

KAPITOLA 8

Ajax

Pravděpodobně nebude příliš zdrženlivým vyjádření, když řeknu, že Ajax revitalizoval web. Rozhodně je jedním z důvodů, který stojí za opětovně ožívajícím zájmem o JavaScript a je možné, že dokonce zapříčinil, že nyní čtete tuto knihu.

Bez ohledu na nadsázky obklopující Ajax, tato technologie dramaticky ovlivnila způsob, jímž lidé mohou komunikovat s webovou stránkou. Schopnost aktualizovat jednotlivé části stránky informacemi ze vzdáleného serveru vám možná nezní jako revoluční změna, nicméně poskytla bezproblémový typ interakce, který jsme v HTML dokumentech postrádali od jejich vzniku.

Schopnost vytvářet plynule se aktualizující webovou stránku upoutala velkou měrou představivost jak webových vývojářů, tak i designérů, z nichž spousta přešla k Ajaxu. Výsledkem ovšem byla další generace webových aplikací, které šly průměrnému uživateli pořádně na nervy. Je známo, že každá nová technologie svádí k jejímu nadměrnému používání, a stejně tak tomu bylo s Ajaxem. Když se ovšem Ajax používá citlivě a v rozumné míře, rozhodně pomáhá vytvářet rozhraní, která jsou k uživateli více vstřícnější, přičemž poskytují mnohem pestřejší a příjemnější zážitky.

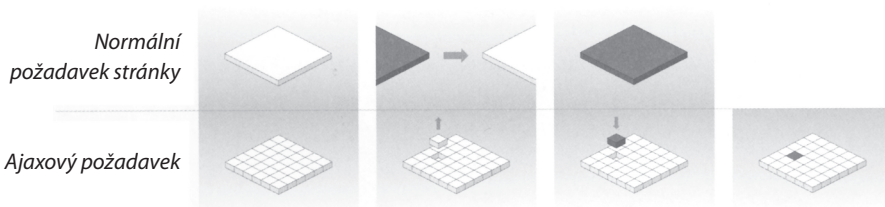
Přestože existuje docela dost "hotových" knihoven JavaScriptu, které nabízejí "kompletní zážitky s Ajaxem", osobně si myslím, že tímto není možné nahradit svobodu, jež přichází společně se znalostmi vztahujícími se k detailnímu principu fungování dané technologie – v tomto případě Ajaxu. Ale už dost povídání a planého filozofování, pojďme si říci více o Ajaxu.

XMLHttpRequest – servírování soust obsahu tak akorát do úst

Klíčovou koncepcí Ajaxu je, že informujete prohlížeč, aby obsah nehltal, ale dával si do úst pouze přijatelně velká sousta. Místo celé stránky, kterou možná požadujete, třeba jenom jediný odstavec.

Přestože cestu k funkcionalitě Ajaxu, která by byla použitelná ve všech webových prohlížečích, prosekaly už dříve plovoucí rámce (prvky `iframes`), aktuální rozmach Ajaxu podnítila až dostupnost objektu `XMLHttpRequest` ve většině webových prohlížečů.

`XMLHttpRequest` je funkcionalita webového prohlížeče, která umožňuje JavaScriptu zavolat server, aniž by bylo nutné projít klasickým mechanismem požadavek-stránka. To znamená, že JavaScript může – zatímco se prohlíží daná stránka – provádět na pozadí dodatečné požadavky na server. V podstatě toto umožňuje vytahovat další data ze serveru a následně manipulovat se stránkou pomocí DOM – tedy nahrazovat sekce, přidávat sekce, nebo odstraňovat sekce podle dat, která právě získáváme. Rozdíl mezi normálními požadavky a požadavky Ajaxu ilustruje obrázek 8.1.



Obrázek 8.1. Porovnání normálního požadavku na stránku (kdy se nahradí celá stránka) s požadavkem Ajaxu (kdy se nahradí pouze konkrétní část stránky).

