KAPITOLA

3

Vytváříme aplikace využívající Ajax

Témata kapitoly:

- Pracujeme s Ajaxem
- Komunikace s kódem běžícím na straně serveru
- Předávání dat skriptům běžícím na straně serveru
- Použití Ajaxu společně s XML
- Shrnutí

Je na čase začít používat Ajax. V této kapitole se dozvíte, jak vytvářet kompletní Ajaxové aplikace od podlahy až po střechu. V prvé řadě uvidíte, jak se vytváří a pracuje se s objektem `XMLHttpRequest`. Protože je tento objekt centrálním objektem celého Ajaxu, bude tento objekt srdcem i tepnami všech vašich Ajaxových aplikací.

Pracujeme s Ajaxem

Tato kapitola začíná detailním pohledem na příklad, který jsme dříve viděli v kapitole 2, konkrétně v souboru index.html, který je zobrazen na obrázku 3.1.

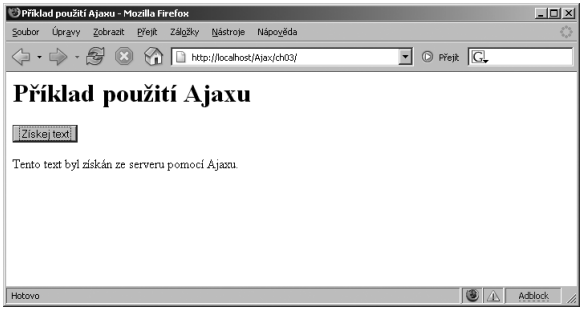
Obrázek 3.1. První Ajaxová aplikace

Tato aplikace nabízí jedno tlačítko s názvem Získej text. Když toto tlačítko stiskneme, použije se Ajax pro získání textu, na pozadí, a jeho následné zobrazení.

Data, která aplikace získává ze serveru, jsou uložena v souboru data.txt. Obsah tohoto souboru je následující:

```
Tento text byl získán ze serveru pomocí Ajaxu.
```

Poté, co stiskneme tlačítko, aplikace získá tento text a zobrazí ho bez jediného obnovení stránky. Výsledek můžete vidět na obrázku 3.2.



Obrázek 3.2. Stažení textu pomocí Ajaxu

Takto vypadá kód této aplikace:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Příklad použití Ajaxu</title>

    <script language="javascript">
      var XMLHttpRequestObjekt = false;

      if (window.XMLHttpRequest) {
        XMLHttpRequestObjekt = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      }

      function ziskejData(zdrojDat, divID)
      {
        if(XMLHttpRequestObjekt) {
          var obj = document.getElementById(divID);
          XMLHttpRequestObjekt.open("GET", zdrojDat);

          XMLHttpRequestObjekt.onreadystatechange = function()
          {
            if (XMLHttpRequestObjekt.readyState == 4 &&
              XMLHttpRequestObjekt.status == 200) {
                obj.innerHTML = XMLHttpRequestObjekt.responseText;
            }
          }

          XMLHttpRequestObjekt.send(null);
        }
      }
    </script>
  </head>

  <body>

    <H1>Příklad použití Ajaxu</H1>

    <form>
```

```
      <input type="button" value="Získej text"
        onclick="ziskejData('data.txt', 'cilovyDiv')">
    </form>

    <div id="cilovyDiv">
      <p>Získaný text se zobrazí na tomto místě.</p>
    </div>

  </body>
</html>
```

Tento kód nyní detailně probereme.

Základ aplikace

Tato aplikace po spuštění zobrazí text Získaný text se zobrazí na tomto místě. v elementu `div` s identifikátorem `cilovyDiv`. Zde se bude odehrávat většina akcí – kód aplikace zobrazí text získaný ze serveru v elementu `div`. Element `div` vypadá následovně:

```
<body>

  <H1>Příklad použití Ajaxu</H1>

  <form>
    <input type="button" value="Získej text"
      onclick="ziskejData('data.txt', 'cilovyDiv')">
  </form>

  <div id="cilovyDiv">
    <p>Získaný text se zobrazí na tomto místě.</p>
  </div>

</body>
```

Dále zde máme tlačítko, jehož parametr `onclick` je svázán s funkcí s názvem `ziskejData`:

```
<body>

  <H1>Příklad použití Ajaxu</H1>

  <form>
    <input type="button" value="Získej text"
      onclick="ziskejData('data.txt', 'cilovyDiv')">
  </form>

  <div id="cilovyDiv">
    <p>Získaný text se zobrazí na tomto místě.</p>
  </div>

</body>
```

Všimněte si, že se funkci `ziskejData` předávají dva řetězce – název souboru, který se má získat ze serveru, a identifikátor elementu `div`, ve kterém se má zobrazit výsledek. Jaký bude další krok? Podíváme se, jak v této aplikaci vypadá kód v JavaScriptu.

Vytváříme kód v JavaScriptu

Funkce `ziskejData` se nachází v elementu `script` webové stránky a bere dva argumenty – `zdrojDat` (název souboru, který se má ze serveru získat) a `divID` (identifikátor elementu `div`, ve kterém se má zobrazit výsledek):

```
<script language="javascript">

function ziskejData(zdrojDat, divID)
{
  .
  .
  .
}
</script>
```

Následně se ověří hodnota proměnné `XMLHttpRequestObjekt`, zda je rovna `null` nebo se jedná o objekt:

```
<script language="javascript">

function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    .
    .
    .
  }
}
</script>
```

Co je vlastně zač již zmíněný objekt `XMLHttpRequest`? Jedná se o objekt, který tvoří jádro Ajaxu. Objekt `XMLHttpRequest` podporují všechny moderní prohlížeče a umožňuje komunikaci se serverem na pozadí.

Tento příklad vytvoří objekt `XMLHttpRequest`, jakmile se webová stránka načte, takže je tento objekt v kódu od začátku dostupný. Pojďme se tedy podívat, jak vytvořit svůj vlastní objekt `XMLHttpRequest`.

Vytvoření objektu XMLHttpRequest

Když se stránka z našeho příkladu načte, vytvoří se proměnná s názvem `XMLHttpRequestObjekt` a nastaví se na hodnotu `false`, následovně:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  function ziskejData(zdrojDat, divID)
  {
    if(XMLHttpRequestObjekt) {
      .
      .
      .
    }
  }
</script>
```

Tato proměnná se inicializuje hodnotou `false` z toho důvodu, aby se dal případný neúspěšný pokus o vytvoření objektu `XMLHttpRequest`, po kterém bude v proměnné stále hodnota `false`, jednoduše otestovat.

Proměnné v JavaScriptu můžeme nastavit také na hodnotu true. Hodnoty `true` a `false` se nazývají Booleanovské hodnoty.

Prohlížeče Netscape Navigator (verze 7.0 a novější), Apple Safari (verze 1.2 a novější), Mozilla a Firefox umožňují vytvoření objektu `XMLHttpRequest` následujícím způsobem:

```
XMLHttpRequestObjekt = new XMLHttpRequest();
```

Jak se dozvíme, že pracujeme s prohlížečem, který je schopen objekt `XMLHttpRequest` tímto způsobem vytvořit? V těchto prohlížečích je objekt `XMLHttpRequest` součástí objektu `window`, takže můžeme otestovat, jestli existuje pomocí podmíněného bloku `if`:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    .
    .
    .
  }
  .
  .
  .
</script>
```

Pokud tento objekt v prohlížeči existuje, můžeme vytvořit objekt `XMLHttpRequest` a uložit ho do proměnné `XMLHttpRequestObjekt` následujícím způsobem:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  }
  .
  .
  .

</script>
```

Tím jsme se postarali o prohlížeče Netscape Navigator, Apple Safari. Mozilla a Firefox. A jak postupovat v případě Microsoft Internet Exploreru? V takovém případě bude objekt `window. XMLHttpRequest` nedefinován. Místo toho budeme ověřovat existenci objektu `window. ActiveXObject`, následovně:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    .
    .
    .
```

```
    }
    .
    .
    .

