací prvek pro snadné nastavování číselných hodnot. Typicky se jím ovládá například jas, hlasitost, barevná složka, zvětšení apod. Základem posuvníku je *jezdec* (též *běžec*; anglicky *handle*), tedy prvek, za který lze zatáhnout kurzorem myši a posunout ho na požadované místo. Na koncích posuvníku bývají tlačítka s šipkami – stisk tlačítka posune jezdec o definovanou hodnotu, obvykle malou. Aktivní bývá také prostor po obou stranách jezdce (směrem k tlačítkům), kliknutí do prostoru posune jezdec daným směrem, obvykle o větší hodnotu než tlačítko.

Existují i varianty posuvníků s více jezdci. Posuvník se dvěma jezdci lze využít k nastavení intervalu, se třemi se používá u obrázků pro nastavení bílého a černého bodu a gamma korekce.

V jQuery UI je posuvník realizován pomocí widgetu Slider. Může být orientován vodorovně či svisle, mít libovolný počet jezdců a pohyb těchto jezdců může být animován. Jako základ opět poslouží například div:

```
<div id="slider"></div>
```

Z něj se vytvoří posuvník:

```
$('#slider').slider();
```

Vodorovný posuvník s jedním jezdcem

Pokud potřebujeme nastavit minimum, maximum a výchozí hodnotu, udělá se to obvyklým způsobem:

```
var opts = {
min: 100,
max: 200,
value: 150
};
$('#slider').slider(opts);
```

Takto vytvořený posuvník je orientován vodorovně. Snadno lze ale vytvořit i svislý:

```
$('#slider').slider({ orientation: 'vertical' });
```

Více jezdců, intervalový posuvník

Posuvník v jQuery UI podporuje v podstatě libovolný počet jezdců. Takto například dostaneme tři jezdce (jeden bude vlevo, druhý uprostřed a třetí vpravo):

```
$('#slider').slider({ values: [ 0, 50, 100] });
```

Posuvník se třemi jezdci (aktivní je levý jezdec – bude reagovat na klávesnici)

Pro intervalové posuvníky je k dispozici zvláštní podpora – interval lze totiž samostatně nastylovat (například obarvit). Slouží k tomu parametr range:

```
var opts = {
values: [0, 100],
range: true
};
$('#slider').slider(opts);
```

Posuvník s intervalem

Interval lze ale využít i při jediném jezdci – bude se počítat od jezdce k některé mezi posuvníku (v následujícím příkladu k maximu):

```
$('#slider').slider({ range: 'max' });
```

Získání hodnoty posuvníku, třídy pro CSS

Můžeme využít dvě cesty. Jednou je, se prostě posuvníku na aktuální hodnotu zeptat:

```
var v = $('#slider').slider('value');
```

Často ale chceme reagovat přímo na změnu hodnoty. Pak je výhodnější využít například reakci na událost change (příklad zobrazí hodnotu ve vyskakovacím okně):

```
var opts = {
change: function(e, ui) {
alert('Nová hodnota:' + ui.value);
}
};
$('#slider').slider(opts);
```

K dispozici je ještě událost slide, která toho umí víc – poradí si s více jezdci a pokud obslužná rutina vrátí false, posun do nové pozice se zruší.

```
var opts = {
slide: function(e, ui) {
if (…) {
return false;
}
…
return true;
}
};
$('#slider').slider(opts);
```

Ohledně CSS tříd je to podobné jako u ukazatele průběhu. Posuvník má třídu ui-slider (a také ui-slider-horizontal pro vodorovný, resp. ui-slider-vertical pro svislý posuvník), každý z jezdců ui-slider-handlea interval ui-slider-range.

Další widgety

Příště se podíváme na dva widgety spjaté s textovými poli – automatické doplňování a kalendář.

jQuery (25) – automatické doplňování v jQuery UI

Textové pole je základ

Přestože s HTML5 přišly i další možnosti pro vstup dat ve formulářích, implementace v prohlížečích není ani dnes stoprocentní. Navíc ne vždy „systémové" řešení vyhovuje a především ne všechny potřebné druhy vstupních prvků jsou součástí specifikace. Proto stále zůstává základem textové pole, na které se navazuje další funkcionalita.

Výhodou tohoto řešení je také to, že i s vypnutým JavaScriptem formulář funguje, byť ne tak komfortně.

Automatické doplňování

S automatickým doplňováním se dnes setkáváme zcela běžně – počínaje adresním řádkem webového prohlížeče, přes různé vyhledávače, až třeba po pole pro vložení PSČ. Data mohou být k dispozici přímo ve formuláři, nebo je lze získávat až během psaní.

Typicky to funguje tak, že při dosažení určité délky napsaného textu a po vypršení prodlevy od napsání posledního znaku se zobrazí nabídka, z níž si lze vybrat, co se do pole vloží. Pokud uživatel nevloží nic a píše dál, nabídka se postupně aktualizuje.

Widget Autocomplete

V jQuery UI je pro automatické doplňování k dispozici widget Autocomplete. Nabízí vše potřebné k realizaci této funkcionality, včetně potřebné interakce s klávesnicí a myší. V kódu HTML potřebujeme mít textové pole:

```
<input type="text" name="city" id="city">
```

Uvedené pole bude sloužit k zadávání města. Pro usnadnění uživateli nabídneme největší česká města, která se budou často používat. Javascriptový kód může vypadat například takto:

```
var props = {
source: ['Praha', 'Brno', 'Ostrava', 'Plzeň', 'Hradec Králové', 'Ústí nad Labem',
'Liberec']
};
$('#city').autocomplete(props);
```

Zobrazení nabídky k automatickému doplnění do textového pole

Předává se pouze seznam k automatickému doplnění, ostatní parametry zůstávají výchozí. Proto se nabídka zobrazí 300 ms od chvíle, kdy uživatel přestane psát a k zobrazení postačí jediný napsaný znak. V daném případě to bude zřejmě plně vyhovující. Někdy to ale můžeme chtít změnit:

```
var props = {
source: …,
minLength: 3,
delay: 700
```

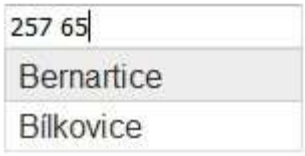
```
};
$('#city').autocomplete(props);
```

Větší minimální délka se hodí hlavně v případech, že je nabízených možností hodně a je potřeba je rozumně omezit, aby jejich nabídnutí mělo vůbec smysl. Delší prodleva se zase hodí v případech, kdy lze očekávat, že bude uživatel psát pomalu, nebo pokud se budou data načítat ze serveru (viz dále) a je žádoucí počet dotazů omezit.

Štítky pro hodnoty

Pokud se zadává například PSČ, může být vhodné u každé nabízené možnosti uvést další informaci (v případě PSČ je to dodací pošta). Ta se do pole po výběru nevloží a slouží jen pro orientaci uživatele – je to tzv. *štítek*. Místo obyčejného pole hodnot použijeme pole objektů:

```
var props = {
source: [{ label: 'Bernartice', value: '257 65'}, { label: 'Bílkovice',
value: '257 26'}, …]
};
$('#city').autocomplete(props);
```



Využití štítků

V tomto případě ovšem nabízení hodnot funguje trochu jinak – do pole je potřeba psát texty, které lze vyhledat ve štítcích. Pokud je potřeba hledat i v samotných hodnotách, musí být ve štítcích obsaženy (například místo „Bernartice" by se použilo „257 65 Bernartice" apod.).

Načítání dat ze serveru

V mnoha případech je vhodnější (ne-li vůbec technicky jediné proveditelné) načítat data v reálném čase ze serveru, místo aby byla načtena předem s formulářem. Představme si třeba hledání v názvech zboží v internetovém obchodě, který může mít desetitisíce položek i více.

Načítání lze realizovat dvěma způsoby – buď se vše nechá na implementaci ve widgetu (což je doporučená cesta, pokud neexistuje nějaká překážka v jejím použití), nebo si to lze realizovat po svém. Využití implementace ve widgetu Autocomplete je velmi snadné:

```
var props = {
source: '/search.php'
};
$('#city').autocomplete(props);
```

Výsledkem je, že se v okamžiku potřeby nabídnout data k vyplnění odešle na server požadavek metodou GET a k URL se přidá parametr „term", v němž bude text napsaný uživatelem. Pokud uživatel napíše například „abc", na server odejde požadavek na /search.php?term=abc. Widget od serveru očekává data ve formátu JSON, která budou vypadat jako ve výše uvedených příkladech.

Vrácená data mohou obsahovat jak samotné hodnoty, tak objekty se štítky a hodnotami. Požadavek může být odeslán jak na stejný server, tak i na jiný (ten ale musí podporovat JSONP).

Potřebujete-li realizovat požadavek na server nějak jinak, případně data nějak zpracovávat i místně, můžete místo URL použít funkci, která vykoná vše potřebné a po asynchronním zpracování vrácených dat zavolá callback response(). Podrobnosti najdete v dokumentaci.

Události a stylování

Při práci s Autocomplete lze reagovat na řadu událostí, například search, response nebo select. Bývá to potřeba málokdy, ale ta možnost tu je. Opět odkazuji na dokumentaci.

Co se týká vlastního stylování (standardně se používá výchozí podoba pro dané téma jQuery UI), jsou k dispozici CSS třídy ui-autocomplete (nabídka textů k doplnění) a ui-autocomplete-input (textové pole). Můžete si tedy upravit, co je potřeba.

Kalendář příště

Původně měl být součástí tohoto dílu i kalendář, nakonec jsem se ale rozhodl ho přesunout až do dílu příštího, protože si zaslouží dostatek prostoru. Pro tentokrát je to tedy vše a pro příště se můžete těšit na ten slíbený kalendář.

jQuery (26) – kalendář v jQuery UI

Datum ve formuláři

Velmi často se setkáváme s potřebou vložit do webového formuláře datum. Pokud se použije obyčejné textové pole, je takové vkládání pracné – jak při vpisování do prázdného pole, tak i následné úpravy. Pokud navíc pole neobsahuje validaci, můžete tam vložit neplatné datum a všimnete si toho až po odeslání celého formuláře (což je zvlášť „zábavné", pokud se s formulářem odesílají na server nějaká větší data).

Mnohem pohodlnější je mít možnost si datum naklikat. Klidně může současně zůstat možnost zadat datum ručně (někdy je to rychlejší, můžete ho třeba vložit přes schránku), ale současně přibude možnost procházet kalendář a v něm si vybírat.

Ve frameworku jQuery UI pro to existuje widget Datepicker.

Widget zobrazuje klasický měsíční kalendář (první den v týdnu je volitelný), v němž lze přecházet mezi měsíci a roky, kliknutím na den se toto datum vybere k dalšímu použití. Vybrané a dnešní datum se obvykle zobrazuje odlišně od ostatních dnů. Dále lze třeba omezit pohyb uživatele v kalendáři (například ho nepustit do minulosti) a různě kalendář přizpůsobovat.