Komunikací se serverem, kdy se nepoužívá mechanismus požadavek-stránka, se říká asynchronní požadavky (asynchronous requests), protože se dají provádět bez toho, aby se přerušila interakce uživatele se stránkou. Normální požadavek na stránku je synchronní v tom smyslu, že prohlížeč čeká na odpověď od serveru, přičemž další interakci povolí až tehdy, až dorazí odpověď.

`XMLHttpRequest` je ve skutečnosti jediným aspektem Ajaxu, který je úplně nový. Všechny ostatní části interakce Ajaxu – posluchač události, který ji spustí, manipulace DOM, jež aktualizuje stránku atd. – to vše jsme už probrali v předchozích kapitolách knihy. Takže – jakmile budete vědět, jak se vydá asynchronní požadavek, bez nadsázky budete umět vše potřebné.

Vytvoření objektu `XMLHttpRequest`

Internet Explorer 5 a 6 byly prvními prohlížeči, které implementovaly objekt `XMLHttpRequest` (prostřednictvím objektu `ActiveX`)¹:

```
var requester = new ActiveXObject("Microsoft.XMLHTTP");
```

Všechny ostatní prohlížeče, které podporují objekt `XMLHttpRequest` (včetně Internet Exploreru 7) to dělají už bez `ActiveX`. Objekt požadavku vypadá v těchto prohlížečích takto:

¹ Objekt `ActiveX` je termín společnosti Microsoft pro opětovně využitelnou softwarovou komponentu, která poskytuje zapouzdřenou, opětovně využitelnou, funkcionalitu. V Internet Exploreru poskytují takové objekty za normálních okolností skriptům na straně klienta přístup k všelijaké výbavě operačního systému, jako je systém souborů, nebo – v případě, že se jedná o `XMLHttpRequest` – k síťové vrstvě.

```
var requester = new XMLHttpRequest();
```

Varování – na ActiveX se nemůžete spoléhat

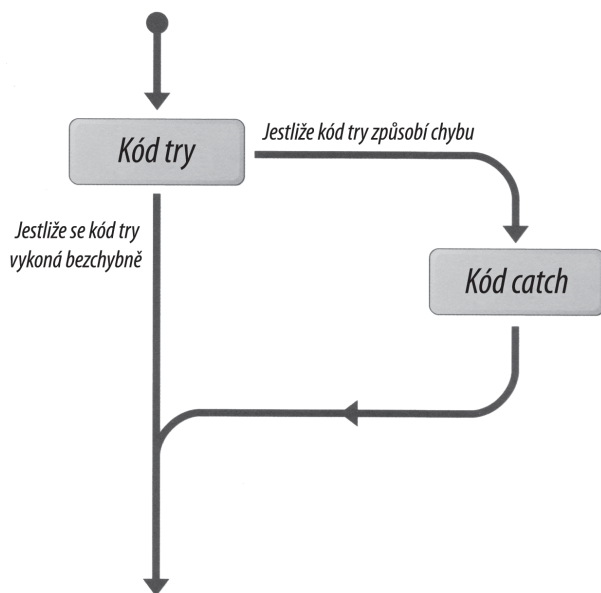
V Internet Exploreru 6 a jeho starších verzích byl XMLHttpRequest implementován tak, že pokud uživatel zakázal důvěryhodné (trusted) ovládací prvky ActiveX, stal se objekt XMLHttpRequest nedostupným, i když JavaScript byl povolen. Ačkoliv mnoho lidí zakazuje nedůvěryhodné (untrusted) ovládací prvky ActiveX, zákaz důvěryhodných ovládacích prvků ActiveX už není tak běžný.

Rozdíly mezi dvěma zmíněnými metodami pro vytvoření objektu můžeme snadno uvést v soulad prostřednictvím příkazu try-catch, který automaticky detekuje správný způsob pro vytvoření objektu XMLHttpRequest:

try-catch_test.js (výňatek)

```
try
{
    var requester = new XMLHttpRequest(); ❶
}
catch (error) ❷
{
    try
    {
        var requester = new ActiveXObject("Microsoft.XMLHTTP"); ❸
    }
    catch (error)
    {
        var requester = null;
    }
}
```

S příkazem try se můžete pokusit vykonat blok kódu, ale jakmile cokoliv uvnitř tohoto kódu způsobí chybu, běh programu neskončí, protože se přesune do příkazu catch a vykoná kód, který je uvnitř tohoto příkazu. Jak vidíte z obrázku 8.2, je to hodně podobné struktuře příkazu if-else, ovšem s tím rozdílem, že větvení neurčuje podmínka, ale výskyt nějaké chyby.



