

KAPITOLA 2

Základy Ajaxu

Hnací silou Ajaxu je interakce mezi klientem (webovým prohlížečem) a serverem. Dříve této komunikaci rozuměli jen ti, kteří vyvíjeli aplikace v jazycích Perl a C na straně serveru. Novější technologie, jako jsou ASP.NET, PHP a JSP, sice softwarovým inženýrům poskytují kombinaci různých technik pro obě strany, ale často jim chybí plná podpora různým technologiím na straně klienta (například JavaScriptu). Nyní se kyvadlo zhouplo opačným směrem a vývojáři na klientské straně potřebují více porozumět technologiím na straně serveru, aby mohli vytvářet ajaxová řešení.

Základy HTTP

Základem pro pochopení technik Ajaxu je pochopení principů hypertextového přenosového protokolu (HTTP), což je protokol pro přenos webových stránek, obrázků a dalších typů souborů přes internet. Zadáte-li do webového prohlížeče nějakou URL adresu začínající na `http://`, specifikujete tím použití HTTP pro přístup k informacím na daném umístění. (Většina prohlížečů podporuje mnoho dalších protokolů – například FTP.)

Mějte na paměti, že tato kapitola pokrývá jenom ty aspekty HTTP, které jsou zajímavé z hlediska vývoje v Ajaxu. Rozhodně nepředstavuje referenční manuál HTTP a ani tutoriál.

HTTP se skládá ze dvou částí: požadavku a odpovědi. Když zadáte nějakou URL adresu, prohlížeč za vás vytvoří požadavek. Tento požadavek obsahuje zadanou URL adresu a další informace o prohlížeči. Server obdrží požadavek a pošle zpět odpověď. Tato odpověď obsahuje informace o požadavku a data z požadované URL adresy (pokud tedy nějaká jsou). Pak je na samotném prohlížeči, aby odpověď ze serveru zpracoval a zobrazil danou webovou stránku (nebo jiný zdroj).

HTTP požadavek

Formát HTTP požadavku je následující:

<řádek-požadavku>

<hlavičky>

<prázdný-řádek>

[<tělo-požadavku>]

První řádek HTTP požadavku musí být řádek požadavku, který specifikuje typ požadavku, požadovaný zdroj a použitou verzi HTTP. Následuje sekce hlaviček udávající doplňující informace, které mohou být serverem využity. Za hlavičkami najdeme prázdný řádek, který může být následován nepovinnými daty (nazvanými jako tělo požadavku).

V HTTP existuje mnoho různých typů požadavků, ale pro vývojáře v Ajaxu jsou důležité pouze dva. Jsou to GET a POST. Kdykoliv zadáte URL adresu, prohlížeč pošle serveru požadavek GET na tuto adresu. Tím v podstatě serveru říká, aby vzal příslušný zdroj a poslal jej zpět. Zde vidíte, jak může vypadat požadavek GET pro adresu `www.wrox.com`:

GET / HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

První část prvního řádku udává, že se jedná o požadavek GET. Druhá část tohoto řádku je lomítko, které říká, že požadavek je na kořenový adresář domény. Poslední část tohoto řádku specifikuje použití HTTP verze 1.1 (alternativou je 1.0). A kam je požadavek odeslán? To je na dalším řádku.

Na tomto druhém řádku je `Host`, což je první hlavička požadavku. Tato hlavička udává cíl požadavku. Spojení hlavičky `Host` s lomítkem z prvního řádku znamená, že požadavek je na `www.wrox.com/`. (Hlavička `Host` je požadována v HTTP 1.1 – starší verze 1.0 ji nevyžaduje.) Třetí řádek obsahuje hlavičku `User-Agent`, která je dostupná jak pro skripty na straně serveru, tak i pro skripty na straně klienta. Je to základní kámen logiky většiny prohlížečů. Tyto informace jsou definovány použitým prohlížečem (v tomto případě se jedná o Firefox 1.0.1) a jsou posílány automaticky při každém požadavku. Posledním řádkem je hlavička `Connection`, která je obvykle nastavena na `Keep-Alive` (může být nastavena na jiné hodnoty, ale to je nad rámec této knihy). Pamatujte, že za poslední hlavičkou následuje jeden prázdný řádek. Tento prázdný řádek je nutný i v případě, kdy nenásleduje žádné tělo požadavku.

