mistrovství v

Jonathan Chaffer
Karl Swedberg

jQuery

Kompletní průvodce vývojáře

- Stylování a animace
- Použití a vývoj zásuvných modulů
- Zpracování událostí a efektů
- Testování a ladění
- Referenční příručka

computer press

[PACKT] PUBLISHING

Jonathan Chaffer, Karl Swedberg

Mistrovství v jQuery

Computer Press
Brno
2013

Mistrovství v jQuery

Jonathan Chaffer, Karl Swedberg

Překlad: Kristýna Baše, Ondřej Baše
Obálka: IMIDEA
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4103-8

Vydalo nakladatelství Computer Press v Brně roku 2013 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 17 911.

1. vydání

ALBATROS MEDIA a.s.

Obsah

Předmluva

Velice mě těší, že Karl Swedberg a Jonathan Chaffer napsali právě tuto knihu. Jelikož se jedná o první knihu o knihovně jQuery, zavádí různé standardy, kterými se snaží řídit také jiné knihy věnované této knihovně a jazyku JavaScript jako celku. Od svého vydání se pravidelně zařazuje mezi nejprodávanější knihy o jazyku JavaScript; jistě za to může její vysoká kvalita a pozornost věnovaná detailům.

Rovněž jsem rád, že ji napsali právě Karl s Jonathanem, protože je oba dobře znám a vím, že se k tomu skvěle hodí. Jako člen vývojového týmu knihovny jQuery jsem měl v uplynulých letech vynikající příležitost poznat Karla, a zejména pak jeho touhu napsat svou vlastní knihu. Když nyní vidím výsledek, tak je mi jasné, že jeho dovednosti vývojáře a bývalého učitele angličtiny jsou výborným základem pro dokončení této úlohy.

Také jsem měl příležitost poznat oba autory osobně, což je ve světě distribuovaných projektů s otevřeným zdrojovým kódem spíše výjimkou, a rozhodně jsou nadále čestnými členy komunity okolo knihovny jQuery.

Knihovnu jQuery používá mnoho různých lidí z komunity okolo této knihovny. Tuto komunitu tvoří návrháři, vývojáři, lidé se zkušenostmi v programování, ale i lidé, kteří tyto zkušenosti nemají. Dokonce i náš vývojový tým knihovny jQuery se skládá ze stejných lidí, kteří vyjadřují své názory na tento projekt. Jedná se o společný znak všech uživatelů knihovny jQuery – jsme komunitou návrhářů a vývojářů, jejichž cílem je usnadnit vývoj aplikací v jazyce JavaScript.

Může se to zdát jako klišé, když řeknu, že projekt s otevřeným zdrojovým kódem těží ze své komunity, nebo že takový projekt pomáhá začít novým uživatelům. U knihovny jQuery se však nejedná jen o plané řeči, jelikož komunita je skutečně životně důležitá pro tento projekt. Ve vývojovém týmu knihovny jQuery máme dokonce více lidí starajících se o tuto komunitu, kteří píší dokumentaci a vyvíjejí zásuvné moduly, než těch, kteří udržují zdrojový kód v jejím jádru. Ačkoliv dobrý stav samotné knihovny je nesmírně důležitý, její komunita je právě tím rozdílem mezi průměrným projektem, jenž se zmítá v problémech, a projektem, který předčí veškerá vaše očekávání.

Způsob, jakým tento projekt funguje a jak používáme náš zdrojový kód, se podstatně liší od jiných projektů s otevřeným zdrojovým kódem, a také od většiny knihoven jazyka JavaScript. Projekt a komunita okolo knihovny jQuery je dobře informovaná; víme, proč knihovna jQuery přináší lepší požitek při programování v jazyce JavaScript a snažíme se tuto znalost předávat dalším uživatelům.

Abyste porozuměli komunitě okolo knihovny jQuery, nestačí, když si o ní přečtete, ve skutečnosti se do ní musíte aktivně zapojit. Doufám, že využijete této příležitosti a připojíte se k nám. Připojte se k našim diskuzním fórům, poštovním seznamům a blogům a umožněte nám, abychom vás důkladně seznámili s knihovnou jQuery.

Pro mě je knihovna jQuery mnohem více než jen blok zdrojového kódu. Jedná se o součet zkušeností posbíraných během několika let, které umožnily vznik této knihovny. Sleduji její výhody a nevýhody, snahu o její rozvoj a těší mě, když vidím, jak roste a stává se úspěšnou. I já se zlepšuji společně s jejími uživateli a bývalými členy týmu, snažím se je pochopit, přizpůsobit se, a tím se také něčemu novému přiučit.

Když jsem poprvé pročítal tuto knihu a zjišťoval jsem, že mluví o knihovně jQuery jako o jednotném nástroji, což bylo přesně něco jiného než to, s čím jsem se dosud setkával, byl jsem překvapený, ale současně také velmi nadšený. Mám radost, když vidím jiné vývojáře, jak se učí vyvíjet své projekty s pomocí knihovny jQuery.

Nejsem samozřejmě jediný, kdo má ke knihovně jQuery vztah, jenž se dosti odlišuje od běžného vztahu uživatele k nástroji. Nevím, jestli to zde budu umět dostatečně vyjádřit, ale představte si ten okamžik, kdy se obličej uživatele rozzáří v okamžiku, kdy zjistí, jak moc mu může knihovna jQuery ulehčit práci.

Každý uživatel knihovny jQuery v jednom okamžiku zjistí, že ten zdánlivě jednoduchý nástroj, co používal, je ve skutečnosti mnohem komplexnější a najednou zcela porozumí tomu, jak psát dynamické webové aplikace. To je prostě neuvěřitelné a je to moje nejoblíbenější část projektu knihovny jQuery.

Doufám, že někdy tuto zkušenost zažijete na vlastní kůži.

John Resig

tvůrce knihovny jQuery

O autorech

Jonathan Chaffer je zaměstnancem společnosti Rapid Development Group sídlící ve městě Grand Rapids ve státě Michigan, jež se zabývá webovým vývojem. V této společnosti dozoruje a implementuje projekty v nejrůznějších technologiích, a to zejména v jazycích PHP a JavaScript s využitím databázového stroje MySQL. Kromě toho pořádá výukové semináře o knihovně jQuery pro webové vývojáře.

V komunitě okolo projektů s otevřeným zdrojovým kódem se angažoval zejména v projektu redakčního systému Drupal, jenž přijal knihovnou jQuery za svůj hlavní framework pro jazyk JavaScript. Je tvůrcem oblíbeného modulu Content Construction Kit pro správu strukturovaného obsahu na webových stránkách poháněných systémem Drupal. Také se podílel na hlavních změnách v systému nabídek a vývojovém rozhraní API tohoto redakčního systému.

Jonathan žije ve městě Grand Rapids se svou ženou Jennifer.

Chtěl bych poděkovat své ženě Jennifer za její neutuchající nadšení a podporu. Dále Karlovi, protože mě motivoval v psaní knihy, když jsem na tom byl psychicky hůře, a také komunitě Ars Technica, že mě inspiruje ve zdokonalování mých technických dovedností. Navíc bych nechtěl zapomenout na Mika Henryho a tým Twisted Pixel, jelikož mi poskytl příjemné rozptýlení během psaní této knihy.

Karl Swedberg je webový vývojář, který pracuje ve společnosti Fusionary Media ve městě Grand Rapids (stát Michigan), kde tráví většinu svého času tím, že vytváří úžasné věci prostřednictvím jazyka JavaScript. Jako člen vývojového týmu knihovny jQuery zodpovídá za údržbu webových stránek jejího rozhraní API, které jsou k dispozici na internetové adrese *http://api.jquery.com/*. Rovněž vydává návody na svém blogu na adrese *http://www.learningjquery.com/* a přednáší na pracovních seminářích a konferencích. Když zrovna neprogramuje, rád tráví čas se svou rodinou, vaří si kávu ve své garáži nebo cvičí v místní posilovně.

Chci poděkovat své ženě Sáře a svým dvěma dětem, Benjaminovi a Lucii, za štěstí, o které obohatili můj život. Také děkuji Jonathanu Chafferovi za trpělivost a ochotu napsat se mnou tuto knihu.

Mnohé díky patří Johnu Resigovi, protože vytvořil nejlepší knihovnu JavaScriptu na světě, a také všem ostatním, kteří přispěli k tomuto projektu svým kódem, časem a zkušenostmi. Děkuji rovněž nakladatelství Packt Publishing, odborným korektorům a spoustě dalších lidí, kteří mě inspirovali a pomáhali mi na mé cestě.

O odborných korektorech

Kaiser Ahmed je profesionální webový vývojář. Svůj bakalářský titul získal na univerzitě Khulna University of Engineering and Technology (KUET). Je také spoluzakladatelem společnosti CyberXpress.Net se sídlem v Bangladéši.

Ovládá širokou škálu technických dovedností, zná Internet a má zkušenosti s webovým vývojem pro řadu klientů. Rád vytváří architekturu a infrastrukturu webových stránek, vyvíjí administrační rozhraní pomocí nástrojů s otevřeným zdrojovým kódem (PHP, MySQL, Apache, Linux apod.) a vytváří prezentační část pomocí jazyků CSS a HTML/XHTML.

Rád bych poděkoval své milující ženě Marii Akter za její podporu.

Kevin Boudloche je webový vývojář pocházející ze státu Mississippi. Webové stránky nejprve vytvářel osm let ve svém volném čase a nyní je vytváří už tři roky profesionálně. Zaměřuje se na vývoj webových aplikací, a to zejména na jejich prezentační část.

Carlos Estebes je zakladatelem softwarové společnosti Ehxioz (*http://ehxioz.com/*) sídlící ve městě Los Angeles, která se specializuje na vývoj moderních webových aplikací, k čemuž používá nejmodernější technologie a metodiky. Má téměř deset let zkušeností s webovým vývojem a je držitelem bakalářského titulu v oboru počítačových věd z univerzity California State University ve městě Los Angeles.

Úvod

V roce 2005 se John Resig inspiroval průkopníky jazyka JavaScript Deanem Edwardsem a Simonem Willisonem a poskládal dohromady knihovnu funkcí, které by mu umožnily jednoduše vyhledat elementy ve webové stránce a přidělit jim chování. Poprvé informoval veřejně o svém projektu v lednu roku 2006, a to už k této knihovně připojil také funkce pro manipulaci s modelem DOM a základní animace. Pojmenoval ji jQuery, aby zdůraznil, že její hlavní rolí je vyhledávání elementů (neboli dotazování se na elementy, jelikož slovo **query** lze přeložit jako **dotaz**) ve stránce, a umožnit tak pracovat s nimi v kódu jazyka JavaScript. V následujících několika letech se pak knihovna jQuery rozrostla o další funkce, vylepšila svou efektivitu a začala se objevovat na řadě nejoblíbenějších webových stránek na Internetu. Přestože John Resig je stále vedoucím vývojářem knihovny jQuery, tento projekt se opravdu zcela transformoval na projekt s otevřeným zdrojovým kódem, takže za knihovnou jQuery nyní stojí spousta špičkových vývojářů, a také rozsáhlá komunita složená z tisíců dalších vývojářů.

Knihovna jQuery může vylepšit vaše webové stránky, ať už je vaše současná situace jakákoliv. V jediném souboru nabízí ohromné množství funkcí, snadno zvládnutelnou syntaxi a robustní kompatibilitu mezi platformami. Navíc však vznikly tisíce zásuvných modulů, díky čemuž se knihovna jQuery stala základním nástrojem pro téměř jakýkoliv požadavek při programování na straně klienta.

Třetí vydání této knihy vás lehce uvede do koncepcí knihovny jQuery, abyste mohli do svých stránek přidávat interakce a animace, přestože jste třeba měli při dřívějších pokusech s psaním kódu v jazyce JavaScript potíže. Dále vás provede úskalími spojenými s používáním technologie Ajax, událostí, efektů a pokročilých funkcí jazyka JavaScript. Stejně tak získáte referenční přehled knihovny jQuery, ke kterému se budete moct kdykoliv vracet.

Čím se tato kniha zabývá

V kapitole 1, „Začínáme," poprvé nakousneme knihovnu jQuery. Popíšeme si tuto knihovnu a řekneme si, k čemu nám může být užitečná. Posléze si vysvětlíme, jak ji stáhnout a začlenit do našeho projektu, a také, jak napsat náš první skript.

V kapitole 2, „Vybíráme elementy," se dozvíme, jak vyhledávat elementy na stránce pomocí selektorů a metod pro procházení modelu DOM knihovny jQuery. Zjistíme, jak pomocí této knihovny měnit vzhled různých skupin elementů, a to někdy způsobem, jaký nezvládá samotný jazyk CSS.

V kapitole 3, „Obsluha událostí," použijeme mechanismus knihovny jQuery pro obsluhu událostí k tomu, abychom provedli určité akce, když nastanou jisté události v prohlížeči. Uvidíme, jak tato knihovna ulehčuje připojování událostí k elementům, a to dokonce i před načtením dané stránky.

V kapitole 4, „Styly a animace," se seznámíme s animačními technikami knihovny jQuery a naučíme se skrývat, zobrazovat a přesouvat elementy s pomocí efektů, které jsou užitečné, ale také příjemné na pohled.

V kapitole 5, „Manipulace s modelem DOM," se dozvíme, jak programově měnit naši stránku. V této kapitole si popíšeme, jak dynamicky měnit strukturu dokumentu HTML, a to včetně jeho obsahu.

V kapitole 6, „Odesíláme data technologií Ajax," objevíme řadu způsobů, jak přistupovat k obsahu na straně serveru, aniž bychom museli obnovovat stránku. Až budeme znát základní komponenty knihovny jQuery, můžeme začít tuto knihovnu rozšiřovat, aby splnila naše požadavky.

V kapitole 7, „Používáme zásuvné moduly," si ukážeme, jak vyhledávat, instalovat a používat zásuvné moduly, a to včetně mocné knihovny jQuery UI obsahující spoustu zásuvných modulů.

V kapitole 8, „Vyvíjíme zásuvné moduly," se naučíme využívat skvělé rozšiřitelnosti knihovny jQuery a vytvoříme vlastní zásuvné moduly od začátku. V této kapitole vytvoříme vlastní pomocné funkce, metody objektů knihovny jQuery a objevíme vynikající nástroj Widget Factory z knihovny jQuery UI.

V kapitole 9, „Pokročilé selektory a procházení," oprášíme své znalosti selektorů a procházení, díky čemuž budeme moct optimalizovat selektory, pracovat se zásobníkem elementů modelu DOM a psát vlastní zásuvné moduly, které budou rozšiřovat schopnosti výběru a procházení.

V kapitole 10, „Pokročilé události," pronikneme hlouběji do technik, jako jsou delegování a přiškrcování událostí, které můžou výrazně vylepšit efektivitu zpracování událostí. Rovněž vytvoříme vlastní události, s nimiž opět rozšíříme dovednosti knihovny jQuery.

V kapitole 11, „Pokročilé efekty," doladíme vizuální efekty, které nabízí knihovna jQuery, takovým způsobem, že vytvoříme vlastní funkce průběhu a nastavíme jednotlivé kroky animace. Budeme moct ovlivňovat průběh animací a plánovat akce pomocí vlastních front.

V kapitole 12, „Pokročilá manipulace s modelem DOM," si prakticky vyzkoušíme úpravu modelu DOM s použitím techniky připojování libovolných dat k elementům. Také se dozvíme, jak rozšířit postup, jímž knihovna jQuery zpracovává vlastnosti jazyka CSS na elementech.

V kapitole 13, „Pokročilá práce s technologií Ajax," lépe pochopíme ajaxové transakce, a to včetně systému odložených objektů pro zpracování dat, která můžou být potřebná později.

V příloze A, „Uzávěry v jazyce JavaScript," se dozvíme, co to jsou uzávěry v jazyce JavaScript a jak je používat ke svému prospěchu.

V příloze B, „Testování kódu jazyka JavaScript pomocí knihovny QUnit," si ukážeme knihovnu QUnit určenou pro jednotkové testování skriptů napsaných v jazyce JavaScript. Tuto knihovnu připojíme k naší sadě nástrojů pro vývoj a údržbu sofistikovaných webových aplikací.

V příloze C, „Rychlá referenční příručka," najdete přehled celé knihovny jQuery, a to včetně všech jejích metod a selektorů. Její skvělý formát oceníte v těch situacích, v nichž sice víte, co máte udělat, ale nejste si jistí, jestli jste vybrali správnou metodu nebo selektor.

Co budete při čtení této knihy potřebovat

Abyste mohli spustit ukázkové zdrojové kódy z této knihy, potřebujete moderní webový prohlížeč, jakým je kupříkladu prohlížeč Firefox od společnosti Mozilla, prohlížeč Safari od společnosti Apple, prohlížeč Google Chrome od společnosti Google nebo prohlížeč Internet Explorer od společnosti Microsoft.

Pokud budete chtít s příklady experimentovat a dělat cvičení z konce kapitol, neobejdete se také bez následujících nástrojů:

- základního textového editoru,
- vývojového nástroje pro prohlížeč; například nástroje Firebug (popsaného v kapitole 1, „Začínáme," v části „Vývojové nástroje"),
- balíčku zdrojových kódů pro jednotlivé kapitoly, jenž obsahuje rovněž kopii knihovny jQuery (jak je patrné v níže uvedené části „Zdrojové kódy ke knize").

Kromě toho pro spuštění některých příkladů technologie Ajax z kapitoly 6, „Odesíláme data technologií Ajax," budete potřebovat webový server s podporou jazyka PHP.

Pro koho je tato kniha určena

Tato kniha je určena pro všechny webové návrháře, kteří chtějí vytvářet interaktivní elementy pro své návrhy, a také pro vývojáře, kteří by chtěli vytvářet nejlepší uživatelská rozhraní pro své webové aplikace. Základní znalost programování v jazyce JavaScript je nezbytná. Měli byste také znát základy jazyků HTML a CSS a orientovat se v kódu jazyka JavaScript. O knihovně jQuery nemusíte vědět vůbec nic, ani o žádné jiné knihovně JavaScriptu.

Při čtení této knihy se seznámíte s funkcemi a syntaxí knihovny jQuery 1.9.x, což byla nejnovější verze v době psaní této knihy.

Historie projektu knihovny jQuery

V této knize si popíšeme funkce a syntaxi knihovny jQuery 1.9.x, jakožto její poslední verze v době psaní této knihy. Její základní cíl, umožnit jednoduše vyhledávat elementy na stránce a pracovat s nimi, se však během jejího vývoje nezměnil; změnily se jen určité syntaktické detaily a funkce. V následujícím přehledu historie této knihovny si vyznačíme nejdůležitější změny mezi jednotlivými verzemi, které zajisté ocení čtenáři pracující s jejími staršími verzemi:

- **Fáze veřejného vývojového procesu**: John Resig se v srpnu roku 2005 poprvé zmínil, že vylepšil knihovnu Behavior. Svou novou knihovnu vydal oficiálně 14. ledna 2006 pod názvem jQuery.