Textové pole s kalendářem

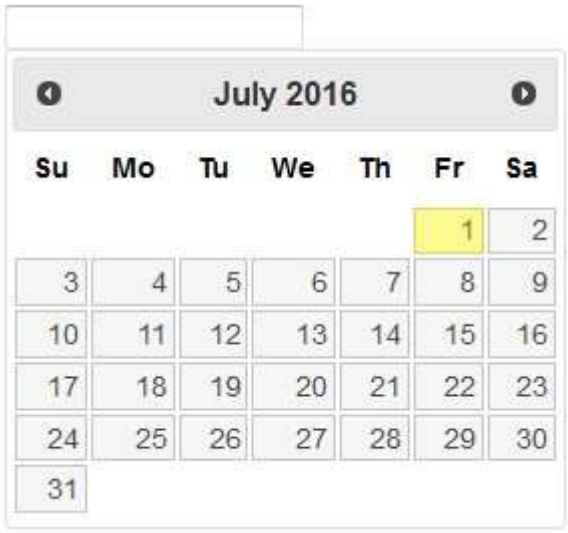
Máme-li formulář s textovým polem pro zadání data, lze k němu snadno přidat zobrazení widgetu kalendáře. Klikne-li uživatel do pole, kalendář se zobrazí a lze s ním pracovat. Mějme například takovéto pole v HTML:

```
<input type="text" name="date" id="date">
```

K němu přidáme kalendář velmi snadno:

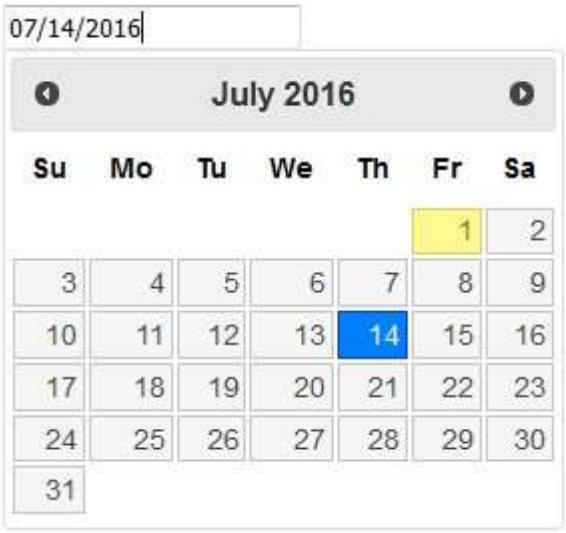
```
$('#date').datepicker();
```

Výsledek po kliknutí do pole potom vypadá zhruba takto (před kliknutím to bylo obyčejné textové pole):



Zobrazení kalendáře u textového pole

Po kliknutí na den v kalendáři se datum vloží do pole a widget se schová. Po opětovném kliknutí do pole se kalendář opět zobrazí a zvýrazní se (v tomto případě modře; žlutě se zobrazuje dnešní datum) v něm datum podle obsahu pole (ať už se tam dostalo jakkoli).



Zvýraznění vybraného dne v kalendáři

Lokalizace

Jak si můžete všimnout, kalendář je v angličtině a také jeho další vlastnosti odpovídají angloamerickému prostředí. My bychom však chtěli českou lokalizaci, tedy české názvy, pondělí jako první den v týdnu a „tečkový" formát data. Toho lze docílit v zásadě dvěma způsoby. Buď se jazyk zvolí přímo pro danou instanci widgetu:

```
$('#date').datepicker($.datepicker.regional['cs']);
```

Druhou možností je nastavit si českou lokalizaci jako výchozí a pak už widgety vytvářet bez specifikace jazyka:

```
$.datepicker.setDefaults( $.datepicker.regional['cs'] );

$('#date').datepicker();
```

Ani ve jednom případě to však nestačí, protože česká lokalizace standardně není definována. Je potřeba si stáhnout příslušný soubor (například z GitHubu) a vložit ho do stránky ke skriptům pro jQuery, jQuery UI atd.:

```
<script type="text/javascript" src="datepicker-cs.js"></script>
```

Lokalizovaný kalendář už vypadá z našeho pohledu poněkud lépe:

Kalendář lokalizovaný do českého prostředí

Obdobně by se postupovalo i pro jiné jazyky. Samozřejmě není problém vložit si více definic a pak z nich vybírat podle jazyka, který si uživatel zvolí.

V kalendáři se lze pohybovat nejen pomocí myši, ale také pomocí klávesnice. Popis klávesových povelů najdete v dokumentaci.

Kalendář bez textového pole

Tam, kde se nepředpokládá vkládání přes textové pole, lze kalendář zobrazit přímo. Stačí místo textového pole použít generický kontejner:

```
<div id="date"></div>
```

Javascriptový kód je stejný jako pro kalendář navázaný na pole. Výsledkem je kalendář zobrazený přímo ve stránce, který po výběru data nezmizí.

inline.png, Widget Datepicker vložený přímo do stránky

Ale jak z tohoto widgetu získáme vybrané datum? Jsou dvě možnosti. Jednak se na něj můžeme zeptat:

```
var d = $('#date').datepicker('getDate');
```

Metoda getDate vrátí datum ve formě instance třídy Date, s níž pak můžeme dále pracovat. Druhou možností je reagovat na událost výběru data (onSelect):

```
var props = {
onSelect: function(d, o) {
…
}
}
$('#date').datepicker(props);
```

Obslužná funkce dostane jako první parametr textový řetězec data, jako druhý potom instanci widgetu. Pokud použití textového řetězce není vhodné (zejména proto, že je lokalizovaný), lze si samozřejmě pomoci metodou getDate.

Opačným problémem je vložení vybraného data do kalendáře (čehož tu nemůžeme dosáhnout textovým polem). K tomu poslouží buď parametry při vytváření widgetu nebo metoda setDate kdykoli později. Jako parametr lze předat jak objekt Date, tak i textový řetězec v lokalizovaném formátu a dokonce i relativní posun vůči aktuálnímu datu:

```
$('#date').datepicker('setDate', new Date());
$('#date').datepicker('setDate', '10.12.2015');
$('#date').datepicker('setDate', '+7d');
```

Další možnosti kalendáře

Kalendář toho umí podstatně víc. Chcete například omezit, v jakém časovém úseku se uživatel může pohybovat? Žádný problém!

```
var props = {
minDate: '-1y',
maxDate: '+1y'
};
$('#date').datepicker(props);
```

Tento kód uživatele pustí v kalendáři rok zpět a rok dopředu. Přestože kalendář ve widgetu Datepicker je měsíční, widget umí zobrazit i více měsíců najednou:

```
var props = {
numberOfMonths: 3
};
$('#date').datepicker(props);
```

Takto se zobrazí tři měsíční kalendáře vedle sebe:

Zobrazení více měsíců najednou

S tím souvisí i krok pro tlačítka přechodu mezi měsíci. Výchozí hodnota je logicky jeden měsíc, ale pokud se zobrazuje více měsíců najednou, může mít logiku i delší krok (například o počtu zobrazených měsíců nebo polovina tohoto počtu).

```
var props = {
numberOfMonths: 3,
```

```
stepMonths: 3
};
$('#date').datepicker(props);
```

Dále lze například měnit texty na tlačítkách, zobrazovat i další měsíce a moci z nich vybírat, reagovat na události (kromě výběru data také na zavření kalendáře a na změnu měsíce či roku) atd. Podrobnosti najdete v dokumentaci.

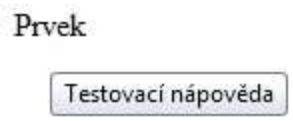
Kontextová nápověda, číselníky

Příště se podíváme na dva druhy widgetů – kontextovou nápovědu (*tooltipy*) a číselníky (*spinnery*).

jQuery (27) – kontextová nápověda a číselníky v jQuery UI

Kontextová nápověda (tooltip)

Každý to jistě zná – najede myší na nějaký objekt a zobrazí se k němu informace, typicky v nějaké „bublině". Umí to i obyčejný HTML kód (atribut title), ale to je víc „šidítko" než cokoli jiného. Pokud chceme lepší zobrazení, je třeba se poohlédnout jinde. Například do jQuery UI, kde k tomu slouží widget Tooltip.

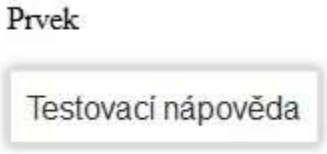


Ukázkat nativní kontextové nápovědy (přímo v HTML)

Funguje kompatibilně s nativními tooltipy, takže v HTML kódu webu není potřeba nic měnit. Lze ho nastylovat podle potřeby (podle grafických témat, podle druhu nápovědy, závažnosti zpráv…), animovat, pohybovat bublinami s myší apod.

```
$('p').tooltip();
```

Toto je nejjednodušší použití widgetu, které jen převede nativní nápovědu na použití widgetu (použijí se zmíněné atributy title; existuje ale možnost text předefinovat). Zobrazení je hodně podobné, ale méně závisí na prohlížeči a operačním systému.



Využití widgetu Tooltip pro zobrazení nápovědy

Animace

Když si vyzkoušíte uvedený kód, všimnete si, že je zobrazování a mizení animované (plynulé). Tuto animaci lze ovlivnit jak co do parametrů (rychlost, zpoždění atd.), ale také co do typu animace (efektu). Pak to může vypadat i úplně jinak:

```
var props = {
show: { effect: "bounce", duration: 500 },
hide: { effect: "slide", duration: 2000 }
};

$('p').tooltip(props);
```

Při zobrazování se nápověda krátce zatřese, při mizení odjede vlevo mimo stránku. Animaci lze také zcela vypnout, pokud se show, resp. hide nastaví na hodnotu false.

Nastylování

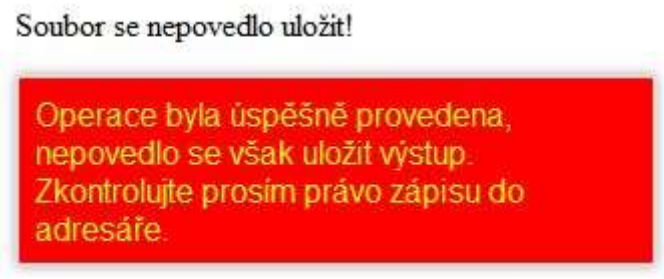
Ke globálnímu nastylování lze použít třídy ui-tooltip (kontejner bubliny) a ui-tooltip-content (obsah bubliny). Je-li ale například potřeba použít jiné barvy pro určitý typ elementů, musí se na to jít trochu jinak. V JavaScriptu bude:

```
$('p.warning').tooltip({ tooltipClass: 'tooltip-warning' });
```

A do CSS se potom umístí něco jako toto:

```
.tooltip-warning {
background-color: red;
color: yellow;
}
```

Zobrazí se tak červená bublina se žlutým textem.



Speciálně nastylovaná kontextová nápověda

Další možnosti

Widget Tooltip toho umí ještě o dost víc. Mějme například tento kód:

```
var props = {
track: true,
content: getTooltipContent
};

$('p').tooltip(props);
```

Parametr track způsobí, že se bude bublina pohybovat s kurzorem myši. Parametr content umožňuje buď přímo předat nějaký obsah (nejen textový, ale obecně HTML), nebo využít funkci, jako je tomu v tomto příkladu (getTooltipContent()). Další možnosti práce s widgetem Tooltip najdete v dokumentaci.

Číselníky

Číselník čili *spinner* je ovládací prvek umožňující snadno nastavovat čísla. Nejčastěji se používá v případech, kdy se číslo mění v malém rozsahu a je proto vhodné umožnit rychlé nastavování myší. Dříve nebyla vůbec možnost pracovat s takovým prvkem v holém HTML, nativní prvek (element input s typem number) v HTML5 není ještě úplně široce podporován (je jen v relativně nových verzích prohlížečů).

Nativní číselník v HTML5

Proto je mnohdy výhodnější použít přímo widget Spinner z jQuery UI, který implementuje široce použitelný číselník s dalšími funkcemi. Mějme HTML kód:

```
<input type="text" id="spinner" value="0">
```

Je to tedy obyčejné textové vstupní pole (číselníkové pole HTML5 **nepoužívejte**, jinak tam budou ovládací tlačítka dvakrát). K němu doplníme funkci číselníku velmi snadno:

```
$('#spinner').spinner();
```

Výsledkem je následující ovládací prvek:

Číselník vytvořený widgetem Spinner

Parametrizace

Jak je výše uvedený příklad jednoduchý, tak je „hloupý". Neumí například takovou základní věc, jako je horní a dolní mez číselných hodnot, v praxi obvykle nezbytnost. Můžeme tam ale limity přidat:

```
var props = {
    min: 0,
    max: 10
    };
$('#spinner').spinner(props);
```

Toto umožní „naklikat" jen čísla 0 – 10, mimo tento rozsah uživatele nepustí.

Pozor: Stále lze do pole zadat číslo mimo rozsah nebo i úplně jiný řetězec! Lze tomu zabránit buď zákazem ruční změny (pole jen k zápisu), nebo lze reagovat na událost změny a zvalidovat obsah, případně tam vrátit poslední platnou hodnotu.