Chcete-li požádat o stránku pod doménou `www.wrox.com`, například `http://www.wrox.com/books`, bude váš HTTP požadavek vypadat následovně:

GET /books/ HTTP/1.1

Host: www.wrox.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)

Gecko/20050225 Firefox/1.0.1

Connection: Keep-Alive

Všimněte si, že se změnil pouze první řádek, který nově obsahuje tu část URL, jež následuje za `www.wrox.com`. Pokud existují nějaké parametry pro požadavek GET, jsou přidány za tuto URL. Obecný formát těchto parametrů vypadá nějak takto:

URL?jmeno1=hodnota1&jmeno2=hodnota2&...&jmenoN=hodnotaN

Tyto informace se nazývají dotazovací řetězec (query string) a jsou zkopírovány do řádku požadavku následujícím způsobem:

GET /books/?name=Professional%20Ajax HTTP/1.1

Host: `www.wrox.com`

User-Agent: `Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)`

Gecko/20050225 Firefox/1.0.1

Connection: `Keep-Alive`

Všimněte si, že text `Professional Ajax` je určitým způsobem zakódován. Mezery jsou nahrazeny znaky `%20`. Říká se tomu zakódování URL (URL encoding) a je to používáno v mnoha oblastech HTTP. (JavaScript má zabudované funkce pro zakódování a dekodování URL, které jsou probírány v pozdějších kapitolách). Dvojice jméno-hodnota (name-value) jsou odděleny znakem ampersand (&). Většina technologií na straně serveru obsah těla požadavku dekóduje a nějakým způsobem poskytne přístup k těmto hodnotám. Samozřejmě záleží na rozhodnutí serveru, co s daty udělá.

Prohlížeče často posílají mnohem více hlaviček, než bylo zmíněno ve výše uvedených příkladech. To znamená, že příklady uvedené v této kapitole byly pro jednoduchost zkráceny.

Požadavek POST poskytuje dodatečné informace ve svém těle. Například když vyplníte nějaký online formulář a odešlete jej, data jsou typicky odeslána prostřednictvím požadavku POST.

Typický POST požadavek vypadá nějak takto:

POST / HTTP/1.1

Host: `www.wrox.com`

User-Agent: `Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)`

Gecko/20050225 Firefox/1.0.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 40

Connection: `Keep-Alive`

name=Professional%20Ajax&publisher=Wiley

Povšimněte si několika odlišností mezi požadavkem POST a GET. Za prvé – první řádek začíná slovem POST namísto GET. Tím je určen typ požadavku. Dále si všimněte, že hlavičky `Host` a `User-Agent` zůstaly a přibýly dvě nové. Hlavička `Content-Type` specifikuje, jak je požadavek zakódován. Prohlížeče vždycky kódují POST data jako `application/x-www-form-urlencoded`,

což je typ MIME pro jednoduché kódování URL. Hlavička `Content-Length` udává délku těla požadavku v bytech. Za hlavičkou `Connection` a prázdným řádkem pak následuje tělo požadavku.

Jako u většiny požadavků POST má tělo podobu dvojic jméno-hodnota (name-value), kde jméno `name` má hodnotu `Professional Ajax` a jméno `publisher` má hodnotu `Wiley`. Měli byste si všimnout, že se jedná o stejný formát jako v případě parametrů v dotazovacím řetězci v URL.

Jak bylo zmíněno dříve, existují i jiné typy HTTP požadavků, nicméně všechny mají stejný základní formát jako HTTP požadavky GET a POST. Podívejme se dále, co server posílá nazpět jako odpověď na HTTP požadavek.

Z bezpečnostních důvodů může být požadavek GET použit pouze pro získávání dat. Pokud je potřeba nějaká data vkládat, editovat nebo mazat, musí být použit HTTP požadavek POST.