- **jQuery 1.0** (srpen 2006): První stabilní verze této knihovny již obsahovala robustní podporu selektorů jazyka CSS, obsluhu událostí a ajaxové interakce.
- **jQuery 1.1** (leden 2007): Tato verze podstatně zjednodušila rozhraní API. Spousta sporadicky používaných metod se sloučila, díky čemuž se zmenšil počet metod pro učení a dokumentaci.
- **jQuery 1.1.3** (červenec 2007): Tato minoritní verze zavedla výrazné zlepšení rychlosti selektorového jádra. Od této verze můžeme porovnávat efektivitu knihovny jQuery s ostatními knihovnami JavaScriptu, jako jsou Prototype, Mootools a Dojo, a to s příznivými výsledky.
- **jQuery 1.2** (září 2007): Tato verze odstranila syntaxi jazyka XPath pro výběr elementů, jelikož její účel se shodoval s účelem syntaxe jazyka CSS. V této verzi se rovněž objevila možnost flexibilnějšího přizpůsobování efektů a zjednodušil se vývoj zásuvných modulů pomocí událostí rozčleněných do jmenných prostorů.
- **jQuery UI** (září 2007): Veřejnost se dozvěděla, že vzniká tato nová sada zásuvných modulů, která by měla nahradit oblíbený, ale zastarávající zásuvný modul Interface. Její součástí je bohatá kolekce hotových nástrojů, a také nástroje pro stavbu sofistikovaných komponent, jako je například rozhraní s možností přesouvání elementů.
- **jQuery 1.2.6** (květen 2008): Funkčnost oblíbeného zásuvného modulu Dimensions od Brandona Aarona se přesunula do jádra této knihovny.
- **jQuery 1.3** (leden 2009): Důkladná revize selektorového jádra Sizzle přinesla obrovský nárůst výkonu knihovny jQuery. Tato knihovna začala oficiálně podporovat **delegování událostí**.
- **jQuery 1.4** (leden 2010): Tato pravděpodobně nejambicióznější verze od verze 1.0 vylepšila efektivitu manipulace s modelem DOM a přidala velké množství nových nebo zdokonalených metod pro téměř všechny aspekty knihovny jQuery. Vydání verze 1.4 doprovázelo 14 propagačních dní plných novinek a videí na internetové adrese *http://jquery14.com/*.
- **jQuery 1.4.2** (únor 2010): Zavedla dvě nové metody určené pro delegování událostí – metody `delegate()` a `undelegate()`. Celý systém událostí knihovny jQuery prodělal důkladnou revizi, aby byl flexibilnější a kompatibilnější napříč různými webovými prohlížeči.
- **jQuery Mobile** (srpen 2010): Tým projektu knihovny jQuery prozradil svou strategii, výsledky výzkumů a záměr vytvářet webová uživatelská rozhraní pro mobilní zařízení s použitím knihovny jQuery a nového frameworku pro mobilní zařízení, který lze najít na internetové adrese *http://jquerymobile.com/*.
- **jQuery 1.5** (leden 2011): Komponenta pro práci s technologií Ajax prodělala výraznou změnu, dále získala lepší rozšiřitelnost a efektivitu. Tato verze rovněž implementovala koncepci **slibů** (**promises**) pro zpracování front synchronních i asynchronních funkcí.
- **jQuery 1.6** (květen 2011): Tato verze obsahuje přepsanou komponentu pro práci s atributy, která lépe reflektuje rozdíly mezi atributy jazyka HTML a vlastnostmi

modelu DOM. Odložený objekt, jenž byl poprvé představen ve verzi 1.5, získal dvě nové metody – `always()` a `pipe()`.

- **jQuery 1.7** (listopad 2011): Nejvýznamnější změnou této verze jsou nové metody `on()` a `off()`, které sjednocují a zefektivňují práci s událostmi v knihovně jQuery.
- **jQuery 1.8** (srpen 2012): Tato verze má rychlejší selektorové jádro a rovněž se dočkala nových funkcí pro animace.
- **jQuery 1.9** (leden 2013): Podporuje některé nové selektory jazyka CSS3 i ve starších webových prohlížečích, opravuje chyby a odstraňuje zastaralé funkce. Vyčištěním svého rozhraní API se chystá na přechod na verzi 2.0, která by, dle vyjádření autorů, měla přijít o podporu starších verzí prohlížeče Internet Explorer (až do verze 8; bude tedy podporovat prohlížeč Internet Explorer 9 a novější). Vzhledem k odstranění některých funkcí je rovněž k dispozici migrační nástroj pro snadnější přechod na novější verzi.

Poznámky ke starším verzím knihovny jQuery najdete na webových stránkách projektu knihovny jQuery na adrese *http://jquery.org/history/*.

Konvence

V této knize se setkáte s mnoha styly textu, které rozlišují různé typy informací. V této části kapitoly najdete některé z nich včetně popisu jejich významu.

Zdrojový kód v textu vypadá následovně: „Tento kód ukazuje, že metodě `console.log()` můžeme předat jakýkoliv výraz."

Blok zdrojového kódu se zobrazuje takto:

```
$('button.show-details').click(function() {
  $('div.details').show();
});
```

Pokud bude nutné upoutat vaši pozornost na určitou část bloku s kódem, tato část bude zvýrazněná tučně:

```
$('#switcher-narrow').on('click', function() {
  $('body').removeClass().addClass('narrow');
});
```

Nové termíny a **důležitá slova** se zobrazují také tučným písmem. Slova, která vidíte kupříkladu na obrazovce, v nabídkách nebo dialogových oknech (prvky uživatelského rozhraní) vypadají následovně: „Kartu **Konzole** budeme při učení se knihovny jQuery používat nejčastěji, jak je patrné na následujícím obrázku."

POZNÁMKA

Varování a důležité poznámky budou vypadat takto.

TIP

Tipy a triky budou vypadat pro změnu takto.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/K2082* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2082* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA

1

Začínáme

Témata kapitoly:

- Co knihovna jQuery dělá
- Proč knihovna jQuery funguje tak skvěle
- Naše první webová stránka poháněná knihovnou jQuery
- Srovnání běžného kódu JavaScriptu s kódem používajícím knihovnu jQuery
- Vývojové nástroje
- Doplněk Firebug

Dnešní web je dynamické prostředí, jehož uživatelé kladou vysoké nároky jak na vzhled, tak na funkce webových stránek. Aby mohli vývojáři vytvářet zajímavé, interaktivní webové stránky, obracejí se na knihovny JavaScriptu, jakou je například knihovna jQuery, aby jim pomohly s běžnými úkoly a zjednodušily ty složitější. Jedním z důvodů, proč je knihovna jQuery oblíbená, je její schopnost pomáhat se širokou škálou úkolů.

Může se zdát, že je jistě těžké začít, jelikož knihovna jQuery vykonává tolik různých funkcí. Návrh této knihovny je však velmi konzistentní; navíc si spoustu koncepcí půjčuje z jazyků HTML a CSS. Návrh této knihovny se snaží o to, aby s ní mohli začít pracovat návrháři s malou zkušeností v programování, protože ti většinou znají právě jazyky HTML a CSS, ale už ne jazyk JavaScript. V této úvodní kapitole napíšeme funkční program na pouhé tři řádky kódu. Pokročilí programátoři na druhou stranu zajisté ocení koncepční jednotnost, jak si ukážeme v pozdějších kapitolách.

Podívejme se, co pro nás může knihovna jQuery udělat.

Co knihovna jQuery dělá

Knihovna jQuery poskytuje víceúčelovou abstraktní vrstvu pro běžné webové skriptování, a je tudíž užitečná skoro ve všech situacích, v nichž potřebujeme skriptovat. Kvůli její rozšiřitelné povaze není možné popsat všechny její možné způsoby použití v jediné knize, protože neustále vznikají nové zásuvné moduly, které ji obohacují o další funkce. Její základní funkce pomáhají s prováděním následujících úloh:

- Přistupovat k elementům dokumentu. Bez knihovny JavaScriptu musí weboví vývojáři často psát spoustu řádků kódu, aby prošli strom **modelu DOM** (Document Object Model) a vyhledali v něm určité části dokumentu HTML. S knihovnou jQuery mají vývojáři k dispozici robustní a efektivní systém selek-

torů, s nímž snadno získají určité části dokumentu, aniž by museli zkoumat, nebo dokonce manipulovat s modelem DOM.

```
$('div.content').find('p')
```

- Upravovat vzhled webové stránky. Jazyk CSS přináší výborný způsob změny vzhledu dokumentu, ale selhává v tom, že ne všechny webové prohlížeče podporují standardy stejně. S knihovnou jQuery můžou vývojáři překlenout tento nedostatek a spoléhat se na stejné standardy napříč rozdílnými webovými prohlížeči. Knihovna jQuery navíc umí měnit třídy a jednotlivé vlastnosti aplikované na různé části dokumentu, a to dokonce i po zobrazení stránky.

```
$('ul > li:first').addClass('active');
```

- Měnit obsah dokumentu. Knihovna jQuery se neomezuje jen na kosmetické změny dokumentu, ale může měnit také jeho obsah. Je možné měnit text, vkládat nebo zaměňovat obrázky, znovu seřazovat seznamy, nebo přepisovat či rozšiřovat celou strukturu dokumentu HTML, a to vše prostřednictvím přehledného **rozhraní API** (Application Programming Interface).

```
$('#container').append('<a href="more.html">více</a>');
```

- Reagovat na akce uživatele. Dokonce i ty nejdokonalejší funkce jsou nám k ničemu, pokud nemůžeme řídit, kdy je chceme spouštět. Knihovna jQuery nabízí elegantní způsob zachytávání událostí, jakou je kupříkladu klepnutí na odkaz, aniž bychom museli „znečišťovat“ náš kód dokumentu HTML obsluhujícími funkcemi. Tato knihovna rovněž odstraňuje rozdíly v rozhraní API pro obsluhu událostí mezi různými webovými prohlížeči.

```
$('button.show-details').click(function() {
  $('div.details').show();
});
```

- Animovat změny v dokumentu. Jestliže chceme zavést takové interaktivní chování, musíme uživatelům poskytnout také vizuální odezvu. Knihovna jQuery nabízí sadu efektů (například prosvítání nebo posouvání), a také nástroje pro tvorbu nových vizuálních efektů.

```
$('div.details').slideDown();
```

- Načítat data ze serveru bez nutnosti obnovení stránky. Tento úkol můžeme provést pomocí technologie Ajax, jejíž název původně vyjadřoval zkratku pro **asynchronní JavaScript a XML** (Asynchronous JavaScript and XML), ale postupně se rozrostla a nyní reprezentuje daleko větší skupinu technologií pro komunikaci mezi klientem a serverem. Knihovna jQuery odstraňuje složitost související se specifickými odchylkami jednotlivých prohlížečů, díky čemuž se vývojáři můžou soustředit více na funkčnost serveru.

```
$('div.details').load('more.html #content');
```

- Zjednodušuje běžné činnosti při programování v jazyce JavaScript. Knihovna jQuery totiž nepřidává pouze nové funkce, ale také rozšiřuje základní konstrukce jazyka JavaScript, jakou je kupříkladu procházení a manipulace s poli.

```
$.each(obj, function(key, value) {
  total += value;
});
```

TIP

Z adresy *http://knihy.cpress.cz/K2082* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Proč knihovna jQuery funguje tak skvěle

Jak sílí zájem lidí o dynamické webové stránky, rozmáhají se rovněž frameworky pro jazyk JavaScript. Některé z nich se specializují pouze na jednu nebo dvě z výše uvedených úloh. Jiné se zase snaží pokrýt všechny myslitelné funkce a animace a naservírovat je uživatelům v jediném balíčku. Aby knihovna jQuery mohla uspokojit všechny tyto požadavky a současně zůstala poměrně kompaktní, používá následující strategie:

- Využívá znalosti jazyka CSS. Jelikož staví na **selektorech** jazyka CSS, získává jednoduchou, avšak velmi užitečnou možnost vyjádřit strukturu dokumentu. Knihovna jQuery se proto stala vstupní bránou do světa profesionálního webového vývoje pro návrháře, kteří chtějí doplnit reakce do svých stránek, jelikož k tomu potřebují pouze znalost syntaxe jazyka CSS.
- Je rozšiřitelná. Knihovna jQuery zabraňuje svému zahlcení funkcemi tak, že umožňuje tvorbu **zásuvných modulů**. Nové zásuvné moduly lze vytvářet jednoduše a k této činnosti existuje výborná dokumentace. Tato vlastnost odstartovala vývoj celé řady užitečných modulů. Většina funkcí v základním balíku knihovny jQuery je také implementována s použitím architektury zásuvných modulů, takže případně je lze z knihovny odstranit a získat ještě menší knihovnu.
- Abstrahuje výstřelky webových prohlížečů. Ve světě webového vývoje se bohužel setkáváme s tím, že každý prohlížeč má nějakou svou skupinu odchylek, jimiž se liší od vydaných standardů. Kvůli tomu musíme věnovat podstatnou část své webové aplikace tomu, abychom používali funkce jinými způsoby na různých platformách. Ačkoliv napříč webovými prohlížeči není možné dosáhnout stoprocentní shody, knihovna jQuery přináší **abstraktní vrstvu**, jež sjednocuje běžné úkoly, a tím zmenšuje velikost zdrojového kódu a zjednodušuje ho.
- Vždy pracuje se skupinami. Když řekneme knihovně jQuery: „Vyhledej mi všechny elementy s třídou `collapsible` a skryj je," nemusíme procházet jednotlivé elementy

v cyklu. Místo toho zavoláme metodu `hide()`, která automaticky zpracovává celou skupinu objektů. Tato technika se nazývá **implicitní iterace** a zaručuje, že nemusíme používat cykly příliš často, což také zmenšuje velikost zdrojového kódu.

- Umožňuje provádět více akcí na jediném řádku. Díky tomu se vyhneme přehlcení dočasnými proměnnými a zbytečnému opakování, jelikož knihovna jQuery aplikuje na většinu svých metod techniku **řetězení**. Funguje to tak, že volaná metoda objektu vrací samotný objekt, takže s ním můžeme hned provést další akci.

Tyto strategie přispívají k poměrně malé velikosti knihovny jQuery (přibližně 90 KB v komprimovaném stavu), a k tomu ještě poskytují techniky, s nimiž udržíme velikost našeho vlastního zdrojového kódu na uzdě.

Elegance knihovny jQuery pochází z části z jejího návrhu a z části z postupného vývojového procesu, který ve své rozrůstající se komunitě neustále podstupuje. Uživatelé této knihovny diskutují nejen o vývoji zásuvných modulů, ale také o vylepšeních samotného jádra knihovny. Uživatelé a vývojáři rovněž pomáhají zdokonalovat její oficiální dokumentaci, která se nachází na internetové adrese *http://api.jquery.com/*.

Navzdory obrovskému množství práce, které stojí za takovým flexibilním a robustním systémem, je konečný produkt volně k dispozici. Knihovnu jQuery můžeme používat, pokud souhlasíme s její licencí MIT, která umožňuje používat tuto knihovnu na libovolné webové stránce a ve vlastních komerčních aplikacích (dříve jsme si mohli vybrat také licenci GNU GPL, ale v průběhu roku 2012 ji autoři odstranili).

Naše první webová stránka poháněná knihovnou jQuery

Protože už víme, s čím nám může knihovna jQuery pomoci, můžeme si popsat, jak ji uvést do provozu. K tomuto účelu potřebujeme kopii knihovny jQuery.

Stažení knihovny jQuery

Knihovnu jQuery nemusíme instalovat, ale vystačíme si s veřejně dostupnou kopií jejího souboru, a to buď na našem vlastním serveru, nebo na externím serveru. Jelikož jazyk JavaScript se řadí mezi interpretované jazyky, nemusíme naši aplikaci kompilovat, ani sestavovat. Jestliže chceme k naší webové stránce připojit knihovnu jQuery, jednoduše se odkážeme na umístění jejího souboru z elementu `script` uvnitř daného dokumentu HTML.

Na oficiálních webových stránkách knihovny jQuery (*http://jquery.com/*) vždy najdeme její nejaktuálnější stabilní verzi, kterou si můžeme stáhnout přímo z domovské stránky. Možná si budeme muset vybírat z více verzí, ale nás jako webové vývojáře zajímá poslední nekomprimovaná verze. V produkčním prostředí bychom ji měli nahradit komprimovanou verzí.

Jak obliba knihovny jQuery roste, některé společnosti poskytují její soubor volně prostřednictvím svých sítí CDN (Content Delivery Network). Zejména společnosti Google (*https://developers.google.com/speed/libraries/devguide?hl=cs*) a Microsoft (*http://www.asp.net/ajaxlibrary/cdn.ashx*) nabízejí tento soubor na výkonných serverech s nízkým zpož-

děním, které jsou distribuované po celém světě, aby měli uživatelé rychlý přístup k datům po celém světě. Přestože kopie knihovny jQuery na síti CDN může urychlit stahování pro koncové uživatele (díky distribuovaným serverům a mezipaměti), při vývoji je pravděpodobně lepší používat její lokální kopii. V této knize budeme používat soubor této knihovny, jejž si uložíme do lokálního systému souborů, abychom mohli spouštět náš kód, ať už jsme připojení k Internetu, nebo nejsme.

Vkládáme knihovnu jQuery do dokumentu HTML

Většina příkladů v této knize se skládá ze tří částí:

1. dokumentu HTML,
2. šablony kaskádových stylů,
3. skriptu jazyka JavaScript.