</script>
```

Pokud objekt `window.ActiveXObject` existuje, můžeme vytvořit objekt `XMLHttpRequest` v Internet Exploreru (verze 5.0 a novějším) následujícím způsobem (v následující kapitole si ukážeme další způsoby, jak v Internet Exploreru vytvářet objekt `XMLHttpRequest`):

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
  }
  .
  .
  .

</script>
```

Nyní máme objekt `XMLHttpRequest`, skvělé. Základní atributy a metody tohoto objektu jsou mezi všemi prohlížeči, které objekt podporují, stejné. Existují však podstatné rozdíly mezi pokročilejšími vlastnostmi tohoto objektu v jednotlivých prohlížečích.. Jaké atributy a metody tohoto objektu jsou tedy v jednotlivých prohlížečích dostupné? Atributy objektu `XMLHttpRequest` dostupné v Internet Exploreru uvádí tabulka 3.1, jeho metody pak tabulka 3.2. Atributy objektu v prohlížečích Mozilla, Netscape Navigator a Firefox uvádí tabulka 3.3, jeho metody pak tabulka 3.4. Společnost Apple dosud nezveřejnila finální seznam atributů a metod, které jejich objekt `XMLHttpRequest` podporuje. Zveřejnila však alespoň sadu běžně užívaných atributů, které najdete v tabulce 3.5, a běžně užívaných metod uvedených v tabulce 3.6.

Tabulka 3.1. Atributy objekty XMLHttpRequest v Internet Exploreru

Atribut	Popis
onreadystatechange	Odkazuje na funkci (funkce pro obsluhu události), která se zavolá při každé změně hodnoty atributu `readyState`. Atribut slouží pro čtení i zápis.
readyState	Obsahuje stav požadavku. Atribut slouží pouze pro čtení.
responseBody	Obsahuje tělo odpovědi v podobě, jak ji odeslal server. Atribut slouží pouze pro čtení.
responseStream	Obsahuje tělo odpovědi ve formě streamu. Atribut slouží pouze pro čtení.
responseText	Obsahuje tělo odpovědi ve formě řetězce. Atribut slouží pouze pro čtení.
responseXML	Obsahuje tělo odpovědi ve formě dokumentu XML. Atribut slouží pouze pro čtení.

Atribut	Popis
status	Obsahuje stavový kód (protokolu HTTP) odpovědi na požadavek. Atribut slouží pouze pro čtení.
statusText	Obsahuje textový popis odpovídající stavovému kódu. Atribut slouží pouze pro čtení.

Tabulka 3.2. Metody objekty XMLHttpRequest v Internet Exploreru

Metoda	Popis
abort	Přeruší požadavek.
getAllResponseHeaders	Vrátí všechny hlavičky protokolu HTTP.
getResponseHeader	Vrátí hodnotu hlavičky protokolu HTTP.
open	Inicializuje požadavek.
send	Odešle serveru požadavek protokolem HTTP.
setRequestHeader	Nastaví název a hodnotu hlavičky HTTP.

Tabulka 3.3. Atributy objekty XMLHttpRequest v prohlížečích Mozilla, Firefox a Netscape Navigator

Atribut	Popis
channel	Obsahuje kanál použití pro odeslání požadavku. Atribut slouží pouze pro čtení.
readyState	Obsahuje stav požadavku. Atribut slouží pouze pro čtení.
responseText	Obsahuje tělo odpovědi ve formě řetězce. Atribut slouží pouze pro čtení.
responseXML	Obsahuje tělo odpovědi ve formě dokumentu XML. Atribut slouží pouze pro čtení.
status	Obsahuje stavový kód (protokolu HTTP) odpovědi na požadavek. Atribut slouží pouze pro čtení.
statusText	Obsahuje textový popis odpovídající stavovému kódu. Atribut slouží pouze pro čtení.
onreadystatechange	Odkazuje na funkci (funkce pro obsluhu události), která se zavolá při každé změně hodnoty atributu `readyState`. Atribut slouží pro čtení i zápis.

Tabulka 3.4. Metody objekty XMLHttpRequest v prohlížečích Mozilla, Firefox a Netscape Navigator

Metoda	Popis
abort	Přeruší požadavek.
getAllResponseHeaders	Vrátí všechny hlavičky protokolu HTTP.
getResponseHeader	Vrátí hodnotu hlavičky protokolu HTTP.
openRequest	Nativní způsob inicializace požadavku.
overrideMimeType	Přepíše typ MIME, který server vrátí.
open	Inicializuje požadavek.
send	Odešle serveru požadavek protokolem HTTP.

Tabulka 3.5. Atributy objekty XMLHttpRequest v Apple Safari

Atribut	Popis
onreadystatechange	Odkazuje na funkci (funkce pro obsluhu události), která se zavolá při každé změně hodnoty atributu `readyState`. Atribut slouží pro čtení i zápis.
readyState	Obsahuje stav požadavku. Atribut slouží pouze pro čtení.
responseText	Obsahuje tělo odpovědi ve formě řetězce. Atribut slouží pouze pro čtení.
responseXML	Obsahuje tělo odpovědi ve formě dokumentu XML. Atribut slouží pouze pro čtení.
status	Obsahuje stavový kód (protokolu HTTP) odpovědi na požadavek. Atribut slouží pouze pro čtení.
statusText	Obsahuje textový popis odpovídající stavovému kódu. Atribut slouží pouze pro čtení.

Tabulka 3.6. Metody objekty XMLHttpRequest v Apple Safari

Metoda	Popis
abort	Přeruší požadavek.
getAllResponseHeaders	Vrátí všechny hlavičky protokolu HTTP.
getResponseHeader	Vrátí hodnotu hlavičky protokolu HTTP.
open	Inicializuje požadavek.
send	Odešle serveru požadavek protokolem HTTP.
setRequestHeader	Nastaví název a hodnotu hlavičky HTTP.

Nyní, když máme vytvořený objekt `XMLHttpRequest`, jakým způsobem s ním budeme pracovat? Začneme tím, že inicializujeme požadavek, což se odehrává ve funkci `ziskejData`.

Inicializace požadavku

Poté, co je objekt `XMLHttpRequest` vytvořen, je připraven k použití a čeká na svůj okamžik, kdy uživatel stiskne tlačítko. Funkce spojená se stiskem tohoto tlačítka nese název `ziskejData`:

```
<script language="javascript">

  function ziskejData(zdrojDat, divID)
  {
    .
    .
    .
  }
</script>
```

Je na čase začít pracovat s objektem `XMLHttpRequest`, který jsme vytvořili. Připojíme se s jeho pomocí k serveru a získáme z něj data. Funkce `ziskejData` začíná ověřením úspěšného vytvoření objektu `XMLHttpRequest` a případným ukončením v případě neúspěchu:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    .
    .
    .
  }
}
```

Pokud chcete, můžete v případě chyby zobrazit chybovou hlášku informující uživatele o tom, že jeho prohlížeč není kompatibilní s Ajaxem (není schopen vytvořit objekt `XMLHttpRequest`). Můžete tak učinit např. získáním objektu korespondujícího s elementem `div` se zadaným identifikátorem a nastavením kódu HTML tohoto elementu na text Váš prohlížeč nepodporuje Ajax:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    .
    .
    .
  }
  else {
    var obj = document.getElementById(divID);
    obj.innerHTML = "Váš prohlížeč nepodporuje Ajax";
  }
}
```

Prvním krokem při práci s objektem XMLHttpRequest je *inicializace požadavku*. Inicializace požadavku připraví požadavek a nakonfiguruje ho pro použití se serverem. Provádí se pomocí metody `open`, jejíž syntaxe je následující (položky v hranatých závorkách,[a], jsou nepovinné):

```
open("metoda", "URL" [, příznakAsynchronníKomunikace[,"uživatelskéJméno"
[,"heslo"]]])
```

Význam jednotlivých argumentů je následující:

- `metoda`: Metoda komunikace protokolem HTTP, jako např. `GET`, `POST`, `PUT`, `HEAD` nebo `PROPFIND`.
- `URL`: Požadované URL.
- `příznakAsynchronníKomunikace`: Booleanovský výraz indikující, zda se jedná o asynchronní volání. Výchozí hodnota je true.
- `uživatelskéJméno`: Uživatelské jméno.
- `heslo`: Heslo.