Následující příklad ukazuje další možnosti parametrizace spinneru:

```
var props = {
    min: 0,
    max: 50,
    step: 2,
    page: 5,
    incremental: false
    };

$('#spinner').spinner(props);
```

Meze jsou jasné. Pak následuje nastavení kroku, o kolik se hodnota změní po kliknutí na tlačítko šipky nebo po stisku klávesy šipky. Parametr page určuje, o kolik změnit hodnotu klávesami **Page Up** a **Page Down**. A konečně parametr incremental říká, zda se má při delším držení tlačítka zrychlovat nebo ne (ve výchozím stavu se zrychluje, zde se to vypíná).

Nastylování a další možnosti

Číselníky lze stylovat pomocí tříd ui-spinner (vnější kontejner), ui-spinner-input (původní element pole), ui-spinner-button (tlačítka pro ovládání hodnoty), ui-spinner-up a ui-spinner-down (tlačítko nahoru, resp. dolů).

Widget Spinner podporuje specifické číselné formáty (včetně finančních hodnot), ovšem potřebuje k tomu podporu frameworku Globalize. Na něj se podíváme někdy v budoucnu.

Nabídky (menu)

U webů a webových aplikací je často potřeba vytvářet různé nabídky (menu). V HTML je lze řešit jen staticky, pro ty dynamické sáhneme k widgetu, který je součástí jQuery UI.

jQuery (28) – nabídky (menu) v jQuery UI

Nulová podpora v HTML

Samotný jazyk HTML žádnou podporu pro nabídky neposkytuje. Jednoduché menu lze udělat například jako seznam (ten může být na druhou stranu víceúrovňový, stromového charakteru), ale tím to končí. Žádné rozbalování, žádná kontextová citlivost, žádné přizpůsobování situaci.

Je tedy třeba se poohlédnout jinde, například v jQuery UI, kde je připraven widget Menu. A již zmíněný seznam se bude hodit, protože umožňuje stát se dobrým základem pro interaktivní menu, současně ale může sloužit i pro ovládání webu bez JavaScriptu.

Pro menu v jQuery UI lze použít téměř libovolné HTML elementy, ale právě seznamy se na to hodí nejvíc a widget Menu s nimi ve výchozím nastavení počítá.

Nabídky v jQuery UI

Mějme HTML kód pro přechod na různé stránky v rámci webu:

```
<ul id="menu">
<li><a href="/">Hlavní stránka</a></li>
<li><a href="/sluzby">Služby</a></li>
    <li>
    <ul>Firma
<li><a href="/o-nas">O nás</a>
<li><a href="/kontakt">Kontakt</a>
    </ul>
    </li>
```

```
</ul>
```
Je to dvouúrovňový seznam s odkazy. Ten nyní převedeme na menu. První úroveň se bude zobrazovat stále, druhá až po najetí kurzorem myši.
```
$('#menu').menu();
```
Pozor na to, že pokud se menu nenastyluje a ponechá se v běžném toku obsahu na stránce, roztáhne se přes celou šířku stránky a druhá úroveň se rozbalí už „za roh". Proto je potřeba buď nastavit absolutní pozicování nebo šířku omezit. První způsob ukazuje následující obrázek:

Zobrazení dvouúrovňové nabídky

Kontextové nabídky

Chceme-li zobrazovat nabídku až po kliknutí na nějaké místo (ať už pravým tlačítkem a potlačit standardní nabídku prohlížeče, nebo tlačítkem levým), kód jednoduše upravíme:
```
$('#menu').menu();
$('#menu').css('display', 'none');
```
Menu teď není vidět a zobrazíme ho až po kliknutí na určitý element:
```
$('#menu-button').click(function() {
$('#menu').css('display', 'block');
$('html, body').click(function() {
$('#menu').css('display', 'none');
$('html, body').off('click');
});
return false;
});
```
Tento kód nastavuje reakci na událost kliknutí, kde zobrazí menu, ale následně také nastaví reakci na kliknutí mimo menu (aby se dalo vůbec zavřít). Tam se menu nabídka schová a reakce se odebere. Obslužná funkce pro kliknutí na element musí bezpodmínečně vracet false, jinak událost „probublá" až na úroveň těla dokumentu, bude zpracována nově nastavenou reakcí a menu se ihned zase zavře.

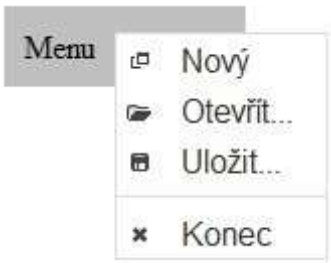
Ještě chybí napozicování kontextové nabídky. Ta se totiž zatím zobrazuje ve výchozí pozici (typicky 0, 0), což ale obvykle nechceme. Do obslužné rutiny přidáme tedy ještě kód, který nabídku otevře u kurzoru myši:
```
$('#menu-button').click(function(e) {
$('#menu').css('display', 'block').offset({ left: e.pageX, top: e.pageY });
…
```
Takto to pak vypadá v prohlížeči:

Kontextová nabídky otevřená kliknutím na určitém elementu stránky

Podobně jako kontextové menu se bude řešit i nabídková lišta.

Ikony, separátor

V nabídkách se často používají ikony symbolizující odkazovanou stránku nebo činnost. Widget Menu ikony podporuje, a to jak zabudované (v grafickém tématu), tak vlastní. Jak použít ty zabudované, je vidět z tohoto HTML kódu:
```
<ul id="menu">
<li id="menu-new"><span class="ui-icon ui-icon-newwin"></span>Nový</li>
<li id="menu-open"><span class="ui-icon ui-icon-folder-open"></span>Otevřít...</li>
<li id="menu-save"><span class="ui-icon ui-icon-disk"></span>Uložit...</li>
<li><span>-</span></li>
<li id="menu-exit"><span class="ui-icon ui-icon-close"></span>Konec</li>
</ul>
```
Všimněte si dvou věcí. Jednak že je element span je prázdný a text je až za ním. Další věc je řádek s neobvyklou položkou („-"). To je separátor, umožňující opticky oddělit různé části menu. Pokud chcete použít vlastní ikony, stačí místo některé z předdefinovaných tříd (např. ui-icon-close) použít třídy s vlastními definicemi.



Menu s ikonami

Widget disponuje vlastností icons, která umožňuje nastavovat ikony pro „režijní" účely. Aktuálně lze nastavit jen ikonu pro rozbalení podnabídky (submenu).

Události

Možná se ptáte, jak odchytávat události kliknutí na položky menu, pokud to nejsou odkazy. Úplně jednoduše – stačí reagovat na událost select:

```
var props = {
select: function(e, ui) {
switch (ui.item.attr('id')) {
case 'menu-open':
…
break;
...
}
}
};
$('#menu').menu(props);
```

Zde se reakce řeší přes identifikátory položek, možností je ale samozřejmě více.

Widget Menu má ještě další události, konkrétně create (jako všechny widgety jQuery UI), focus (získání fokusu) a blur (ztráta fokusu). Lze je využít například pro zobrazování podrobnějších informací o položkách.

Další informace o práci s nabídkami najdete, jako vždy, v dokumentaci.

Tak trochu jiná nabídka

Příště se podíváme na „trochu jinou nabídku", která se označuje jako *accordion* (akordeon, harmonika). Na webech se používá relativně málo, ale je to zajímavý prvek umožňující přiblížit webové rozhraní tomu tradičnímu desktopovému.

jQuery (29) – když uživatel „hraje na harmoniku"

Co je accordion

Každého nemusí napadnout podle názvu, jak vypadá tento ovládací prvek, ale drtivá většina uživatelů ho někdy použila. Svým chováním trochu připomíná ouška pro ovládání karet (listů, panelů atd., podle konkrétního názvosloví; v angličtině se říká *tabs*). V základním stavu připomíná stlačenou harmoniku – je vidět jen nejvyšší úroveň. Při roztažení – může jít roztáhnout vždy jen jeden prvek nebo i víc – se objeví další úroveň. *Accordion* může mít více úrovní než jen ty minimální dvě, stejně tak se může lišit chování při rozbalování (buď se zbytek harmoniky sbalí nebo zůstane rozbalený).

K podobným účelům se v uživatelském rozhraní používá také strom. Dá se říci, že strom je vhodnější při velkém počtu položek nebo složitější struktuře, kdežto accordion pro případy jednodušší, s menším počtem položek.

Widget Accordion v jQuery UI

Harmoniku v jQuery UI implementuje widget Accordion. Lze ho ovládat jak myší, tak i klávesnicí (pomocí šipek, Home/End a Enter či mezerníku), může být animovaný. Jeho chování lze přizpůsobit různým potřebám a vzhled nastylovat podle přání.

Widget implementuje pouze dvouúrovňový akordeon. Není ale problém do sebe widgety vnořit.

V kódu HTML je potřeba připravit kontejner, v němž budou jednak nadpisy jednotlivých sekcí a také obsah jejich panelů. Může to vypadat například takto:

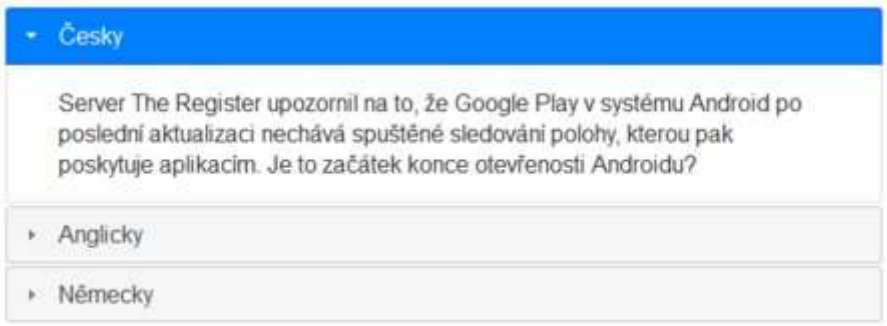
```
<div id="accord">
<h3>Česky</h3>
<p>…</p>
<h3>Anglicky</h3>
<p>…</p>
<h3>Německy</h3>
<p>…</p>
</div>
```

Není nutné používat tyto druhy elementů. Výchozí chování je takové, že se liché elementy používají jako nadpisy a sudé jako obsah. Lze to ale změnit.

Uvedený kus kódu reprezentuje akordeon pro zobrazení textu v různých jazycích. V této podobě je to ale jen běžný, neživý kód, který je potřeba teprve oživit:

```
$('#accord').accordion();
```

Výsledek můžete vidět na následujícím obrázku:



„Harmonika" realizovaná widgetem Accordion

Nastavení vlastností

V mnoha případech si lze vystačit s úplně základním použitím widgetu. Někdy ale můžeme mít specifické požadavky. Například lze měnit způsob, jak se pracuje s výškou. Výchozí je automatické nastavení jednotlivých „panelů", kdy se výška widgetu přizpůsobí výšce nejvyššího panelu. Pokud ale například potřebujeme vyplnit akordeonem určitou oblast, lze to zajistit:

```
var props = {
heightStyle: 'fill'
};

$('#accord').accordion(props);
```

Tato hodnota parametru heightStyle zajistí, že celý widget výškově vyplní rodičovský kontejner.

Výškové vyplnění rodičovského kontejneru

Opakem je, že každý panel bude jen tak vysoký, jak odpovídá jeho obsahu. Pro heightStyle se použije hodnota content. Pozor na to, že se bude celková výška widgetu měnit podle toho, který panel bude aktuálně zobrazen.

Jednoduchým způsobem můžeme také docílit, že lze celou harmoniku stlačit:

```
var props = {
collapsible: true
};
```

Stlačená harmonika (žádný panel se nezobrazuje)

Dále lze přímo určit, který panel bude ve výchozím stavu vidět (zde to bude 2; pokud je collapsiblenastaveno na true, dá se místo indexu použít false a harmonika zůstane stlačená).

```
var props = {
index: 2
};
```

Animace

Už ve výchozím nastavení je změna stavu widgetu Accordion animovaná. Animaci lze ale vypnout nebo změnit. Například takto:

```
var props = {
animate: 1000
};
```

Použitím číselné hodnoty se nastaví čas animace v milisekundách. Je ale možné použít i text označující druh animace (křivku), objekt s položkami duration a easing (kombinace dvou předchozích variant) a případně také down obsahující zmíněné dvě položky (použije se pro animaci při změně směrem k nižšímu indexu), poslední možností je hodnota false k úplnému vypnutí animace.