HTTP odpověď

Formát HTTP odpovědi je velmi podobný formátu HTTP požadavku:

```
<řádek-se-stavem>
<hlavičky>
<prázdný-řádek>
[<tělo-odpovědi>]
```

Jak můžete vidět, jediná odlišnost je v prvním řádku, který nyní místo řádku s požadavkem obsahuje informaci o stavu. Tento řádek se stavem obsahuje stavový kód pro požadovaný zdroj. Příklad HTTP odpovědi následuje:

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 122
```

```
<html>
  <head><title>Wrox Homepage</title></head>
  <body>
    <!-- zde následuje tělo WWW stránky -->
  </body>
</html>
```

V tomto příkladu je v řádku se stavem uveden HTTP kód 200 a zpráva OK. Nemohou nastat žádné nejasnosti, protože řádek se stavem vždy obsahuje stavový kód a odpovídající krátkou zprávu. Nejpožívanější stavové kódy jsou následující:

- **200 (OK).** Zdroj byl nalezen a všechno je v pořádku.

- **304 (NOT MODIFIED)**. Zdroj nebyl od posledního požadavku modifikován (upravován). Tento kód nejčastěji využívají různé cache webových prohlížečů.
- **401 (UNAUTHORIZED)**. Klient nemá oprávnění k přístupu ke zdroji. Tento kód často vyvolá dotaz na uživatelské jméno a heslo pro přihlášení se k serveru.
- **403 (FORBIDDEN)**. Klient nezískal oprávnění. Tato situace typicky nastane, když se nezdaří přihlášení s uživatelským jménem a heslem po kódu 401.
- **404 (NOT FOUND)**. Zdroj na dané adrese neexistuje.

Za prvním řádkem se stavem následují hlavičky. Server obvykle vrátí hlavičku `Date` reprezentující datum a čas, kdy byla odpověď vygenerována. (Seravery často také vrací nějaké informace o sobě, i když to není požadováno.) Další dvě hlavičky jsou vám už známé – je to `Content-Type` a `Content-Length`, stejně jako v požadavku `POST`. V tomto případě `Content-Type` udává MIME typ pro HTML (`text/html`) s kódováním ISO-88-59-1, což je standard pro zdroje v americké angličtině. Tělo odpovědi obsahuje HTML kód požadovaného zdroje (ale pro jiné typy zdrojů může obsahovat i prostý text nebo binární data). Právě tato data pak prohlížeč zobrazuje uživateli.

Všimněte si, že v odpovědi nenaleznete žádné označení požadavku, který si tuto odpověď ze serveru vyžádal – pro server toto nemá žádný význam. Je na klientovi, aby věděl, jaký typ dat má pro daný požadavek v odpovědi očekávat, a je pouze na něm, aby se rozhodl, co s těmito daty udělat.

Techniky komunikace pro Ajax

Nyní už rozumíte základům HTTP, takže je čas se podívat na techniky komunikace na webové stránce. Jak už sami víte, existuje mnoho požadavků, které jsou posílány mezi klientem a serverem, zatímco vy surfujete na webu. Všechny tyto požadavky vzniknou pokaždé, když uživatel učiní nějakou akci, která tyto požadavky vyžaduje. Techniky Ajaxu osvobozují vývojáře od čekání, než uživatel tuto akci provede, protože umožňují vytvořit pro volání serveru kdykoliv.

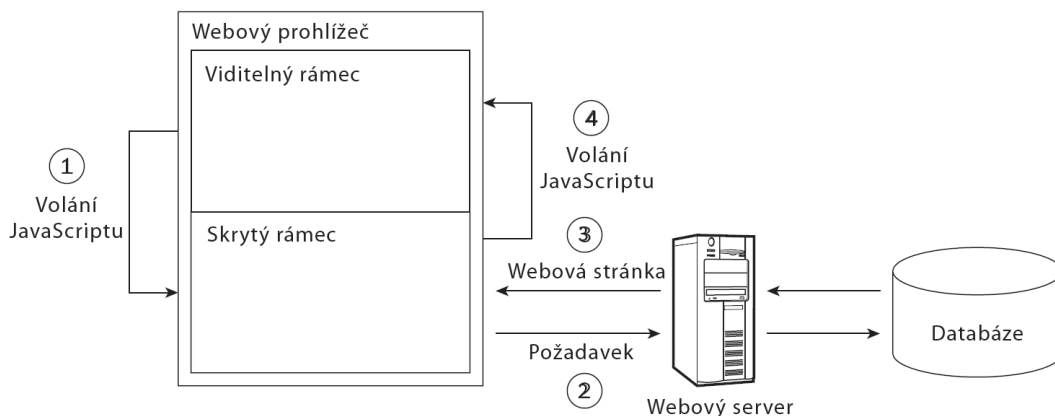
Jak bylo podrobně popsáno v kapitole 1, Ajax podporuje velké množství různých komunikačních technik. Každá z nich má výhody a nevýhody a je velmi důležité pochopit, kterou z nich je vhodné v dané situaci použít.