Obrázek 8.2. Logická struktura příkazu try-catch.

Pro vytváření objektů ActiveX se neobejdeme bez příkazu try - catch, protože test detekce objektu bude indikovat, že ovládací prvky ActiveX jsou dostupné i v případě, kdy je uživatel zakázal (takže skript vygeneruje chybu, až se pokusíte objekt ActiveX vytvořit).

Poznámka – ke škodě už došlo

Dojde-li k chybě uvnitř příkazu try, program se nevrátí do stavu, v jakém byl předtím, než se příkaz try vykoná – místo toho okamžitě přejde do příkazu catch. Důsledkem je, že všechny proměnné, které se vytvořily předtím, než došlo k chybě, budou pořád existovat. Jestliže ovšem došlo k chybě v okamžiku, kdy se proměnná přiřazovala, tato proměnná se vůbec nevytvoří.

A zde následuje popis toho, co všechno se bude dít v kódu uvedeném výše:

1. Pokusíme se vytvořit objekt XMLHttpRequest metodou použitelnou ve většině prohlížečů. Bude-li pokus úspěšný, bude proměnná requester novým objektem XMLHttpRequest. Pokud bude objekt XMLHttpRequest nedostupný, kód sice způsobí chybu, nicméně se budeme moci pokusit vytvořit objekt jiným způsobem (v příkazu catch).
2. Příkazy catch zachytávají výjimky, které způsobily, že příkaz try se nezdařil. Výjimka se specifikuje názvem proměnné, který se uvede v závorkách za klíčovým slovem catch a je povinná, protože přiděluje výjimce název (nezáleží na tom, zdali tento název použijete či nikoliv). Výjimce můžete přidělit jakýkoliv název – myslím si, že error je docela zřejmý.
3. Uvnitř příkazu catch se pokusíme vytvořit objekt XMLHttpRequest přes ActiveX. Pokud se to povede, bude requester platným objektem XMLHttpRequest. Jestliže se ale opět nepoda-

ří objekt vytvořit, nastaví druhý příkaz `catch` proměnnou `requester` na `null`. V takovém případě je snadné otestovat, zdali aktuální uživatelský agent podporuje `XMLHttpRequest`. Pokud to bude nutné, vykoná se nějaký nouzový kód, který nespadá pod Ajax (například to může být normální odeslání formuláře):

```
if (requester == null)
{
    kód pro klienty bez Ajaxu
}
else
{
    kód pro klienty se zapnutým Ajaxem
}
```

Významné odlišnosti, které se objevují v implementaci objektu `XMLHttpRequest` v různých webových prohlížečích, naštěstí končí vytvořením tohoto objektu. Všechny základní komunikační metody pro práci s daty se volají prostřednictvím stejné syntaxe – bez ohledu na prohlížeč, kde běží.

Volání serveru

Jakmile máme vytvořen objekt `XMLHttpRequest`, musíme zavolat dvě samostatné metody `open` a `send`, abychom byli schopni získávat data ze serveru.

Metoda `open` inicializuje připojení, přebírá dva povinné a několik nepovinných argumentů. Prvním povinným argumentem je typ HTTP požadavku, který chcete odeslat (`GET`, `POST`, `DELETE` atd.). Druhým je pak umístění, z něhož požadujeme data. Například – pokud bychom chtěli použít požadavek `GET` pro přístup k souboru `feed.xml`, který se nachází v kořenovém adresáři webu, inicializovali bychom objekt `XMLHttpRequest` takto:

```
requester.open("GET", "/feed.xml", true);
```

URL sice může být relativní nebo absolutní, ale kvůli záležitostem týkajícím se bezpečnosti musí cíl sídlit ve stejné doméně jako stránka, která ho požaduje.