Události, nastylování

Pro widget Accordion jsou zajímavé hlavně události activate a beforeActivate. První nastává v poté, co se aktivuje (zobrazí) nový panel nebo se harmonika stlačí. Druhá událost nastává před aktivací (po uživatelově akci) a lze ji využít k zablokování změny stavu harmoniky.

Událostí lze s úspěchem využít například v případech, kdy se akordeon používá jen jako ovládací prvek a hlavní věci se dějí někde jinde. Například bude widget po levé straně od hlavní oblasti aplikace.

Co se týká vlastního nastylování, je k dispozici celá řada CSS tříd – například ui-accordion, ui-accordion-header nebo ui-accordion-content-active. Můžete si tak widget například přebarvit, změnit zaoblení nebo něco provést s panelem.

Záložky a panely

Příště se dostane na ony panely (karty, listy) se záložkami zmíněné na začátku. Principy fungování jsou prakticky stejné jako u harmoniky, na některé věci je ale potřeba si dát pozor.

jQuery (30) – záložky a panely

Terminologický „babylon“

Kdo pracuje na počítači, jistě se s panely setkal, například ve webových prohlížečích. Panely jsou jakoby nad sebou, takže je vždycky vidět jen ten, který je nahoře. Myší, případně klávesnicí, lze vybrat k zobrazení jiný panel.

Bohužel ale nepanuje zrovna shoda na tom, jak tyto prvky uživatelských rozhraní nazývat. Setkáváme se hlavně s panely, listy a kartami, k jejich ovládání se pak používají záložky či ouška. V angličtině se obvykle používá termín tab (označující především tu ovládací část, používá se ale i v kontextu celého panelu; pro samotný panel se někdy používá termín panel) a celé řešení pak tabbed document interface (TDI).

Následující odstavce budou důsledně používat termín panel pro celou plochu a záložka pro ovládací část.

Widget Tabs

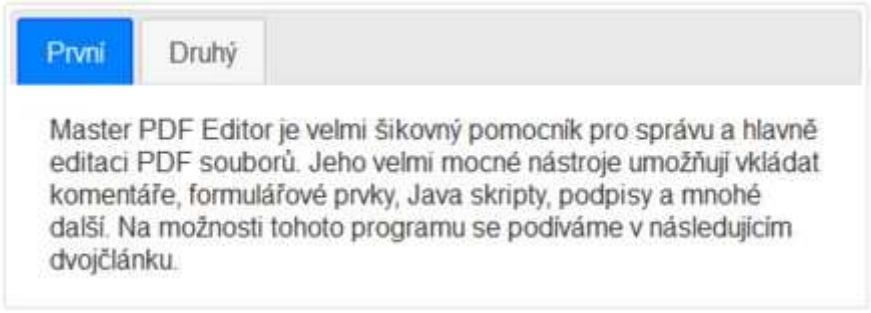
V jQuery UI se pro panely používá widget Tabs. Podobně jako u harmoniky, také tady se standardně využívají seznamy definované v HTML. Ty zde ale fungují tak, že v každém prvku seznamu je záložka definovaná jako odkaz na panel, vlastní obsah panelu je pak v jiném elementu. Kód může vypadat například takto:

```
<div id="tabs">
<ul>
<li><a href="#tab1">První</a></li>
<li><a href="#tab2">Druhý</a></li>
</ul>
<div id="tab1">
...
</div>
<div id="tab2">
...
</div>
</div>
```

V tomto případě jsou tu dva panely s obsahem a jsou ovládány příslušnými záložkami. Ke zprovoznění použijeme jednoduchý javascriptový kód:

```
$('#tabs').tabs();
```

Výsledkem je fungující panelové rozhraní:



Panelové rozhraní

Přepínat panely lze kliknutím na záložku požadovaného panelu (případně najetím, pokud se to nastaví), lze ale použít i klávesnici. Zde k ovládání slouží směrové šipky, které mění fokus záložek a při jeho ponechání na určité záložce se panel po chvilce aktivuje (přenese do popředí). Přidržením Ctrl lze tuto automatickou aktivaci vypnout, panel se pak aktivuje mezerníkem nebo klávesou Enter.

Další možnosti ovládání klávesnicí najdete v dokumentaci.

Vzdálený obsah panelů, styly

Panely nemusejí mít nutně statický obsah načítaný se stránkou, obsah lze také načítat ze vzdáleného serveru v okamžiku aktivace panelu. Stačí do odkazu v záložce jednoduše místo kotvy nastavit URL, odkud se má obsah načíst.

```
<div id="tabs">
<ul>
<li><a href="/robots.txt">Robots</a></li>
<li><a href="/README">Read me</a></li>
</ul>
</div>
```

Všimněte si, že tu nejsou elementy pro panely. Ty se vytvoří automaticky. Výsledek může vypadat například takto:

Panely se vzdáleně načítaným obsahem

Aby se soubory zobrazily správně (jsou čistě textové), bylo potřeba nastylovat třídu .ui-tabs-panel (nastavit neproporcionální písmo a interpretaci bílých znaků). Podobně si lze nastylovávat i další třídy, například ui-tabs (hlavní kontejner), ui-tabs-nav (seznam záložek) nebo ui-tabs-active (aktivní záložky).

Pozor na to, že při vzdáleném načítání prohlížeč kontroluje zdroj podle politiky původu. Data z „cizích" serverů se tedy ve výchozím stavu nenačtou. Pro jejich načítání je potřeba pomocí hlaviček nastavit CORS.

Změna vlastností panelů

Přestože si lze mnohdy vystačit se základními vlastnostmi panelového rozhraní (a jen si přizpůsobit CSS), někdy se hodí využít i možnost tyto vlastnosti změnit. Tady je příklad:

```
var props = {
heightStyle: 'auto',
collapsible: true
};

$('#tabs').tabs(props);
```

První parametr definuje výškový styl – je to úplně stejné jako u harmoniky, v příkladu výše se výška automaticky nastavuje podle nejvyššího obsahu ze všech panelů. Druhý parametr aktivuje možnost skrytí aktivního panelu kliknutím nebo klávesou Enter (mezerník ho neschová). Po skrytí aktivního panelu není zobrazen žádný panel.

Také lze – podobně jako u harmoniky – měnit animaci pro zobrazení a skrytí panelu. Ve výchozím stavu se neanimuje, lze ale nastavit různé efekty a měnit i dobu trvání:

```
var props = {
show: {
effect: 'fadeIn',
duration: 1000
```

```
}
};

$('#tabs').tabs(props);
```

Takto se zapne animace „roztmívání" trvající 1000 ms. Místo objektu s parametry lze uvést i pouze číslo s dobou trvání, řetězec s názvem efektu nebo i jen hodnotu true, která zapne výchozí animaci.

Ovládání panelů

Panely lze také přímo ovládat, a to jak ve smyslu výběru aktivního panelu (na počátku nebo kdykoli později), tak i ve smyslu zákazu či povolení některých panelů (zakázaný panel nelze aktivovat). Výchozí aktivní panel se nastaví parametrem active:

```
$('#tabs').tabs({ active: 3 });
```

Takto se jako aktivní nastaví panel s indexem 3. Později použijeme k témuž tento příkaz (neexistuje na to tedy samostatná metoda):

```
$('#tabs').tabs('option', 'active', 3);
```

Samostatné metody má ale zákaz a povolení panelů:

```
$('#tabs').tabs('disable', 2);
$('#tabs').tabs('disable', '#tab2');
$('#tabs').tabs('disable');

$('#tabs').tabs('option', 'disabled', [0, 3]);

$('#tabs').tabs('enable');
```

V příkladu jsou nejprve tři volání metody disable(). První zakáže panel s indexem 2 (pokud není aktivní; aktivní panel nelze zakázat), druhý s daným URL (v tomto případě kotvou, ale může to být i URL vzdáleného obsahu), třetí zakáže všechny panely. Pak následuje zákaz pomocí změny vlastnosti objektu – má tu výhodu, že lze v poli předat i více indexů panelů. Poslední příkaz povolí všechny panely (pro metodu enable() lze používat stejné parametry jako pro disable()).

Dále stojí za zmínku ještě metoda load(), která načte ze vzdáleného zdroje obsah panelu určeného indexem nebo URL.

Události

Běžně není události potřeba řešit, protože si je spravuje samotný widget Tabs (včetně načítání obsahu). Ale někdy je žádoucí na některé události reagovat. Widget podporuje nastavení reakce na vytvoření, aktivaci panelu, načtení obsahu a na přípravu k aktivaci a k načtení.

Poslední dvě události jsou „vetovatelné", čili umožňují zabránit chystané operace v provedení. To může být výhodné například v situaci, kdy se na panelu provedly změny a je potřeba se zeptat uživatele, jestli se mají opustit a jít jinam.

```
$('#tabs').tabs({
beforeActivate: function() {
return confirm('Opravdu chcete přepnout panel?');
}
});
```

Dialog

Tímto končí celá část seriál věnovaná jednoduchým widgetům a příště se podíváme na něco „výživnějšího": dialogy. Právě dialogy bývají často impulsem k použití jQuery. A widget Dialog toho umí opravdu hodně.

jQuery (31) – dialogy v jQuery UI

Co je dialog?

Důležité je si uvědomit, co to vůbec dialog ve webové aplikaci je. Je to ohraničená oblast okna (případně i samostatné okno, nezávislé na okně, kde je otevřena stránka), která se zobrazí a umožňuje interakci s uživatelem. Pro dialog je typické, že se zobrazuje jen po omezený čas, kdy je potřeba.

Dialog může mít modální charakter (neumožnit interakci se zbytkem aplikace, dokud je otevřen), též může mít – a často mívá – horní lištu s ovládacími tlačítky (nejčastěji se zavíracím tlačítkem, může mít i minimalizační, maximalizační, případně nabídkové), může mít měnitelné rozměry, téměř vždy bývá posouvatelný.

Dialog může být připraven předem nebo konstruován dynamicky, může komunikovat jen se zbytkem aplikace nebo i přímo se serverem apod. Typicky dialog obsahuje různé formulářové komponenty (textová pole, zaškrtávátka, tlačítka…), může být ale tvořen třeba i obrázkovou plochou (například náhledovou mapou).

Dialog v jQuery UI

Widget Dialog v jQuery UI zajišťuje základní dialogovou funkcionalitu – tedy vytvoření okna na stránce. Okno má horní lištu (standardně se zavíracím tlačítkem), lze ho posouvat a měnit jeho rozměry. Lze také přidat dolní tlačítka. O zbývající obsah dialogu se ale už musíme postarat my.

Posouvání dialogu za horní lištu lze zakázat, změnu rozměrů lze také buď zcela zakázat nebo jen nastavit meze.

Element pro dialog může být připraven v HTML předem nebo vytvořen až později za běhu. Staticky připraveno to může vypadat třeba takto:

```
<div id="dialog"></div>
```

Nejjednodušší dialog se potom vytvoří prostým zavoláním:

```
$('#dialog').dialog();
```

Jak si můžete vyzkoušet, zobrazí se prázdný dialog, se kterým lze pracovat pomocí myši a zavřít ho kromě tlačítka křížku i klávesou Esc.