Náš příklad používá metodu `open` objektu `XMLHttpRequest` následovně. Použije metodu `GET` protokolu HTTP a předá URL souboru, který se má získat a je předán funkci `ziskejData` jako argument `zdrojDat`:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
     XMLHttpRequestObjekt.open("GET", zdrojDat);
    .
    .
    .
  }
}
```

Dvěma hlavními metodami komunikace protokolem HTTP jsou `GET` a `POST` a tento příklad používá metodu `GET`. Použití metody `POST` si ukážeme později v této kapitole. V případě použití metody `POST` se s objektem `XMLHttpRequest` pracuje trochu odlišně.

Všimněte si také, že URL předané metodě `open` je pouhým názvem souboru, data.txt, který se funkci `ziskejData` předává jako argument:

```
<form>
  <input type="button" value="Získej text"
    onclick="ziskejData('data.txt', 'cilovyDiv')">
</form>
```

Aby tento kód pracoval správně, je třeba se ujistit, že se soubor data.txt nachází na serveru ve stejném adresáři jako webová stránka index.html. Pokud se tento soubor nachází v jiném adresáři, např. podadresáři adresáře, ve kterém se nachází soubor index.html, s názvem data, je třeba URL zadat takto:

```
<form>
  <input type="button" value="Získej text"
    onclick="ziskejData('data/data.txt', 'cilovyDiv')">
</form>
```

Jak data.txt, tak data/data.txt jsou relativními URL. Je možné použít také absolutní URL, která specifikuje úplnou cestu k souboru data.txt, např. takto:

```
<form>
  <input type="button" value="Získej text"
    onclick="ziskejData('http://localhost/ch03/data.txt', 'cilovyDiv')">
</form>
```

Pokud si však přejeme přistupovat k datům v jiné doméně, než ve které se nachází soubor index.html, např. takto,

```
<form>
  <input type="button" value="Získej text"
    onclick="ziskejData('http://www.jinaDomena.cz/data.txt', 'cilovyDiv')">
</form>
```

budeme mít problémy, protože toto chování považuje prohlížeč za potenciální bezpečnostní riziko.

ODKAZ

Více se o tomto bezpečnostním omezení a způsobech, jak ho obejít, dozvíme v kapitole 4.

V rámci zachování jednoduchosti získávají příklady v této knize data ze stejného adresáře, ve kterém se nachází webová stránka. Pro tento příklad se tedy ujistěte, že se soubor data.txt nachází ve stejném adresáři jako soubor index.html (v kódu, který je možné k této knize získat, jsou oba soubory uloženy v témže adresáři).

Zpracování požadavku

Data přicházející od serveru se zpracovávají *asynchronně*, tzn. aplikace se po odeslání požadavku nezablokuje a nečeká na odpověď serveru. To také značí ono velké „A“ v názvu Ajax, Asynchronous. Co to pro nás, programátory, znamená?

Znamená to, že musíme vytvořit tzv. funkci *callback* (funkci pro obsluhu události), která se zavolá po dokončení požadavku popř. v jeho průběhu. Objekt `XMLHttpRequest` zavolá funkci callback při každé změně stavu požadavku.

Funkci callback můžeme objektu `XMLHttpRequest` přiřadit, a tím ho s ní propojit, s využitím jeho atributu `onreadystatechange`. To může vypadat např. takto, za předpokladu, že funkce, kterou přiřazujeme atributu `onreadystatechange`, nese název `callbackFunkce`:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
     XMLHttpRequestObjekt.open("GET", zdrojDat);

     XMLHttpRequestObjekt.onreadystatechange = callbackFunkce;
    .
    .
    .
  }
}

function callbackFunkce()
{
  .
  .
  .
}
```

Tato konstrukce se však v Ajaxu příliš často nepoužívá. Častěji se jako funkce callback používají tzv. anonymní funkce, které se vytváří pomocí klíčového slova `function` následovaného složenými závorkami uzavírajícími tělo funkce:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", zdrojDat);

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      .
      .
      .
    }
  }
}
```

Takovéto funkce se nazývají *anonymní*, protože nemají název. Jednoduše zadáte jejich tělo přímo do složených závorek za klíčovým slovem `function`.

Tato nová anonymní funkce callback se za nás postará o zpracování požadavku a odpovědi na něj – je upozorněna při každé změně stavu požadavku. Objekt `XMLHttpRequest` nabízí dva atributy pro ověření stavu požadavku – `readyState` a `status`.

Atribut `readyState` specifikuje aktuální stav požadavku. Možné hodnoty tohoto atributu jsou následující:

- 0 – neinicializováno
- 1 – načítání
- 2 – načteno
- 3 – interaktivní
- 4 – dokončeno

Nás bude zajímat hlavně hodnota 4, která značí, že byl požadavek zcela dokončen.

Atribut `status` obsahuje stavový kód odpovědi na požadavek. Jedná se o běžný stavový kód protokolu HTTP, který vám webový server odesílá i při běžném procházení webových stránek. Pokud se např. nepodařilo nalézt požadovaná data, bude hodnota atributu `status` 404. Zde jsou některé obvyklé hodnoty tohoto atributu:

- 200 OK
- 201 Created
- 204 No Content
- 205 Reset Content
- 206 Partial Content
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden

- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout
- 411 Length Required
- 413 Requested Entity Too Large
- 414 Requested URL Too Long
- 415 Unsupported Media Type
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway
- 503 Service Unavailable
- 504 Gateway Timeout
- 505 HTTP Version Not Supported

POZNÁMKA

Přejeme si vidět stavový kód 200, který značí, že odpověď na požadavek proběhla úspěšně.

Ve funkci callback začneme tím, že ověříme hodnotu atributu `readyState` objektu `XMLHttpRequest` a ujistíme se, že je rovna hodnotě 4, což značí dokončení požadavku:

```
<script language="javascript">
  .
  .
  .

  function ziskejData(zdrojDat, divID)
  {
    if(XMLHttpRequestObjekt) {
      XMLHttpRequestObjekt.open("GET", zdrojDat);

      XMLHttpRequestObjekt.onreadystatechange = function()
      {
        if (XMLHttpRequestObjekt.readyState == 4)
        {
          .
          .
          .
        }
      }
    }
  }
</script>
```

Následně ověříme, že atribut `status` objektu `XMLHttpRequest` obsahuje hodnotu 200 signalizující, že odpověď na požadavek byla úspěšná:

```
<script language="javascript">
  .
  .
  .

  function ziskejData(zdrojDat, divID)
  {
    if(XMLHttpRequestObjekt) {
      XMLHttpRequestObjekt.open("GET", zdrojDat);

      XMLHttpRequestObjekt.onreadystatechange = function()
      {
        if (XMLHttpRequestObjekt.readyState == 4 &&
          XMLHttpRequestObjekt.status == 200) {
          .
          .
          .
        }
      }
    }
  }
</script>
```

Požadavek i odpověď na něj proběhly úspěšně a jsou tímto dokončeny. Máme data, která jsme si pomocí Ajaxu vyžádali od serveru. Jak nyní k těmto datum přistoupit?

Zpracování odpovědi

Pokud atribut `status` objektu `XMLHttpRequest` obsahuje hodnotu 200 a atribut `readyState` obsahuje hodnotu 4, úspěšně jsme dokončili požadavek na server, skvělé. Jakým způsobem je nyní možné přistupovat k odpovědi serveru a datům v ní obsaženým?

Existují dva způsoby, oba využívající objekt `XMLHttpRequest`:

- Pokud představují získaná data prostý text, můžete je získat z atributu `responseText` objektu `XMLHttpRequest`, tj. `XMLHttpRequestObjekt.responseText`.
- Pokud jsou získaná data ve formátu XML, můžete je získat z atributu `responseXML` objektu `XMLHttpRequest`, tj. `XMLHttpRequestObjekt.responseXML`.

V našem příkladu jsou požadovaná data uložena v souboru data.txt, takže použijeme atribut `responseText`. Cílem této první aplikace je zobrazit získaná data na webové stránce, k čemuž použijeme trochu dynamického HTML. Začneme získáním objektu korespondujícího s elementem `div` s identifikátorem `cilovyDiv` webové stránky, ve kterém tento příklad zobrazuje text:

```
<script language="javascript">
  .
  .
  .

  function ziskejData(zdrojDat, divID)
  {
```

```
    if(XMLHttpRequestObjekt) {
      var obj = document.getElementById(divID);
      XMLHttpRequestObjekt.open("GET", zdrojDat);

      XMLHttpRequestObjekt.onreadystatechange = function()
      {
        if (XMLHttpRequestObjekt.readyState == 4 &&
          XMLHttpRequestObjekt.status == 200) {
          .
          .
          .
        }
      }
    }
  }
</script>
```

Následně můžeme získaný text, nacházející se v atributu `responseText` objektu `XMLHttpRequest`, zobrazit v elementu `div` s identifikátorem `cilovyDiv` následujícím způsobem:

```
<script language="javascript">
  .
  .
  .