Poznámka – pouze HTTP

Hodně prohlížečů povoluje volání objektu `XMLHttpRequest` pouze přes URL s prefixem `http://` a `https://`. Jinak řečeno, pokud si vaše stránky prohlížíte na lokálním umístění prostřednictvím URL s prefixem `file://`, je možné, že volání objektu `XMLHttpRequest` nebude povoleno.

Třetím argumentem `open` je logická hodnota, která specifikuje, zdali je požadavek asynchronní (`true`), nebo synchronní (`false`). Synchronní požadavek způsobí, že prohlížeč nebude reagovat až do té doby, dokud nebude požadavek dokončen. To znamená, že v této době není povolena interakce s uživatelem. Asynchronní požadavek se vykonává na pozadí, takže simultánně mohou běžet

jiné skripty, přičemž uživatel může používat prohlížeč obvyklým způsobem. Já doporučuji, abyste vždy používali asynchronní požadavky, jinak riskujete, že prohlížeče uživatelů budou zablokovány při čekání na požadavek, který nefunguje řádně. Metoda `open` dále poskytuje čtvrtý a pátý argument. Oba jsou nepovinné. Umožňují specifikovat uživatelské jméno a heslo pro potřeby autentizace (obvykle v situacích, kdy chcete přistoupit k URL, která je chráněna heslem).

Jakmile jste metodou `open` inicializovali připojení, metoda `send` připojení aktivuje a vydá požadavek. Metoda `send` přebírá jeden argument, který umožňuje odeslat společně s požadavkem POST data, jež jsou zakódována ve stejném formátu, jako když se odesílá formulář:

```
requester.setRequestHeader("Content-Type",  
    "application/x-www-form-urlencoded");  
requester.open("POST", "/query.php", true);  
requester.send("name=Clark&email=superman@justiceleague.xmp");
```

Poznámka – požadovaný Content-Type

Prohlížeč Opera požaduje, abyste metodou `setRequestHeader` nastavili záhlaví `Content-Type` požadavku POST. Ostatní prohlížeče to sice nepožadují, nicméně se jedná o nejbezpečnější přístup, který byste měli používat u všech webových prohlížečů.

Chcete-li simulovat odeslání formuláře požadavkem GET, musíte explicitně zakódovat názvy a jejich hodnoty do URL metody `open` a pak vykonat `send` s argumentem `null`:

```
requester.open("GET",  
    "query.php?name=Clark&email=superman@justiceleague.xmp", true);  
requester.send(null);
```

Internet Explorer nepožaduje, abyste předávali do `send` nějakou hodnotu, prohlížeče Mozilla ovšem vrátí chybu, nepředáte-li nic, takže proto je v kódu výše uveden argument `null`.

Jakmile jste zavolali metodu `send`, objekt `XMLHttpRequest` bude kontaktovat server a získá data, která jste požadovali. V případě, že se jednalo o asynchronní požadavek, funkce, která vytvořila připojení, pravděpodobně dokončí svůj běh během získávání dat.

V termínech toku programu se volání `XMLHttpRequest` hodně podobá volání `setTimeout`, na které se určitě pamatujete z kapitoly 5.

K oznámení toho, že server vrátil odpověď, použijeme obsluhu události. V tomto konkrétním případě potřebujeme zpracovat změny v hodnotě vlastnosti `readyState` objektu `XMLHttpRequest`, která specifikuje stav připojení objektu. Může nabývat těchto hodnot:

- 0 (uninitialized, neinicializováno).
- 1 (loading, nahráváno).
- 2 (loaded, nahráno).

- 3 (interactive, probíhá dialog).
- 4 (complete, kompletní).

Změny vlastnosti `readyState` můžeme monitorovat obsluhou událostí `readystatechange`, které se odpalují při každé změně hodnoty této vlastnosti:

```
requester.onreadystatechange = readystatechangeHandler;  
function readystatechangeHandler()  
{  
    Někjaký kód zpracovávající změny hodnoty vlastnosti  
    readyState objektu XMLHttpRequest  
}
```

Hodnota `readyState` se inkrementuje od 0 do 4, přičemž událost `readystatechange` se spouští při každé inkrementaci. My se ovšem potřebujeme pouze dozvědět, kdy se připojení dokončí (tj. kdy se bude `readyState` rovnat 4), takže naše funkce bude muset kontrolovat tuto hodnotu.