Nejjednodušší dialog

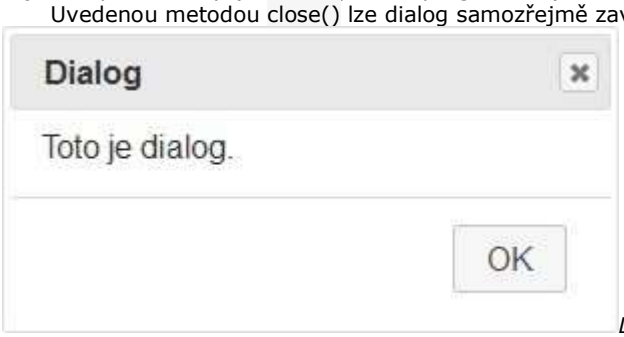
Jistě ale chceme víc, pro začátek třeba titulek na liště, nějaký text v dialogu a dole potvrzovací tlačítko. Nejdřív tedy HTML:

```
<div id="dialog">Toto je dialog.</div>
```

A teď JavaScript:

```
var props = {
title: 'Dialog',
buttons: [
{
text: "OK",
click: function() {
$(this).dialog('close');
}
}
]
};
$('#dialog').dialog(props);
```

Zprávu v dialogu zde definujeme opět staticky (lze ji ale přidat i programově), ostatní parametry potom při vytváření widgetu. Uvedenou metodou close() lze dialog samozřejmě zavřít i odjinud.



Dialog se zprávou

Pokročilé možnosti

V některých případech potřebujeme, aby byl dialog modální a nešla měnit jeho velikost. Aby to bylo zajímavější, přidáme si také animaci otevření a zavření dialogu.

```
var props = {
title: 'Dialog',
modal: true,
resizable: false,
show: 'fade',
hide: 'explode'
};
$('#dialog').dialog(props);
```

Na obrázku můžete vidět jeden okamžik animace zavírání dialogu efektem „explode“.

Zavírání dialogu – explode

Podobně lze nastavit i další parametry, například rozměry a jejich meze, podrobnosti najdete v dokumentaci.

Manipulace s dialogem

Zatím všechny příklady ukazovaly otevření dialogu zároveň s vytvořením widgetu. Operace lze ale rozdělit – pouze si widget připravit (otevření se potlačí nastavení autoOpen na hodnotu false) a otevřít ho až později.

```
$('#dialog').dialog({ autoOpen: false });
...
$('#dialog').dialog('open');
```

Metodu open() lze volat i poté, co byl dialog zavřen (opětovně se tím otevře). Prostředky pro dialog zůstávají alokované (ve formě objektů v DOM) až do chvíle, než se zavolá destroy(). Potom už samozřejmě dialog otevřít nelze a je potřeba ho znovu vytvořit.

Zda je dialog otevřen, lze snadno zjistit metodou isOpen().

Pokud máme dialogů na stránce více, může být užitečné poslat nějaký dialog nahoru, aby ho měl uživatel na očích. Zajistí se to snadno:

```
$('#dialog').dialog('moveToTop');
```

Události a styly

Widget Dialog umožňuje reagovat na celou škálu událostí. Při vytváření je to create, při otevírání open, před zavřením beforeClose a při zavření close. Událost beforeClose lze vetovat (vrátit false), dialog pak zůstane otevřený – to se velmi hodí jako ochrana proti ztrátě neuložených změn.

Dále jsou tu sady událostí souvisejících s posouváním dialogu (drag, dragStart, dragStop), změnou rozměrů (resize, resizeStart, resizeStop) a se získáním fokusu (focus). Hlavně poslední událost je užitečná, lze na ni „pověsit“ například aktualizaci dat.

Vzhled dialogu lze měnit pomocí CSS, k dispozici je řada tříd. Například ui-dialog představuje celý dialog, ui-dialog-titlebar titulkovou lištu, ui-dialog-titlebar-close zavírací tlačítko a ui-dialog-contentkontejner pro obsah dialogu. Tedy kdo si chce kupříklad obarvit lištu na modrou barvu, může tak snadno učinit:

```
.ui-dialog-titlebar {
background-color: blue
}
```

Vlastní widget

Tímto je u konce procházka přes widgety, které jQuery UI nabízí. Příště si vytvoříme widget vlastní, protože je to občas potřeba a není to vůbec složité.

jQuery (32) – vlastní widget v jQuery UI

Opravdu potřebujeme něco vlastního?

Na podobnou otázku se celkem logicky nabízí odpověď „ne". A často to tak je. Kromě widgetů, které jsou obsaženy v jQuery UI, totiž existuje spousta takových, které už někdo vytvořil, protože řešil v minulosti stejný problém. Háček je ale v tom, že většina takových widgetů je nějak licenčně omezena – jejich licence je proprietární a za použití se platí buď vždy, nebo třeba při komerčním užití.

Základní jQuery a jQuery UI mají licenci MIT. Ta je svobodná, ale není copyleftová. Proto lze prakticky bez omezení vytvářet widgety s libovolnými licencemi.

Proto je leckdy vhodnější si widgety vytvořit a nakládat s nimi podle svého – tedy používat je ve svých projektech nebo je i nabídnout k dalšímu použití, ať už pod jakoukoli licencí.

Továrna na widgety

Nejsnazší cesta, jak si vytvořit vlastní widget, je „továrna na widgety", tedy tovární metodu widget() objektu jQuery. Této metodě se předá název widgetu, bázový typ widgetu a objekt pro použití jako prototyp. Název je jasný (musí být ale jedinečný). Od bázového typu bude odvozen nový widget – pokud se nezadá explicitně, použije se jQuery.Widget. A konečně prototypový objekt bude ten, podle něhož se budou vyrábět instance widgetu.

Možná to není úplně srozumitelné, takže bude nejlepší se podívat na příklad. Vytvoříme si nový widget odvozením od tlačítka, tedy widgetu Button:

```
$.widget('mynamespace.mybutton', $.ui.button,
{
_create: function() {
this.element.button();
}
});
```

Název je žádoucí vytvářet v samostatném jmenném prostoru (zde mynamespace). Pro odvození se použije třída $.ui.button (standardní widget Button) a v rámci prototypového objektu si připravíme metodu _create(), která zajistí vytvoření widgetu. V tomto případě jen vytvoří standardní tlačítko. Widget by se následně použil například takto (bez uvedení jmenného prostoru):

```
$('#button').mybutton();
```

Pracujeme s parametry

Jako parametry budeme chtít nastavovat barvu textu a pozadí. Definujeme si tedy výchozí hodnoty, které si tvůrce instance případně předefinuje podle potřeby:

```
$.widget('mynamespace.mybutton', $.ui.button,
{
options: {
foreground: 'black',
background: 'yellow'
},
_create: function() {
this.element.button();
this.element.css('color', this.options.foreground)
.css('background-color', this.options.background);
}
});
```

Při následném vytvoření widgetu bez parametrů bude text černý a pozadí žluté. Můžeme ale zadat některý z parametrů či oba dva, jako u běžných widgetů:

```
$('#button').mybutton({ foreground: 'white', background: 'black'});
```

Zjišťování a změna parametrů za běhu

Zjišťování hodnot parametrů není potřeba vůbec řešit. Už tím, že vytvoříme v prototypu objekt options, hodnoty parametrů zpřístupníme. Problém je se změnou hodnot, protože ta musí často mít vedlejší efekt (nestačí jen nastavit hodnoty, ale musí se stát ještě něco dalšího – tedy u toho tlačítka změna barev). Bude proto potřeba si definovat metody _setOption() a _setOptions().

Zbývající část kódu prototypu se nemění.

```
_setOption: function(key, value) {
switch (key) {
case 'foreground':
this.options.foreground = value;
this.element.css('color', this.options.foreground);
break;
case 'background':
this.options.background = value;
this.element.css('background-color', this.options.background);
break;
this._super(key, value);
}
},

_setOptions: function(options) {
```

```
this._super(options);
this.element.css('color', this.options.foreground)
.css('background-color', this.options.background);
}
```

Rozeberme si to. Metoda _setOption() zde pro jednotlivé parametry vždy nejdřív nastaví hodnotu a pak ji ještě uplatní na elementu. Následně je potřeba zavolat stejnou metodu na předkovi. U metody _setOptions()je to podobné, nejdřív se ale volá metoda předka (která vše nastaví) a pak se hodnoty nastaví na elementu. Netestuje se, zda se něco změnilo, protože v tomto případě to nevadí. Někdy by ale bylo žádoucí otestovat (třeba už jen výkonových důvodů), zda došlo ke změně, a teprve potom provádět nějaký výkonný kód.

Metody widgetu

Podobně jako standardní widgety, také ty vlastnoručně implementované mohou mít různé metody pro ovládání. Widget přebere do svého prototypu metody z předka, ty lze poté předefinovat nebo si nadefinovat metody zcela nové.
Například Button má metodu showLabel(), které se můžeme chtít zbavit, tak ji implementujeme s prázdným tělem (správně by bylo potřeba zamezit i změně přes parametry, ale to už jistě každý dokáže dotvořit sám). Naopak si přidáme metodu invert(), která prohodí barvu popředí a pozadí.

```
showLabel: function() {},
invert: function() {
this._setOptions({
foreground: this.options.background,
background: this.options.foreground
});
}
```

Takto by se pak zvenčí nastavilo, aby se barevnost invertovala na kliknutí:

```
$('#button').click(function() {
$(this).mybutton('invert');
});
```

Další informace o metodách a další práci s widgety najdete v dokumentaci.

Úklid

Pokud si vytvoříme nějaká data v DOM (atributy, elementy…), která by po skončení práce s widgetem neměla přetrvat, je potřeba je uvolnit. Totéž se týká například i tříd, které nastavujeme elementu, s nímž pracujeme.

```
_create: function() {
this.element.addClass('mybutton');
},

_destroy: function() {
this.element.removeClass('mybutton');
}
```

Příklad ukazuje odstranění třídy přidané elementu při vytvoření widgetu.

Události

Pro interakci mezi widgetem a okolím (iniciovanou ve widgetu) využijeme události. Není třeba se starat o to, zda na události někdo reaguje, událost prostě nastane (spustí se).

```
if (this.options.foreground == this.options.background) {
this._trigger('coloridentity', null, { color: this.options.foreground });
}
```

Kód příkladu bychom přidali jako ověření nastavených barev. Pokud budou obě barvy shodné, nastane událost coloridentity, na niž může někdo reagovat. Jako parametr do obslužné rutiny se předá konkrétní barva. Lze předávat také objekt události (zde je null).
Widget může také reagovat na cizí události. Netřeba dodávat, že nastavené reakce je potřeba při rušení widgetu zrušit (v metodě _destroy()).

Mobilní prostředí

Úsek seriálu o jQuery UI je u konce. Příště už přejdeme do mobilního prostředí, které je u jQuery reprezentováno frameworkem jQuery Mobile. Tvorba webových aplikací pro mobilní telefony a tablety s jeho použitím je doslova hračkou.

jQuery (33) – úvod do jQuery Mobile

Co je jQuery Mobile

Při vývoji mobilních či responzivních webových aplikací si lze vystačit s běžnými frameworky jQuery a jQuery UI. Ty však neobsahují podporu pro uživatelské interakce specifické pro mobilní telefony, tablety, navigace a další zařízení s dotykovými displeji a zároveň bez vstupních zařízení používaných u „velkých" počítačů.
Proto vznikl framework jQuery Mobile, který specifika mobilních zařízení dokáže využít. Co všechno tento framework nabízí?

- speciální widgety vhodné pro mobilní přístroje (rozbalovatelné widgety, posuvný přepínač, panel, tabulka s automaticky skrývanými sloupci…)
- události typické pro dotykové ovládání (krátký/dlouhý dotyk, posun, změna orientace…)
- ikony ve formátech SVG a PNG
- sloupcový layout, responzivní mřížku
- grafická témata a CSS třídy

Na speciálních stránkách se můžete podívat, jak je na tom jQuery Mobile s podporou různých prohlížečů. Dá se říci, že až na výjimky (velmi staré prohlížeče) je framework podporován výborně.

Kde získat a jak použít

Podobně jako jQuery UI, také jQuery Mobile pro svoji činnost potřebuje jQuery. Není ale třeba se o to starat, protože ať už využijete kterýkoli postup, jeho součástí je i použití tohoto frameworku.
Nejjednodušší je samozřejmě vložit soubory z CDN, protože není nutné nic stahovat ani umísťovat na server. Projekt jQuery má vlastní CDN využívající službu MaxCDN, takže logickou volbou je tato možnost. Dále můžete využít i CDN firem Google nebo Microsoft. Odkazy najdete na stahovací stránce.
Pro ladění použijete nekomprimovanou verzi, pro ostrý provoz je vhodnější verze minifikovaná a komprimovaná. U CSS si můžete navíc vybrat, zda má soubor obsahovat i standardní téma či nikoli. Druhý případ využijete v případě, že máte téma vlastní.