V našem prvním příkladu napíšeme stránku s výňatkem z knihy, jehož části označíme řadou tříd. V této stránce se odkážeme na poslední verzi knihovny jQuery, kterou jsme si stáhli, přejmenovali ji na *jquery.js* a uložili do lokální složky našeho projektu:

```
<!DOCTYPE html>

<html lang="cs">
<head>
  <meta charset="utf-8">
  <title>Povídky z jedné kapsy</title>

  <link rel="stylesheet" href="01.css" type="text/css" />

  <script src="jquery.js"></script>
  <script src="01.js"></script>
</head>
<body>
  <h1>Povídky z jedné kapsy</h1>
  <div class="author">Karel Čapek</div>

  <div class="chapter" id="chapter-12">
    <h2 class="chapter-title">12. Básník</h2>
    <p><span class="spoken">"Ukažte mi to,"</span> řekl dr. Mejzlík
      shovívavě.</p>
    <p><span class="spoken">"To nic není,"</span> bránil se básník.
      <span class="spoken">"Ale jestli chcete, já vám to přečtu."</span>
      Načež vykuliv nadšeně oči a zpěvavě protahuje dlouhé slabiky
      recitoval:</p>
    <div class="poem">
      <div>"marš tmavých domů ráz dva zastavit stát</div>
      <div>úsvit na mandolínu hrá</div>
```

```
      <div>proč dívko proč se červenáš</div>
      <div>pojedem vozem 120 HP na konec světa</div>
      <div>nebo do Singapore"</div>
    </div>
    <p>a po kratší odmlce pokračoval:</p>
    <div class="poem">
      <div>"zastavte zastavte vůz letí</div>
      <div>naše veliká láska v prachu leží</div>
      <div>dívka zlomený květ</div>
      <div>labutí šíje ňadra buben a činely</div>
      <div>proč tolik pláču"</div>
    </div>
    <p><span class="spoken">"A to je celé,"</span> prohlásil Jaroslav
      Nerad.</p>
  </div>
</body>
</html>
```

TIP

Na uspořádání souborů na serveru příliš nezáleží. Pokud se ale odkazujeme z jednoho souboru na jiný soubor, musíme se řídit aktuální strukturou souborů. Ve většině příkladů z této knihy budeme používat relativní cesty k odkazovaným souborům (*../images/foo.png*) častěji než absolutní cesty (*/images/foo.png*). Díky tomu budeme moct spouštět zdrojový kód lokálně bez webového serveru.

Pomocí běžného kódu jazyka HTML načítáme naši šablonu stylů. U tohoto příkladu použijeme níže uvedenou šablonu stylů:

```
body {
  background-color: #fff;
  color: #000;
  font-family: Helvetica, Arial, sans-serif;
}

h1, h2 {
  margin-bottom: .2em;
}

.poem {
  margin: 0 2em;
}

.highlight {
  background-color: #ccc;
  border: 1px solid #888;
```

```
    font-style: italic;
    margin: 0.5em 0;
    padding: 0.5em;
}
```

Po šabloně stylů načítáme skripty jazyka JavaScript. Na skript knihovny jQuery se musíme odkazovat před vlastními skripty, protože v opačném případě bychom nemohli v našem zdrojovém kódu používat tuto knihovnu.

POZNÁMKA

Ve zbytku této knihy narazíte pouze na důležité části souborů jazyka HTML a CSS. Kompletní zdrojové kódy si můžete stáhnout z adresy *http://knihy.cpress.cz/K2082* po klepnutí na odkaz Soubory ke stažení.

Nyní máme stránku, která vypadá podobně jako na následujícím obrázku:

Povídky z jedné kapsy

Karel Čapek

12. Básník

"Ukažte mi to," řekl dr. Mejzlík shovívavě.

"To nic není," bránil se básník. "Ale jestli chcete, já vám to přečtu." Načež vykuliv nadšeně oči a zpěvavě protahuje dlouhé slabiky recitoval:

"marš tmavých domů ráz dva zastavit stát
úsvit na mandolínu hrá
proč dívko proč se červenáš
pojedem vozem 120 HP na konec světa
nebo do Singapore"

a po kratší odmlce pokračoval:

"zastavte zastavte vůz letí
naše veliká láska v prachu leží
dívka zlomený květ
labutí šíje ňadra buben a činely
proč tolik pláču"

"A to je celé," prohlásil Jaroslav Nerad.

Pomocí knihovny jQuery aplikujeme nové pravidlo stylu na text básně.

POZNÁMKA

Tento příklad pouze demonstruje jednoduchost knihovny jQuery. Ve skutečné webové stránce bychom stejného výsledku dosáhli jen s kódem jazyka CSS.

Přidáváme kód knihovny jQuery

Ve výše uvedeném dokumentu HTML vkládáme prozatím prázdný skript *01.js*, a to pomocí elementu `<script src="01.js"></script>`. V našem příkladu si vystačíme se třemi řádky kódu:

```
$(document).ready(function() {
  $('div.poem').addClass('highlight');
});
```

Vyhledání básně

Základním cílem knihovny jQuery je usnadnit vyhledávání částí dokumentu. To provádíme voláním funkce `$()`. Této funkci předáváme obvykle textový řetězec, který může obsahovat libovolný selektor jazyka CSS. V tomto případě chceme vyhledat všechny elementy `div` s třídou `poem`, takže volíme jednoduchý selektor `div.poem`. V průběhu této knihy budeme však používat i složitější selektory. Vyhledávání částí dokumentu si popíšeme podrobněji v kapitole 2, „Vybíráme elementy."

Funkce `$()` vrací novou instanci objektu knihovny jQuery, což je základní stavební prvek, s nímž budeme od této chvíle pracovat. Tento objekt obaluje žádný nebo více elementů modelu DOM a umožňuje nám přistupovat k nim nejrůznějšími způsoby. Tentokrát potřebujeme změnit vzhled vyhledaných částí stránky, a toho dosáhneme změnou tříd aplikovaných na text básně.

Přidáváme novou třídu

Název metody `addClass()` je sebepopisný (podobně jako názvy většiny metod knihovny jQuery) – tato metoda přiřazuje danou třídu vybraným částem stránky. Jejím jediným parametrem je název třídy, kterou přidáváme. Díky této metodě a jí opačné metodě `removeClass()` můžeme snadno nahlédnout do knihovny jQuery, až budeme zkoumat různé selektory. Prozatím přidáváme třídu `highlight`, přičemž v naší šabloně stylů jsme pro příslušné pravidlo stylu definovali kurzívu, šedou barvu pozadí a rámeček.

Povšimněte si, že nemusíte používat cyklus, když chcete přidat třídu všem básním. Jak už víte, knihovna jQuery používá techniku **implicitní iterace** u metod, jakou je například metoda `addClass()`, proto si vystačíte s jediným voláním, a tím změníte všechny vybrané části dokumentu.

Spuštění našeho zdrojového kódu

Jestliže spojíme volání funkcí `$()` a `addClass()`, mělo by to stačit, abychom dosáhli změny vzhledu básně. Kdybychom však tento jediný řádek kódu vložili do záhlaví našeho dokumentu, nic by se nestalo. Webový prohlížeč totiž spouští kód jazyka JavaScript, jakmile na něj narazí při parsování dokumentu HTML. Když zpracovává záhlaví tohoto dokumentu, žádné části těla dokumentu, jejichž vzhled bychom chtěli měnit, ještě nejsou načtené. Musíme tedy odsunout spuštění našeho kódu do okamžiku, kdy bude dostupný model DOM.

Knihovna jQuery nám umožňuje naplánovat volání funkcí po načtení celého modelu DOM prostřednictvím konstrukce `$(document).ready()` – aniž bychom museli čekat na načtení

všech obrázků. Ačkoliv úkoly lze plánovat i bez pomoci knihovny jQuery, konstrukce `$(document).ready()` představuje elegantní řešení, které funguje ve všech webových prohlížečích:

- Pokud je to možné, používá nativní událost o načtení modelu DOM daného webového prohlížeče, přičemž jako zálohu přidává obsluhující funkci k vlastnosti `window.onload`.
- Konstrukci `$(document).ready()` je možné zapisovat vícekrát. V takovém případě se jí předané funkce volají v pořadí, v jakém jsme je uvedli.
- Knihovna jQuery spouští funkce předané konstrukci `$(document).ready()` i v případech, v nichž jsme ji přidali až poté, co nastala v prohlížeči událost načtení modelu DOM.
- Nakládá s plánováním této události asynchronně, aby ji mohly skripty odložit, kdyby to bylo nutné.
- Simuluje vznik události načtení modelu DOM ve starších webových prohlížečích tak, že opakovaně sleduje, jestli existuje metoda modelu DOM, jež bývá k dispozici současně s načtením modelu DOM.

Metodě `ready()` můžeme předávat dříve definovanou funkci jako argument, jak je patrné v níže uvedeném kódu.

Výpis 1.1

```
function addHighlightClass() {
  $('div.poem-stanza').addClass('highlight');
}

$(document).ready(addHighlightClass);
```

Jak jsme ale viděli v původní verzi tohoto kódu, kterou opakujeme ve výpisu 1.2, tato metoda přijímá rovněž **anonymní funkci** (někdy také nazývanou **lambda funkce**).

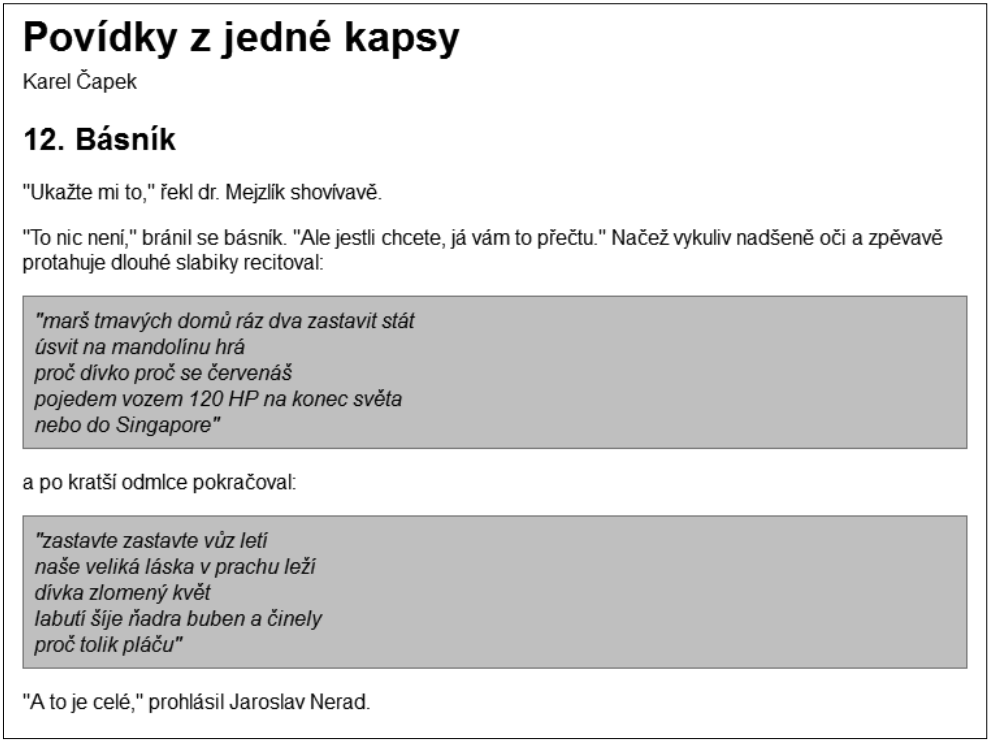
Výpis 1.2

```
$(document).ready(function() {
  $('div.poem').addClass('highlight');
});
```

Koncepce anonymních funkcí je v kódu knihovny jQuery poměrně běžná – objevují se všude tam, kde chceme předat metodě jako argument funkci, kterou už nebudeme posléze potřebovat nikde jinde. Anonymní funkce navíc vytváří nový **uzávěr**, jejž můžeme používat jako užitečný nástroj. Pokud však nebudeme pracovat s anonymními funkcemi obezřetně, mohou mít negativní dopad na využití paměti. Uzávěrům se budeme detailně věnovat v příloze A, „Uzávěry v jazyce JavaScript."

Výsledný produkt

Jakmile vložíme náš kód jazyka JavaScript, stránka bude vypadat podobně jako na následujícím obrázku.



Básně se nyní vypisují kurzívou a uvnitř rámečku, protože tento vzhled jsme definovali v šabloně stylů *01.css* a v kódu jazyka JavaScript jsme jim přidělili třídu `highlight`.

Srovnání běžného kódu JavaScriptu s kódem používajícím knihovnu jQuery

Dokonce i tak jednoduchá úloha, jakou jsme teď splnili, by byla bez knihovny jQuery složitá. V prostém jazyce JavaScript bychom přidali třídu `highlight` jako v níže uvedeném bloku zdrojového kódu.

Výpis 1.3

```
window.onload = function() {
  var divs = document.getElementsByTagName('div');
  for (var i = 0; i < divs.length; i++) {
    if (hasClass(divs[i], 'poem') && !hasClass(divs[i], 'highlight')) {
      divs[i].className += ' highlight';
    }
  }

  function hasClass(elem, cls) {
```

```
    var reClass = new RegExp(' ' + cls + ' ');
    return reClass.test(' ' + elem.className + ' ');
  }
};
```

Toto řešení nepokrývá, navzdory své délce, všechny situace jako knihovna jQuery ve výpisu 1.2, k nimž se řadí kupříkladu následující:

- Správně se vypořádat s dalšími obsluhujícími funkcemi pro událost načtení modelu DOM.
- Spouštět kód okamžitě po načtení modelu DOM.
- Optimalizovat načítání elementů a další operace prostřednictvím moderních metod modelu DOM.

Je patrné, že kód, v němž používáme knihovnu jQuery, lze snadněji napsat, přečíst a rychleji spustit než obdobný kód v jazyce JavaScript.

Vývojové nástroje

Na předchozím srovnání kódů jsme si ukázali, že kód používající knihovnu jQuery je obyčejně kratší a čistější než s ním srovnatelný prostý kód jazyka JavaScript. To ale neznamená, že budeme neustále psát bezchybný kód, nebo že pokaždé budeme intuitivně tušit, co se na našich stránkách odehrává. Programování s knihovnou jQuery bude mnohem jednodušší, když nám budou pomáhat vývojové nástroje.

Kvalitní vývojové nástroje jsou dostupné ve všech moderních webových prohlížečích. Můžeme si vybrat nástroj, jenž nám bude nejlépe vyhovovat. K dispozici máme kupříkladu tyto nástroje:

- Nástroje pro vývojáře prohlížeče Internet Explorer: *http://msdn.microsoft.com/en-us/library/dd565628.aspx*.
- Nástroje pro vývojáře prohlížeče Safari: *https://developer.apple.com/technologies/safari/developer-tools*.
- Nástroje pro vývojáře prohlížeče Google Chrome: *https://developers.google.com/chrome-developer-tools/?hl=cs*.
- Doplněk Firebug prohlížeče Firefox: *https://getfirebug.com/*.

Všechny tyto sady nástrojů nabízejí podobné funkce:

- Možnost zkoumat a upravovat části modelu DOM.
- Sledovat vztah mezi pravidly stylů jazyka CSS a jejich vlivem na stránku.
- Procházet skript postupně pomocí speciálních metod.
- Pozastavit provádění skriptu a zkoumat hodnoty jeho proměnných.

Přestože jednotlivé prohlížeče implementují tyto nástroje mírně odlišně, hlavní koncepce zůstávají stejné. Některé příklady v této knize vyžadují, abychom nějaký z těchto nástrojů

používali – v takových případech použijeme doplněk Firebug, ale stejnému účelu by posloužily i jiné nástroje.

Doplněk Firebug

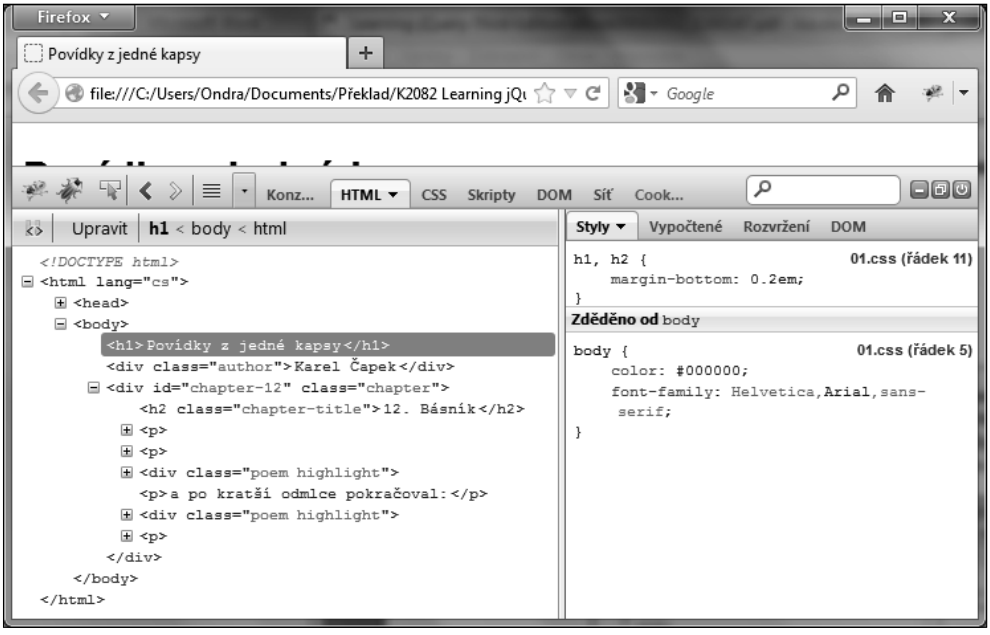
Aktuální návod pro instalaci a používání doplňku Firebug je možné najít na jeho domovské stránce na internetové adrese *http://getfirebug.com/*. Tento nástroj je příliš komplikovaný na to, abychom ho zde mohli podrobně zkoumat, ale přehled jeho základních funkcí bude jistě užitečný.

TIP

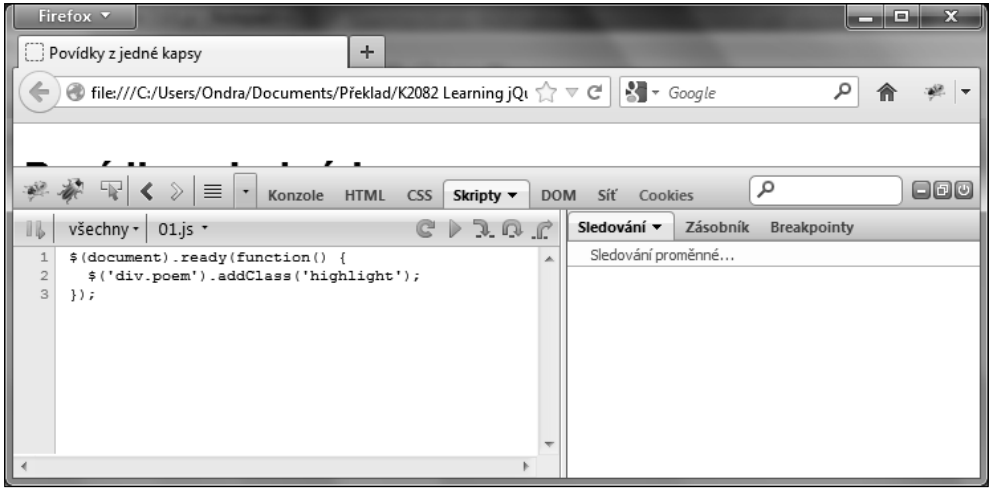
Doplněk Firebug se neustále rozrůstá a mění, takže následující obrázky nemusí přesně odpovídat jeho uživatelskému rozhraní. Některé zobrazené textové popisky můžou pocházet z volitelného doplňku FireQuery: *http://firequery.binaryage.com/*.

Když aktivujeme nástroj Firebug, objeví se nový panel s informacemi o aktuální stránce.