  function ziskejData(zdrojDat, divID)
  {
    if(XMLHttpRequestObjekt) {
      var obj = document.getElementById(divID);
      XMLHttpRequestObjekt.open("GET", zdrojDat);

      XMLHttpRequestObjekt.onreadystatechange = function()
      {
        if (XMLHttpRequestObjekt.readyState == 4 &&
          XMLHttpRequestObjekt.status == 200) {
            obj.innerHTML = XMLHttpRequestObjekt.responseText;
        }
      }
    }
  }
</script>
```

Jsme tímto hotovi? Bude aplikace nyní pracovat? Ne tak rychle, stále zbývá ještě jeden krok. Inicializovali jsme požadavek tak, aby pomocí metody `GET` protokolu HTTP požádal o soubor data.txt, a vytvořili jsme kód pro zpracování požadavku a zobrazení odpovědi. Stále jsme však požadavek neodeslali a právě to nyní napravíme.

Odeslání požadavku

Úspěšně jsme inicializovali metodou `open` objektu `XMLHttpRequest` požadavek a přiřadili atributu `onreadystatechange` anonymní funkci callback. Jakým způsobem se nyní připojíme k serveru a odešleme mu tento požadavek?

Použijeme metodu `send` objektu `XMLHttpRequest` pro odeslání požadavku serveru. Pokud pro komunikaci se serverem použijete metodu `GET` protokolu HTTP, provádí se veškerá

konfigurace přímo v metodě `open`. Metodě `send` se pak jako argument předává pouze hodnota `null`. Kód vypadá takto:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
  }

  function ziskejData(zdrojDat, divID)
  {
    if(XMLHttpRequestObjekt) {
      var obj = document.getElementById(divID);
      XMLHttpRequestObjekt.open("GET", zdrojDat);

      XMLHttpRequestObjekt.onreadystatechange = function()
      {
        if (XMLHttpRequestObjekt.readyState == 4 &&
          XMLHttpRequestObjekt.status == 200) {
            obj.innerHTML = XMLHttpRequestObjekt.responseText;
        }
      }

      XMLHttpRequestObjekt.send(null);
    }
  }
</script>
```

A to je vše. Právě jste vytvořili svou první aplikaci využívající Ajax. Pracuje tak, jak ukazují obrázky 3.1 a 3.2.

Zopakujme si kroky, podle kterých jsme postupovali při vytváření naší první, velmi důležité, aplikace využívající Ajax:

1. Vytvořili jsme objekt `XMLHttpRequest`.
2. Inicializovali jsme požadavek pomocí metody open objektu XMLHttpRequest.
3. Přiřadili jsme anonymní funkci callback atributu `onreadystatechange` objektu `XMLHttpRequest`, která zpracuje změny stavu požadavku.
4. Protože jsme použili metodu `GET` protokolu HTTP, předali jsme metodě `send`, sloužící pro odeslání požadavku, pouze hodnotu `null`.

Tento postup je základním stavebním blokem Ajaxu.

Teď, když jsme dokončili naši první aplikaci využívající Ajax, je na čase ji trochu více rozvést. Nyní byste měli disponovat základními vědomostmi týkajícími se daného tématu, ale to je teprve začátek. Existují např. i další způsoby, jak vytvořit objekt `XMLHttpRequest`.

Další způsoby vytvoření objektu XMLHttpRequest

V ukázkové aplikaci, kterou jsme právě dokončili, jste měli možnost vidět jeden způsob vytvoření objektu `XMLHttpRequest`, který vypadá následovně:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  if (window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
  }
```

Tento kód je možné ještě dále rozšířit, protože v Internet Exploreru existuje více způsobů, jak vytvořit objekt `XMLHttpRequest` (také je toho více, co je třeba říct o vytváření objektu `XMLHttpRequest` v prohlížečích Mozilla a Firefox v souvislosti s prací s XML, jak se dozvíte později v této kapitole). Uvedený kód je pro potřeby této knihy dostačující, ale je třeba zmínit, že je v Internet Exploreru dostupných více verzí objektů `XMLHttpRequest`. Běžnou verzi tohoto objektu je možné vytvořit za pomocí konstruktoru `ActiveX`, dosazením argumentu `Microsoft.XMLHTTP`. Existují však i novější verze – `MSXML2.XMLHTTP`, či ještě novější `MSXML2.XMLHTTP.3.0`, `MSXML2.XMLHTTP.4.0` a nyní i `MSXML2.XMLHTTP.5.0`.

Trik spočívá v tom, ověřit, jestli prohlížeč uživatele dokáže vytvořit tyto novější verze objektu `XMLHttpRequest`, aniž by, v případě neúspěchu, došlo k výjimce. Využít k tomuto účelu můžeme konstrukci `try/catch` JavaScriptu, která dovoluje otestovat kritický kód a pokračovat, i když v něm dojde k výjimce. Řekněme, že si přejete vytvořit objekt `MSXML2.XMLHTTP` s využitím klíčového slova `try` následovně:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  try {
    XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP");
  }
</script>
```

Pokud při provádění tohoto kódu, pokoušejícího se o vytvoření objektu `XMLHttpRequest`, vznikne výjimka, je třeba ji zachytit s využitím klíčového slova `catch`. Obecně tento systém pracuje následujícím způsobem. Pokud při provádění kódu v bloku `try` dojde k výjimce, předá se řízení bloku `catch`. V případě výskytu výjimky se bloku `catch`, pomocí objektu výjimky, předává také informace o tom, proč blok `try` selhal, takto:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  try {
    XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP");
  } catch (vyjimka1) {
    .
    .
    .
  }
</script>
```

Pokud vznikne při pokusu o vytvoření objektu `XMLHttpRequest` pomocí konstruktoru `ActiveXObject("MSXML2.XMLHTTP")` výjimka, můžeme se pokusit vytvořit objekt `XMLHttpRequest` standardním způsobem, pomocí konstruktoru `ActiveXObject("Microsoft.XMLHTTP")`, opět s využitím klíčového slova `try`, následovně:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  try {
    XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP");
  } catch (vyjimka1) {
      try {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      }
  }
</script>
```

Pokud kód proběhne v pořádku, máme objekt `XMLHttpRequest`. V opačném případě můžeme explicitně nastavit proměnnou `XMLHttpRequestObjekt` na hodnotu `false`, takto:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  try {
    XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP");
  } catch (vyjimka1) {
      try {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (vyjimka2) {
        XMLHttpRequestObjekt = false;
      }
  }
</script>
```

Pokud je na konci tohoto kódu hodnota proměnné `XMLHttpRequest` rovna `false`, víme, že nemáme tu čest s prohlížečem Internet Explorer verze 5.0 nebo novějším. To znamená, že je možné vytvořit objekt `XMLHttpRequest` metodou specifickou pro prohlížeč Mozilla/Firefox, jak ukazuje následující zdrojový kód uložený v souboru index2.html:

```
<script language="javascript">
  var XMLHttpRequestObjekt = false;

  try {
    XMLHttpRequestObjekt = new ActiveXObject("MSXML2.XMLHTTP");
  } catch (vyjimka1) {
      try {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (vyjimka2) {
        XMLHttpRequestObjekt = false;
      }
  }