Dále je samozřejmě možnost si soubory stáhnout a uložit na svůj server, a to buď v kompletní podobě, nebo v podobě připravené pomocí Download Builderu (podobného jako u jQuery UI). Ten umožňuje zaškrtnout jen ty komponenty, které chcete do balíčku zahrnout (pokud využijete jen malou část a chcete mít soubory co nejmenší).
Ve všech případech pak samozřejmě musíte vložit do HTML odkazy na jednotlivé soubory, ať už vzdálené nebo místní (myšleno na stejném serveru).
Nevyhovuje-li vám základní grafické téma, můžete si vytvořit téma vlastní. Pohodlně k tomu poslouží nástroj ThemeRoller– s ním si můžete témata doslova „naklikávat", prohlížet vzájemně porovnávat atd.

Základní koncepty jQuery Mobile

Přestože jQuery Mobile vychází ze stejných základů jako jQuery a jQuery UI, zavádí zcela nové koncepty, s nimiž je pro efektivní práci potřeba se seznámit. Každá stránka je tvořena stromovou strukturou odpovídající následující podobě:

- stránka (nejvyšší úroveň, přímo pod body)
  - hlavička (horní lišta)
  - obsahová část (hlavní obsah)
  - patička (dolní lišta)

Příště se podíváme, jak tato struktura vypadá v HTML. Pro tuto chvíli je důležité si říct, že žádná z těchto součástí není povinná. Navíc stránek (a tedy celých struktur) může být v jednom souboru více. Rozlišují se pomocí identifikátorů (atribut id) a lze na ně odkazovat pomocí kotev (#něco).
Další specifickým konceptem je využití atributu data-role. Ten se využívá k vytváření widgetů z elementů v HTML. Jde tedy o deklarativní přístup, kdy se základ programu tvoří deklaracemi (v tomto případě pomocí speciálního atributu) namísto volání metod, které se využívá u imperativního přístupu, jak ho známe z jQuery UI.
Obdobně se deklarují také ikony (atribut data-icon), případně jejich pozice (data-iconpos) a dokonce i téma (data-theme).
Používání popsaných konceptů není povinné, ale je nejjednodušší a zároveň nejčistší cestou k využití frameworku. Pokud chce někdo něco udělat jinak, měl by k tomu mít pádné důvody.

Začínáme s jQuery Mobile

Obecný úvod je u konce, příště už se podíváme na to, jak s jQuery Mobile vytvořit jednoduchou fungující aplikaci.

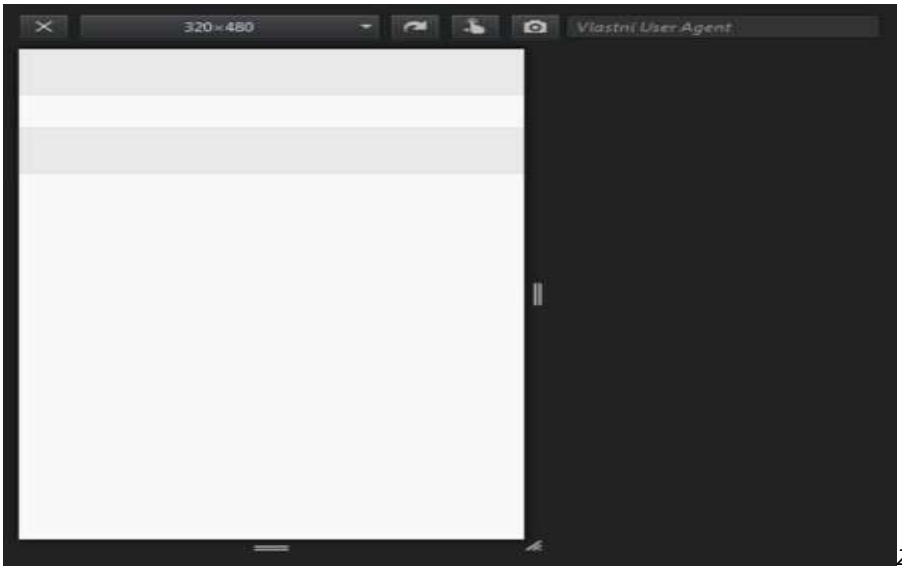
jQuery (34) – základ mobilní aplikace v jQuery Mobile

Kostra aplikace

Když se podíváte do minulého dílu seriálu, najdete tam informace o základních konceptech frameworku. Ty nyní „přetavíme" do skutečné mobilní webové aplikace. Kostra úplně nejjednodušší aplikace vypadá nějak takto:

```
<!doctype html>
<html>
<head>
<title>Nejjednodušší aplikace</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="css/jquery.mobile.min.css">
<link rel="stylesheet" href="css/app.css">
<script src="js/jquery.min.js"></script>
<script src="js/jquery.mobile.min.js"></script>
<script src="js/app.js"></script>
</head>
<body>
<div data-role="page">
<div data-role="header"></div>
<div role="main" class="ui-content"></div>
<div data-role="footer"></div>
</div>
</body>
</html>
```

Zmínku si zaslouží hned několik věcí. V první řadě je to meta tag viewport, který říká, jak se má stránka zobrazovat. V tomto případě se bude vycházet z šířky zařízení a výchozí škálování bude této šířce odpovídat. Dále tu máme odkazy na javascriptové soubory – tam je potřeba vložit základní jQuery, jQuery Mobile a soubor(y) aplikace; nesmí se zapomenout ani na stylopisové soubory CSS.
Soubory jQuery a jQuery Mobile lze samozřejmě vložit i z CDN, nemusejí se vkládat lokálně, jako je tomu v příkladu.
Vlastní kostra aplikace má tady úplně základní podobu (bez dalších obsahových součástí), ale s uvedením všech tří hlavních prvků. Jak se můžete přesvědčit, zobrazí se tři prázdné prvky s výchozím nastylováním.
Pro testování mobilních aplikací můžete používat například funkci *Responzivní design* z vývojářských nástrojů v prohlížeči Mozilla Firefox. Obdobnou funkcí disponují i prohlížeče Google Chrome a Chromium.



Základ aplikace bez obsahu

Aplikace s obsahem

Do aplikace nyní přidáme nějaký obsah. V hlavičce stránky bude titulek, v patičce stavový řádek, v těle nějaký text. Na obrázku je vidět, jak zafunguje základní nastylování. Pokud vám nevyhovuje, můžete si ho změnit podle potřeby.

```
<body>
<div data-role="page">
<div data-role="header">
<h1>Aplikace</h1>
</div>
<div role="main" class="ui-content">
<p>
Lorem ipsum…
</p>
</div>
<div data-role="footer">
<p>Aplikace připravena</p>
</div>
</div>
</body>
```

Aplikace s obsahem

Změnou rozlišení se můžete přesvědčit, jak funguje responzibilita. Vzhled aplikace se automaticky přizpůsobí (i když v tomto případě je to velmi primitivní).

Aplikace po změně rozlišení displeje

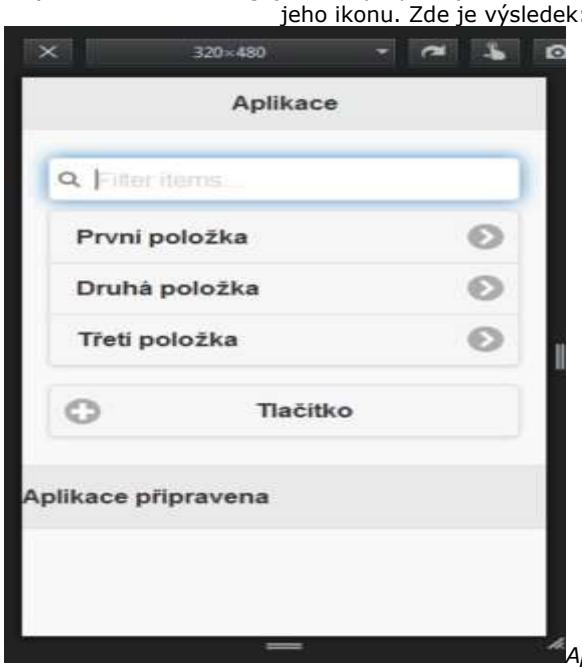
Ve výchozím stavu aplikace stále vypadá jako stránka, čili patička následuje za tělem a není navázána na dolní hranu obrazovky. Totéž se týká i hlavičky, ta je zase vždy na začátku stránky. V budoucnu se podíváme i na to, jak to změnit.

Ovládací prvky

Nyní zkusíme přidat do aplikace nějaké ovládací prvky. Používá se k tomu deklarativní přístup, do HTML se tedy vloží požadované elementy a u nich se nastaví atribut data-role. Framework jQuery Mobile z nich automaticky vytvoří widgety. Takto může vypadat tělo stránky:

```
<div role="main" class="ui-content">
<ul data-role="listview" data-inset="true" data-filter="true">
<li><a href="#">První položka</a></li>
<li><a href="#">Druhá položka</a></li>
<li><a href="#">Třetí položka</a></li>
</ul>
<div>
<a href="#" data-role="button" data-icon="plus">Tlačítko</a>
</div>
</div>
```

Toto tělo obsahuje seznam položek a tlačítko. Že se z obyčejného seznamu (ul) stane seznam zpracovávaný frameworkem, říká jeho role listview. Nastavení data-inset na true způsobí, že položky „vylezou" nad pozadí, nastavení atributu data-filter přidá možnost položky filtrovat (filtr standardně funguje hned při psaní). Ohledně tlačítka lze říct jen to, že atribut data-icon nastaví jeho ikonu. Zde je výsledek:



Aplikace s ovládacími prvky

Oživení aplikace

Příště se podíváme, jak takto vytvořenou aplikaci oživit, přidat jí interaktivitu.

jQuery (35) – interaktivita mobilní aplikace v jQuery Mobile

Události v mobilní aplikaci

Základem interaktivity jsou v jQuery Mobile samozřejmě události. K dispozici máme klasické události známé z jQuery a jQuery UI (například načtení či připravenost objektu, kliknutí, stisk klávesy apod.), ale zajímavější jsou události obsažené přímo v jQuery Mobile. Ty totiž odpovídají dotykovému ovládání a dalším vlastnostem mobilních aplikací.

K dispozici je také sada virtualizovaných událostí ukazovacích zařízení – tyto události jsou abstrakcí nad událostmi myši a dotykového displeje. Další sada událostí souvisí se „stránkami" v rámci jedné webové stránky (data-role="page").

Dotykové ovládání

Kdo používá chytrý telefon nebo tablet, určitě zná ovládání mobilních aplikací pomocí dotykového displeje. Základem jsou akce:

- krátký dotyk (ťuknutí na displej)

- dlouhý dotyk (podržení prstu na displeji)
- svislý posun (nahoru nebo dolů)
- vodorovný posun (vlevo nebo vpravo) – „listování"

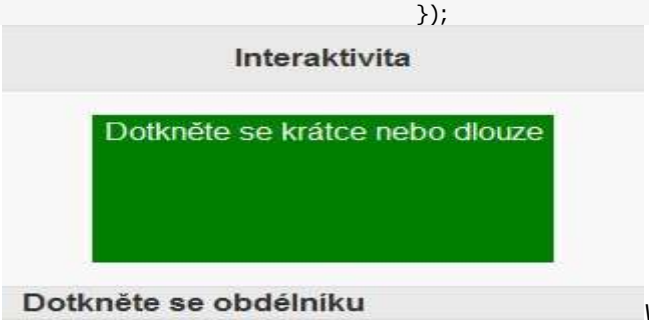
Tyto akce mají přímou podporu v událostech jQuery Mobile. K dalším známým mobilním akcím patří opakovaný krátký dotyk (poťukání), současný dotyk více prstů, oddalování nebo přibližování prstů (ve webovém prohlížeči mění zvětšení stránky), přetažení přes okraj, všelijaká gesta (kruh, „fajfka", střih…) apod. Tyto akce v jQuery Mobile podporu nemají, ale některé z nich si lze v případě potřeby doimplementovat.