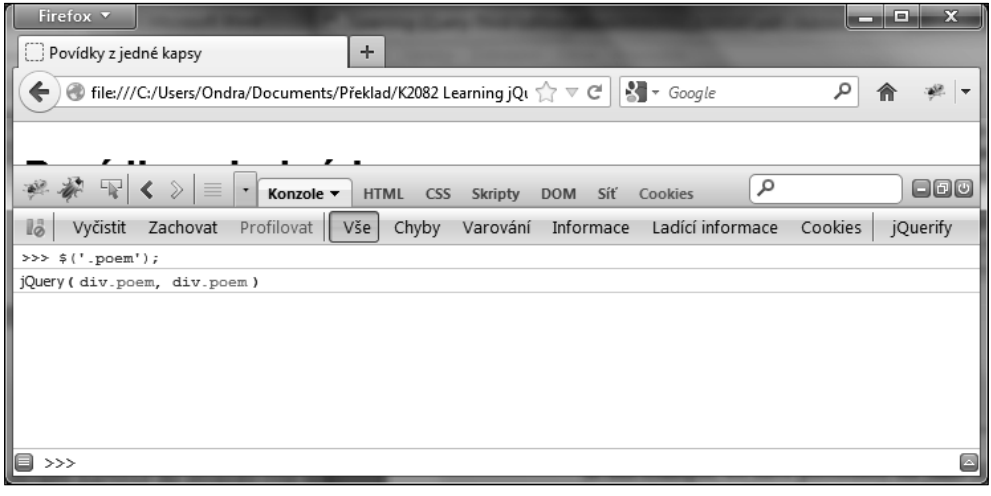
Karta **HTML** tohoto panelu obsahuje stromovou reprezentaci struktury aktuální stránky vlevo a podrobné informace o vybraném elementu (například na něj aplikovaná pravidla stylů) na pravé straně. Tato záložka je užitečná při prohlížení struktury stránky a ladění problémů s kódem jazyka CSS, jak lze vidět na následujícím obrázku.



Na kartě **Skripty** můžeme prohlížet všechny skripty načtené do stránky (viz následující obrázek). Klepnutím na číslo řádku můžeme nastavit **bod přerušení** – jakmile se skript dostane k řádku s bodem přerušení, nástroj Firebug přeruší jeho provádění, dokud neklepneme na některé tlačítko v horní části tohoto panelu.



Kartu **Konzole** budeme používat při učení se knihovně jQuery nejčastěji, jak je patrné na níže uvedeném obrázku. Do pole ve spodní části tohoto panelu můžeme zadat příkaz jazyka JavaScript a posléze tento panel zobrazí výsledek daného příkazu.



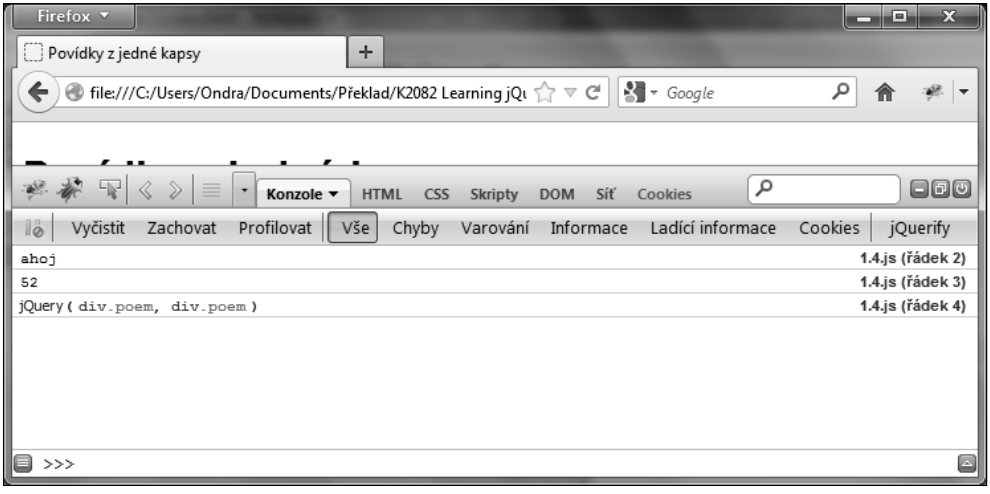
V tomto příkladu jsme použili stejný selektor knihovny jQuery jako ve výpisu 1.2, ale tentokrát jsme s vybranými elementy nic neprovedli. Přesto jsme spuštěním tohoto příkazu získali zajímavou informaci – víme, že výsledkem tohoto výběru je objekt knihovny jQuery ukazující na dva elementy s třídou `poem`. V této konzoli si můžeme kdykoliv rychle vyzkoušet kód napsaný s použitím knihovny jQuery.

Navíc můžeme s touto konzolí pracovat přímo v našem zdrojovém kódu, a to prostřednictvím metody `console.log()`:

Výpis 1.4

```
$(document).ready(function() {
  console.log('ahoj');
  console.log(52);
  console.log($('div.poem'));
});
```

Na předchozím kódu jsme si předvedli, že metodě `console.log()` můžeme předat libovolný výraz. Jednoduché hodnoty (například textové řetězce) vypisuje konzole přímo, ale složitější hodnoty (kupříkladu objekty knihovny jQuery) naformátuje tak, abychom je mohli pohodlně zkoumat, jak ukazuje následující obrázek.



Metoda `console.log()` představuje běžnou alternativu k metodě `window.alert()` jazyka JavaScript a budeme ji používat při testování našeho kódu knihovny jQuery.

Shrnutí

Nyní už víme, proč by měl webový vývojář používat nějaký framework pro jazyk JavaScript místo toho, aby psal veškerý kód od začátku. Ukázali jsme si také, v jakých ohledech knihovna jQuery vyniká jako framework pro jazyk JavaScript a které úlohy usnadňuje. Jednoduše řečeno – proč bychom si ji měli vybrat z tolika možností.

V této kapitole jsme se dozvěděli, jak načíst knihovnu jQuery do našeho kódu jazyka JavaScript, jak pomocí funkce `$()` vyhledat části aktuální stránky s danou třídou, jak změnit jejich vzhled voláním funkce `addClass()` a jak spustit celý náš kód až po načtení stránky prostřednictvím konstrukce `$(document).ready()`. Představili jsme si rovněž vývojové nástroje, na které se budeme spoléhat při psaní, testování a ladění našeho zdrojového kódu.

Tento jednoduchý příklad demonstruje, jak skvěle funguje knihovna jQuery, ale ve skutečné aplikaci by nebyl příliš užitečný. V příští kapitole budeme tento kód rozšiřovat, když budeme zkoumat selektory knihovny jQuery, přičemž si ukážeme praktické příklady této techniky.

KAPITOLA

2

Vybíráme elementy

Témata kapitoly:

- Model DOM
- Funkce $()
- Selektory jazyka CSS
- Vlastní selektory
- Metody pro procházení modelu DOM
- Přistupujeme k elementům modelu DOM

Knihovna jQuery používá sílu **selektorů** jazyka CSS (Cascading Style Sheets), aby nám umožnila snadno přistupovat k elementům v modelu DOM (Document Object Model). V této kapitole si popíšeme několik takových selektorů, a navíc ještě **vlastní selektory** knihovny jQuery. Ukážeme si také metody knihovny jQuery pro procházení modelu DOM, s nimiž získáme ještě větší volnost v tom, jak vyhledat to, co potřebujeme.

Model DOM

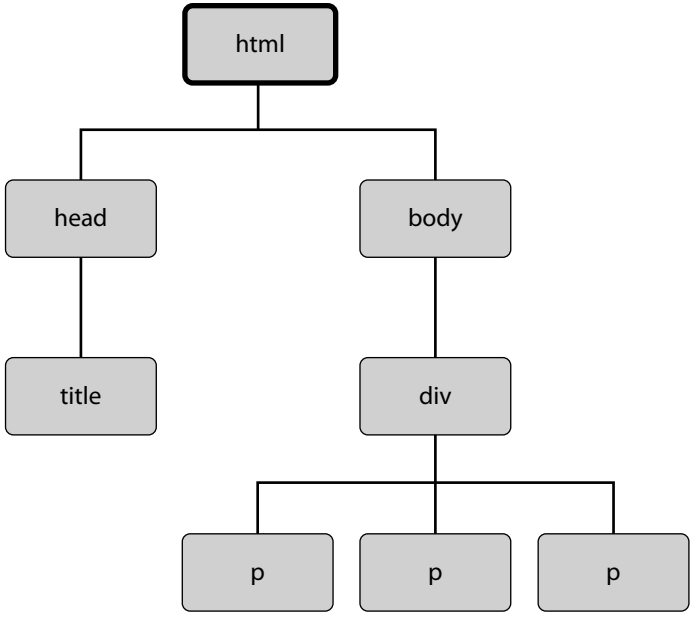
Nejužitečnější schopnost knihovny jQuery spočívá v rychlém vyhledávání elementů v modelu DOM. Model DOM slouží jako rozhraní mezi jazykem JavaScript a webovou stránkou – reprezentuje strukturu dokumentu HTML jako síť objektů.

Tato síť vypadá jako „genealogický strom“ („rodinný strom“) elementů na stránce. Když mluvíme o vztazích mezi elementy, používáme stejná označení jako u rodinných vztahů – rodiče, děti atd. Ukažme si jednoduchý příklad:

```
<html>
<head>
  <title>Název</title>
</head>
<body>
  <div>
    <p>Toto je odstavec.</p>
    <p>Toto je další odstavec.</p>
    <p>Toto je ještě jeden odstavec.</p>
  </div>
</body>
</html>
```

Element `html` je **předkem** pro všechny ostatní elementy; jinými slovy – všechny ostatní elementy jsou **potomky** ele-

mentu `html`. Elementy `head` a `body` nejsou pouze potomky elementu `html`, ale také jeho **dceřinými elementy**. Obdobně – element `html` není jen předkem elementů `head` a `body`, ale rovněž jejich rodičovským elementem. Elementy `p` jsou dceřinými elementy (a současně potomky) elementu `div`, dále pak potomky elementů `body` a `html` a sourozeneckými elementy mezi sebou navzájem (viz níže uvedený diagram).



K vizualizaci rodinného stromu dokumentu můžeme použít celou řadu nástrojů – například doplněk Firebug pro prohlížeč Firefox, nebo nástroje pro vývojáře v prohlížečích Safari a Chrome.

Ve stromu elementů budeme moct efektivně vyhledávat jakoukoliv skupinu elementů na stránce. K tomuto účelu máme k dispozici **selektory** a **procházecí metody** knihovny jQuery.

Než začneme, měli bychom si říct, že výsledná skupina elementů bude vždy zabalená do **objektu knihovny jQuery**. V případě, že budeme chtít s vyhledanými elementy něco provést, s těmito objekty lze pracovat velmi jednoduše. Opravdu snadno můžeme s těmito objekty svazovat **události**, připojovat k nim **efekty**, a také **řetězit** více uprav a efektů dohromady. Objekty knihovny jQuery se přesto liší od obyčejných elementů modelů DOM, a proto nemusí v určitých situacích poskytovat stejné metody a vlastnosti. Na konci této kapitoly si vysvětlíme, jak přistupovat přímo k elementům, které jsou zabalené do objektu knihovny jQuery.

Funkce $()

Bez ohledu na to, jaký selektor knihovny jQuery, se chystáme použít, vždy musíme začít stejnou funkcí, a to funkcí `$()`. Tato funkce většinou přijímá selektor jazyka CSS jako svůj argument a posléze funguje jako „továrna", jež vrací nový objekt knihovny jQuery, který odkazuje na odpovídající elementy na stránce. Této funkci můžeme předat libovolný selektor, který obyčejně zapisujeme do šablony stylů a následně můžeme na vyhledanou skupinu elementů volat metody knihovny jQuery.

TIP

V knihovně jQuery představuje znak `$` pouhý alias pro název `jQuery`. Protože funkce `$()` je v knihovnách JavaScriptu poměrně rozšířená, může dojít ke konfliktu, pokud do jediné stránky načteme více takových knihoven. Těmto konfliktům se můžeme vyhnout kupříkladu tak, že ve svém zdrojovém kódu budeme psát slovo `jQuery` místo znaku `$`. Na další řešení tohoto problému se zaměříme v kapitole 10, „Pokročilé události."

Třemi základními stavebními bloky selektorů jsou **názvy elementů**, **identifikátory** a **třídy**. Tyto stavební bloky můžeme používat samostatně nebo je kombinovat s ostatními. Na následujících jednoduchých příkladech si ukážeme, jak tyto selektory zapisovat do našeho kódu:

Typ selektoru	Jazyk CSS	Knihovna jQuery	Význam
název elementu	`p {}`	`$('p')`	Vybírá všechny odstavce v dokumentu.
identifikátor	`#nejaky-id {}`	`$('#nejaky-id')`	Vybírá jediný element s identifikátorem `nejaky-id`.
třída	`.nejaka-trida {}`	`$('.nejaka-trida')`	Vybírá všechny elementy, které mají třídu `nejaka-trida`.

Z kapitoly 1, „Začínáme," víme, že když voláme metody objektu knihovny jQuery, tato knihovna obvykle automaticky (implicitně) prochází všechny elementy, na něž se odkazujeme selektorem, jejž jsme předali funkci `$()`. Díky tomu se vyhneme **explicitní iteraci** (například cyklu `for`), s níž se běžně setkáváme při práci s modelem DOM.

Právě jsme si popsali všechny základy knihovny jQuery, takže můžeme začít zkoumat mocnější selektory.

Selektory jazyka CSS

Knihovna jQuery podporuje téměř všechny selektory uvedené ve specifikacích 1 až 3 pro jazyk CSS, které sepsalo konsorcium W3C (World Wide Web Consortium): *http://www.w3.org/Style/CSS/specs*. Tato podpora umožňuje vývojářům vylepšovat své webové stránky,

aniž by se museli starat o to, které webové prohlížeče rozumí pokročilým selektorům (za předpokladu, že uživatelé mají ve svých webových prohlížečích povolený jazyk JavaScript).

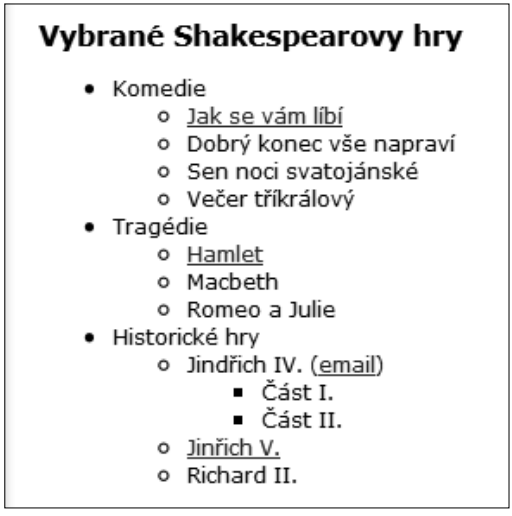
TIP

Zodpovědní vývojáři pracující s knihovnou jQuery by měli vždy aplikovat koncepci **postupného vylepšování** na svůj zdrojový kód. To znamená, že by se měli snažit, aby se jejich stránky zobrazily uživatelům, kteří mají vypnutý jazyk JavaScript, třebaže ne tak pěkně jako se zapnutým jazykem JavaScript. S touto koncepcí se budeme seznamovat napříč touto knihou.

Abychom se naučili, jak používat selektory jazyka CSS v knihovně jQuery, napíšeme kód jazyka HTML, jenž slouží na spoustě webových stránek jako navigace – víceúrovňový neuspořádaný seznam:

```
<ul id="selected-plays" class="clear-after">
  <li>Komedie
    <ul>
      <li><a href="/asyoulikeit/">Jak se vám líbí</a></li>
      <li>Dobrý konec vše napraví</li>
      <li>Sen noci svatojánské</li>
      <li>Večer tříkrálový</li>
    </ul>
  </li>
  <li>Tragédie
    <ul>
      <li><a href="hamlet.pdf">Hamlet</a></li>
      <li>Macbeth</li>
      <li>Romeo a Julie</li>
    </ul>
  </li>
  <li>Historické hry
    <ul>
      <li>Jindřich IV. (<a href="mailto:henryiv@king.co.uk">email</a>)
        <ul>
          <li>Část I.</li>
          <li>Část II.</li>
        </ul>
      </li>
      <li><a href="http://www.shakespeare.co.uk/henryv.htm">
        Jinřich V.</a></li>
      <li>Richard II.</li>
    </ul>
  </li>
</ul>
```

Povšimněte si, že první element `ul` má identifikátor `selected-plays`, ale žádný element `li` nemá přidělenou třídu. Bez pravidel stylů vypadá tento seznam jako na níže uvedeném obrázku.



Víceúrovňový seznam vypadá přesně tak, jako bychom čekali – skupina položek s odrážkami, které jsou uspořádané svisle a odsazené podle jejich úrovně zanoření.

Měníme vzhled úrovní seznamu

Předpokládejme, že chceme uspořádat položky seznamu první úrovně vodorovně (avšak pouze položky seznamu první úrovně). Pro začátek definujeme pravidlo stylu pro třídu `horizontal` v naší šabloně stylů:

```
.horizontal {
  float: left;
  list-style: none;
  margin: 10px;
}
```

Pravidlem stylu pro třídu `horizontal` nastavujeme elementy jako plovoucí vlevo, odstraňujeme z nich odrážky za předpokladu, že se jedná o položky seznamu, a k tomu ještě přidáváme vnější okraj o velikosti 10 pixelů na všechny jejich strany.

Třídu `horizontal` nebudeme přidělovat v našem dokumentu HTML, ale přidáme ji dynamicky k položkám seznamu první úrovně – tj. k položkám **Komedie**, **Tragédie** a **Historické hry**. Na následujícím výpisu si ukážeme, jak toho dosáhnout prostřednictvím selektorů v knihovně jQuery.