Technika neviditelných rámců

Technika neviditelných rámců se zrodila se vznikem HTML rámců. Základní myšlenkou v pozadí této techniky je vytvoření skupiny rámců s jedním neviditelným rámcem, který se používá pro komunikaci mezi klientem a serverem. Rámec lze skrýt tak, že mu nastavíte šířku nebo výšku na 0 pixelů, čímž jej efektivně skryjete na stránce. Tato technika je mezi vývojáři stále značně oblíbená, ačkoliv některé starší prohlížeče (jako třeba Netscape 4) neuměly rámce úplně schovat a často zobrazovaly jejich tlusté okraje. Tyto starší prohlížeče jsou ovšem již dávno minulostí.

Vzor

Technika neviditelného rámce používá speciální čtyřkrokový vzor (viz obrázek 2-1). První krok vždy začíná viditelným rámcem, kde se uživateli zobrazuje webová stránka. Uživatel ovšem neví, že stránka také obsahuje neviditelný rámec, který v moderních prohlížečích není viditelný. Se stránkou pracuje obvyklým způsobem. Ve stejný moment, kdy uživatel požaduje nějaká další data ze serveru, nastane první krok tohoto procesu – funkce JavaScriptu zavolá skrytý rámec. Toto volání může být buď velmi jednoduché, například přeměrování skrytého rámce na jinou adresu, nebo složitější, například zaslání dat prostřednictvím formuláře. Bez ohledu na složitost této funkce je výsledkem druhý krok procesu – odeslání požadavku na server.



Obrázek 2-1. Čtyřkrokový vzor.

Třetím krokem ve vzoru je odpověď obdržena od serveru. Protože pracujeme s rámci, musí být v odpovědi poslána kompletní webová stránka. Ta musí obsahovat jednak data požadovaná od serveru a dále kód JavaScriptu, který tato data přenese do viditelného rámce. Typicky se to děje odchycením události `onload` vrácené stránky, což znamená, že po načtení stránky se zavolá funkce ve viditelném rámci (to je čtvrtý krok). V tomto okamžiku se data nachází ve viditelném rámci.

Když nyní chápeme techniku neviditelných rámců, je čas se o ní dozvědět více informací. Stejně jako u každé nové techniky vede nejlepší cesta k jejímu vysvětlení přes názorný příklad. Budeme vytvářet jednoduchou vyhledávací stránku, kde pracovník zákaznického servisu může vyhledávat informace o zákaznících. Tento příklad bude velmi jednoduchý: uživatel zadá ID zákazníka a obdrží informace o něm. Bude používána databáze, takže se neobejdeme bez programování na straně serveru. Naštěstí toho programování nebude moc. Tento příklad je založen na PHP, což je výtečný open-source jazyk pro programování na straně serveru a open-source databázi MySQL, která je zdarma dostupná na adrese www.mysql.org.

V PHP 5 je standardně podpora databáze MySQL vypnuta. Pro informace, jak ji zapnout, navštivte webovou adresu <http://www.php.net/mysql/>.

Za prvé – než bude možné vyhledávat v databázi informace o nějakém uživateli, musíte mít tabulku, která bude tato data obsahovat. Tuto tabulku vytvoříte pomocí následujícího SQL skriptu:

```
CREATE TABLE 'Customers' (  
  'CustomerId' int(11) NOT NULL auto_increment,  
  'Name' varchar(255) NOT NULL default '',  
  'Address' varchar(255) NOT NULL default '',  
  'City' varchar(255) NOT NULL default '',  
  'State' varchar(255) NOT NULL default '',  
  'Zip' varchar(255) NOT NULL default '',  
  'Phone' varchar(255) NOT NULL default '',  
  'Email' varchar(255) NOT NULL default '',  
  PRIMARY KEY ('CustomerId')  
) TYPE=MyISAM COMMENT='Sample Customer Data';
```

Nejdůležitější políčko v tabulce je CustomerId, které budete používat pro vyhledání informací o daném zákazníkovi.

Tento skript, včetně vzorových dat, si můžete stáhnout z adresy www.zonerpress.cz.

S existující databázovou tabulkou je čas přesunout se k HTML kódu. Pro použití techniky neviditelných rámců musíte začít s definicí skupiny rámců:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">  
<html>  
  <head>  
    <title>Hidden Frame GET Example</title>  
  </head>  
  <frameset rows="100%,0" style="border: 0px">  
    <frame name="displayFrame" src="DataDisplay.php" noresize="noresize" />  
    <frame name="hiddenFrame" src="about:blank" noresize="noresize" />  
  </frameset>  
</html>
```

Důležitou součástí tohoto kódu je atribut rows prvku <frameset>. Jeho nastavením na hodnotu 100%, 0 se prohlížeč dozví, že tělo rámce pojmenovaného jako hiddenFrame se nemá ve stránce zobrazovat. Atribut style se používá pro nastavení nulového orámování rámce – tím se zabezpečí, že rámce nebudou mít kolem sebe nějaké orámování. Posledním důležitým krokem v této definici skupiny rámců je použití atributu noresize každého rámce, takže uživatel nebude schopen změnit velikost rámců a zjistit, co se nachází v tom skrytém rámci. Obsah neviditelného rámce by měl zůstat před uživatelem vždy skrytý.

Následuje stránka pro zadání požadavku a pro zobrazení informací o zákazníkovi (`Display.php`). Tato stránka je poměrně jednoduchá – skládá se z textového pole formuláře pro zadání ID zákazníka, tlačítka pro odeslání požadavku a prvku `<div>` pro zobrazení získaných informací:

```
<p>Enter customer ID number to retrieve information:</p>
<p>Customer ID: <input type="text" id="txtCustomerId" value="" /></p>
<p><input type="button" value="Get Customer Info"
    onclick="requestCustomerInfo()" /></p>
<div id="divCustomerInfo"></div>
```

Povšimněte si, že odesílací tlačítko volá funkci `requestCustomerInfo()`, která pomocí skrytého rámce zařídí získání informací. Jednoduše vezme hodnotu textového pole a přidá ji do dotazovacího řetězce stránky `GetCustomerData.php`, čímž tak vytvoří URL ve tvaru `GetCustomerData.php?id=23`. Tato URL je následně přiřazena skrytému rámcu. Kód této funkce je následující:

```
function requestCustomerInfo() {
    var sId = document.getElementById("txtCustomerId").value;
    top.frames["hiddenFrame"].location = "GetCustomerData.php?id=" + sId;
}
```

První krok v této funkci je získání identifikačního čísla zákazníka z textového pole. Toho je dosaženo voláním `document.getElementById()` s ID textboxu, což je `txtCustomerId`. (Hodnota proměnné obsahuje text v textovém poli.) Potom je ID přidáno do řetězce `GetCustomerData.php?id=` pro vytvoření kompletní URL. Druhý řádek vytvoří URL a nastaví ji do skrytého rámce. Pro získání odkazu na skrytý rámeček musíte nejprve pomocí objektu `top` přistoupit k nejvyššímu oknu v prohlížeči. Tento objekt má pole rámců, v němž najdeme i skrytý rámeček. Protože každý rámeček je jiný objekt okna, můžete nastavit umístění každého z nich na požadovanou URL.

