Kapitola 16

Knihovna jQuery, technologie Ajax a datový formát HTML, XML, JSON a JSONP

Jonathan Sharp

16.1 Úvod

Weboví vývojáři pracují s mnoha datovými formáty a protokoly při přenosu informací mezi webovým prohlížečem a serverem. V této kapitole se seznámíme s několika recepty pro práci s nejběžnějšími datovými formáty, s technologií Ajax a knihovnou jQuery.

16.2 Knihovna jQuery a technologie Ajax

Problém

Chceme odeslat na server požadavek na další data, aniž by návštěvník musel opustit aktuální stránku.

Řešení

Zde je jednoduchý ajaxový požadavek:

```
(function($) {
  $(document).ready(function() {
    $('#update').click(function() {
      $.ajax({
        type: 'GET',
        url: 'hello-ajax.html',
        dataType: 'html',
        success: function(html, textStatus) {
          $('body').append(html);
        },
        error: function(xhr, textStatus, errorThrown) {
          alert('Nastala chyba. ' + (errorThrown? errorThrown :
            xhr.status );
        }
```

```
      });
    });
  });
})(jQuery);
```

Diskuze

Ve středu ajaxové architektury knihovny jQuery stojí metoda `jQuery.ajax()`. Ta představuje hlavní součást pro odesílání požadavků na server a přijímání odpovědí z něj ve všech moderních webových prohlížečích. Požadavek odešleme na server tak, že předáme objekt s možnostmi nastavení pro daný požadavek metodě `$.ajax()`. Je k dispozici široká nabídka možností nastavení, z nichž nejběžnější jsou `type`, `url`, `complete`, `dataType`, `error` a `success`:

```
var options = {
  type: 'GET'
};
```

První možnost nastavení, jež musíme uvést před odesláním ajaxového požadavku, je typ požadavku protokolu HTTP, kterým většinou bude typ GET nebo POST:

```
var options = {
  type: 'GET',
  url: 'hello-ajax.html',
  dataType: 'html'
};
```

Dále si vysvětlíme možnosti nastavení `url` a `dataType`. Název možnosti nastavení `url` mluví sám za sebe, ale stojí za zmínku následující interakce. Když nastavíme hodnotu `false` možnosti nastavení `cache`, knihovna jQuery přidá definici proměnné _=<náhodné číslo> k řetězci dotazu v adrese URL (například */server-ajax-gateway?_=6273551235126*), díky čemuž zabrání tomu, aby prohlížeč, server proxy nebo webový server vrátil odpověď uloženou v mezipaměti. Konečně pomocí možnosti nastavení `dataType` specifikujeme formát dat, který by měla mít odpověď ze serveru. Jestliže kupříkladu očekáváme, že server vrátí data ve formátu XML, použijeme hodnotu `xml`:

```
var options = {
  type: 'GET',
  url: 'hello-ajax.html',
  dataType: 'html',
  error: function(xhr, textStatus, errorThrown) {
    alert('Nastala chyba. ' + errorThrown);
  },
  success: function(data, textStatus) {
    $('body').append(data);
  }
};
```

Nyní definujeme další dvě možnosti nastavení, a to dvě funkce zpětného volání – jednu s názvem `error()` a druhou s názvem `success()`. Obě fungují přesně, jak napovídají jejich názvy – funkce `error()` se volá, pokud nastane chyba požadavku a funkce `success()` se volá při úspěšném přijetí odpovědi (to knihovna jQuery pozná podle toho, že server vrátí odpověď se stavem 200). Mezi další běžné možnosti nastavení patří možnost nastavení `complete`, jež definuje funkci zpětného volání, která se spustí při úspěšné i chybné odpovědi:

```
var options = {
  type: 'GET',
  url: 'hello-ajax.html',
  dataType: 'html',
  complete: function(xhr, textStatus) {
    // Kód pro zpracování odpovědi.
  }
};
```

Jakmile definujeme možnosti nastavení, můžeme odeslat náš požadavek:

```
var options = {
  type: 'GET',
  url: 'hello-ajax.html',
  dataType: 'html',
  complete: function(xhr, textStatus) {
    // Kód pro zpracování odpovědi.
  }
};
$.ajax(options);
```

Rovněž můžeme nastavit naše možnosti nastavení přímo uvnitř volání metody `$.ajax()`:

```
$.ajax({
  type: 'GET',
  url: 'hello-ajax.html',
  dataType: 'html',
  complete: function(xhr, textStatus) {
    // Kód pro zpracování odpovědi.
  }
});
```