  if (!XMLHttpRequestObjekt && window.XMLHttpRequest) {
    XMLHttpRequestObjekt = new XMLHttpRequest();
  }
</script>
```

A to je vše. Tento kód se pokusí v Internet Exploreru vytvořit objekt `XMLHttpRequest` verze `MSXML2.XMLHTTP`.

Doposud jsme získávali nějaký text z textového souboru, ale Ajax umí mnohem více. Ajax umožňuje připojit se k serveru na pozadí a komunikovat s ním. A to znamená také určité programování na straně serveru.

Komunikace s kódem běžícím na straně serveru

Skutečná síla Ajaxu tkví v jeho kombinaci s kódem běžícím na straně serveru, protože to je to, o čem celý Ajax také je – komunikace se serverem. A aby se na serveru odehrávaly nějaké akce, je třeba kódu běžícího na straně serveru. Tato kniha používá pro kód běžící na straně serveru skriptovací jazyk PHP, ale nemusíte se bát – není třeba znát jazyk PHP, abyste tuto knihu mohli číst (kapitola 12 pro jistotu obsahuje i stručný úvod do PHP v kombinaci s Ajaxem). Kód v jazyce PHP použitý v této knize je velmi jednoduchý, a pokud znáte JavaScript, budete schopni vydedukovat, co kód provádí.

Tato kniha používá jazyk PHP, protože se jedná o nejčastější volbu v kombinaci s Ajaxem. Jazyk PHP se snadno používá a je jednoduchý na učení. Existují tisíce webových serverů podporujících jazyk PHP, takže pokud se chcete na nějakém zaregistrovat, není obtížné ho najít. I váš stávající webový server může podporovat PHP, protože ho dnes podporuje většina serverů – stačí se dotázat. Pro účely testování si můžete nainstalovat PHP na svůj vlastní počítač. Interpreter jazyka PHP si můžete zdarma stáhnout na adrese *www.php.net*, včetně kompletních instrukcí pro jeho instalaci (ve Windows může instalace představovat pouhé spuštění souboru .exe).

Namísto získávání textu ze statického souboru, data.txt, mějme na serveru např. následující skript PHP:

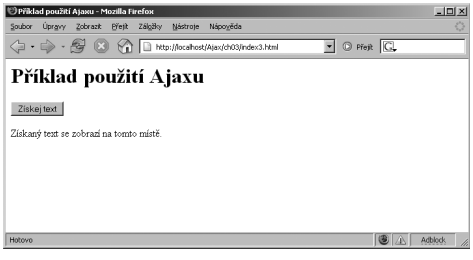
```
<?php
    echo 'Tento text byl také získán ze serveru pomocí Ajaxu.';
?>
```

Tento skript zobrazí v prohlížeči text Tento text byl také získán ze serveru pomocí Ajaxu. Neměl by tedy být problém získat tento text ze serveru a zobrazit ho v naší Ajaxové aplikaci. Získání textu ze serveru je jednoduché. Vše, co stačí pro získání textu udělat, je specifikovat URL souboru data.php (namísto data.txt), jak ukazuje příklad uložený v souboru index3.html:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Příklad použití Ajaxu</title>

    <script language="javascript">
      var XMLHttpRequestObjekt = false;

      if (window.XMLHttpRequest) {
        XMLHttpRequestObjekt = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      }

      function ziskejData(zdrojDat, divID)
      {
        if(XMLHttpRequestObjekt) {
          var obj = document.getElementById(divID);
          XMLHttpRequestObjekt.open("GET", zdrojDat);

          XMLHttpRequestObjekt.onreadystatechange = function()
```

```
            {
              if (XMLHttpRequestObjekt.readyState == 4 &&
                XMLHttpRequestObjekt.status == 200) {
                  obj.innerHTML = XMLHttpRequestObjekt.responseText;
              }
            }

            XMLHttpRequestObjekt.send(null);
          }
        }
      </script>
    </head>

    <body>

      <H1>Příklad použití Ajaxu</H1>

      <form>
        <input type="button" value="Získej text"
          onclick="ziskejData('data.php', 'cilovyDiv')">
      </form>

      <div id="cilovyDiv">
        <p>Získaný text se zobrazí na tomto místě.</p>
      </div>

    </body>
  </html>
```

Výsledek zobrazení této stránky v prohlížeči ukazuje obrázek 3.3.



Obrázek 3.3. Získání textu ze skriptu PHP pomocí Ajaxu

Když uživatel stiskne tlačítko, získá se text ze skriptu PHP, data.php, a zobrazí se, jak ukazuje obrázek 3.4. Gratuluji, právě jste úspěšně zvládli komunikaci se skriptem běžícím na straně serveru pomocí Ajaxu.

Obrázek 3.4. Získání textu ze skriptu PHP pomocí Ajaxu

Předávání dat skriptům běžícím na straně serveru

Skript data.php tedy pracuje. Stále jsou však data, která server vrací, statická. Ani tomu nemůže být jinak, protože nepředáváme skriptu data.php žádná data, která by určitým způsobem ovlivnila jeho výstup. Nic nám však nebrání serveru předat potřebná data a způsob, jakým to provedeme, se liší v závislosti na tom, kterou z metod protokolu HTTP, `GET` nebo `POST`, použijeme.

Použití metody `GET` je jednodušší, tou proto začneme.

Předávání dat serveru metodou GET

Řekněme, že je naším cílem, aby nám server vrátil jednu ze dvou zpráv v závislosti na datech, která mu předáme. Např. pokud předáme serveru hodnotu 1, odpoví server zprávou Zadali jste hodnotu 1, a pokud předáme hodnotu 2, odpoví zprávou Zadali jste hodnotu 2.

Implementovat tuto funkcionalitu v PHP je snadné. Data se serveru předávají pomocí pojmenovaných parametrů. V tomto případě nazveme pro jednoduchost náš parametr `data`. Skript PHP v tomto případě nazveme zpracovaniDat.php. Protože předáváme skriptu data s použitím metody `GET` protokolu HTTP, můžeme k těmto datům ve skriptu PHP přistupovat pomocí speciálního pole s názvem `$_GET`. Pokud např. přiřadíme parametru `data` hodnotu 1 a předáme ho skriptu zpracovaniDat.php, vyhodnotí se podmínka následujícího podmíněného bloku `if` jako pravdivá:

```
<?
if ($_GET["data"] == "1") {
  .
  .
  .
}
?>
```

V tomto případě odešleme prohlížeči nazpět zprávu Zadali jste hodnotu 1 následujícím způsobem:

```
<?
if ($_GET["data"] == "1") {
  echo 'Zadali jste hodnotu 1';
}
?>
```

Pokud má naopak parametr `data` hodnotu 2, odešleme nazpět zprávu Zadali jste hodnotu 2 následujícím způsobem:

```
<?
if ($_GET["data"] == "1") {
  echo 'Zadali jste hodnotu 1';
}
if ($_GET["data"] == "2") {
  echo 'Zadali jste hodnotu 2';
}
?>
```

Tento krátký kód v jazyce PHP zvládne vše potřebné. Když mu předáme parametr s názvem `data` a hodnotou 1, zobrazí se první zpráva. Pokud naopak předáme parametr s názvem `data` a hodnotou 2, zobrazí se druhá zpráva.

Nyní je vhodnou otázkou, jak se předávají data serveru, s nímž komunikujeme metodou `GET` protokolu HTTP, pomocí Ajaxu. V případě použití metody GET se data odesílají serveru *zakódovaná v URL*, což znamená, že se data připojí k vlastnímu URL, na které se odkazujeme.

Máme-li např. webovou stránku s textovým polem s názvem `a` obsahujícím hodnotu 5, textovým polem `b` obsahujícím hodnotu 6 a textovým polem `c` obsahujícím text hodnota pole c, pak se v případě použití metody `GET` všechna tato data přidají do URL, ke které přistupujeme. Názvy textových polí, `a`, `b` a `c`, se stanou parametry předávanými serveru a obsah jednotlivých textových polí se přiřadí odpovídajícím parametrům.

Pokud se data předávají v URL, přidá se na konec URL otazník (?) následovaný daty ve formátu `parametr=hodnota`. Mezery v textu se zakódují jako znaky + a jednotlivé páry parametrů a jejich hodnot se od sebe oddělí pomocí znaku &. Pro předání obsahu našich textových polí s názvy `a`, `b` a `c` na adresu *http://www.nazevServeru.cz/uzivatel/skript* metodou `GET` bychom tedy použili následující URL:

http://www.nazevServeru.cz/uzivatel/skript?a=5&b=6&c=hodnota+pole+c

POZNÁMKA

Data, která tímto způsobem předáváme, se vždy považují za textové řetězce. I když odesíláme čísla, bude se s nimi nakládat jako s řetězci.