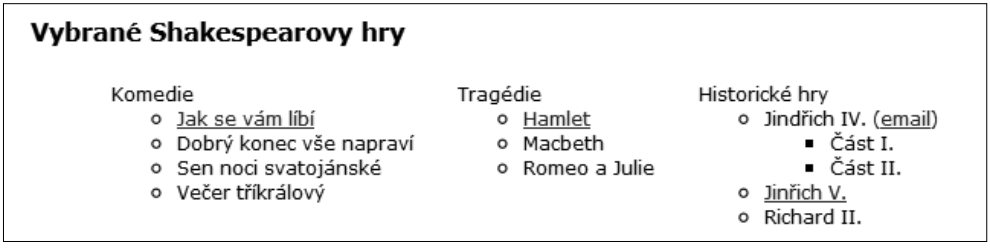
Výpis 2.1

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
});
```

Stejně jako v kapitole 1, „Začínáme," nejprve voláme metodu `$(document).ready()`, která spouští jí předanou funkci po načtení modelu DOM.

Na druhém řádku používáme **selektor dceřiného elementu** (se znakem >), s jehož pomocí přidělujeme třídu `horizontal` všem položkám seznamu první úrovně. Selektorem předaným funkci `$()` totiž říkáme: „Vyhledej všechny položky seznamu (`li`), které jsou dceřinými elementy (>) elementu s identifikátorem `selected-plays` (`#selected-plays`)."

Protože jsme aplikovali naši třídu, projeví se pravidlo stylu, které jsme definovali v naší šabloně stylů. Náš seznam tudíž bude vypadat podobně jako na následujícím obrázku.



Vzhled ostatních položek – to znamená, těch položek, které nejsou v seznamu první úrovně – je možné změnit více způsoby. Jelikož položkám na první úrovni jsme již přidělili třídu `horizontal`, můžeme použít **pseudotřídu negace**, s níž vybereme všechny položky, které nemají tuto třídu. Do našeho skriptu tudíž doplníme třetí řádek.

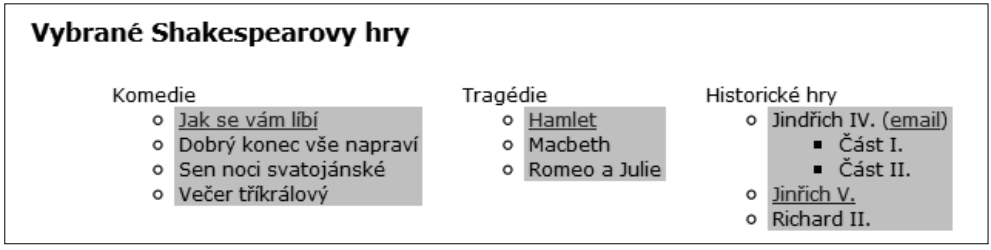
Výpis 2.2

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

Tentokrát hledáme všechny položky seznamu (`li`), které:

- jsou potomky elementu s identifikátorem `selected-plays` (`#selected-plays`),
- nemají třídu `horizontal` (`:not(.horizontal)`).

Když přidáme položkám seznamu třídu `sub-level`, získají šedé pozadí, které jsme jim definovali v naší šabloně stylů. Vnořené seznamy nyní vypadají jako na níže uvedeném obrázku.



Selektory atributů

Selektory atributů jsou velmi užitečnou skupinou selektorů jazyka CSS. Umožňují nám specifikovat element pomocí některého jeho atributu – například odkaz pomocí atributu `title` nebo obrázek prostřednictvím atributu `alt`. Všechny obrázky, které mají atribut `alt`, bychom vybrali kupříkladu takto:

```
$('img[alt]')
```

V selektorech atributů můžeme používat zástupné znaky (knihovna jQuery se v tomto ohledu inspirovala regulárními výrazy), abychom rozpoznali hodnoty na začátku (`^`) nebo na konci (`$`) textového řetězce. Pomocí hvězdičky (`*`) v selektoru atributu lze vyhledat hodnotu na libovolné pozici daného textového řetězce, nebo pomocí vykřičníku (`!`) je možné označit negovanou hodnotu.

Stylujeme odkazy

Řekněme, že budeme chtít měnit vzhled různých typů odkazů. Nejdříve definujeme příslušná pravidla stylů v naší šabloně stylů:

```
a {
  color: #00c;
}
a.mailto {
  background: url(images/email.png) no-repeat right top;
  padding-right: 18px;
}
a.pdflink {
  background: url(images/pdf.png) no-repeat right top;
  padding-right: 18px;
}
a.henrylink {
  background-color: #fff;
  padding: 2px;
  border: 1px solid #000;
}
```

Posléze přiřadíme tři třídy – `mailto`, `pdflink` a `henrylink` – vhodným odkazům prostřednictvím knihovny jQuery.

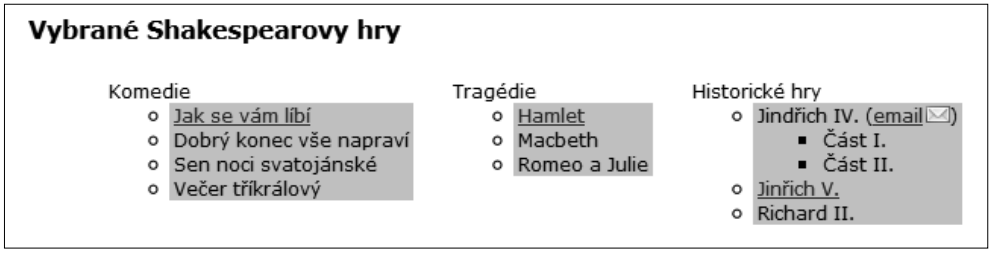
Abychom mohli přidat třídu `mailto` všem odkazům na e-mailové adresy, sestavíme selektor, který bude vyhledávat všechny odkazy (`a`) s atributem `href` (`[href]`), jehož hodnota začíná výrazem `mailto:` (), a to jako v níže uvedeném výpisu.

Výpis 2.3

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');

  $('a[href^="mailto:"]').addClass('mailto');
});
```

Díky pravidlům stylů obsaženým v šabloně stylů této stránky se objeví obrázek obálky za všemi odkazy typu `mailto:` na dané stránce, což dokazuje taktéž tento obrázek:

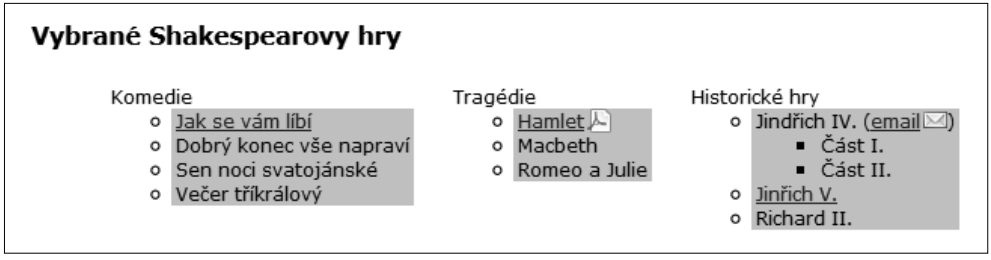


Když budeme přidělovat třídu všem odkazům na soubory PDF, použijeme znak dolaru místo znaku stříšky. Je tomu tak proto, že hledáme odkazy, jejichž atribut `href` končí výrazem `.pdf` (viz výpis 2.4).

Výpis 2.4

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');

  $('a[href^="mailto:"]').addClass('mailto');
  $('a[href$=".pdf"]').addClass('pdflink');
});
```



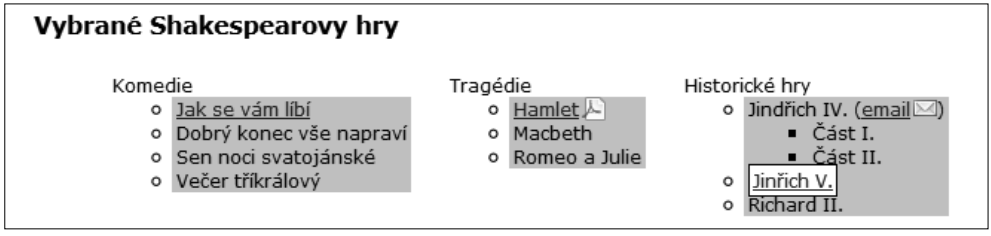
Selektory atributů je možné také kombinovat. Například můžeme přidat třídu `henrylink` všem odkazům, jejichž atribut `href` začíná výrazem `http` a současně kdekoliv obsahuje slovo `henry`.

Výpis 2.5

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');

  $('a[href^="mailto:"]').addClass('mailto');
  $('a[href$=".pdf"]').addClass('pdflink');
  $('a[href^="http"][href*="henry"]').addClass('henrylink');
});
```


Už jsme přidělili všechny tři třídy, takže naše stránka by měla vypadat podobně jako na následujícím obrázku.



Na výše uvedeném obrázku si všimněte ikony souboru typu PDF vpravo od odkazu **Hamlet**, ikony obálky vedle odkazu **email** a bílého pozadí s černým rámečkem okolo odkazu **Jindřich V**.

Vlastní selektory

Knihovna jQuery přidává k široké škále selektorů jazyka CSS ještě své vlastní selektory. Těmito vlastními selektory rozšiřuje už tak úchvatné schopnosti jazyka CSS, a tím nám umožňuje prohledávat stránku novými způsoby.

TIP

Knihovna jQuery vyhledává elementy v dokumentu pomocí nativního selektorového jádra modelu DOM aktuálního webového prohlížeče, kdykoliv je to možné. Tento neuvěřitelně rychlý přístup však nemůže používat u vlastních selektorů. Z tohoto důvodu bychom se měli vyhýbat používání vlastních selektorů, pokud existuje alternativní řešení daného problému, u něhož si vystačíme s běžnými selektory jazyka CSS. Tím napomůžeme zachování dobrého výkonu naší aplikace.

Většina vlastních selektorů nám dovoluje vybírat elementy podle jejich aktuální pozice v dříve vybrané skupině elementů. Tyto selektory zapisujeme stejně jako **pseudotřídy** jazyka

CSS – takové selektory začínají dvojtečkou (`:`). Kdybychom kupříkladu potřebovali vybrat druhý element `div` ze všech elementů `div` s třídou `horizontal`, napsali bychom tento kód:

```
$('div.horizontal:eq(1)')
```

Zapamatujte si, že selektorem `:eq(1)` vybíráte druhý element ze skupiny, protože jazyk JavaScript čísluje pole od nuly; tzn. první element má indexové pořadí `0`. Naproti tomu jazyk CSS čísluje pole od jedné, takže například selektorem `div:nth-child(1)` byste vybrali všechny elementy `div`, které jsou prvními dceřinými elementy svých rodičovských elementů (tento selektor byste také mohli napsat jednoduše jako `div:first-child`).

Měníme vzhled lichých řádků tabulky

Dvěma velmi užitečnými vlastními selektory knihovny jQuery jsou selektory `:odd` a `:even`. Podívejme se, jak s jejich pomocí můžeme zvýraznit liché řádky níže uvedených tabulek:

```
<h2>Shakespearovy hry</h2>
<table>
  <tr>
    <td>Jak se vám líbí</td>
    <td>komedie</td>
    <td></td>
  </tr>
  <tr>
    <td>Dobrý konec vše napraví</td>
    <td>komedie</td>
    <td>1601</td>
  </tr>
  <tr>
    <td>Hamlet</td>
    <td>tragédie</td>
    <td>1604</td>
  </tr>
  <tr>
    <td>Macbeth</td>
    <td>tragédie</td>
    <td>1606</td>
  </tr>
  <tr>
    <td>Romeo a Julie</td>
    <td>tragédie</td>
    <td>1595</td>
  </tr>
  <tr>
    <td>Jinřich IV., část I.</td>
    <td>historická hra</td>
```

```
      <td>1596</td>
    </tr>
    <tr>
      <td>Jindřich V.</td>
      <td>historická hra</td>
      <td>1599</td>
    </tr>
  </table>
  <h2>Shakespearovy sonety</h2>
  <table>
    <tr>
      <td>Krásný mladík</td>
      <td>1–126</td>
    </tr>
    <tr>
      <td>Černá paní</td>
      <td>127–152</td>
    </tr>
    <tr>
      <td>Básnický soupeř</td>
      <td>78–86</td>
    </tr>
  </table>
```


S minimálním množstvím uplatněných pravidel stylů naší šablony stylů vypadají nadpisy a tabulky velmi hole. Tabulky mají bílé pozadí a jejich řádky nejsou ničím rozdělené, jak je patrné na následujícím obrázku.

Shakespearovy hry

Jak se vám líbí	komedie	
Dobrý konec vše napraví	komedie	1601
Hamlet	tragédie	1604
Macbeth	tragédie	1606
Romeo a Julie	tragédie	1595
Jindřich IV., část I.	historická hra	1596
Jindřich V.	historická hra	1599

Shakespearovy sonety

Krásný mladík	1–126
Černá paní	127–152
Básnický soupeř	78–86

Nyní můžeme rozšířit naši šablonu stylů o pravidla stylů pro všechny řádky tabulky a pro liché řádky tabulky, které přidělíme třídu `alt`:

```
tr {
  background-color: #fff;
}
.alt {
  background-color: #ccc;
}
```

Ještě doplníme náš skript o řádek, s nímž přiřadíme tuto třídu všem lichým řádkům tabulky (elementy `tr`):

Výpis 2.6

```
$(document).ready(function() {
  ...
  $('a[href^="http"][href*="henry"]').addClass('henrylink');

  $('tr:even').addClass('alt');
});
```

Není něco špatně? Proč bychom měli používat selektor `:even` (**even** v překladu znamená **sudé číslo**), když chceme vybrat liché řádky? Selektory `:even` a `:odd` totiž používají číslování polí od nuly stejně jako selektor `:eq()`. První řádek tabulky má tudíž indexové pořadí `0` (sudé číslo), druhý řádek má indexové pořadí `1` (liché číslo) atd. S naším novým řádkem by měly tyto tabulky vypadat podobně jako na níže uvedeném obrázku.

Shakespearovy hry

Jak se vám líbí	komedie	
Dobrý konec vše napraví	komedie	1601
Hamlet	tragédie	1604
Macbeth	tragédie	1606
Romeo a Julie	tragédie	1595
Jindřich IV., část I.	historická hra	1596
Jindřich V.	historická hra	1599

Shakespearovy sonety

Krásný mladík	1–126
Černá paní	127–152
Básnický soupeř	78–86

Povšimněte si, že druhá tabulka neodpovídá původnímu záměru. Jelikož poslední řádek první tabulky má šedé pozadí, první řádek druhé tabulky má bílé pozadí. Tomuto problému se lze vyhnout například používáním selektoru `:nth-child()`, jenž určuje pozici elementu vzhledem k jeho rodičovskému elementu, a ne ke všem vybraným elementům. Tento selektor přijímá jako svůj argument číslo, slovo `odd` nebo slovo `even`.

Výpis 2.7

```
$(document).ready(function() {
  ...
  $('a[href^="http"][href*="henry"]').addClass('henrylink');

  $('tr:nth-child(odd)').addClass('alt');
});
```


Zapamatujte si, že selektor `:nth-child()` je jediným selektorem knihovny jQuery, u něhož tato knihovna začíná číslovat pole jedničkou. Abychom barevně odlišili střídající se řádky tabulky (avšak tentokrát správně také u druhé tabulky), použijeme tentokrát jako argument slovo `odd` místo slova `even`. Nyní už se obě tabulky zobrazují správně, jak je patrné na následujícím obrázku.

Shakespearovy hry

Jak se vám líbí	komedie	
Dobrý konec vše napraví	komedie	1601
Hamlet	tragédie	1604
Macbeth	tragédie	1606
Romeo a Julie	tragédie	1595
Jindřich IV., část I.	historická hra	1596
Jindřich V.	historická hra	1599

Shakespearovy sonety

Krásný mladík	1–126
Černá paní	127–152
Básnický soupeř	78–86

Nyní si představíme ještě jeden vlastní selektor knihovny jQuery – předpokládejme, že potřebujeme zvýraznit všechny buňky tabulky, které obsahují některou z her o Jindřichovi. Nejprve tedy rozšíříme naši šablonu stylů o toto pravidlo stylu:

```
.highlight {
  font-weight: bold;
  font-style: italic;
}
```

a potom přidáme řádek kódu se selektorem `:contains()` do našeho skriptu:

Výpis 2.8

```
$(document).ready(function() {
  ...
  $('tr:nth-child(odd)').addClass('alt');
  $('td:contains(Jindřich)').addClass('highlight');
});
```

Úspěšně jsme zvýraznili všechny hry o Jindřichovi v naší tabulce.

Shakespearovy hry

Jak se vám líbí	komedie	
Dobrý konec vše napraví	komedie	1601
Hamlet	tragédie	1604
Macbeth	tragédie	1606
Romeo a Julie	tragédie	1595
Jindřich IV., část I.	historická hra	1596
Jindřich V.	historická hra	1599

Shakespearovy sonety

Krásný mladík	1–126
Černá paní	127–152
Básnický soupeř	78–86

Důležitou vlastností selektoru `:contains()` je, že rozlišuje velikost písmen. To znamená, že příkazem `$('td:contains(jindřich)')` bychom v tomto případě nevybrali žádnou buňku tabulky.

Samozřejmě – je možné dosáhnout zvýrazňování řádků a buněk tabulky i bez knihovny jQuery, nebo jakéhokoliv jiného programování na straně klienta. Nicméně knihovna jQuery spolu se svými selektory jazyka CSS představuje skvělou alternativu pro stylování v situacích, v nichž nemáme přístup k dokumentu HTML, ani ke kódu na straně serveru, který ho generuje.

Selektory pro formuláře

Mezi vlastní selektory knihovny jQuery nepatří pouze selektory pro vyhledávání elementů podle jejich pozice. Kupříkladu při práci s formuláři nám můžou velmi ulehčit práci některé vlastní selektory knihovny jQuery ve spojení se selektory jazyka CSS3. Následující tabulka obsahuje přehled několika takových **formulářových selektorů**:

Selektor	Vyhledává
`:input`	Elementy `input`, `textarea`, `select` a `button`.
`:button`	Vyhledává elementy `button`, a také elementy `input`, jejichž atribut `type` má hodnotu `button`.
`:enabled`	Povolené formulářové elementy.
`:disabled`	Zakázané formulářové elementy.
`:checked`	Přepínače a zaškrtávací políčka, která jsou zaškrtnutá.
`:selected`	Vybrané položky seznamů.

Pokud se chceme vyjádřit specifičtěji, můžeme formulářové selektory kombinovat s jinými selektory. Mohli bychom kupříkladu vyhledat všechny zaškrtnuté přepínače (nikoliv však zaškrtávací pole) pomocí volání `$('input[type="radio"]:checked')` nebo všechna vstupní pole pro hesla spolu se zakázanými textovými poli prostřednictvím kódu `$('input[type="password"], input[type="text"]:disabled')`. Dokonce i u vlastních selektorů můžeme používat základní principy jazyka CSS při sestavování seznamu hledaných elementů.

V této kapitole jsme si pouze nastínili přehled dostupných selektorů. Do této problematiky pronikneme hlouběji v kapitole 9, „Pokročilé selektory a procházení."

Metody pro procházení modelu DOM

Se selektory knihovny jQuery, které jsme si doposud popsali, můžeme vybírat elementy napříč modelem DOM, a také směrem dolů; případně ještě filtrovat výsledky. Kdyby se jednalo o jediný způsob výběru elementů, naše možnosti by byly značně omezené (třebaže selektory jsou robustní samy o sobě, a to zejména ve srovnání s běžným procházením modelu DOM). V řadě případů ale potřebujeme vybrat rodičovský element nebo vzdálenějšího předka – v tu chvíli oceníme metody knihovny jQuery pro procházení modelu DOM. S těmito metodami můžeme procházet stromovou strukturu dokumentu směrem nahoru, dolů a do stran.