Podívejme se na následující příklad, který ukazuje reakce na dotyky. V HTML bude definován element na stránce:

```
<div id="rectangle">
Dotkněte se krátce nebo dlouze
</div>
```

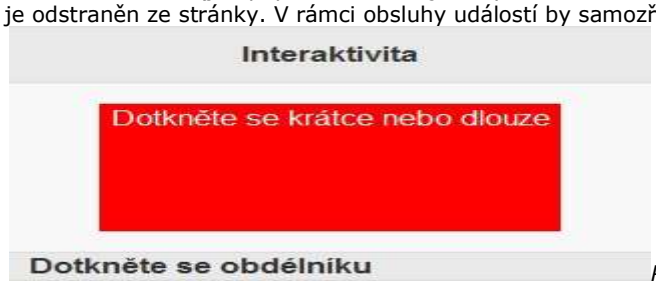
A obsluha událostí může vypadat například takto:

```
$('#rectangle').tap(function() {
$(this).css('background-color', 'red');
}).taphold(function() {
$(this).remove();
});
```

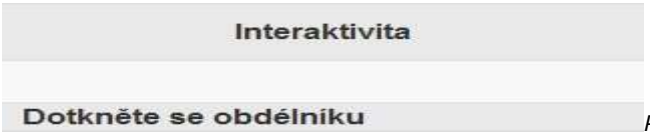


Výchozí podoba GUI

Krátkým klepnutím element zčervená (jen poprvé; samozřejmě by šlo snadno napsat, aby se na další dotyky barva dále měnila), dlouhým dotykem je odstraněn ze stránky. V rámci obsluhy událostí by samozřejmě bylo možno aktualizovat i aplikační texty.



Podoba po dotyku obdélníku



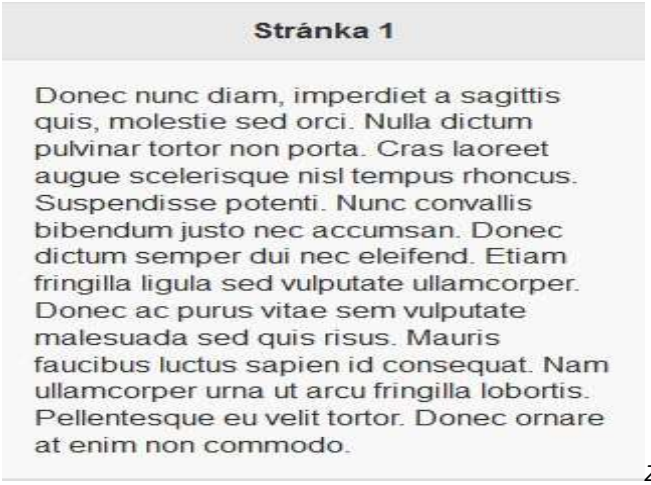
Podoba po podržení prstu na obdélníku

Svislý posun je obdobný jako u běžného jQuery, nemá proto smysl se jím zvlášť zabývat. Zajímavější je posun vodorovný, který lze s výhodou využít například pro listování ve stránkách. Mějme například následující HTML:

```
<div data-role="page" id="page1">
…
</div>
<div data-role="page" id="page2">
…
</div>
<div data-role="page" id="page3">
…
</div>
```

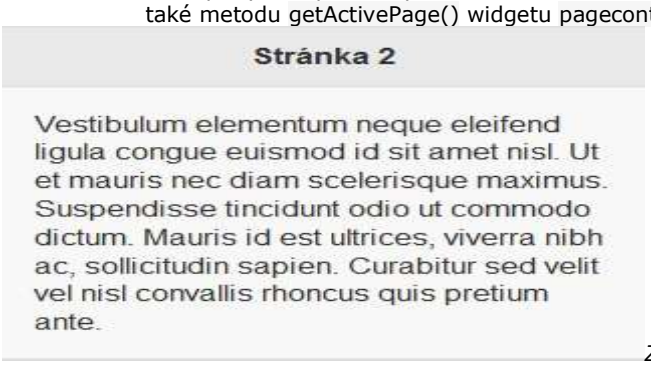
Každá stránka má své obvyklé součásti (hlavičku, obsah, patičku). Nyní instalujeme obsluhu události vodorovného posunu, kterým budeme stránky přepínat.

```
$('[data-role=page]').swipeleft(function() {
var n = Number($(this).attr('id').replace('page', '')) - 1;
if (n > 0) {
$(':mobile-pagecontainer').pagecontainer('change', '#page' + n);
}
}).swiperight(function() {
var n = Number($(this).attr('id').replace('page', '')) + 1;
if (n <= $('[data-role=page]').length) {
$(':mobile-pagecontainer').pagecontainer('change', '#page' + n);
}
});
```



Zobrazení stránky 1

Jak to funguje? Nastaví se obsluha událostí pro posun vlevo a vpravo (existuje i událost swipe pro posun libovolným směrem), v ní se pak přepíná na stránku s nižším, resp. vyšším pořadovým číslem. Místo získávání čísla z identifikátoru stránky lze použít také metodu getActivePage() widgetu pagecontainer.



Zobrazení stránky 2

Vodorovný posun nastává, pokud dojde k přemístění prstu minimálně o 30 pixelů vodorovně a maximálně o 30 pixelů svisle. Tyto hodnoty lze změnit.

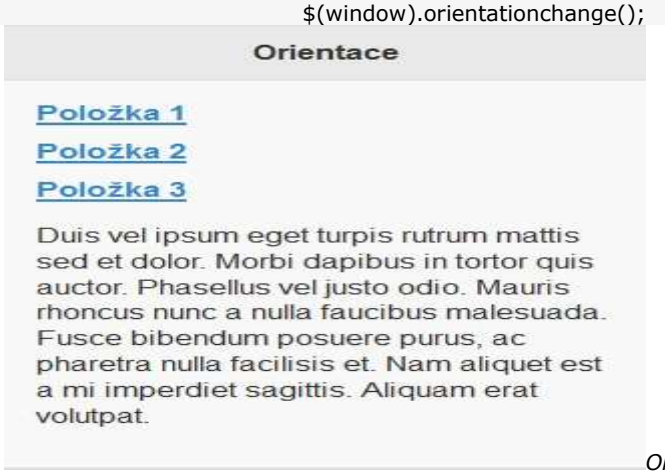
Obsah stránek lze také načítat přes AJAX, ale to si necháme na příště.

Orientace displeje

API jQuery Mobile poskytuje i další události. Jednou z nich je změna orientace displeje. Můžeme ji využít například pro zobrazení/skrytí (případně změnu polohy, velikosti, způsobu zobrazení apod.) nějakého prvku GUI pro konkrétní orientaci.

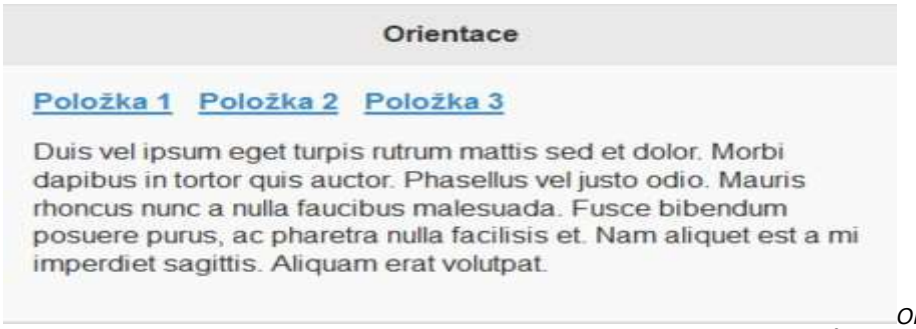
```
$(window).orientationchange(function(e) {
    switch (e.orientation) {
        case 'portrait':
$('#menu').removeClass('landscape').addClass('portrait');
            break;
        case 'landscape':
$('#menu').removeClass('portrait').addClass('landscape');
            break;
    }
});

$(window).orientationchange();
```



Orientace displeje na výšku

Při změně orientace se upraví způsob zobrazení menu nastavením příslušné CSS třídy. Pokud zařízení nepodporuje detekci orientace displeje (nebo je její použití zakázáno v nastavení frameworku), nastává tato událost při změně velikosti displeje. S tím je potřeba počítat, tedy například opakované zavolání obsluhy události by nemělo vést k odlišným výsledkům.



Orientace displeje na šířku

Po nastavení obsluhy je potřeba událost vyvolat, aby se počáteční stav přizpůsobil aktuální orientaci displeje.

Načítání stránek a související události

Stránky se mohou načítat ze serveru, na což se podíváme příště, společně s událostmi se stránkami spojenými. Touto cestou lze vytvářet velmi flexibilní aplikace.

jQuery (36) – interaktivita a AJAX v jQuery Mobile

Dynamické načítání stránek

Jednotlivé stránky v rámci aplikace nemusejí být definovány jen napevno. Lze je také načítat ze serveru technologií AJAX – koncepce jQuery Mobile s tím přímo počítá. Hodí se to hlavně v situacích, kdy obsah některých stránek závisí na uživatelské činnosti na jiných stránkách.

Existují v zásadě dva způsoby načítání stránek:

1. při přepínání na stránku,
2. načtení nové stránky a přidání do kontejneru.

Postupně se na oba způsoby podíváme.

Načtení stránky při přepínání

Přepínání mezi stránkami znáte už z minulého dílu seriálu. Tam se ale přepínalo jen mezi staticky načtenými stránkami (obsaženými přímo v základním souboru). Lze ale obsah stránky také načíst až v okamžiku, kdy se na ni přepíná.

V některých případech bývá vhodnější načítat jen nějaké jednoduché hodnoty, které se ve stránce změní. Tam je pak postup obdobný jako v běžném jQuery.

Údaje potřebné pro načtení stránky se jednoduše předají do metody change() objektu kontejneru stránek:

```
var opts = {
dataUrl: '/page.php?p=1',
reload: true,
showLoadMsg: true
};

$(':mobile-pagecontainer').pagecontainer('change', '#page1', opts);
```

Parametry se – jako obvykle – definují v objektu. Položka dataUrl říká, odkud se stránka načte.

Nastavením reload na hodnotu true (výchozí je false) se vynutí nové načtení i v případě, že už byla stránka dříve načtena (vhodné pro aktualizaci dat). A konečně showLoadMsg říká, že se má během načítání zobrazovat informace, že se data načítají.

Další použitelné parametry najdete v dokumentaci.

Načítání nové stránky

Kromě existujících stránek můžeme načítat i úplně nové a vkládat je do DOM (do kontejneru stránek). Je to velmi podobné jako při načítání existujících stránek, jen se použije jiná metoda a zároveň musíme říct, jak stránku vytvořit.

```
var opts = {
type: 'post',
showLoadMsg: true
};

$(':mobile-pagecontainer').pagecontainer('load', '/createpage.php?p=5', opts);
```

Tady se URL předává přímo jako argument metody. V objektu s parametry lze nastavit další vlastnosti, v tomto případě je to mj. typ požadavku (HTTP metoda). Identifikátor stránky je potřeba přenést v atributu hlavního elementu. Může být nastaven i atribut data-role, ten lze ale změnit i parametrem role v objektu předávaném metodě load() – toto platí i pro načítání při přepínání stránek.

Události při načítání a přepínání

Přestože si lze leckdy vystačit i s pouhým načtením stránky a případně vložením do kontejneru, mnohdy se nevyhneme dalším operacím, které s načítáním souvisejí – už proto, že je vhodné reagovat alespoň na chybové stavy a dát uživateli vědět, že se něco nepovedlo. Další události souvisejí s přepínáním stránek.