V našem výsledném řešení požadujeme soubor `hello-ajax.html` a připojujeme jeho obsah k elementu `body` bezprostředně po úspěšném přijetí odpovědi. Jestliže požadavek selže, spustí se metoda `error()`, v níž uživateli zobrazíme výstražnou zprávu:

```
(function($) {
  $(document).ready (function() {
    $('#update').click(function() {
      $.ajax({
        type: 'GET',
        url: 'hello-ajax.html',
        dataType: 'html',
        success: function(html, textStatus) {
        $('body').append(html);
        },
        error: function(xhr, textStatus, errorThrown) {
          alert('Nastala chyba. ' + errorThrown);
        }
      });
    });
  });
})(jQuery);
```

16.3 Používání technologie Ajax na celém webu

Problém

Máme velkou webovou aplikaci se spoustou odeslaných ajaxových požadavků a chceme definovat výchozí možnosti pro všechny tyto požadavky v rámci celé aplikace.

Řešení

```
(function($) {
  $(document).ready(function() {
    $('#loadingIndicator')
      .bind('ajaxStart', function() {
        $(this).show();
      })
      .bind('ajaxComplete', function() {
        $(this).hide();
      });
    $.ajaxSetup({
      cache: true,
      dataType: 'json',
      error: function(xhr, status, error) {
        alert('Nastala chyba: ' + error);
      },
      // Časový limit o délce 60 sekund.
      timeout: 60000,
      type: 'POST',
      url: 'ajax-gateway.php'
    }); // Uzavřeme volání metody $.ajaxSetup().
  }); // Uzavřeme volání metody ready().
})(jQuery);
```

Diskuze

Když pracujeme s velkými aplikacemi, často se setkáme se společnou ajaxovou bránou, přes kterou proudí všechny požadavky. Pomocí metody `$.ajaxSetup()` můžeme nastavit výchozí možnosti nastavení všech ajaxových požadavků. Tímto způsobem zjednodušíme ajaxové požadavky v celé aplikaci; například následovně:

```
$.ajaxSetup({
  data: {
    // Naše data požadavku pro server.
  },
  success: function(data) {
    // Nyní aktualizujeme uživatelské rozhraní.
  }
});
```

Měli bychom vědět, že možnost nastavení časového intervalu obsahuje hodnotu v milisekundách (tisícina sekundy), takže časový interval o hodnotě `6000` představuje šest sekund. Když nastavujeme tuto hodnotu, musíme si uvědomit, do jakých rozměrů se Internet rozrostl. Někteří

návštěvníci se můžou nacházet v místech, které mají výrazně větší zpoždění, než bychom očekávali v naší oblasti. Nenastavujme tedy tuto hodnotu příliš malou (například pět sekund). Větší časový interval (30 nebo 60 sekund) umožní, aby si funkce naší aplikace vychutnali i uživatelé se spojením s větším zpožděním (kupříkladu se satelitním spojením).

Ve výše uvedeném příkladu odesíláme požadavek typu POST skriptu `ajax-gateway.php`. Pokud vznikne chyba, zpracuje ji funkce `error()`, kterou definujeme v metodě `$.ajaxSetup()`. Stále je ale možné přepsat možnosti nastavení v konkrétním požadavku:

```
$.ajax({
  url: 'another-url.php',
  data: {
    // Naše data požadavku pro server.
  },
  success: function(data) {
    // Nyní aktualizujeme uživatelské rozhraní.
  }
});
```

Předchozí skript bychom odeslali skriptu `another-url.php` místo skriptu `ajax-gateway.php`. Skvělou vlastností ajaxové architektury knihovny jQuery jsou dostupné globální události – například `ajaxComplete`, `ajaxError`, `ajaxSend`, `ajaxStart`, `ajaxStop` a `ajaxSuccess`. Tyto události můžeme nastavit pomocí volání `.bind('událost', funkceZpětnéhoVolání)` nebo zkrácené verze `.event(funkceZpětnéhoVolání)`. Následující příklad ukazuje dvě volání metody pro svázání funkce zpětného volání s událostí `ajaxError`:

```
(function($) {
  $(document).ready(function() {
    $('#loadingIndicator')
      .ajaxError(function() {
        // Náš zdrojový kód.
      });
      // Nebo použijeme metodu bind().
      $('#loadingIndicator')
        .bind('ajaxError', function() {
        // Náš zdrojový kód.
      });
  });
})(jQuery);
```

Následuje stručný přehled a popis dostupných událostí a také pořadí, v jakém se spouštějí:

- `ajaxStart`: spouští se na začátku ajaxového požadavku, když se nezpracovávají žádné jiné požadavky.
- `ajaxSend`: spouští se před odesláním každého požadavku.
- `ajaxSuccess` nebo `ajaxError`: spouští se při úspěšném nebo neúspěšném dokončení požadavku.
- `ajaxComplete`: spouští se pokaždé, když se dokončí požadavky (bez ohledu na to, zda úspěšně, nebo neúspěšně).
- `ajaxStop`: spouští se po dokončení požadavku, pokud se nezpracovávají žádné další ajaxové požadavky.