Některé z těchto metod mají svou alternativu přímo v selektorech. Například řádek, s nímž jsme prvně přidělovali třídu `alt` řádkům tabulky – `$('tr:even').addClass('alt')` – bychom mohli přepsat s použitím metody `filter()` takto:

```
$('tr').filter(':even').addClass('alt');
```

Obvykle se však oba uvedené způsoby výběru elementů spíše doplňují. Navíc metoda `filter()` toho umí více, jelikož jí můžeme předávat rovněž funkci. S pomocí této funkce můžeme složitě testovat, které elementy by měly zůstat ve výsledné skupině. Předpokládejme kupříkladu, že chceme přidat třídu všem externím odkazům. Knihovna jQuery neposkytuje žádný vyhovující selektor pro tento úkol. Bez **filtrovací funkce** bychom museli explicitně procházet všechny elementy a testovat je zvlášť. S níže uvedenou filtrovací funkcí ale zachováme implicitní iteraci, a tím i malou velikost zdrojového kódu.

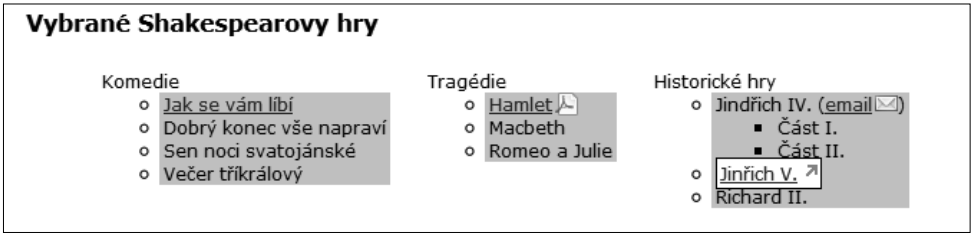
Výpis 2.9

```
$(document).ready(function() {
  ...
  $('a').filter(function() {
    return this.hostname && this.hostname != location.hostname;
  }).addClass('external');
});
```

Příkazem `return` filtrujeme elementy `a` podle dvou kritérií:

1. Odkaz musí mít atribut `href` s doménovým jménem (uloženým ve vlastnosti `this.hostname`). Pomocí tohoto kroku vyřazujeme například odkazy typu `mailto`.
2. Doménové jméno odkazu (`this.hostname`) musí být různé od (`!=`) doménového jména aktuální stránky (`location.hostname`).

Přesněji řečeno – metoda `filter()` prochází vybranou skupinu elementů v cyklu, přičemž pro každý z nich volá jí předanou funkci a posléze ověřuje její návratovou hodnotu. Pokud tato funkce vrátí hodnotu `false`, odstraní příslušný element z vybrané skupiny. Jestliže vrátí hodnotu `true`, ponechá daný element ve vybrané skupině.



V následující části této kapitoly se opět vrátíme ke zvýrazňování řádků tabulky, abychom zjistili, k čemu ještě můžeme používat procházecí metody.

Měníme vzhled vybraných buněk tabulky

Dříve v této kapitole jsme přidělili třídu `highlight` všem buňkám tabulky, které obsahují text `Jindřich`. Kdybychom chtěli přiřadit tuto třídu všem sousedním buňkám, které následují za buňkami s textem `Jindřich`, začali bychom výběrem elementů jako předtím, ale na výslednou skupinu bychom zavolali metodu `next()` jako v níže uvedeném kódu.

Výpis 2.10

```
$(document).ready(function() {
  ...
  $('tr:nth-child(odd)').addClass('alt');
  $('td:contains(Jindřich)').next().addClass('highlight');
  ...
});
```

Naše tabulky by měly nyní vypadat jako na následujícím obrázku:

Metoda `next()` vybírá pouze sousední (nejbližší) sourozenecké elementy. Kdybychom chtěli zvýraznit všechny sourozenecké elementy, které následují za buňkami s textem `Jindřich`, mohli bychom použít metodu `nextAll()`:

Výpis 2.11

```
$(document).ready(function() {
  ...
  $('tr:nth-child(odd)').addClass('alt');
  $('td:contains(Jindřich)').nextAll().addClass('highlight');
  ...
});
```

Jelikož buňky obsahující text `Jindřich` se nacházejí v prvním sloupci tabulky, pomocí předchozího zdrojového kódu zvýrazňujeme zbylé buňky v příslušných řádcích, jak je patrné na tomto obrázku:

Určitě není příliš překvapující, že existují opačné metody k metodám `next()` a `nextAll()`, a to metody `prev()` a `prevAll()`. Kromě toho – prostřednictvím metody `siblings()` je

možné vyhledávat všechny zbývající elementy na stejné úrovni modelu DOM (všechny sourozenecké elementy), ať už se nacházejí před, nebo za vybraným elementem.

Abychom zahrnuli do skupiny vybraných elementů také původní buňku (s textem `Jindřich`), můžeme doplnit volání metody `andSelf()`:

Výpis 2.12

```
$(document).ready(function() {
  ...
  $('tr:nth-child(odd)').addClass('alt');
  $('td:contains(Jindřich)').nextAll().andSelf().addClass('highlight');
  ...
});
```

Po této úpravě už jsou zvýrazněné všechny buňky na stejném řádku jako buňka s textem `Jindřich`.

Shakespearovy hry

Jak se vám líbí	komedie	
Dobrý konec vše napraví	komedie	1601
Hamlet	tragédie	1604
Macbeth	tragédie	1606
Romeo a Julie	tragédie	1595
Jindřich IV., část I.	***historická hra***	***1596***
Jindřich V.	***historická hra***	***1599***

Shakespearovy sonety

Krásný mladík	1–126
Černá paní	127–152
Básnický soupeř	78–86

Jednu skupinu elementů lze samozřejmě vyhledat mnoha různými kombinacemi selektorů a procházecích metod. Zde je kupříkladu další způsob, jak vybrat všechny buňky na řádku, který obsahuje buňku s textem `Jindřich`:

Výpis 2.13

```
$(document).ready(function() {
  ...
  $('tr:nth-child(odd)').addClass('alt');
  $('td:contains(Jindřich)').parent().children().addClass('highlight');
  ...
});
```

Zde nepřecházíme na vedlejší sourozenecké elementy, ale o úroveň výše k elementu `tr` voláním metody `parent()` a následně vybíráme všechny buňky tohoto řádku, když voláme metodu `children()`.

Řetězení

U právě popsaných metod pro procházení modelu DOM jsme se setkali s dovedností knihovny jQuery řetězit volání svých metod. S knihovnou jQuery můžeme vybrat skupinu mnoha elementů a potom s nimi dělat spoustu různých akcí, a to na jediném řádku kódu. Toto řetězení nepomáhá jen zkrácení zdrojového kódu, ale také může zlepšit efektivitu aplikace (například oproti tomu, kdybychom znovu vyhledávali elementy).

TIP

Později se důkladně seznámíte s tím, jak řetězení funguje. Prozatím si povšimněte, že téměř všechny metody knihovny jQuery vrací objekt této knihovny, na nějž můžete zavolat další metodu.

Řádek kódu se spoustou zřetězených volání metod můžeme rozdělit na více řádků, abychom zlepšili čitelnost našeho kódu. Například následující sekvenci metod bychom mohli napsat na jediný řádek.

Výpis 2.14

```
$('td:contains(Jindřich)').parent().find('td:eq(1)').addClass('highlight')
  .end().find('td:eq(2)').addClass('highlight');
```

Nebo na sedm řádků:

Výpis 2.15

```
$('td:contains(Jindřich)') // Vyhledáme všechny buňky s textem Jindřich.
  .parent() // Vybereme jejich rodičovské elementy.
  .find('td:eq(1)') // Vyhledáme druhou dceřinou buňku.
    .addClass('highlight') // Přidáme jí třídu highlight.
  .end() // Vrátíme se k rodičovským elementům buněk s textem Jindřich.
  .find('td:eq(2)') // Vyhledáme třetí dceřinou buňku.
    .addClass('highlight'); // Přidáme jí třídu highlight.
```

Procházení modelu DOM jsme v tomto příkladu dotáhli až k absurditě. Samozřejmě takové řešení rozhodně nelze doporučit pro skutečnou aplikaci, jelikož existují mnohem jednodušší způsoby. Cílem tohoto příkladu je demonstrovat skvělou flexibilitu řetězeného volání metod.

Řetězení lze přirovnat k tomu, jako bychom se snažili vyřknout celý odstavec textu na jeden nádech – s úlohou budeme rychle hotoví, ale jen málokdo nám bude rozumět. Rozdělení dlouhého řádku na více řádků, které navíc okomentujeme, nám může dlouhodobě ušetřit spoustu času.

Přistupujeme k elementům modelu DOM

Volání funkce `$()` se selektorem a většiny metod knihovny jQuery vrací objekt knihovny jQuery. S tímto přístupem si obvykle vystačíme, protože nabízí úžasnou implicitní iteraci a řetězení.

Občas ale potřebujeme přistupovat k **elementům modelu DOM** přímo; například když chceme sestavit skupinu elementů pro jinou knihovnu JavaScriptu, nebo pokud chceme zjistit název daného elementu, což je přímo vlastnost elementu modelu DOM (vlastnost `tagName`). Pro tyto výjimečné situace poskytuje knihovna jQuery metodu `get()`. K prvnímu elementu modelu DOM uvnitř objektu knihovny jQuery se dostaneme tak, že na něj aplikujeme volání `.get(0)`. Kdybychom zpracovávali všechny tyto elementy uvnitř cyklu, pak bychom získali aktuální element voláním `.get(indexovePoradi)`. V případě, že bychom chtěli určit název elementu s identifikátorem `my-element`, napsali bychom tento kód:

```
var myTag = $('#my-element').get(0).tagName;
```

Knihovna jQuery nabízí také zkrácený zápis bez nutnosti volat metodu `get()`. Jednoduše stačí za vybranou skupinu elementů přidat hranaté závorky, které obsahují indexové pořadí elementu:

```
var myTag = $('#my-element')[0].tagName;
```

Určitě není na škodu, že tento zápis vypadá, jako bychom s objektem knihovny jQuery zacházeli stejně jako s polem elementů modelu DOM – pomocí hranatých závorek převádíme objekt knihovny jQuery na seznam uzlů a indexovým pořadím (v tomto případě `0`) z něj vytrhneme samotný element.

Shrnutí

S použitím technik z této kapitoly umíme vyhledávat elementy na stránce mnoha různými způsoby. Naučili jsme se, jak stylovat položky víceúrovňových seznamů pomocí selektorů jazyka CSS, jak měnit vzhled různých typů odkazů prostřednictvím selektorů atributů, jak střídavě zvýrazňovat řádky tabulky pomocí vlastních selektorů knihovny jQuery `:odd` a `:even`, nebo pokročilého selektoru `:nth-child()`, a také, jak zvýrazňovat text v určitých buňkách tabulky prostřednictvím řetězeného volání metod.

Dosud jsme vybraným elementům přidávali třídu uvnitř funkce předané konstrukci `$(document).ready()`. V příští kapitole objevíme, jak přidávat třídu v reakci na nejrůznější uživatelské akce.

Další informace

Tématu selektorů a procházecích metod se budeme věnovat v kapitole 9, „Pokročilé selektory a procházení.“ Kompletní seznam selektorů a procházecích metod je k dispozici v příloze C, „Rychlá referenční příručka,“ nebo v oficiální dokumentaci knihovny jQuery na adrese *http://api.jquery.com/*.

Cvičení

Abyste mohli dokončit tato cvičení, potřebujete soubory *index.html* a *complete.js* ze složky zdrojových kódů pro tuto kapitolu.

Možná budete potřebovat rovněž informace z oficiální dokumentace knihovny jQuery, která je dostupná na internetové adrese *http://api.jquery.com/*.

1. Přidejte třídu `special` všem elementům `li` na druhé úrovni víceúrovňového seznamu.
2. Přidělte třídu `year` všem buňkám v třetím sloupci tabulek.
3. Přidejte třídu `special` prvnímu řádku tabulky, který obsahuje slovo `tragédie`.
4. **Výzva:** Vyberte všechny položky seznamů (elementy `li`), které obsahují odkaz (element `a`). Přidejte třídu `afterlink` všem sourozeneckým položkám, které následují za vybranými položkami.
5. **Výzva:** Přidejte třídu `tragedy` nejbližšímu elementu `ul`, jenž je předkem jakéhokoliv odkazu na soubor s příponou *.pdf*.

KAPITOLA

3

Obsluha událostí

Jazyk JavaScript nabízí několik způsobů, jak zpracovat interakci s uživatelem a jiné události. Pokud chceme vytvářet dynamické a responzivní stránky, musíme tuto schopnost využít, abychom mohli používat techniky knihovny jQuery, s nimiž jsme se již seznámili, a také další triky, které si představíme později. Třebaže bychom mohli tuto činnost provádět pouze v jazyce JavaScript, knihovna jQuery vylepšuje mechanismus obsluhy událostí pomocí elegantnější syntaxe, a navíc ho zdokonaluje.

Témata kapitoly:

- **Provádění úloh při načtení stránky**
- **Základní události**
- **Složené události**
- **Cesta události**
- **Změna cesty – objekt události**
- **Odstraňujeme obsluhující funkci události**
- **Simulujeme uživatelskou interakci**

Provádění úloh při načtení stránky

Už jsme si ukázali, jak v knihovně jQuery reagovat na načtení webové stránky. Konstrukce `$(document).ready()` pro zpracování této události umí spustit námi uvedenou funkci, avšak měli bychom si ji popsat podrobněji.

Čas spouštění zdrojového kódu

V kapitole 1, „Začínáme," jsme si řekli, že konstrukce `$(document).ready()` představuje hlavní způsob, jak v knihovně jQuery provádět úlohy po načtení stránky. Nejedná se však o jedinou metodu sloužící k tomuto účelu. S vestavěnou vlastností `window.onload` můžeme dosáhnout podobného výsledku. Přestože jsou si oba tyto způsoby podobné, musíme znát rozdíl v tom, kdy se jim předané funkce spouštějí, ačkoliv tento rozdíl může být zanedbatelný (závisí to na počtu načítaných prostředků).

Funkce předaná vlastnosti `window.onload` se spouští po kompletním stažení daného dokumentu do webového prohlížeče. To znamená, že všechny elementy na stránce jsou připravené na manipulaci v jazyce JavaScript, což je základ pro psaní složitějšího kódu, aniž bychom se museli zabývat pořadím načítání prostředků.

Na druhou stranu – obsluhující funkce, kterou předáme metodě `$(document).ready()`, se spouští v okamžiku, kdy

je k dispozici celý model DOM. To znamená, že všechny elementy jsou rovněž dostupné v našich skriptech, ale nemusí být ještě stažené všechny soubory do webového prohlížeče. Jakmile webový prohlížeč stáhne samotný dokument HTML a rozparsuje ho do modelu DOM, spustí se náš kód.

TIP

Abychom měli jistotu, že webový prohlížeč aplikuje kaskádové styly na danou stránku dříve, než s ní začneme pracovat v kódu jazyka JavaScript, měli bychom umístit všechny elementy `link` a `style` (určené pro načítání kaskádových stylů) uvnitř elementu `head` před elementy `script`.

Představme si kupříkladu stránku s fotogalerií, jež obsahuje spoustu velkých obrázků, které můžeme v knihovně jQuery skrývat, zobrazovat, přesouvat a jinak s nimi manipulovat. Kdybychom použili vlastnost `window.onload`, uživatelé by mohli používat naše funkce až poté, co by webový prohlížeč stáhl všechny tyto obrázky. Situace by ale mohla být mnohem horší, protože za předpokladu, že by delší dobu prohlížeč nenastavil obsluhu událostí pro elementy, které mají nějaké výchozí chování (například odkazy), naše aplikace by mohla začít fungovat špatně. Pokud ale použijeme konstrukci `$(document).ready()`, naše rozhraní bude připravené dříve na uživatelské akce.

TIP

Téměř vždy bychom měli upřednostňovat konstrukci `$(document).ready()` před vlastností `window.onload`, ale musíme si zapamatovat, že prohlížeč ještě nemusel načíst některé podpůrné soubory, takže nemusí být k dispozici hodnoty některých vlastností – například šířka nebo výška obrázku. Kdybychom tyto hodnoty potřebovali, museli bychom raději zvolit způsob s vlastností `window.onload` (nebo použít knihovnu jQuery ke svázání události `load`); obě tyto koncepce lze používat současně.

Více skriptů na stránce

V jazyce JavaScript běžně nastavujeme obsluhující funkce událostí tak, že je přiřazujeme příslušným vlastnostem elementů modelu DOM (případně je voláme v příslušných atributech elementů uvnitř dokumentu HTML). Mějme kupříkladu tuto funkci:

```
function udelejNeco() {
  // Proveď úlohu...
}
```

Uvnitř dokumentu HTML bychom ji mohli zavolat následovně:

```
<body onload="udelejNeco();">
```

Nebo bychom ji mohli přiřadit v našem kódu jazyka JavaScript takto:

```
window.onload = udelejNeco;
```

Oběma výše uvedenými přístupy dosáhneme toho, že se naše funkce spustí po načtení stránky. Výhodou druhého přístupu je, že chování je lépe oddělené od kódu jazyka HTML.

TIP

Povšimněte si, že když přiřazujete obsluhující funkci v kódu JavaScriptu, vynecháváte závorky na konci. Se závorkami na konci tuto funkci bezprostředně **voláte**, zatímco bez nich se na ni pouze **odkazujete** a prohlížeč ji spustí v jiný okamžik (v tomto případě po načtení stránky).

S jedinou obsluhující funkcí nemá tato strategie žádnou chybu. Předpokládejme ale, že máme také druhou funkci:

```
function udelejNecoDalsiho() {
  // Proveď další úlohu...
}
```

Posléze zkusíme nastavit, aby se tato funkce rovněž spouštěla po načtení stránky:

```
window.onload = udelejNecoDalsiho;
```

Tímto přiřazením ale přebijeme to předchozí. Vlastnost `onload` umí ukládat pouze jeden odkaz na funkci, takže nemůžeme nastavit dvě obsluhující funkce.

Metoda `$(document).ready()` zvládá tuto situaci skvěle. Každým voláním této metody přidáme novou obsluhující funkci do interní fronty. Jakmile se načte daná stránka, knihovna jQuery zavolá všechny zaregistrované funkce. Tyto funkce se spouštějí v pořadí, ve kterém jsme je registrovali.