Takto se tedy odesílají data z ovládacích prvků, jako jsou textová pole – použitím názvů prvků coby parametrů a přiřazením obsahu prvků těmto parametrům. V našem příkladu, zpracovaniDat.php, však žádná textová pole nemáme, takže jen přiřadíme hodnotu 1, resp. 2 do parametru `data`, aby mohl skript PHP zobrazit adekvátní text. Např. následující URL předává parametr s názvem `data` a hodnotou 1 skriptu zpracovaniDat.php:

```
zpracovaniDat.php?data=1
```

Takto vypadá kód, uložený v souboru zpracovaniDat.html, který zobrazí dvě tlačítka, s jejichž pomocí může uživatel rozhodnout, kterou ze dvou zpráv si přeje zobrazit:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Příklad použití Ajaxu</title>

    <script language="javascript">
      var XMLHttpRequestObjekt = false;

      if (window.XMLHttpRequest) {
        XMLHttpRequestObjekt = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      }

      function ziskejData(zdrojDat, divID)
      {
        if(XMLHttpRequestObjekt) {
```

```
            var obj = document.getElementById(divID);
            XMLHttpRequestObjekt.open("GET", zdrojDat);

            XMLHttpRequestObjekt.onreadystatechange = function()
            {
              if (XMLHttpRequestObjekt.readyState == 4 &&
                XMLHttpRequestObjekt.status == 200) {
                  obj.innerHTML = XMLHttpRequestObjekt.responseText;
              }
            }

            XMLHttpRequestObjekt.send(null);
          }
        }
      </script>
    </head>

    <body>

      <H1>Příklad použití Ajaxu</H1>

      <form>
        <input type="button" value="Získej text 1"
          onclick="ziskejData('zpracovaniDat.php?data=1', 'cilovyDiv')">
        <input type="button" value="Získej text 2"
          onclick="ziskejData('zpracovaniDat.php?data=2', 'cilovyDiv')">
      </form>

      <div id="cilovyDiv">
        <p>Získaný text se zobrazí na tomto místě.</p>
      </div>

    </body>
  </html>
```

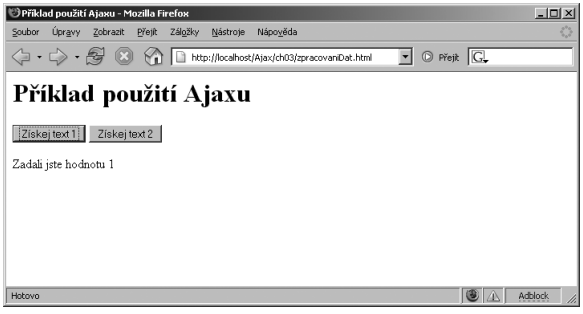
Výsledek zobrazení této stránky v prohlížeči ukazuje obrázek 3.5.

Obrázek 3.5. Předávání dat serveru metodou GET

Poté, co uživatel stiskne tlačítko, získá se ze serveru odpovídající text a zobrazí se, jak ukazuje obrázek 3.6. Právě jsme úspěšně předali data skriptu běžícímu na straně serveru pomocí Ajaxu a metody `GET`.

Tento příklad využívající metodu `GET` pracuje správně, ale má jednu nevýhodu – odesílaná data jsou příliš na očích. To je zřejmé, neboť jsou přidaná na konec URL, ke které se přistupuje,

což znamená, že je může každý snadno přečíst. O něco bezpečnější je v tomto smyslu metoda `POST`, která vkládá data do těla zprávy HTTP.



Obrázek 3.6. Zobrazení textu získaného ze serveru

Předávání dat serveru metodou POST

Metoda `POST` vkládá veškerá přenášená data do těla zprávy HTTP, která tak nejsou přímo na očích (což přirozeně neznamená, že jsou lépe zabezpečena), jako je tomu v případě metody `GET`. Právě z tohoto důvodu se v některých situacích metoda `POST` před metodou `GET` preferuje.

V případě použití metody `POST` v kombinaci s Ajaxem je třeba trochu pozměnit kód, jak také uvidíme. Stejně tak se změní kód skriptu PHP, kde se namísto pole `$_GET`:

```
<?
if ($_GET["data"] == "1") {
  echo 'Zadali jste hodnotu 1';
}
if ($_GET["data"] == "2") {
  echo 'Zadali jste hodnotu 2';
}
?>
```

použije pole $_POST, tak jak ukazuje soubor `zpracovaniDatPost.php`:

```
<?
if ($_POST["data"] == "1") {
  echo 'Zadali jste hodnotu 1';
}
if ($_POST["data"] == "2") {
  echo 'Zadali jste hodnotu 2';
}
?>
```

Nyní modifikujeme kód klientské aplikace tak, aby používala metody `POST` namísto metody `GET`. První krok je zřejmý, změníme řádek kódu, na kterém specifikujeme použití metody `GET`:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("GET", zdrojDat);
    .
    .
    .
```

tak, aby se použila metoda `POST`:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("POST", zdrojDat);
    .
    .
    .
```

Aby bylo možné použít metodu `POST`, je také zapotřebí nastavit hlavičku požadavku HTTP, konkrétně hlavičku `Content-type`, a to na hodnotu `application/x-www-form-urlencoded`. Pokud nevíte, co to znamená, jen se ujistěte, že jste přidali následující kód:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("POST", zdrojDat);
    XMLHttpRequestObjekt.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');
    .
    .
    .
```


Následuje standardní anonymní funkce pro zpracování změn stavů požadavku, která vypadá následovně:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("POST", zdrojDat);
    XMLHttpRequestObjekt.setRequestHeader('Content-Type',
     'application/x-www-form-urlencoded');

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
          XMLHttpRequestObjekt.status == 200) {
            obj.innerHTML = XMLHttpRequestObjekt.responseText;
      }
    }
    .
    .
    .
```

Zatím to jde dobře, ale pokud se v případě metody `POST` nevkládají data do URL, jak tedy přiřadíme parametru `data` hodnotu 1, resp. 2 v závislosti na tom, jaké tlačítko uživatel stiskne, a odešleme tato data serveru?

Začneme modifikací obou tlačítek tak, aby předávala funkci `ziskejData` informaci o zprávě, která se má získat, 1 nebo 2, následovně:

```
function ziskejData(zdrojDat, divID)
{
  if(XMLHttpRequestObjekt) {
```

```
        var obj = document.getElementById(divID);
        XMLHttpRequestObjekt.open("POST", zdrojDat);
        XMLHttpRequestObjekt.setRequestHeader('Content-Type',
          'application/x-www-form-urlencoded');
         XMLHttpRequestObjekt.onreadystatechange = function()
        {
          if (XMLHttpRequestObjekt.readyState == 4 &&
            XMLHttpRequestObjekt.status == 200) {
              obj.innerHTML = XMLHttpRequestObjekt.responseText;
          }
        }
        .
        .
        .
      }
    }
  </script>
</head>

<body>

  <H1>Příklad použití Ajaxu a metody POST</H1>

  <form>
    <input type="button" value="Získej text 1"
      onclick="ziskejData('zpracovaniDatPost.php', 'cilovyDiv', 1)">
    <input type="button" value="Získej text 2"
      onclick="ziskejData('zpracovaniDatPost.php', 'cilovyDiv', 2)">
  </form>
  .
  .
  .
```

Následně modifikujeme funkci `ziskejData` tak, aby přijímala třetí argument s názvem `data`, tj. hodnotu 1 nebo 2, v závislosti na tom, jaké tlačítko uživatel stisknul a kterou zprávu si tedy přeje vidět:

```
function ziskejData(zdrojDat, divID, data)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("POST", zdrojDat);
    XMLHttpRequestObjekt.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
          obj.innerHTML = XMLHttpRequestObjekt.responseText;
      }
    }
    .
    .
    .
  }
}
```

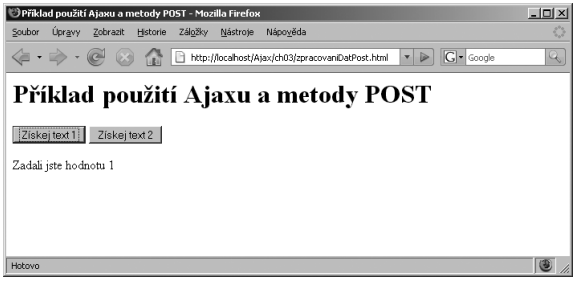
Nyní přichází hlavní změna. Namísto toho, aby se metodou `send` objektu `XMLHttpRequest` odesílala hodnota `null`, jako je tomu v případě metody `GET`, tentokrát odešleme serveru metodou `send` vlastní data. Data se odesílají ve stejném formátu, jako kdyby se vkládala do URL. Např. parametr s názvem `data` a jeho hodnotu představovanou argumentem `data` funkce `ziskejData` odešleme následujícím způsobem:

```
function ziskejData(zdrojDat, divID, data)
{
  if(XMLHttpRequestObjekt) {
    var obj = document.getElementById(divID);
    XMLHttpRequestObjekt.open("POST", zdrojDat);
    XMLHttpRequestObjekt.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
          obj.innerHTML = XMLHttpRequestObjekt.responseText;
      }
    }