V další kapitole se seznámíme s těmito událostmi podrobněji.

16.4 Používání jednoduchých ajaxových požadavků se zpětnou odezvou uživateli

Problém

Potřebujeme zobrazit indikátor stavu uživateli, když se zpracovávají ajaxové požadavky a schovat jej po jejich dokončení.

Řešení

```
(function($) {
  $(document).ready(function() {
    $('#ajaxStatus')
      .ajaxStart(function() {
        $(this).show();
      })
      .ajaxStop(function() {
        $(this).hide();
      });

    // Zahájíme ajaxový požadavek, když uživatel klepne na tlačítko
    // s identifikátorem doAjaxButton.
    $('#doAjaxButton').click(function() {
      $.ajax({
        url: 'ajax-gateway.php',
        data: {val: "Ahoj světe"},
        dataType: 'json',
        success: function(json) {
          // Kód pro zpracování dat.
          $('body').append('Hodnota odpovědi: ' + json.val);
        }
      });
    });
  });
})(jQuery);
```

Diskuze

Obrovskou výhodou implementace technologie Ajax v knihovně jQuery je vystavení globálních ajaxových událostí, které se spouští na všech elementech při každém ajaxovém požadavku. Ve výše uvedeném řešení svazujeme dvě události (`ajaxStart` a `ajaxStop`) pomocí zkrácených metod s elementem jazyka XHTML s identifikátorem `ajaxStatus`. Když uživatel spustí ajaxový požadavek klepnutím na tlačítko s identifikátorem `doAjaxButton`, spustí se taktéž událost `ajaxStart` a její obsluhující funkce zavolá metodu `show()` na element s identifikátorem `ajaxStatus`. Povšimněte si, že tyto události se spouští automaticky a jsou takovým vedlejším produktem metody `$.ajax()` (nebo jiných zkrácených metod – například `$.get()`). Díky tomu můžeme elegantním způsobem vytvořit indikátor stavu požadavků, který zobrazuje stav ajaxových požadavků v celé aplikaci:

```
(function($) {
  $(document).ready(function() {
    $('#ajaxStatus')
      .ajaxStart(function() {
        $(this).show();
      })
      .ajaxStop(function() {
        $(this).hide();
      });

    // Zahájíme ajaxový požadavek, když uživatel klepne na tlačítko
    // s identifikátorem doAjaxButton.
    $('#doAjaxButton').click(function() {
      $.ajax({
        url: 'ajax-gateway.php',
        data: {val: 'Ahoj světe'},
        dataType: 'json',
        success: function(json) {
          // Zdrojový kód pro zpracování dat.
          $('body').append('Hodnota odpovědi: ' + json.val);
        }
      });
    });
  });
})(jQuery);
```

Podívejte se na další události a rozdíl mezi lokálními a globálními ajaxovými událostmi. Mezi lokální ajaxové události patří události `beforeSend`, `success`, `error` a `complete`. Tyto události definujeme přímo při odesílání ajaxového požadavku a jsou s ním tudíž úzce spojené. Globální ajaxové události se spouští střídavě s lokálními událostmi, ale spouští se pro všechny elementy, které se s nimi svážou, a také používají přirozenou architekturu obsluhy událostí v knihovně jQuery. Zde je stručný příklad, jak nakládat s lokálními ajaxovými událostmi (například s událostí `complete`):

```
$.ajax({
  type: 'GET',
  url: 'ajax-gateway.php',
  dataType: 'html',
  complete: function(xhr, textStatus) {
    // Zdrojový kód pro zpracování odpovědi.
  }
});
```

Nyní prozkoumáme rozdělení, pořadí a oblast, v jaké se události spouští při úspěšném ajaxovém požadavku:

- `ajaxStart` (globální),
- `beforeSend` (lokální),
- `ajaxSend` (globální),
- `success` (lokální),
- `ajaxSuccess` (globální),

- `complete` (lokální),
- `ajaxComplete` (globální),
- `ajaxStop` (globální).

U neúspěšného ajaxového požadavku by bylo pořadí spuštěných událostí následující, přičemž události `success` a `ajaxSuccess` se nahradí událostmi `error` a `ajaxError`:

- `ajaxStart` (globální),
- `beforeSend` (lokální),
- `ajaxSend` (globální),
- `error` (lokální),
- `ajaxError` (globální),
- `complete` (lokální),
- `ajaxComplete` (globální),
- `ajaxStop` (globální).