Tohle je vše, co je potřeba pro získání informací. Protože se jedná o požadavek GET, který přenáší informace v dotazovacím řetězci, je jeho provedení velmi jednoduché. V této kapitole si samozřejmě ukážeme, jak s použitím skrytého rámce provést i požadavek typu POST.

Dále budeme potřebovat funkci, která zobrazí informace o zákazníkovi. Funkce `displayCustomerInfo()` bude zavolána skrytým rámcem, jakmile budou data načtena. Jediným jejím argumentem je řetězec obsahující informace, které mají být zobrazeny:

```
function displayCustomerInfo(sText) {
    var divCustomerInfo = document.getElementById("divCustomerInfo");
    divCustomerInfo.innerHTML = sText;
}
```

Ve druhém řádku této funkce je získána reference na prvek `<div>`, ve kterém mají být získaná data zobrazena. Ve třetím řádku je řetězec s informacemi o zákazníkovi (`sText`) přiřazen do vlastnosti `innerHTML` prvku `<div>`. Použití vlastnosti `innerHTML` umožní naformátovat řetězec pomocí HTML. Tím máme hotový kód hlavní stránky. Nyní je čas vytvořit logiku na straně serveru.

Stránka `GetCustomerData.php` je jednoduchá HTML stránka s PHP kódem na dvou místech:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>Get Customer Data</title>
  <?php
    // kód PHP
  ?>
  </head>
  <body><div id="divInfoToReturn"><?php echo $sInfo ?></div></body>
</html>
```

První blok PHP na této stránce obsahuje logiku pro získání dat o zákazníkovi. Druhý PHP blok pošle proměnnou `$sInfo` s daty o zákazníkovi do prvku `<div>`. Pomocí tohoto prvku `<div>` jsou data poslána do viditelného rámce. Je potřeba vytvořit javascriptovou funkci, která bude zavolána, jakmile je stránka kompletně načtena:

```
window.onload = function () {
  var divInfoToReturn = document.getElementById("divInfoToReturn");
  top.frames["displayFrame"].displayCustomerInfo(divInfoToReturn.innerHTML);
};
```

Tato funkce je přiřazena k události `windows.onload`. Nejprve získá odkaz na prvek `<div>`, který obsahuje informace o zákazníkovi. Potom s použitím pole `top.frames` přistoupí k rámci a zavolá funkci `displayCustomerInfo()` pro předání informací do vlastnosti `innerHTML` tohoto prvku `<div>`. Celé zasílání informací má na svědomí JavaScript. Ale odkud se tyto informace vezmou? Je k tomu potřeba krátký PHP kód, který je získá z databáze.

Prvním krokem při vytvoření tohoto PHP kódu je definice všech dat, která budeme potřebovat. Zde se jedná o ID zákazníka, proměnnou `$sInfo` pro vrácení informací a nezbytné informace pro přístup k databázi (databázový server, jméno databáze, uživatelské jméno, heslo a SQL dotaz):