Poté, co se připojení dokončí, budeme muset také zkontrolovat, zdali objekt `XMLHttpRequest` úspěšně získal data, nebo zdali obdržel nějaký chybový kód, jako je např. 404 (stránka nenalezena). To můžeme určit z vlastnosti `status` objektu `XMLHttpRequest`, která obsahuje celočíselnou hodnotu. Hodnota 200 znamená splněný požadavek, takže tuto hodnotu je potřeba zkontrolovat společně s hodnotou 304 (nic se nezměnilo), protože obě indikují úspěšně získaná data. Vlastnost `status` ovšem může mít hodnotu libovolného kódu HTTP, který může server vrátit, takže asi budete muset napsat několik podmínek, abyste zpracovali i několik dalších kódů. Obvykle ovšem stačí, když specifikujete, co má program podniknout za akce, jestliže byl požadavek neúspěšný:

```
requester.onreadystatechange = readystatechangeHandler;  
function readystatechangeHandler()  
{  
    if (requester.readyState == 4)  
    {  
        if (requester.status == 200 || requester.status == 304)  
        {  
            kód zpracovávající úspěšný požadavek  
        }  
        else  
        {  
            kód zpracovávající neúspěšný požadavek  
        }  
    }  
}
```

Místo toho, abyste přiřazovali funkci, která je někde definována jako obsluha události `readystatechange`, můžete přímo (inline) deklarovat novou anonymní (tzn. nepojmenovanou) funkci:

```

requester.onreadystatechange = function()
{
    if (requester.readyState == 4)
    {
        if (requester.status == 200 || requester.status == 304)
        {
            kód zpracovávající úspěšný požadavek
        }
        else
        {
            kód zpracovávající neúspěšný požadavek
        }
    }
}

```

Předností této přímé (inline) specifikace funkce zpětného volání `readystatechange` je to, že objekt `requester` bude dostupný uvnitř této funkce pomocí uzávěry (closure). Jestliže se obsluha události `readystatechange` deklaruje samostatně, musíte proskákat několika brankami, chcete-li obdržet referenci na objekt `requester` uvnitř funkce pro obsluhu události.

Poznámka – XMLHttpRequest není recyklovatelný

Přestože objekt `XMLHttpRequest` povoluje volat metodu `open` několikrát, každý objekt se dá použít pouze pro jedno volání, protože událost `readystatechange` se odmítne odpálit znovu, jakmile se `readyState` změní na 4 (v prohlížečích Mozilla). Takže vždy, když budete chtít získat nějaká data ze serveru, budete muset vytvořit nový objekt `XMLHttpRequest`.

Práce s daty

Jestliže jste vydali požadavek, který skončil úspěšně, je dalším logickým krokem přechíst odpověď serveru. Pro tento účel se dají využít dvě vlastnosti objektu `XMLHttpRequest`:

<code>responseXML</code>	<i>Do této vlastnosti se uloží strom DOM reprezentující získaná data, ale pouze tehdy, pokud server pro odpověď indikoval <code>content-type</code> rovno <code>text/xml</code>. Tento strom DOM se dá zkoumat a modifikovat pomocí standardních metod a vlastností pro přístup k DOM JavaScriptu, což je téma, které jsme probírali ve třetí kapitole (patří mezi ně <code>getElementsByTagName</code>, <code>childNodes</code> a <code>parentNode</code>).</i>
<code>responseText</code>	<i>Do této vlastnosti se uloží data odpovědi v podobě jediného řetězce. Jestliže je <code>content-type</code> dat dodaných serverem <code>text/plain</code> nebo <code>text/html</code>, bude to jediná vlastnost obsahující data. V případě odpovědi s <code>text/xml</code> bude vlastnost obsahovat XML, také v podobě textového řetězce (alternativa k vlastnosti <code>responseXML</code>).</i>