Události `ajaxStart` a `ajaxStop` jsou speciální události v globální oblasti. Liší se v tom, že pracují současně s více požadavky. Událost `ajaxStart` se spouští po vytvoření požadavku ale jen tehdy, když se nezpracovávají další požadavky. Událost `ajaxStop` se spouští po dokončení požadavku, pokud se nezpracovávají žádné jiné požadavky. Tyto události se tedy spouští jen jednou, když současně odešleme více požadavků:

```
(function($) {
  $(document).ready(function() {
    $('#ajaxStatus')
      .ajaxStart(function() {
        $(this).show();
      })
      .ajaxStop(function() {
        $(this).hide();
      });

    // Zahájíme ajaxový požadavek, když uživatel klepne na tlačítko
    // s identifikátorem doAjaxButton.
    $('#doAjaxButton').click(function() {
      $.ajax({
        url: 'ajax-gateway.php',
        complete: function() {
          // Zdrojový kód pro zpracování dat.
        }
      });
      $.ajax({
        url: 'ajax-data.php',
        complete: function() {
          // Zdrojový kód pro zpracování dat.
        }
      });
    });
  });
})(jQuery);
```

Jednou z možností nastavení, kterou můžeme předat metodě `$.ajax()`, je možnost nastavení `global`, jež lze nastavit na hodnotu `true` nebo `false`. Nastavením hodnoty `false` možnosti nastavení `global` je možné potlačit spouštění globálních událostí.

Pokud ve své aplikaci narazíte na problémy s efektivitou, může je způsobovat propagace událostí ve stránce s velkým počtem elementů. V tomto případě můžete nastavením hodnoty `false` možnosti nastavení `global` výrazně vylepšit efektivitu.

Funkce zpětného volání lokální události `beforeSend` umožňuje změnit objekt `XMLHTTPRequest` (předává se jako argument této funkci zpětného volání) před odesláním požadavku. V následujícím příkladu definujeme vlastní hlavičky protokolu HTTP pro náš požadavek. Odeslání požadavku můžeme zrušit vrácením hodnoty `false` v této funkci zpětného volání:

```
(function($) {
  $(document).ready(function() {
    // Zahájíme ajaxový požadavek, když uživatel klepne na tlačítko
    // s identifikátorem doAjaxButton.
    $('#doAjaxButton').click(function() {
      $.ajax({
        url: 'ajax-gateway.php',
        beforeSend: function(xmlHttpRequest) {
          xmlHttpRequest.setRequestHeader('X-SampleHeader',
            'Ahoj světe');
        },
        complete: function() {
          // Zdrojový kód pro zpracování dat.
        }
      });
    });
  });
})(jQuery);
```

Nyní použijeme všechny tyto události a předěláme naše řešení na následující:

```
(function($) {
  $(document).ready(function() {
    $('#ajaxError')
      .ajaxError(function(evt, xhr, ajaxOptions, error) {
        $(this)
          .html('Chyba: ' + (xhr? xhr.status: '')
            + ' ' + (error? error :'neznámá'))
          .show();
      })
      .ajaxSuccess(function() {
        $(this).hide();
      });
    $('#ajaxStatus')
      .ajaxStart(function() {
        $(this).show();
      })
```

```
      .ajaxSend(function() {
        $(this).html('Odesílání požadavku...');
      })
      .ajaxStop(function() {
        $(this).html('Požadavek dokončen...');
        var t = this;
        setTimeout(function() {
          $(t).hide();
        }, 1500);
      });

    // Zahájíme ajaxový požadavek, když uživatel klepne na tlačítko
    // s identifikátorem doAjaxButton.
    $('#doAjaxButton').click(function() {
      $.ajax({
        url: 'ajax-gateway.php',
        complete: function() {
          // Zdrojový kód pro zpracování dat.
        }
      });
    });
  });
})(jQuery);
```

16.5 Používání zkrácených ajaxových metod a datových typů

Problém

Potřebujeme odeslat ajaxový požadavek typu GET na server a uložit získaný kód jazyka HTML do elementu `div` s identifikátorem `contents`.

Řešení

```
(function($) {
  $(document).ready(function() {
    $('#contents').load('hello-world.html');
  });
})(jQuery);
```

Diskuze

Tento recept se bude mírně lišit od ostatních v tom, že v něm prozkoumáme nejrůznější zkrácené funkce knihovny jQuery, aby vyplynuly rozdíly mezi nimi.