POZNÁMKA

Knihovna jQuery samozřejmě nemá monopol na řešení tohoto problému. V jazyce JavaScript můžeme napsat funkci, jež vytváří novou funkci, která volá současnou obsluhující funkci přiřazenou vlastnosti `window.onload` a posléze námi předanou obsluhující funkci. Tímto přístupem zabraňujeme konfliktům mezi obsluhujícími funkcemi podobně jako s metodou `$(document).ready()`, ale ztrácíme několik dalších výhod, o kterých jsme se bavili. V moderních webových prohlížečích (včetně prohlížeče Internet Explorer 9) můžeme spouštět událost `DOMContentLoaded` prostřednictvím metody `document.addEventListener()`. Pokud však chceme podporovat rovněž starší prohlížeče, knihovna jQuery za nás překlene rozdíly mezi prohlížeči, abychom se o to nemuseli starat.

Zkrácená verze pro stručnost kódu

Pomocí konstrukce `$(document).ready()` ve skutečnosti voláme metodu `ready()` na objektu knihovny jQuery, jejž jsme sestavili z elementu modelu DOM s názvem `document`. Funkce `$()` představuje zkrácenou variantu pro tuto běžnou úlohu. V případě, že této funkci předáme obsluhující funkci jako argument, knihovna jQuery automaticky zavolá metodu `ready()`. Kdybychom chtěli dosáhnout stejného cíle jako v tomto bloku kódu:

```
$(document).ready(function() {
  // Sem patří náš zdrojový kód...
});
```

Mohli bychom také napsat níže uvedený zdrojový kód:

```
$(function() {
  // Sem patří náš zdrojový kód...
});
```

Ačkoliv druhá verze je kratší, z delší verze snadněji poznáme, co se v kódu odehrává. Z tohoto důvodu budeme v této knize používat delší zápis.

Předáváme argument metodě ready()

V některých případech můžeme chtít použít více knihoven JavaScript na jediné stránce. Protože spousta těchto knihoven používá název `$` (jelikož je krátký a zavedený), musíme zabránit kolizím mezi knihovnami.

V knihovně jQuery je naštěstí možné vrátit kontrolu nad tímto znakem jiným knihovnám prostřednictvím metody `jQuery.noConflict()`. Typický příklad jejího použití vypadá takto:

```
<script src="prototype.js"></script>
<script src="jquery.js"></script>
<script>
  jQuery.noConflict();
</script>
<script src="mujskript.js"></script>
```

Nejprve vkládáme jinou knihovnu (v tomto případě knihovnu Prototype) a následně samotnou knihovnu jQuery, která přebere vládu nad názvem `$`. Následně uvolňujeme znak `$` voláním metody `noConflict()`, čímž se kontrola nad ním navrátí první knihovně (Prototype). Nyní můžeme v našem skriptu používat obě knihovny, ale kdykoliv chceme volat metodu knihovny jQuery, musíme napsat název `jQuery` místo názvu `$`.

Metoda `ready()` má však v tomto ohledu ještě jedno eso v rukávě. Funkce zpětného volání, kterou jí předáváme, má volitelný parametr – samotný objekt `jQuery`. Díky tomu ho můžeme libovolně přejmenovat, aniž bychom se museli obávat vzniku konfliktů, jak je patrné na následujícím kódu:

```
jQuery(document).ready(function($) {
  // Zde můžeme používat název $ jako obvykle.
});
```

Nebo můžeme použít kratší zápis:

```
jQuery(function($) {
  // Zde můžeme používat název $ jako obvykle.
});
```

Základní události

Existuje spousta jiných okamžiků (kromě načtení stránky), v nichž bychom mohli chtít provést nějakou úlohu. Jazyk JavaScript nám neumožňuje zachytávat pouze událost načtení stránky pomocí atributu `onload` elementu `body` nebo vlastnosti `window.onload`, ale také kupříkladu události klepnutí myši (`onclick`), úpravy formulářových polí (`onchange`) a změnu velikosti okna (`onresize`). Přiřazování těchto událostí přímo elementům v dokumentu má podobné nevýhody, jaké jsme si popisovali u atributu `onload`. Z tohoto důvodu umožňuje knihovna jQuery lépe zacházet s těmito událostmi.

Jednoduchý přepínač vzhledu

Abychom si mohli ukázat některé techniky pro obsluhu událostí, předpokládejme, že máme stránku, jejíž vzhled by si uživatel mohl sám měnit. Umožníme uživateli vybírat si vzhled klepáním na tlačítka – bude moct volit mezi normálním pohledem, pohledem s textem vměstnaným do úzkého sloupce a pohledem s textem psaným velkým písmem.

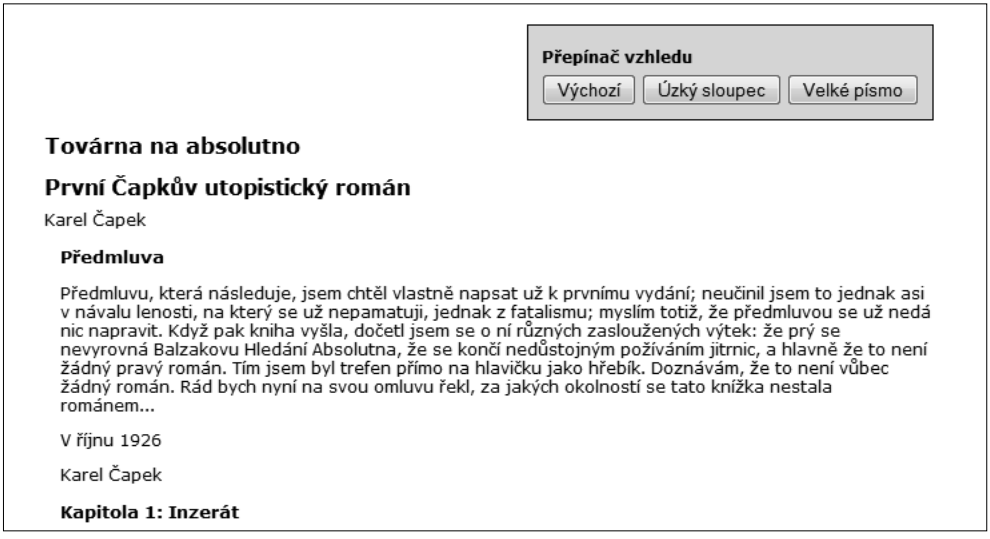
TIP

Ve skutečné aplikaci by každý dobrý vývojář použil techniku **postupného vylepšování**. Například přepínač vzhledu stránky může být schovaný, pokud není povolený jazyk JavaScript, nebo ještě lépe – může fungovat prostřednictvím odkazů na alternativní verze dané stránky. Pro zachování jednoduchosti tohoto příkladu budeme však předpokládat, že mají všichni uživatelé povolený jazyk JavaScript.

Kód jazyka HTML pro náš přepínač vzhledu vypadá následovně:

```
<div id="switcher" class="switcher">
  <h3>Přepínač vzhledu</h3>
  <button id="switcher-default">
    Výchozí
  </button>
  <button id="switcher-narrow">
    Úzký sloupec
  </button>
  <button id="switcher-large">
    Velké písmo
  </button>
</div>
```

V kombinaci se zbytkem dokumentu HTML a základními pravidly stylů získáme stránku, která vypadá jako na tomto obrázku:



Pro začátek zprovozníme tlačítko **Velké písmo**. Potřebujeme kousek kódu jazyka CSS, s nímž změníme vzhled naší stránky:

```
body.large .chapter {
  font-size: 1.5em;
}
```

Naším dalším cílem je aplikovat třídu `large` na element `body`. To umožní předchozímu pravidlu stylu změnit vzhled naší stránky odpovídajícím způsobem. Nyní zužitkujeme to, co jsme se naučili v kapitole 2, „Vybíráme elementy“ – s následujícím příkazem dosáhneme vytyčeného cíle:

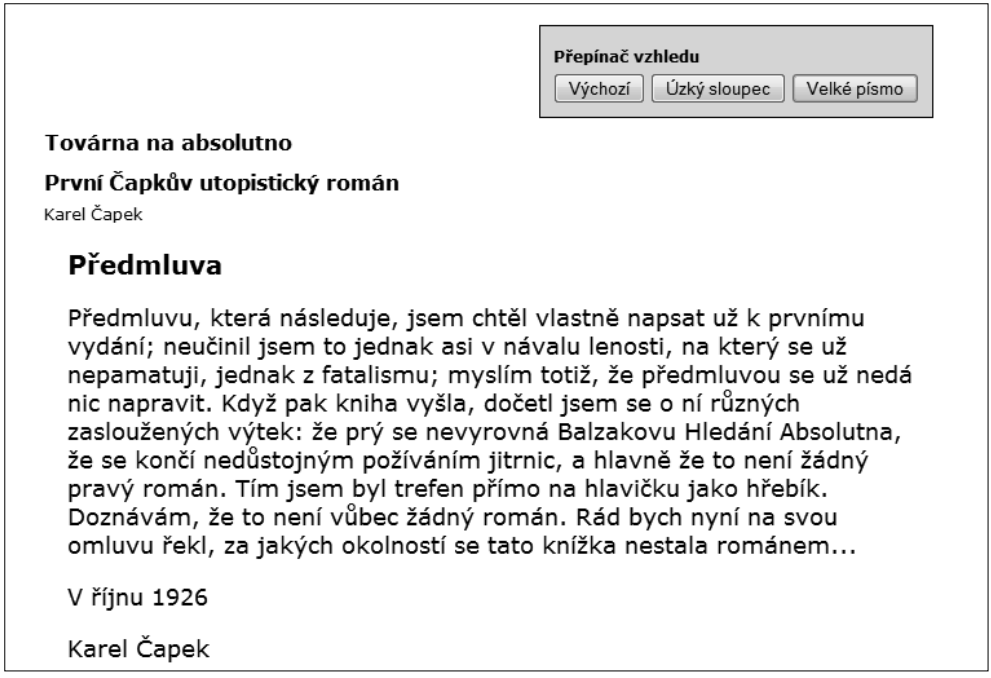
```
$('body').addClass('large');
```

Tentokrát však chceme, aby se tento příkaz provedl po klepnutí na tlačítko, a ne po načtení stránky, jako tomu bylo doposud. Za tímto účelem si představíme metodu `on()`. S její pomocí můžeme svazovat libovolnou událost s chováním. V tomto případě je touto událostí událost `click` a chováním obsluhující funkce s řádkem, který jsme si uvedli výše:

Výpis 3.1

```
$(document).ready(function() {
  $('#switcher-large').on('click', function() {
    $('body').addClass('large');
  });
});
```

Když teď uživatel klepne na tlačítko **Velké písmo**, spustí se náš kód, a tím se zvětší písmo na stránce, jak lze vidět na následujícím obrázku.



Stačilo jen svázat chování s událostí. Zde máme stejné výhody, které jsme si popisovali u metody `ready()`. Můžeme volat metodu `on()` vícekrát, a tím připojovat další obsluhující funkce pro stejnou událost.

Není to však nejelegantnější, ani nejefektivnější řešení tohoto úkolu – jak budeme postupovat touto kapitolou, rozšíříme a přepíšeme náš kód do podoby, na kterou budeme moct být pyšní.

Zprovozňujeme ostatní tlačítka

Tlačítko **Velké písmo** už funguje správně, ale musíme se obdobně postarat také o ostatní tlačítka (tlačítka **Výchozí** a **Úzký sloupec**). Řešení je prosté – pomocí metody `on()` s nimi svážeme obsluhující funkce pro událost `click`, v nichž odstraníme, nebo přidáme třídy dle potřeby. Náš nový zdrojový kód bude vypadat jako v následujícím výpisu.

Výpis 3.2

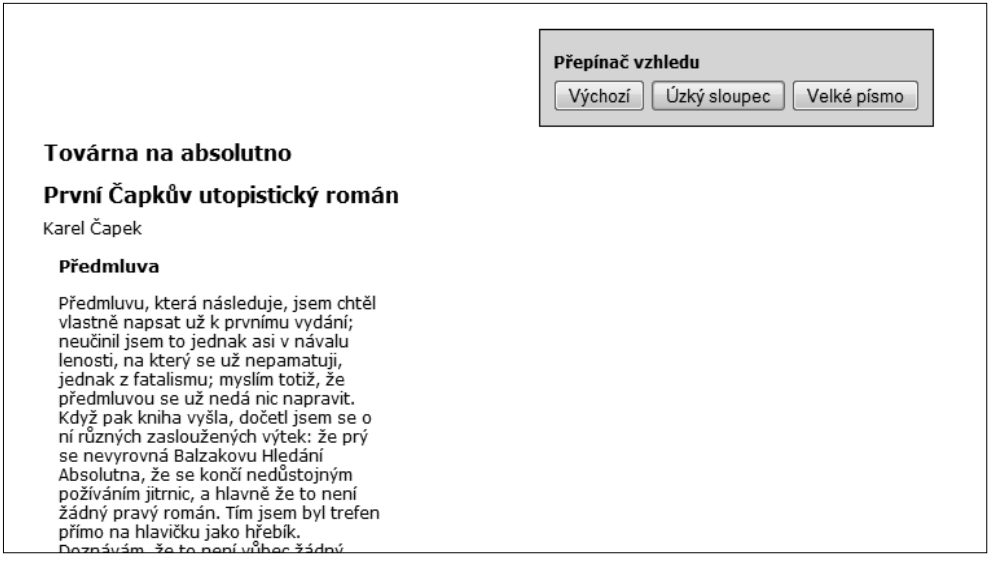
```
$(document).ready(function() {
  $('#switcher-default').on('click', function() {
    $('body').removeClass('narrow');
    $('body').removeClass('large');
  });
  $('#switcher-narrow').on('click', function() {
    $('body').addClass('narrow');
    $('body').removeClass('large');
  });
```

```
  $('#switcher-large').on('click', function() {
    $('body').removeClass('narrow');
    $('body').addClass('large');
  });
});
```

Spojíme tento kód s pravidlem stylu pro třídu `narrow`:

```
body.narrow .chapter {
  width: 250px;
}
```

Po klepnutí na tlačítko **Úzký sloupec** se uplatní výše uvedené pravidlo stylu, takže text bude rozvržený jinak:



Klepnutím na tlačítko **Výchozí** odstraníme obě tyto třídy z elementu `body`, takže stránka bude vypadat jako na začátku.

Kontext obsluhující funkce

Náš přepínač se chová správně, ale neposkytujeme uživateli žádnou zpětnou odezvu, podle níž by poznal právě aktivní tlačítko. Za tímto účelem přidáme třídu `selected` tlačítku, na které uživatel klepnul, přičemž ji odstraníme z ostatních. Třída `selected` jednoduše vypíše popisek aktuálního tlačítka tučným písmem:

```
.selected {
  font-weight: bold;
}
```

Předchozí třídu bychom mohli přidávat a odstraňovat tak, že bychom se odkázali na jednotlivá tlačítka prostřednictvím jejich identifikátorů, ale raději si ukážeme elegantnější a rozšiřitelnější řešení, které bude používat **kontext**, v němž obsluhující funkce pracují.

Uvnitř obsluhující funkce odkazuje klíčové slovo `this` na element modelu DOM, s nímž jsme tuto funkci svázali. Dříve jsme si vysvětlili, že funkce `$()` může přijímat element jako svůj argument a nyní se setkáváme s jedním z klíčových způsobů použití této dovednosti. Když napíšeme uvnitř obsluhující funkce výraz `$(this)`, vytvoříme tím objekt knihovny jQuery obsahující daný element, a můžeme s ním pracovat stejně, jako bychom ho vyhledali pomocí selektoru jazyka CSS.

Díky tomuto poznatku napíšeme následující řádek kódu:

```
$(this).addClass('selected');
```

Umístíme-li tento řádek všech tří obsluhujících funkcí, přidáme tak danou třídu tlačítku, na které uživatel klepnul. Při odstraňování této třídy z ostatních tlačítek použijeme implicitní iteraci knihovny jQuery, a proto si vystačíme s tímto řádkem kódu:

```
$('#switcher button').removeClass('selected');
```

Předchozím řádkem kódu odstraníme třídu `selected` ze všech tlačítek uvnitř přepínače vzhledu.

Po načtení daného dokumentu bychom měli rovněž přidat tuto třídu tlačítku **Výchozí**. Když sestavíme právě popsané části kódu ve správném pořadí, dostaneme níže uvedený blok zdrojového kódu:

Výpis 3.3

```
$(document).ready(function() {
  $('#switcher-default')
  .addClass('selected')
  .on('click', function() {
    $('body').removeClass('narrow');
    $('body').removeClass('large');
    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-narrow').on('click', function() {
    $('body').addClass('narrow');
    $('body').removeClass('large');
    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-large').on('click', function() {
    $('body').removeClass('narrow');
    $('body').addClass('large');
    $('#switcher button').removeClass('selected');
```

```
    $(this).addClass('selected');
  });
});
```

Nyní náš přepínač vzhledu poskytuje rovněž zpětnou odezvu.

Pokud zobecníme uvedené příkazy pomocí kontextu obsluhující funkce, můžeme být ještě efektivnější. Proces zvýrazňování tlačítek lze vyčlenit do samostatné obsluhující funkce (protože se shoduje u všech tří tlačítek), jak je patrné na výpise 3.4.

Výpis 3.4

```
$(document).ready(function() {
  $('#switcher-default')
  .addClass('selected')
  .on('click', function() {
    $('body').removeClass('narrow').removeClass('large');
  });
  $('#switcher-narrow').on('click', function() {
    $('body').addClass('narrow').removeClass('large');
  });
  $('#switcher-large').on('click', function() {
    $('body').removeClass('narrow').addClass('large');
  });

  $('#switcher button').on('click', function() {
    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Při této optimalizaci využíváme hned tři koncepce knihovny jQuery, které jsme si popsali dříve. V prvé řadě opět zapojujeme **implicitní iteraci**, když svazujeme stejnou obsluhující funkci pro událost `click` se všemi třemi tlačítky, a to jediným voláním metody `on()`. Zadruhé – pomocí **řazení obsluhujících funkcí do fronty** můžeme svázat dvě funkce s jedinou událostí `click`, aniž by druhá funkce přepsala první. V neposlední řadě používáme **řetězení**, s nímž můžeme spojit přidávání a odstraňování tříd na jediný řádek pro jednotlivá tlačítka.

Další slučování

Právě provedená optimalizace zdrojového kódu je příkladem **refaktorování** – úpravy současného kódu tak, aby prováděl stejnou úlohu efektivnějším nebo elegantnějším způsobem. V redaktorování kódu budeme ještě pokračovat – podíváme se na obsluhující funkce, které jsme připojili k našim tlačítkům. Parametr metody `removeClass()` je volitelný; když jej vynecháme, tato metoda odstraní všechny třídy z daného elementu. Díky tomu můžeme zestručnit náš zdrojový kód následujícím způsobem:

Výpis 3.5

```
$(document).ready(function() {
  $('#switcher-default')
  .addClass('selected')
  .on('click', function() {
    $('body').removeClass();
  });
  $('#switcher-narrow').on('click', function() {
    $('body').removeClass().addClass('narrow');
  });
  $('#switcher-large').on('click', function() {
    $('body').removeClass().addClass('large');
  });

  $('#switcher button').on('click', function() {
    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```


Všimněte si, že uspořádání operaci se trochu změnilo, abyste mohli použít obecnější způsob odstraňování tříd – nejprve musíte spustit metodu `removeClass()`, aby neodstranila i výsledek volání metody `addClass()`.