    XMLHttpRequestObjekt.send("data=" + data);
  }
}
```

A tím jsme hotovi. Nyní používáme metodu `POST` namísto metody `GET` pro odeslání dat serveru. Stránku zpracovaniDatPost.html ukazuje obrázek 3.7.

Obrázek 3.7. Předávání dat serveru metodou POST

Stejně jako tomu bylo v případě metody `GET`, pokud uživatel stiskne tlačítko, získá se ze serveru odpovídající text a zobrazí se, tak jak ukazuje obrázek 3.8.



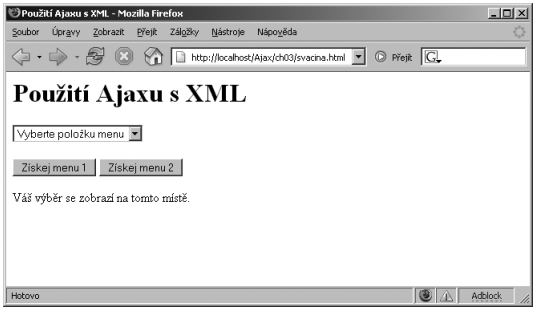
Obrázek 3.8. Zobrazení textu získaného ze serveru

Nyní již máte docela dobré základy Ajaxu, ale stále jsme neprobrali jeden důležitý aspekt. Co XML? Zkratka Ajax přeci znamená Asynchronous JavaScript and XML. XML je věnovaná následující část kapitoly.

Použití Ajaxu společně s XML

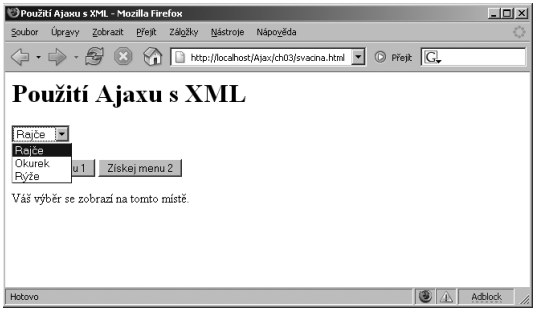
Příklady na Ajax, které jsme v této kapitole doposud měli možnost vidět, získávají ze serveru textová data. Samozřejmě se však dají použít i data ve formátu XML. To vyžaduje jednak použití atributu `responseXML` objektu `XMLHttpRequest` namísto atributu `responseText`, ale je toho ještě více. Musíte např. použít JavaScript pro parsing dokumentu XML získaného od serveru, a to zdaleka není triviální (této problematice se věnuje kapitola 9).

Pojďme se podívat na příklad, který pomocí Ajaxu získá ze serveru dokument XML. Jeden takový, uložený v souboru svacina.html, můžete vidět na obrázku 3.9.



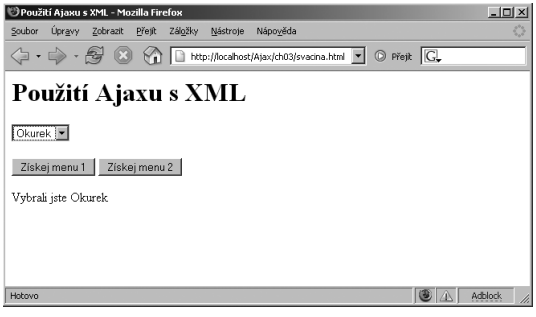
Obrázek 3.9. Webová stránka svacina.html

Tuto stránku můžete použít pro výběr mezi dvěma menu. Když vyberete menu stisknutím jednoho z tlačítek, získají se položky vybraného menu ve formátu XML ze serveru a zobrazí se v nabídce, jak ukazuje obrázek 3.10.



Obrázek 3.10. Získání položek menu ze serveru ve formátu XML

Když vyberete z nabídky nějakou položku, zobrazí se váš výběr ve spodní části stránky, jak ukazuje obrázek 3.11.



Obrázek 3.11. Výběr položky menu

Jakým způsobem tento příklad pracuje?

Vytvoření dokumentu XML

Potřebný kód XML pro tento příklad bude uložen ve dvou dokumentech XML – menu1.xml a menu2.xml. Každý z těchto dokumentů XML začíná, stejně jako jakýkoli jiný dokument XML, deklarací XML:

```
<?xml version="1.0" ?>
.
.
.
```

Kořenovým elementem v obou dokumentech XML je element `menu` (v XML se musí všechny elementy nacházet v jednom, kořenovém, elementu):

```
<?xml version="1.0" ?>
<menu>
.
.
.
</menu>
```

Jednotlivé položky menu pak vložíme do elementu `polozkamenu`, stejně jako je tomu v souboru menu1.xml:

```
<?xml version="1.0" ?>
<menu>
  <polozkamenu>Párek</polozkamenu>
  <polozkamenu>Kuře</polozkamenu>
  <polozkamenu>Steak</polozkamenu>
</menu>
```

Takto vypadá soubor menu2.xml:

```
<?xml version="1.0" ?>
<menu>
  <polozkamenu>Rajče</polozkamenu>
  <polozkamenu>Okurek</polozkamenu>
  <polozkamenu>Rýže</polozkamenu>
</menu>
```

Tyto dokumenty XML uložíme do stejného adresáře, ve kterém se nachází webová stránka svacina.html. Tím máme připraveny potřebné dokumenty XML. Jak je získáme ze serveru a jakým způsobem je použijeme?

Získání dokumentu XML ze serveru

Je na čase vytvořit kód v JavaScriptu, který získá a dekóduje dokumenty XML s názvy menu1. xml a menu2.xml. První změnu oproti práci s běžným textem je třeba provést již při vytváření objektu `XMLHttpRequest`. Pokud máme co do činění s prohlížeči Netscape/Firefox, je třeba pro práci s XML přidat následující řádek kódu:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Použití Ajaxu s XML</title>

    <script language="javascript">
      var menu;

      var XMLHttpRequestObjekt = false;

      if (window.XMLHttpRequest) {
        XMLHttpRequestObjekt = new XMLHttpRequest();
        XMLHttpRequestObjekt.overrideMimeType("text/xml");
      } else if (window.ActiveXObject) {
        XMLHttpRequestObjekt = new ActiveXObject("Microsoft.XMLHTTP");
      }
      .
      .
      .
```

Proč je uvedený řádek kódu zapotřebí a k čemu slouží? V některých případech mohou prohlížeče Mozilla, Firefox a Netscape nastavit typ Mime dat přenášených objektem `XMLHttpRequest` na hodnotu nepodporující XML, takže je při práci s XML vhodné použít uvedený řádek kódu pro nastavení očekávaného typu dat na XML.

Následně se podívejme na obsah webové stránky. Nabídka představovaná elementem `select`, dvě tlačítka a element `div` pro zobrazení výsledku:

```
<body>

  <h1>Použití Ajaxu s XML</h1>

  <form>
    <select size="1" id="seznamPolozek" onchange="vyberPolozku()">
      <option>Vyberte položku menu</option>
    </select>
    <br>
    <br>
    <input type="button" value="Získej menu 1"
      onclick="ziskejMenu1()">
    <input type="button" value="Získej menu 2"
      onclick="ziskejMenu2()">
  </form>

  <div id="cilovyDiv" width="100" height="100">Váš výběr se zobrazí na
    tomto místě.</div>

</body>
```


Všimněte si funkcí v JavaScriptu, na které se tyto ovládací prvky odkazují. Tlačítka, pomocí kterých uživatel vybírá mezi prvním a druhým menu, jsou svázána s funkcemi `ziskejMenu1`, resp. `ziskejMenu2`. Nabídka zobrazující položky vybraného menu je svázána s funkcí `vyberPolozku`, která zobrazí výběr uživatele v elementu `div`.