Knihovna jQuery nabízí několik zkrácených metod pro odesílání ajaxových požadavků. Ke koncepcím z předchozích receptů o technologii Ajax existují následující zkrácené metody: `load()`, `$.get()`, `$.getJSON()`, `$.getScript()` a `$.post()`. Nejprve si ale popíšeme naše řešení:

```
$('#contents').load('hello-world.html');
```

Pomocí metody `load(url)` odesíláme ajaxový požadavek typu GET na soubor `hello-world.html` a umístíme obsah odpovědi do elementu s identifikátorem `contents`. Metoda `load()` má dva volitelné parametry – parametr `data` a parametr `callback`. Jako parametr `data` lze zadat parametr objekt JavaScriptu a od verze 1.3 knihovny jQuery také textový řetězec. V následujícím příkladu předáváme proměnnou `ahoj` s hodnotou `světe` (to odpovídá následující adrese URL: *hello-world.html? ahoj=sv%ECte*).

```
$('#contents').load('hello-world.html', {ahoj: 'světe'});
```

Třetí nepovinný parametr je funkce zpětného volání, která se volá při dokončení požadavku (ať už úspěšném, nebo neúspěšném). V následujícím příkladu zobrazíme výstražné dialogové okno po dokončení požadavku:

```
$('#contents').load('hello-world.html', {ahoj: 'světe'}, function() {
  alert('Požadavek dokončen!');
});
```

Dále se podíváme na dvě metody – `$.get()` a `$.post()`. Obě přijímají stejné argumenty, přičemž metoda `$.get()` odesílá požadavek typu GET protokolu HTTP, zatímco metoda `$.post()` odesílá požadavek typu POST protokolu HTTP. Ukážeme si ukázkový příklad použití metody `$.get()` k odeslání požadavku. Metoda `$.get()` má parametry `url`, `data`, `callback` a `type`. První tři parametry funkce se shodují s parametry předchozí metody `load()`, takže si popíšeme pouze parametr `type`:

```
$.get(
  'hello-world.html',
  {ahoj: 'světe'},
  function(data) {
    alert('Požadavek dokončen!');
  },
  'html'
);
```

Parametr `type` přijímá některou z následujících hodnot: `xml`, `html`, `script`, `json`, `jsonp` a `text`. Hodnoty parametru `type` určují, jak se mají data odpovědi na ajaxový požadavek zpracovat předtím, než se předají funkci zpětného volání. Jelikož jsme v předchozím příkladu specifikovali hodnotu `html`, argumentem s daty pro naši funkci zpětného volání bude objekt modelu DOM. Zadání hodnoty `xml` jako argumentu `type` bude mít za následek předání objektu modelu DOM ve formátu XML. Jestliže zadáme jako argument `type` hodnotu `script`, data vrácená serverem se provedou předtím, než se spustí funkce zpětného volání. Pomocí argumentu `json` a `jsonp` naformátujeme výsledek do objektu JavaScriptu, který se předá funkci zpětného volání, přičemž argument `jsonp` se liší v tom, že knihovna jQuery předá název metody z požadavku a spojí funkci zpětného volání s anonymní funkcí definovanou spolu s požadavkem. To umožňuje provádět požadavky mezi doménami. Formát `text` je přesně takový, jak jeho název napovídá – funkci zpětného volání se předá prostý text v podobě textového řetězce.

Nyní si popíšeme poslední dvě zkrácené metody – metodu `$.getJSON()` a metodu `$.getScript()`. Metoda `$.getJSON()` přijímá argumenty `url`, `data` a `callback`. Metoda `$.getJSON()` je kombinací metody `$.get()` a příslušných parametrů pro formát JSON nebo JSONP. Následující příklad by odeslal požadavek typu JSONP na server Flickr, po němž by žádal fotografie z veřejného časového přehledu:

```
$.getJSON(
  'http://www.flickr.com/services/feeds/photos_public.gne?
    format=json&jsoncallback=?',
```

```
  function(json) {
  }
);
```

Protože se jedná o požadavek mezi doménami, knihovna jQuery s ním automaticky nakládá jako s požadavkem typu JSONP a vyplní vhodný název funkce pro parametr `callback`. To také znamená, že knihovna jQuery zahajuje požadavek vložením elementu `script` do dokumentu místo použití objektu `XMLHttpRequest`. Rozhraní API služby Flickr umožňuje definovat název funkce zpětného volání nastavením proměnné `jsoncallback` v řetězci dotazu. Povšimněte si výrazu `jsoncallback=?` v adrese URL. Knihovna jQuery inteligentně nahradí otazník vhodným názvem funkce. Knihovna jQuery běžně přidává proměnnou tvaru `callback=`, ale dovoluje tento tvar snadno upravit. Nahrazování parametru `callback` funguje jak s požadavky typu GET, tak s požadavky typu POST, ale nefunguje s parametry předanými v datovém objektu. Více se o práci s formátem JSON dozvíte v receptech „Vytváření dat ve formátu JSON“ a „Analýza dat ve formátu JSON“ a úplnou verzi našeho příkladu se serverem Flickr najdete v receptu „Používání dat ve formátu JSONP v knihovně jQuery.“