POZNÁMKA

Všechny třídy můžeme bezpečně odstranit jen z toho důvodu, že známe i kód jazyka HTML. Kdybychom psali opětovně použitelné obecné řešení (například pro zásuvný modul), museli bychom respektovat případné další třídy na elementech a v žádném případě je neodstraňovat.

Teď provádíme stejnou část kódu ve všech obsluhujících funkcích tlačítek. Můžeme ji tudíž vyčlenit do obecné obsluhující funkce pro událost `click`, což dokazuje níže uvedený výpis kódu.

Výpis 3.6

```
$(document).ready(function() {
  $('#switcher-default').addClass('selected');

  $('#switcher button').on('click', function() {
    $('body').removeClass();
    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
```

```
  $('#switcher-narrow').on('click', function() {
    $('body').addClass('narrow');
  });
  $('#switcher-large').on('click', function() {
    $('body').addClass('large');
  });
});
```

Všimněte si, že přesouváte obecnou obsluhující funkci nad ty specifické. Metoda `removeClass()` se musí spustit před funkcí `addClass()`, a na to se můžete spolehnout, protože knihovna jQuery vždy spouští obsluhující funkce v pořadí, v jakém je registrujete. Konečně můžete zcela odstranit všechny specifické obsluhující funkce, a to s použitím **kontextu obsluhující funkce**. Jelikož kontextové klíčové slovo `this` vrací element modelu DOM místo objektu knihovny jQuery, je možné zjistit jeho identifikátor pomocí běžné vlastnosti modelu DOM. Můžete tudíž svázat jedinou obsluhující funkci se všemi tlačítky a uvnitř ní vykonávat různé akce pro jednotlivá tlačítka jako v následujícím výpisu.

Výpis 3.7

```
$(document).ready(function() {
  $('#switcher-default').addClass('selected');

  $('#switcher button').on('click', function() {
    var bodyClass = this.id.split('-')[1];

    $('body').removeClass().addClass(bodyClass);

    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Proměnná `bodyClass` bude obsahovat hodnotu `default`, `narrow` nebo `large` – v závislosti na tom, na které tlačítko uživatel klepnul. Tentokrát se trochu odchylujeme od dřívějšího kódu, v němž jsme přidělovali třídu `default` elementu `body`, když uživatel klepnul na tlačítko **Výchozí**. Protože tuto třídu aplikovat nemusíme, menší velikost kódu určitě dostatečně vykompenzuje skutečnost, že jsme nepoužili jednu třídu.

Zkrácené metody událostí

Svazování obsluhujících funkcí s událostmi (jako například jednoduchá událost `click`) je natolik běžnou úlohou, že knihovna jQuery nabízí ještě stručnější způsob, jak toho dosáhnout – **zkrácené metody událostí** fungují stejně jako metoda `on()`, ale stačí napsat méně znaků.

Kód pro náš přepínač vzhledu bychom mohli přepsat s použitím metody `click()` místo metody `on()`, což dokazuje níže uvedený výpis.

Výpis 3.8

```
$(document).ready(function() {
  $('#switcher-default').addClass('selected');

  $('#switcher button').click(function() {
    var bodyClass = this.id.split('-')[1];

    $('body').removeClass().addClass(bodyClass);

    $('#switcher button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Zkrácené metody událostí jsou k dispozici pro všechny standardní události modelu DOM, jak je patrné na následujícím seznamu:

- `blur()`,
- `change()`,
- `click()`,
- `dblclick()`,
- `error()`,
- `focus()`,
- `keydown()`,
- `keypress()`,
- `keyup()`,
- `load()`,
- `mousedown()`,
- `mousemove()`,
- `mouseout()`,
- `mouseover()`,
- `mouseup()`,
- `resize()`,
- `scroll()`,
- `select()`,
- `submit()`,
- `unload()`.

S každou z těchto metod svazujeme obsluhující funkci se stejnojmennou událostí.

Složené události

Většina metod pro obsluhu událostí knihovny jQuery přesně odpovídá standardním událostem modelu DOM. Tato knihovna však přináší vlastní události pro ještě větší pohodlí a optimalizaci napříč webovými prohlížeči. Jednu z nich – metodu `ready()` – už jsme si podrobně popsali. K dalším takovým metodám se řadí metody `mouseenter()`, `mouseleave()`, `focusin()` a `focusout()`, které normalizují stejnojmenné události prohlížeče Internet Explorer. Metodu `hover()` označujeme jako **složenou metodu pro obsluhu událostí**, protože zachytává kombinaci uživatelských akcí a reaguje na ně více než jednou funkcí.

Zobrazování a skrývání pokročilých funkcí

Předpokládejme, že budeme chtít skrýt náš přepínač vzhledu, pokud ho nebude uživatel potřebovat. Obvyklý způsob implementace skrývání pokročilých funkcí spočívá v tom, že je umožníme sbalovat a rozbalovat. Po prvním klepnutí na popisek přepínače skryjeme jeho tlačítka a ponecháme viditelný pouze samotný popisek. Po dalším klepnutí na něj tato tlačítka zobrazíme. Potřebujeme pravidlo stylu pro třídu, které by skrylo naše tlačítka; například toto:

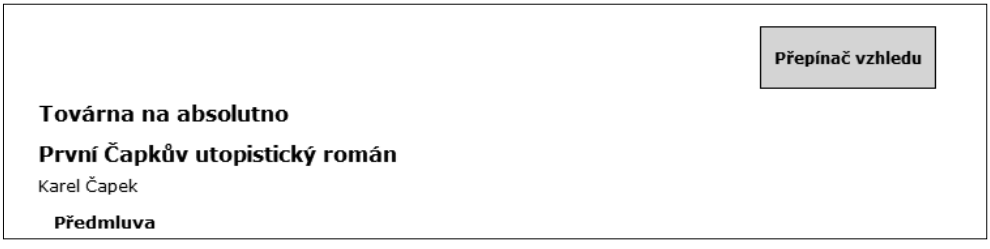
```
.hidden {
  display: none;
}
```

Tuto funkčnost bychom mohli implementovat tak, že bychom si uložili aktuální stav zobrazení tlačítek do proměnné a ověřovali její hodnotu po každém klepnutí na popisek, abychom věděli, jestli máme tlačítku přidat, nebo odebrat třídu `hidden`. Rovněž bychom mohli přímo ověřit přítomnost této třídy na tlačítku a na základě zjištění informace se rozhodnout, co udělat. Knihovna jQuery naštěstí poskytuje metodu `toggleClass()`, jež tuto činnost vykoná za nás.

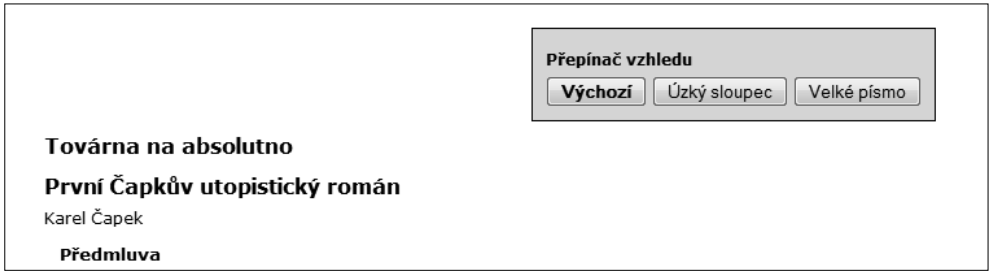
Výpis 3.9

```
$(document).ready(function() {
  $('#switcher h3').click(function() {
    $('#switcher button').toggleClass('hidden');
  });
});
```

Po prvním klepnutí se skryjí všechna tlačítka.



Druhým klepnutím je zase zobrazíme.



Opět se spoléháme na implicitní iteraci – tentokrát zobrazujeme a schováváme všechna tlačítka v jediném kroku.

Zvýrazňujeme prvky s možností klepnutí

V současné době uživatel jen stěží pozná, že může klepnout na element stránky, na který normálně není možné klepnout – v tomto případě na popisek přepínače vzhledu, což je obyčejný nadpis `h3`. To napravíme tak, že popisek našeho přepínače dynamicky zvýrazníme, abychom uživateli naznačili, že s ním může pracovat pomocí myši:

```
.hover {
  cursor: pointer;
  background-color: #afa;
}
```

Specifikace jazyka CSS obsahuje pseudotřídu `:hover`, s jejíž pomocí je možné měnit vzhled elementu, nad nějž uživatel přesunul ukazatel myši. Prohlížeč Internet Explorer 6 však tuto dovednost omezuje pouze na elementy odkazů, proto ji nemůžeme použít pro jiné elementy, jestliže chceme podporovat tento starší prohlížeč. Navíc – pokud se chceme řídit principem postupného vylepšování, měli bychom zvýraznit daný nadpis `h3` jako prvek s možností klepnutí jen za předpokladu, že jsme mu skutečně přiřadili událost `click` v kódu jazyka JavaScript. Naštěstí s pomocí metody `hover()` knihovny jQuery můžeme měnit vzhled elementu (ve skutečnosti vykonávat libovolnou akci), když ukazatel myši přechází nad tento element i tehdy, když ho opouští.

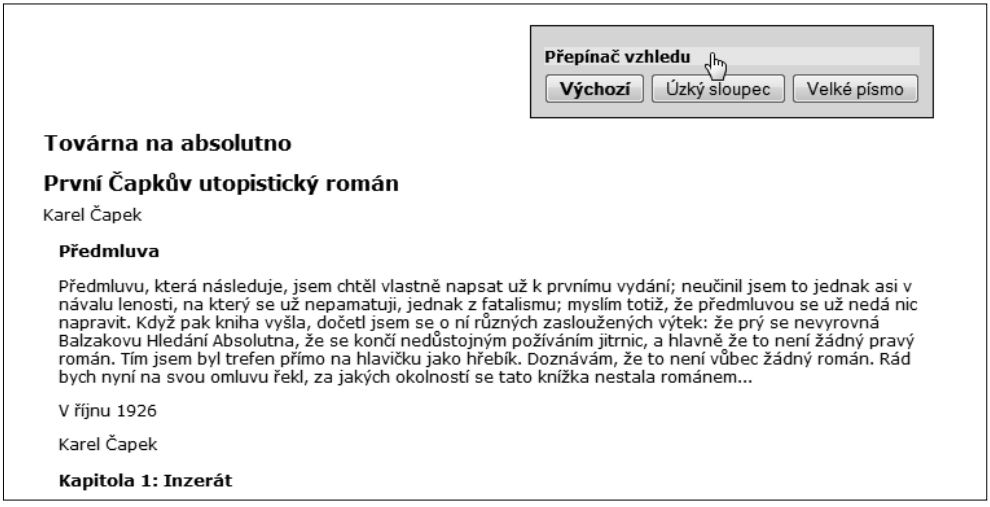
Metoda `hover()` přijímá dvě funkce jako své argumenty. První funkce se spustí, když ukazatel myši vstupuje nad vybraný element, zatímco druhá funkce se provádí v okamžiku, kdy jej ukazatel myši opouští. V těchto okamžicích můžeme odstraňovat nebo přidávat třídy našim tlačítkům, abychom dosáhli požadovaného efektu.

Výpis 3.10

```
$(document).ready(function() {
  $('#switcher h3').hover(function() {
    $(this).addClass('hover');
  }, function() {
```

```
    $(this).removeClass('hover');
  });
});
```

Zase zkracujeme náš zdrojový kód prostřednictvím implicitní iterace a kontextu obsluhující funkce. V případě, že nyní přesuneme ukazatel myši nad daný nadpis `h3`, uplatní se naše třída jako na následujícím obrázku:



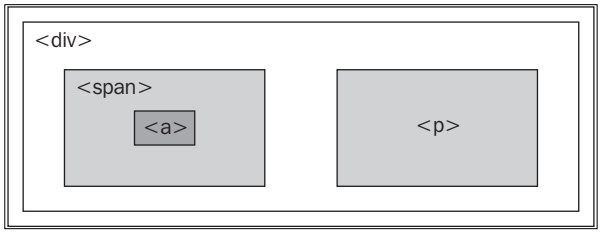
Při používání metody `hover()` se rovněž vyhneme problémům, které můžou nastávat při **delegování událostí** v jazyce JavaScript. Abychom pochopili tuto problematiku, musíme si vysvětlit, jak se jazyk JavaScript rozhoduje, na kterém elementu danou událost zpracuje.

Cesta události

Jakmile nastane událost, celý strom elementů modelu DOM získá šanci, aby ji obsloužil. Mějme kupříkladu níže uvedenou strukturu stránky:

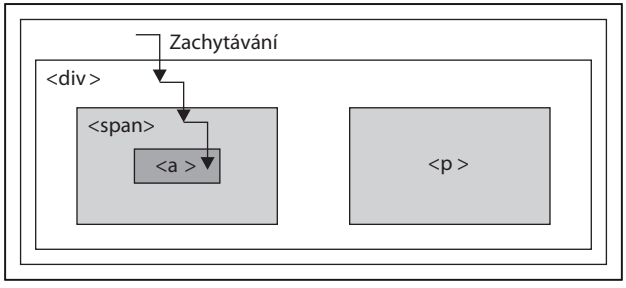
```
<div class="foo">
  <span class="bar">
    <a href="http://www.example.com/">
      Příliš žluťoučký kůň úpěl ďábelské ódy.
    </a>
  </span>
  <p>
    Zvlášť zákeřný učeň s ďolíčky běží podél zóny úlů.
  </p>
</div>
```

Tento kód lze znázornit jako skupinu nořených elementů (viz následující obrázek).



Pro každou událost existuje několik elementů, které na ni můžou reagovat. Když uživatel například klepne na odkaz na této stránce, na toto klepnutí můžou odpovědět elementy `div`, `span` nebo `a`. Koneckonců – všechny tyto tři elementy se v daný okamžik nacházejí pod ukazatelem myši. Na druhou stranu – element `p` se této interakce nezúčastňuje.

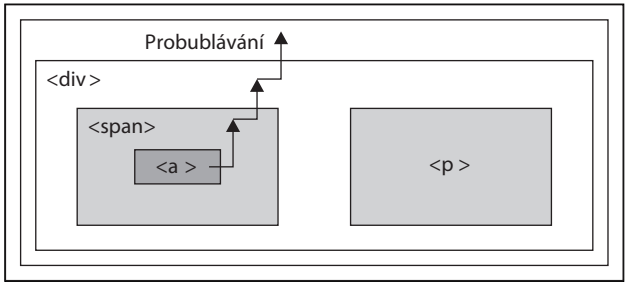
Jedna ze strategií, která umožňuje elementům reagovat na uživatelskou akci, je **zachytávání událostí**. U zachytávání událostí získává událost nejprve vnější element a posléze specifičtější elementy. V našem příkladu tudíž může tuto událost zpracovat nejdříve element `div`, následně element `span` a nakonec element `a`, jak je patrné z následujícího diagramu:



POZNÁMKA

Technicky vzato – zachytávání událostí je v prohlížečích implementované tak, že určité elementy naslouchají událostem, které nastávají mezi jejich potomky. Zde uvedený přibližný popis nám však zcela dostačuje.

Opačnou strategií je **probublávání událostí**. V tomto případě prohlížeč odešle událost nejspecifičtějšímu elementu a až poté, co tento element získá příležitost na ni reagovat, předává ji nahoru (probublává) k obecnějším elementům. V našem příkladu by získal danou událost nejprve element `a`, následně element `span` a nakonec element `div`, což dokazuje taktéž níže uvedený diagram.



Není tedy nikterak udivující, že vývojáři různých webových prohlížečů zvolili různé strategie propagování událostí. Standard pro model DOM proto specifikoval, že prohlížeče by měly používat obě strategie – nejdříve zachytávat události z obecnějších elementů ke specifičtějším a potom ji probublávat zpět nahoru stromem modelu DOM. Obsluhující funkce lze registrovat na kteroukoliv část tohoto procesu.

Ne všechny webové prohlížeče se ale začaly řídit tímto novým standardem. Kromě toho – v těch prohlížečích, které podporují zachytávání událostí, musíme zachytávání obvykle specificky povolit. Aby knihovna jQuery zavedla konzistenci mezi webovými prohlížeči, registruje obsluhující funkce vždy pro fázi **probublávání**. Můžeme tudíž počítat s tím, že nejspecifičtější element může reagovat na událost jako první.

Vedlejší účinky probublávání událostí

Probublávání událostí může způsobovat neočekávané chování, a to zejména tehdy, když na událost `mouseover` nebo `mouseout` reaguje špatný element. Představme si, že jsme přiřadili obsluhující funkci pro událost `mouseout` k našemu elementu `div`. Jakmile ukazatel myši opustí element `div`, obsluhující funkce události `mouseout` se spustí dle očekávání. Na druhou stranu – když ukazatel myši opustí element `a`, událost `mouseout` začne probublávat. Nejprve se spustí její obsluhující funkce na daném elementu `a`, potom na elementu `span` a nakonec na elementu `div`. Tuto posloupnost volání pravděpodobně nechceme.

Události `mouseenter` a `mouseleave` s těmito problémy probublávání počítají (ať už je svazujeme samostatně, nebo současně pomocí metody `hover()`), takže můžeme ignorovat problém výběru špatných elementů, který běžně vzniká u událostí `mouseover` a `mouseout`.

Příklad neočekávaného chování události `mouseout` dokládá, že v některých situacích bychom měli omezovat oblast události. Přestože metoda `hover()` řeší tento konkrétní případ, v některých situacích potřebujeme omezit událost prostorově (zabránit tomu, aby ji zachytávaly určité elementy) nebo dočasně (zabránit tomu, aby se daná událost odesílala v určitých chvílích).

Změna cesty – objekt události

Už jsme viděli jednu situaci, v níž může probublávání událostí způsobovat problémy. Abychom si představili případ, kdy nám metoda `hover()` nepomůže, změníme způsob rozbalování našeho přepínače vzhledu.

Předpokládejme, že chceme rozšířit oblast, na kterou může uživatel klepnout, když potřebuje sbalit nebo rozbalit náš přepínač vzhledu. Můžeme kupříkladu přesunout obsluhující funkci události z popisku (elementu `h3`) na obalující element `div`.

Výpis 3.11

```
// Nedokončený zdrojový kód
$(document).ready(function() {
  $('#switcher').click(function() {
    $('#switcher button').toggleClass('hidden');
```