Jak tedy vypadají funkce `ziskejMenu1` a `ziskejMenu2`? Hned v úvodu funkce `ziskejMenu1` inicializujeme požadavek tak, aby metodou `GET` požádal o soubor menu1.xml:

```
function ziskejMenu1()
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", "menu1.xml");
    .
    .
    .
```

Při zpracování odpovědi serveru použijeme namísto atributu `responseText` objektu `XMLHttpRequest` atribut `responseXML`. Potřebná změna v kódu, který nyní uloží získaný dokument XML do proměnné s názvem `xmlDokument`, vypadá takto:

```
function ziskejMenu1()
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", "menu1.xml");

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
        var xmlDokument = XMLHttpRequestObjekt.responseXML;
        .
        .
        .
```

```
        }
      }

      XMLHttpRequestObjekt.send(null);
    }
  }
```

Tímto způsobem se dokument XML uložený do proměnné `xmlDokument`, který se získá ze serveru, uloží ve formě objektu DOM, který JavaScript podporuje. V průběhu této knihy si ukážeme, jakým způsobem je možné s dokumenty XML nakládat. Dokument XML, který jsme v tomto případě získali vypadá následovně:

```
<?xml version="1.0" ?>
<menu>
  <polozkamenu>Párek</polozkamenu>
  <polozkamenu>Kuře</polozkamenu>
  <polozkamenu>Steak</polozkamenu>
</menu>
```

Trik spočívá v tom extrahovat z tohoto dokumentu data z elementů `polozkamenu` a zobrazit je v nabídce. Pole elementů `polozkamenu` můžeme získat pomocí metody `getElementsByTagName` tak, že jí jako argument předáme název elementu, tj. `polozkamenu`:

```
function ziskejMenu1()
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", "menu1.xml");

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
        var xmlDokument = XMLHttpRequestObjekt.responseXML;
        menu = xmlDokument.getElementsByTagName("polozkamenu");
        .
        .
        .
      }
    }

    XMLHttpRequestObjekt.send(null);
  }
}
```

Nyní potřebujeme extrahovat z pole s názvem `menu` názvy jednotlivých položek menu nacházející se v elementech `polozkamenu`. K tomuto účelu vytvoříme novou funkci s názvem `vypisMenu`:

```
function ziskejMenu1()
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", "menu1.xml");

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
```

```
            var xmlDokument = XMLHttpRequestObjekt.responseXML;
            menu = xmlDokument.getElementsByTagName("polozkamenu");
            vypisMenu();
          }
        }

        XMLHttpRequestObjekt.send(null);
    }
}
```

Funkce `vypisMenu` projde cyklem přes všechny elementy `polozkamenu` v poli `menu`:

```
function vypisMenu()
{
  var citac;

  for (citac = 0; citac < menu.length; citac++ )
  {
    .
    .
    .
  }
}
```

Tělo cyklu je tím pravým místem, kde se budou vkládat do nabídky, tj. elementu `select`, nové položky menu. K obsahu elementu `select`, tj. k jednotlivým položkám nabídky, je možné přistupovat pomocí pole `options` následujícím způsobem:

```
function vypisMenu()
{
  var citac;
  var prvekSelect = document.getElementById('seznamPolozek');

  for (citac = 0; citac < menu.length; citac++ )
  {
    prvekSelect.options[citac] =
    .
    .
    .
  }
}
```

Zbývá nějakým způsobem extrahovat názvy jednotlivých položek menu, tedy obsah elementů `polozkamenu`:

```
<polozkamenu>Párek</polozkamenu>

<polozkamenu>Kuře</polozkamenu>

<polozkamenu>Steak</polozkamenu>
```

Na tomto místě už je nutná určitá znalost formátu XML. Např. text „Párek“ v prvním elementu `polozkamenu` se ve skutečnosti považuje za textový uzel a je prvním potomkem elementu `polozkamenu`. Abychom tedy k tomuto textovému uzlu přistoupili, použijeme výraz `menu[0].firstChild`. Tím to však nekončí. Protože potřebujeme data uložená v textovém uzlu („Párek“), nestačí nám přistoupit k uzlu jako takovému, ale musíme přistoupit k jeho obsahu. To zajistí výraz `menu[0].firstChild.data`.

Na základě těchto informací můžeme získat názvy položek menu a vložit je do elementu `select` následovně:

```
function vypisMenu()
{
  var citac;
  var prvekSelect = document.getElementById('seznamPolozek');

  for (citac = 0; citac < menu.length; citac++ )
  {
    prvekSelect.options[citac] = new
      Option(menu[citac].firstChild.data);
  }
}
```

Tímto způsobem vyplníme nabídku položkami menu získanými z dokumentu XML.

Poslední částí kódu tohoto příkladu je funkce `vyberPolozku`, která slouží ke zpracování výběru uživatele z nabídky. Když uživatel učiní výběr z nabídku, zobrazíme jeho výběr tak, jak už bylo dříve znázorněno na obrázku 3.11. Funkce `vyberPolozku` vypadá následovně:

```
function vyberPolozku()
{
  document.getElementById('cilovyDiv').innerHTML =
    "Vybrali jste " + menu[document.getElementById
      ('seznamPolozek').selectedIndex].firstChild.data;
}
```

A tím jsme hotovi, příklad je dokončen.

Na druhou stranu, tento příklad používal pouze statické dokumenty XML. Co když si ale přejete od serveru získat dokument XML generovaný na základě dat, která serveru odešlete? Co kdybychom např. chtěli upravit tento příklad tak, aby namísto dvou dokumentů XML, menu1.xml a menu2.xml, používal jeden soubor PHP s názvem menu.php? Soubor menu.php může odeslat prohlížeči data jak pro první, tak pro druhé menu. Stačí tomuto skriptu předat parametr s názvem `menu` s hodnotou 1, pokud máme zájem o první menu, nebo hodnotou 2, pokud máme zájem o druhé menu.

Soubor menu.php začíná nastavením hlavičky s názvem `Content-type` na hodnotu `text/xml`, aby prohlížeč věděl, že se jedná o dokument XML:

```
<?
header("Content-type: text/xml");
.
.
.
?>
```

Následně se v souboru menu.php vytvoří pole položek menu v závislosti na hodnotě parametru `menu`:

```
<?
header("Content-type: text/xml");
if ($_GET["menu"] == "1")
  $polozkymenu = array('Párek', 'Kuře', 'Steak');
if ($_GET["menu"] == "2")
  $polozkymenu = array('Rajče', 'Okurek', 'Rýže');
```

```
.
.
.
?>
```

Zbytek skriptu PHP cyklem projde přes jednotlivé prvky pole a vytvoří dokument XML, který se odešle zpět prohlížeči:

```
<?
header("Content-type: text/xml");
if ($_GET["menu"] == "1")
  $polozkymenu = array('Párek', 'Kuře', 'Steak');
if ($_GET["menu"] == "2")
  $polozkymenu = array('Rajče', 'Okurek', 'Rýže');

echo '<?xml version="1.0" ?>';
echo '<menu>';
foreach ($polozkymenu as $polozka)
{
  echo '<polozkamenu>';
  echo $polozka;
  echo '</polozkamenu>';
}
echo '</menu>';
?>
```

Tím jsme se postarali o serverovou část aplikace. Jak bude nyní vypadat klientská část aplikace využívající Ajax, kterou v tomto případě nazveme např. svacina2.html?

Provedené změny jsou jednoduché. Namísto funkcí `ziskejMenu1` a `ziskejMenu2` můžeme použít jedinou funkci `ziskejMenu`, které argumentem předáme číslo menu, o které máme zájem. Při stisknutí tlačítka uživatelem pak stačí zavolat funkci `ziskejMenu` s patřičným číslem menu jako argumentem:

```
<form>
  <select size="1" id="seznamPolozek" onchange="vyberPolozku()">
    <option>Vyberte položku menu</option>
  </select>
  <br>
  <br>
  <input type="button" value="Získej menu 1"
    onclick="ziskejMenu('1')">
  <input type="button" value="Získej menu 2"
   onclick="ziskejMenu('2')">
</form>
```

Ve funkci `ziskejMenu` zakódujeme číslo menu, které si přejeme získat do URL, a předáme ho tak skriptu menu.php:

```
function ziskejMenu(cisloMenu)
{
  if(XMLHttpRequestObjekt) {
    XMLHttpRequestObjekt.open("GET", "menu.php?menu=" + cisloMenu);

    XMLHttpRequestObjekt.onreadystatechange = function()
    {
      if (XMLHttpRequestObjekt.readyState == 4 &&
        XMLHttpRequestObjekt.status == 200) {
```

```
            var xmlDokument = XMLHttpRequestObjekt.responseXML;
            menu = xmlDokument.getElementsByTagName("polozkamenu");
            vypisMenu();
          }
        }

        XMLHttpRequestObjekt.send(null);
      }
    }
```

A to je vše, co bylo zapotřebí. Nyní když uživatel stiskne první tlačítko, získá se menu č.1, když stiskne druhé tlačítko, získá se menu č.2. Skvěle, právě jsme úspěšně získali dynamická data (dokument XML) ze serveru.

Shrnutí

V této kapitole jsme si ukázali, jak vytvářet objekt `XMLHttpRequest` a jak začít vytvářet Ajaxové aplikace. Viděli jsme, jak se používají metody `GET` a `POST` protokolu HTTP, a ukázali jsme si, jak se ze serveru získává text i dokumenty XML. Tyto informace by měly být dobrým základem pro zbytek této knihy.