```
<?php
  $sID = $_GET["id"];
  $sInfo = "";
  $sDBServer = "your.databaseserver";
  $sDBName = "your_db_name";
  $sDBUsername = "your_db_username";
  $sDBPassword = "your_db_password";
  $sQuery = "Select * from Customers where CustomerId=".$sID;
  // zde bude další kód
?>
```

Začátek tohoto kódu slouží pro získání argumentu `id` z dotazovacího řetězce. PHP umístí všechny argumenty dotazovacího řetězce do pole `$_GET`. Tento identifikátor je pak uložen do `$sID` a používá se k vytvoření SQL dotazu uloženého v `$sQuery`. Do proměnné `$sInfo` je vložen prázdný řetězec. Všechny zbývající proměnné v tomto kódu slouží pro vložení přístupových informací k vaší databázi, takže je nahraďte vašimi vlastními hodnotami.

Po získání informací od uživatele a nastavení přístupu k databázi bude naším dalším krokem spojení s databází, vykonání dotazu a vrácení výsledků. Pokud se v tabulce nachází zákazník s daným ID, proměnná `$sInfo` je naplněna HTML řetězcem obsahujícím všechna jeho data, včetně odkazu pro e-mailovou adresu. Pokud je ID zákazníka neplatné, proměnná `$sInfo` je naplněna chybovou hláškou, která je pak poslána zpět do viditelného rámce:

```
<?php
    $sID = $_GET["id"];
    $sInfo = "";
    $sDBServer = "your.databaseserver";
    $sDBName = "your_db_name";
    $sDBUsername = "your_db_username";
    $sDBPassword = "your_db_password";
    $sQuery = "Select * from Customers where CustomerId=".$sID;
    $oLink = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
    @mysql_select_db($sDBName) or $sInfo="Unable to open database";
    if ($sInfo == "") {
        if($oResult = mysql_query($sQuery) and mysql_num_rows($oResult) > 0) {
            $aValues = mysql_fetch_array($oResult,MYSQL_ASSOC);
            $sInfo = $aValues['Name']."<br />".$aValues['Address']."<br />".
                $aValues['City']."<br />".$aValues['State']."<br />".
                $aValues['Zip']."<br /><br />Phone: ".$aValues['Phone']."<br />".
                "<a href=\"mailto:".$aValues['Email']."\">".
                $aValues['Email']."</a>";
            mysql_free_result($oResult);
        } else {
            $sInfo = "Customer with ID $sID doesn't exist.";
        }
    }
    mysql_close($oLink);
?>
```

První dva řádky ve zvýrazněné části kódu obsahují připojení k databázi MySQL z PHP. Následuje volání funkce `mysql_query()` pro vykonání SQL dotazu. Pokud tato funkce vrátí výsledek s alespoň jedním řádkem, pak kód pokračuje získáním informací a jejich uložením do `$sInfo`. V opačném případě je do `$sInfo` vložena chybová hláška. Předposlední řádek tohoto kódu pak ukončí spojení s databází.

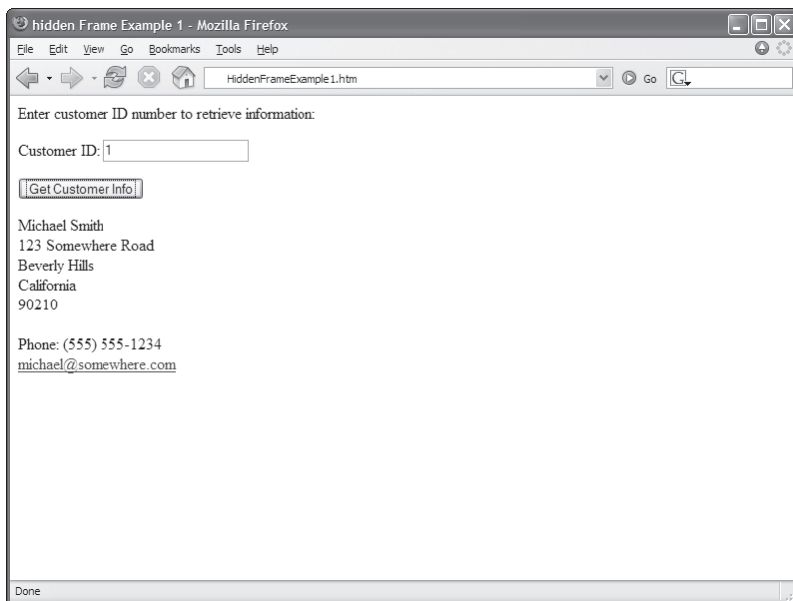
Vysvětlení detailů ohledně programování v PHP a MySQL je mimo rozsah této knihy. Pokud se chcete dozvědět více, zvažte pořízení těchto titulů z vydavatelství Zoner Press: Velká kniha PHP a MySQL 5 – kompendium znalostí pro začátečníky i profesionály nebo PhpMyAdmin – efektivní správa MySQL.

Než se posuneme dále, je nezbytný ještě jeden krok. Předcházející kód má z hlediska bezpečnosti jeden nedostatek. Protože ID zákazníka je přenášeno v dotazovacím řetězci, není bezpečné přidat jeho hodnotu přímo do SQL dotazu. Co když do něj uživatel vložil nějaký jiný SQL příkaz? Toto je tzv. útok injektáží SQL (SQL injection attack) a jedná se o velmi nebezpečný typ útoku. Oprava našeho příkladu je jednoduchá: ujistit se, že ID zákazníka je jen číslo a nic více. PHP nabízí velmi užitečnou funkci `is_numeric()`, která určí, zdali řetězec reprezentuje pouze číslo:

```
<?php
    $sID = $_GET["id"];
    $sInfo = "";
    if (is_numeric($sID)) {
        $sDBServer = "your.databaseserver";
        $sDBName = "your_db_name";
        $sDBUsername = "your_db_username";
        $sDBPassword = "your_db_password";
        $sQuery = "Select * from Customers where CustomerId=".$sID;
        $oLink = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
        @mysql_select_db($sDBName) or $sInfo="Unable to open database";
        if ($sInfo == "") {
            if($oResult = mysql_query($sQuery) and mysql_num_rows($oResult) > 0) {
                $aValues = mysql_fetch_array($oResult,MYSQL_ASSOC);
                $sInfo = $aValues['Name']."<br />".$aValues['Address']."<br />".
                    $aValues['City']."<br />".$aValues['State']."<br />".
                    $aValues['Zip']."<br /><br />Phone: ".$aValues['Phone']."<br />".
                    "<a href=\"mailto:".$aValues['Email']."\">".
                    $aValues['Email']."</a>";
                mysql_free_result($oResult);
            } else {
                $sInfo = "Customer with ID $sID doesn't exist.";
            }
        }
    } else {
        $sInfo = "Invalid customer ID.";
    }
    mysql_close($oLink);
?>
```

Přidáním této jednoduché kontroly dat se vyhnete případným útokům založeným na injektáži SQL tak, že místo zobrazení informací v takovém případě vrátíte z databáze chybovou zprávu.

Když je proměnná `@sInfo` přenesena, bude prvek `<div>` obsahovat požadované informace. Zachycení události `onload` přečte data a pošle je zpět do zobrazeného rámce ve stránce. Pokud je nějaký zákazník nalezen, budou zobrazeny informace jako na obrázku 2-2.



Obrázek 2-2. Zobrazení konkrétních informací o zákazníkovi.

Pokud daný zákazník neexistuje nebo zadané ID není číslo, bude ve stránce zobrazena chybová zpráva. Každopádně – ať už to dopadne tak, či onak, pracovník zákaznického servisu bude mít hezké uživatelské dojmy při práci s touto aplikací. A tímto končí náš první příklad v Ajaxu.

Tento příklad (a vlastně i všechny ostatní příklady uvedené v této knize) je rovněž dostupný i ve verzi pro ASP.NET a JSP. Naleznete je ke stažení na adrese www.zonerpress.cz.

Požadavek POST a neviditelný rámec

Předchozí příklad používal pro získání informací z databáze požadavek GET. Bylo to docela jednoduché, protože ID zákazníka bylo přidáno do URL a jako dotazovací řetězec posláno serveru. Ale co když chcete poslat požadavek typu POST? S použitím techniky neviditelného rámce je to rovněž možné, pouze si to vyžádá o trochu práce navíc.

Požadavek POST je obvykle použit v případě, kdy je potřeba poslat na server nějaká data (na rozdíl od požadavku GET, který data ze serveru pouze získává). A další podstatný rozdíl je ve velikosti