Metoda `$.getScript()` provádí požadavek buď prostřednictvím technologie Ajax, nebo dynamickým vložením elementu `script` do stránky pro komunikaci mezi doménami; potom zpracuje vrácená data a nakonec spustí zadanou funkci zpětného volání. V následujícím příkladu vkládáme skript do dokumentu a posléze voláme jednu jeho funkci ve funkci zpětného volání:

```
// Soubor hello-world.js.
function helloWorld(msg) {
  alert('Ahoj světe. Máte zde zprávu: ' + msg);
}

// Soubor hello-world.html.
(function($) {
  $(function() {
    $.getScript('hello-world.js', function() {
      helloWorld('Je nádherný den.');
    });
  });
})(jQuery);
```

16.6 Použití úryvků kódu jazyka HTML v knihovně jQuery

Problém

Chceme z textového řetězce s kódem jazyka HTML udělat skupinu uzlů modelu DOM, a ty potom vložit do dokumentu.

Řešení

```
(function($) {
  $(document).ready(function() {
    $('<div>Ahoj světe.</div>')
      .append('<a href="http://jquery.com">odkaz</a>')
```

```
      .appendTo('body');
  });
})(jQuery);
```

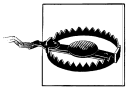
Diskuze

Práce s řetězci s kódem jazyka HTML je v knihovně jQuery běžnou úlohou. V srdci knihovny jQuery je elegantní rozhraní pro převod textového řetězce se značkovacím kódem na jeho odpovídající reprezentaci v modelu DOM. Funkci `jQuery()` můžeme místo selektoru předat textový řetězec s kódem jazyka HTML (následující způsob nefunguje pro kód jazyka XML; recept „Převod textového řetězce s kódem jazyka XML na objekt modelu DOM“ ukazuje, jak převést textový řetězec s kódem jazyka XML na objekt modelu DOM).

```
$('<div>Hello World</div>');
```

Nyní jsme kód jazyka HTML převedli na odpovídající reprezentaci v modelu DOM a můžeme s ním pracovat v knihovně jQuery. Na tento úryvek kódu můžeme volat libovolné metody knihovny jQuery:

```
$('<div>Ahoj světe.</div>')
  .append('<a href="http://jquery.com">odkaz</a>')
  .appendTo('body');
```



Stojí za upozornění, že před připojením úryvku kódu jazyka HTML do dokumentu mohou být některé vizuální vlastnosti nedostupné – například `width` a `height`. Proto v následujícím příkladu vrátí volání metody `width()` hodnotu `0`.

```
$('<div>Ahoj světe.</div>').width();
// Vrací hodnotu '0'.
```

16.7 Převod textového řetězce s kódem jazyka XML na objekt modelu DOM

Problém

Potřebujeme převést textový řetězec s kódem jazyka XML na objekt modelu DOM, s nímž můžeme pracovat v knihovně jQuery.

Řešení

```
<h1 id="title"></h1>

(function($) {
  $(document).ready(function() {
    var xml = '<myxml><title>Ahoj světe</title></myxml>';
    var title = $.xmlDOM( xml ).find('myxml > title').text();
    $('#title').html( title );
  });
})(jQuery);
```

Diskuze

Často se v poštovním seznamu knihovny jQuery objevuje otázka, jak převést kód jazyka XML do odpovídající reprezentace modelu DOM, kterou umí zpracovat knihovna jQuery. Pokud odesíláme ajaxový požadavek typu `xml`, webový prohlížeč automaticky analyzuje vrácený kód jazyka XML a převede jej na objekt modelu DOM.

Co tedy máme udělat, když chceme v knihovně jQuery zpracovat textový řetězec s kódem jazyka XML? Zásuvný modul `xmlDom` nabízí analýzu textového řetězce s kódem jazyka XML pro všechny moderní webové prohlížeče a vrací objekt model DOM obalený objektem knihovny jQuery. Díky tomu můžeme převést a začít používat kód jazyka XML v jediném kroku:

```
(function($) {
  $(document).ready(function() {
    var xml = '<myxml><title>Ahoj světe.</title></myxml>';
    var title = $.xmlDOM(xml).find('myxml > title').text();
    $('#title').html(title);
  });
})(jQuery);
```

Dalším obvyklým postupem je předání objektu modelu DOM jako druhého argumentu funkci `jQuery()` (kontextový parametr):

```
(function($) {
  $(document).ready(function() {
    var $xml = $.xmlDOM('<myxml><title>Ahoj světe.</title></myxml>');
    var title = $('myxml > title', $xml).text();
    $('#title').html( title );
  });
})(jQuery);
```

V předchozím kódu provádíme výběr knihovnou jQuery nad předaným objektem kontextu; v opačném případě (bez tohoto objektu kontextu) by knihovna jQuery provedla výběr nad objektem dokumentu.