Asi nejlepší je, se na to podívat z hlediska časového – jak postupně události nastávají:

- beforechange – před změnou (přepnutím) stránky včetně načítání,
- beforeload – před načítáním stránky ze serveru,
- load – po úspěšném načtení stránky,
- loadfailed – po neúspěšném načtení stránky,
- changefailed – po neúspěšném přepnutí stránky (pokud se načítala ze serveru a to selhalo),
- beforetransition – před zahájením přepínací animace,
- beforehide – před zahájením animace mizení stránky,
- beforeshow – před zahájením animace zobrazení stránky,
- transition – po dokončení přepínací animace,
- hide – po dokončení animace mizení stránky,
- show – po dokončení animace zobrazení stránky,
- change – po úspěšném dokončení změny (přepnutí stránky).

Do obsluhy událostí se předává řada parametrů, které lze využívat. Takto se například obslouží událost loadfailed:

```
$(document).on('pageloadfailed', function(e, data) {
switch (data.textStatus) {
```

```
case 'timeout':
...
break;
case 'parseerror':
...
break;
default:
...
}
};
```

V tomto příkladu se obsluha nastavuje přímo celému dokumentu, šla by ale samozřejmě nastavit i kontejneru stránek (který je zatím vždy jen jeden, i když se do budoucna počítá s tím, že jich bude moci být více). Selže-li načtení stránky, podle druhu chyby se na událost různým způsobem zareaguje.

Odstranění stránky

Pokud už stránku nepotřebujeme, můžeme ji odstranit – a to úplně stejně jako jiný objekt z DOM:

```
$('#page9').remove();
```

Specialitou je tu jen to, že lze kontejneru nastavit obsluhu události remove, která dostane i informaci o tom, na kterou stránku se po odstranění přejde.

Podrobněji o stránkách

Příště se na stránky podíváme podrobněji. Nemusejí totiž fungovat jen jako „nějaký obsah", nýbrž podporují i určitou vlastní inteligenci.

jQuery (37) – další možnosti stránek v jQuery Mobile

Chytřejší stránky

Minulý díl seriálu ukázal stránky v aplikacích s jQuery Mobile v „dynamickém světle", tedy s možností načítat jejich obsah přes AJAX. Stránky toho ale umějí ještě víc. Dá se jim přidat určitá inteligence.

Podívejme se tedy nyní na některé metody, které umožňují ze stránek dostat ještě více.

Cache v DOM

Je to zdánlivě bezvýznamná drobnost, která ale umožňuje podstatně zrychlit aplikaci. Ve výchozím stavu se totiž stránky, které uživatel opustí, odstraňují z DOM. Při návratu se musí celá struktura stránky vytvořit znovu. To může být hlavně u složitějších stránek poměrně neefektivní.

Proto lze zapnout, aby se opuštěné stránky v DOM ponechávaly (ten tedy slouží v podstatě jako cache). Vrátí-li se uživatel na již dříve navštívenou stránku, ta se přímo zobrazí, bez nutnosti znovu vytvářet celý strom objektů.

```
$.mobile.page.prototype.options.domCache = true;
```

Tímto se zapne ponechávání všech stránek. Chceme-li ponechávat jen některé, nastavíme jim to přímo:

```
$('.cached').page('option', 'domCache', true);
```

V tomto příkladu se vyberou všechny stránky s CSS třídou cached a těm se cachování zapne. Případně to lze udělat i deklarativně, pomocí atributu v HTML:

```
<div data-role="page" data-dom-cache="true">
```

Pozor na to, že ponechávání stránek v DOM spotřebovává operační paměť. Její nedostatek (hlavně u starších a levnějších mobilních zařízení) může vést ke zpomalení běhu, případně k ukončování běžících aplikací, v krajním případě k pádu webového prohlížeče.

Přednačítání stránek

Podobným „urychlovačem" je přednačítání stránek. Jednou z možností je využití javascriptového načítání, popsaného v minulém dílu. Urychlení spočívá v tom, že se stránky načítají s předstihem, tedy ne až ve chvíli kdy jsou potřeba.

Druhou možností jsou pak odkazy v HTML, kterým se nastaví atribut data-prefetch:

```
<a href="page2.html" data-prefetch="true">Druhá stránka</a>
```

I při tomto způsobu načítání stránek lze reagovat na události (nastavit jejich obsluhu) a uživateli například načítané stránky zpřístupňovat, měnit způsob zobrazení přepínacích odkazů apod.

Dialogy

Stránka má možnost se chovat i jiným než základním způsobem. Jednou z velmi častých úprav stránky je dialog. Ve starších verzích jQuery Mobile se na dialogy používal speciální widget Dialog, od verze 1.4 je ale jeho použití zavržené a v 1.5 bude zřejmě odstraněn.

Nejvhodnější je použití stránky jako dialogu deklarovat pomocí atributu v HTML:

```
<div data-role="page" data-dialog="true">
```

Případně to lze řešit i v JavaScriptu, byť to znepřehledňuje aplikaci a proto je lepší se tomu vyhnout:

```
$('.dlg').page('option', 'dialog', true);
```

Zobrazení stránky jako dialogu

Výsledkem konverze stránky na dialog je, že se zobrazí trochu jiným způsobem a standardně také přibude uzavírací tlačítko. To se ve výchozím stavu zobrazí vlevo nahoře, ale lze ho také přesunout na pravou stranu, případně zcela odstranit:

```
<div data-role="page" data-dialog="true" data-close-btn="right">
$('.dlg').page({  closeBtn: 'none' });
<div data-role="page" data-dialog="true" data-close-btn-text="Zavřít, až žerná">
```

První příklad ukazuje deklarativní přesun tlačítka na pravou stranu. Ve druhém příkladu se tlačítko úplně odstraní, a to imperativně z JavaScriptu. Třetí příklad – opět deklarativně – mění text tlačítka.

Text uzavíracího tlačítka se běžně nezobrazuje. Hodí se ale například pro hlasové rozhraní nebo jiné podobné situace.

Pro zavření dialogu jiným způsobem než uzavíracím tlačítkem, vytvoří se odkaz s cílem „#" a nastaví se mu atribut data-rel="back". Kliknutí na něj uzavře dialog a způsobí návrat o úroveň výš (tedy na nadřazenou stránku, případně do nadřazeného dialogu). Pokud je potřeba udělat ještě nějaké akce, dá se uzavření vyvolat programově:

```
$.mobile.back();
```

Další widgety

Příští díl bude věnován dalším widgetům, které jsou v jQuery Mobile k dispozici. Není jich sice tolik jako v jQuery UI, i tak z nich ale lze sestavit uživatelské rozhraní k téměř libovolné aplikaci.

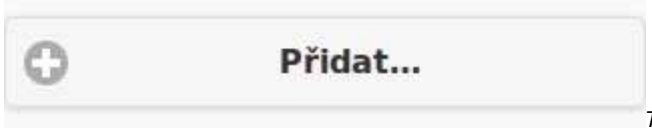
jQuery (38) – widgety v jQuery Mobile

Tlačítka

Tento druh widgetů důvěrně známe z jQuery UI. V mobilním frameworku jsou prakticky stejné, i když rozdíly se najdou. První je možnost deklarativního použití, s nímž už jste měli možnost se setkat v jednom z předchozích dílů. Další rozdíly se týkají způsobu zobrazení.

```
<a href="#" data-role="button" data-icon="plus" id="button-add">Přidat…</a>
```

Tohle už znáte. Je to deklarativně vytvořené tlačítko pro přidání něčeho.



Tlačítko v jQuery Mobile

V javascriptovém kódu je pak samozřejmě potřeba reagovat na stisk tlačítka.

```
$('#button-add').click(function() {
    …
});
```

Může to být ovšem ještě jednodušší, protože jQuery Mobile automaticky zpracovává elementy input s typem button, submit a reset. Tam ani není potřeba nastavovat atribut data-role. Na druhou stranu lze tlačítko vytvořit i imperativně z libovolného elementu:

```
$('#button').button();
```

Tlačítka nemusejí mít jen výchozí vlastnosti (plus věci typu ikon, viz výše), ale lze je i upravovat – a to opět jak deklarativně, tak imperativně.

```
<input type="button" value="Tlačítko" data-icon="star" data-corners="false" data-iconpos="right" data-mini="true">
```

Toto tlačítko (vytvořené z klasického HTML elementu) bude mít hranaté rohy, ikonu umístěnou vpravo a bude zmenšené (ve svislém směru). Uvedené parametry (samozřejmě bez data-) lze také předat ve formě položek objektu do metody button(), jak je to v takových případech obvyklé.

Modifikované tlačítko

Zaškrtávátka a přepínače

Zaškrtávátko (*checkbox*) a přepínač (*radio button*) jsou dvě podoby téhož widgetu Checkboxradio. Podívejme se nejdřív na první z podob. Opět lze postupovat deklarativně:

```
<label><input type="checkbox" name="permlogin" id="permlogin">trvalé přihlášení</label>
```

Zaškrtávací políčko v jQuery Mobile

Jak vidíte, oproti úplně obyčejnému checkboxu z HTML se to vůbec neliší. Podobně jako tlačítko, také zaškrtávátko má svoji zmenšenou verzi. Chcete-li checkbox vytvořit imperativně, je to také jednoduché, ale v podstatě není důvod to dělat. Zajímavější je seskupování prvků a přidání legendy:

```
<fieldset data-role="controlgroup">
    <legend>Zaškrtněte:</legend>
<input type="checkbox" name="permlogin" id="permlogin">
    <label for="permlogin">trvalé přihlášení</label>
<input type="checkbox" name="ipaddr" id="ipaddr">
    <label for="ipaddr">kontrolovat IP adresu</label>
</fieldset>
```

Legenda se zde standardně zobrazí nad zaškrtávátky. Seskupení má jen grafický význam, nikoli funkční. Příklad ukazuje i to, že pokud popisek není kontejnerem checkboxu, je potřeba použít atribut for.

Skupina zaškrtávátek

Přepínače se vytvářejí úplně stejně. Nemá valný smysl vytvářet jen jeden prvek, čili obvykle budeme pracovat přímo s celou skupinou tvořící přepínač:

```
<fieldset data-role="controlgroup">
    <legend>Zvolte barvu:</legend>
<input type="radio" name="color" value="red" id="red" checked="checked">
    <label for="red">červená</label>
<input type="radio" name="color" value="blue" id="blue">
    <label for="blue">modrá</label>
</fieldset>
```

Přepínač v jQuery Mobile

Standardní seskupení je vertikální, tedy zaškrtávátka nebo pole přepínače jsou nad sebou. To lze ale snadno změnit – horizontální seskupení je v mnoha případech velmi užitečné.

```
<fieldset data-role="controlgroup" data-type="horizontal">
...
</fieldset>
```

Horizontální uspořádání skupiny prvků

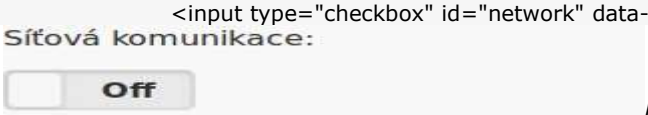
Události a jejich obsluha fungují stejně jako v klasickém jQuery UI.

Posuvné přepínače

Pro ovládání stavu typu ano/ne, resp. zapnuto/vypnuto (obecně zkrátka dva stavy) se s oblibou využívá posuvný přepínač (který je někdy názornější než zaškrtávátko nebo běžný přepínač). V jQuery Mobile je reprezentován widgetem Flipswitch.

Základem pro posuvný přepínač může být checkbox. Deklaruje se úplně normálně, jen se mu změní role:

```
<label for="network">Síťová komunikace:</label>
<input type="checkbox" id="network" data-role="flipswitch">
```



Posuvný přepínač v jQuery Mobile

Tento přepínač má stavy on/off. Potřebujeme-li nějaké jiné, je možnost je nastavit takto:

```
<input type="checkbox" id="speed" data-role="flipswitch" data-on-text="10" data-off-text="100">
```

Posuvný přepínač s jinými texty

Jako základ může posloužit také element select. To se hodí v případech, kdy potřebujeme nastavit nejen popisky stavů, ale i přímo odpovídající hodnoty.

```
<select id="wifi" data-role="flipswitch">
<option value="24g">2,4 GHz</option>
<option value="5g">5 GHz</option>
</select>
```

Více widgetů

Pojednání o widgetech není rozhodně u konce, pokračovat budeme příště a podíváme se například na panely nástrojů.