Zásuvný modul `xmlDOM` můžete stáhnout na adrese *http://outwestmedia.com/jquery-plugins/xmldom/*.

16.8 Vytváření dat ve formátu JSON

Problém

Máme objekt JavaScriptu s daty, která chceme zakódovat do textového řetězce, abychom je mohli snadněji ukládat a načítat.

Řešení

```
(function($) {
  $(document).ready(function() {
    var messageObject = {title: 'Ahoj světe.',
      body: 'Je skvělé být na živu.'};
    var serializedJSON = JSON.stringify(messageObject);
```

```
  });
})(jQuery);
```

Diskuze

Formát JSON (JavaScript Object Notation) je rozšířený datový formát používaný pro výměnu dat mezi webovým prohlížečem a serverem. Tento formát je odlehčený a můžeme jej snadno používat a analyzovat v jazyce JavaScript. Podívejte se nejprve na jednoduchý objekt:

```
var messageObject = {title: 'Ahoj světe.',
  body: 'Je skvělé být na živu.'};
```

V tomto příkladu máme jednoduchý objekt se dvěma vlastnostmi – `title` a `body`. Zakódovanou verzi objektu je možné ukládat poměrně jednoduše. Tato verze vypadá následovně:

```
var messageObject = '{"title":"Ahoj světe.",
  "body":"Je skvělé být na živu."}';
```

Dvě běžné úlohy při práci s formátem JSON jsou zakódování objektu do textového řetězce a dekódování textového řetězce do literálu objektu. Pouze několik webových prohlížečů má v současnosti vestavěné zpracování formátu JSON (například webový prohlížeč Firefox 3.1 a novější nebo prohlížeč Internet Explorer 8). Další prohlížeče se chystají podporu formátu JSON taktéž přidat, protože formát JSON je součástí specifikace ECMA 3.1. Zatím jsou k dispozici dva přístupy pro práci s daty ve formátu JSON. Douglas Crockfor napsal kód v jazyku JavaScript pro zakódování a dekódování dat ve formátu JSON, který najdete na adrese *http://json.org/*. Nyní budeme kódovat do textového řetězce předchozí objekt pomocí knihovny JSON:

```
var serializedJSON = JSON.stringify(messageObject);
```

Teď máme k dispozici textový řetězec, který můžeme odeslat na server v ajaxovém požadavku nebo ve formuláři.

16.9 Analýza dat ve formátu JSON

Problém

Chceme převést textový řetězec s daty ve formátu JSON na objekt.

Řešení

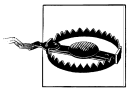
```
(function($) {
  $(document).ready(function() {
    var serializedJSON = '{"title":"Ahoj světe.",
      "body":"Je skvělé být na živu."}';
    var message = JSON.parse(serializedJSON);
  });
})(jQuery);
```

Poznámka redakce českého vydání: Od verze 1.4.1 knihovny jQuery je k dispozici metoda `jQuery.parseJSON()`. Místo výše uvedeného příkazu lze tedy použít příkaz:

```
var message = jQuery.parseJSON(serializedJSON);
```

Diskuze

Jak jsme si řekli v předchozím receptu, nyní se podíváme, jak analyzovat (dekódovat) textový řetězec s daty ve formátu JSON.



Je důležité poznamenat, že některé zde uvedené postupy nejsou bezpečné a můžou představovat potenciální bezpečnostní rizika. Ujistěte se, že důvěřujete zdroji svých dat.

Nejjednodušší způsob, jak převést textový řetězec s daty ve formátu JSON, je zavolat na něj funkci `eval()`. Tento přístup s sebou neodmyslitelně přináší bezpečnostní rizika, protože funkce `eval()` zahrnuje kompletní specifikaci jazyka JavaScript, a ne jen jeho část pro data ve formátu JSON. To znamená, že zákeřná osoba by mohla do textového řetězce s daty ve formátu JSON podstrčit jakýkoliv kód JavaScriptu. Proto tento přístup není vhodný. Místo toho použijeme dříve zmíněnou knihovnu JSON od Douglase Crockforda (tato knihovna rovněž používá funkci `eval()`, ale předtím, než ji předá data, je zpracuje tak, aby byla bezpečná).

```
var serializedJSON = '{"title":"Ahoj světe.",
  "body":"Je skvělé být na živu."}';
var message = JSON.parse(serializedJSON);
```

Nyní můžeme s naší zprávou pracovat, jako by se jednalo o běžný objekt JavaScriptu:

```
alert("Nová zpráva.\nPředmět: " + message.title + "\nObsah: " +
  message.body);
```

Knihovnu JSON můžete spolu s dalšími zdroji informací o formátu JSON stáhnout z adresy *http://json.org/*.

Poznámka redakce českého vydání: Od verze 1.4.1 knihovny jQuery můžeme analyzovat textový řetězec s daty ve formátu JSON přímo pomocí metody `parseJSON()` (*http://api.jquery.com/jQuery.parseJSON/*). Používáte-li tedy knihovnu jQuery ve verzi alespoň 1.4.1, pak nemusíte stahovat žádnou další knihovnu.

16.10 Používání dat ve formátu JSONP v knihovně jQuery

Problém

Chceme stáhnout seznam fotografií z veřejného datového proudu serveru Flickr a zobrazit první tři obrázky.

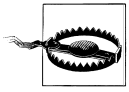
Řešení

```
(function($) {
  $(document).ready(function() {
    var url = 'http://www.flickr.com/services/feeds/photos_public.gne?
      jsoncallback=?';
```

```
    var params = {format: 'json'};
    $.getJSON(url, params, function(json) {
      if (json.items) {
        $.each(json.items, function(i, n) {
          var item = json.items[i];
          $('<a href="' + item.link + '"></a>')
            .append('<img src="' + item.media.m + '" />')
            .appendTo('#photos');
          // Zobrazíme první tři fotografie (vrácením hodnoty false
          // ukončím cyklus)
          return i < 2;
        });
      }
    });
  });
})(jQuery);
```

Diskuze

Bezpečnost představuje kritický aspekt, když vytváříme webové stránky nebo aplikaci, a to zejména s příchodem technologie Ajax. Webové prohlížeče uplatnily jednu zásadu zabezpečení na požadavky, a ta spočívá v tom, že lze odesílat požadavky jen na stejnou doménu (nebo subdoménu), kterou obsahuje adresa URL aktuální stránky. Takže kupříkladu ze stránky na adrese *http://www.priklad.cz* je možné odesílat ajaxové požadavky na doménu *http://www.priklad.cz* nebo na doménu *http://x.www.priklad.cz*, ale ne na doménu *http://priklad.cz* nebo na doménu *http://y.priklad.cz*. Jakmile se objevil sémantický web a webové stránky typu Flickr zveřejnily své rozhraní API pro další uživatele nebo služby, zásady zabezpečení webových prohlížečů se staly překážkou. Jedním místem, kde se tyto zásady zabezpečení nikdy neprojevily, je element `script` a jeho atribut `src`. Na doméně *http://www.priklad.cz* je možné vložit skript z domény *http://static.priklad2.cz*, ale vzniká problém s dynamickým vkládáním skriptů a řízením toku programu. Z tohoto důvodu se objevil formát JSONP jako standardní způsob pro překonání tohoto omezení.



Je důležité poznamenat, že některé zde uvedené postupy nejsou bezpečné a můžou představovat potenciální bezpečnostní rizika. Ujistěte se, že důvěřujete zdroji svých dat. Když vkládáte skript do stránky, tak bude mít přístup k celému modelu DOM a všem soukromým a citlivým datům v něm obsaženým. Zákeřný skript by mohl tato data odeslat nedůvěryhodné straně. Měli byste předem zavést bezpečnostní opatření – například umístit skript do pískoviště. Pokročilé možnosti zabezpečení jsou nad rámec tohoto receptu, ale je důležité, že o této nutnosti víte.

Formát JSONP v knihovně jQuery umožňuje odesílat požadavky pomocí elementu `script` s jeho atributem `src` a řídit tok programu obalením dat do funkce zpětného volání, kterou může vývojář implementovat. Podívejte se nejprve na jednoduchou zprávu ve formátu JSON:

```
{"title":"Ahoj světe.","body":"Je skvělé být na živu."}
```

Zde je stejná zpráva obalená funkcí zpětného volání:

```
myCallback({"title":"Ahoj světe.","body":"Je skvělé být na živu."})
```

Když žádáme prostředek, funkce `myCallback()` se zavolá po načtení dat do prohlížeče a jako první argument obdrží objekt JSON. Vývojář pak může implementovat funkci `myCallback()` pro zpracování dat takto:

```
function myCallback(json) {
  alert( json.title );
}
```

Nyní si popíšeme naše řešení pro server Flickr. Nejprve definujeme adresu URL webové služby Flickr a potom deklarujeme objekt `params` s proměnnými řetězce dotazu. Služba Flickr specifikuje speciální parametr `jsoncallback`, v němž můžeme zadat název naší funkce. Jelikož jsme nastavili tomuto parametru hodnotu ?, knihovna jQuery automaticky vygeneruje jméno funkce a sváže jej s naší funkcí zpětného volání.

Knihovna jQuery pozná, že se jedná o požadavek typu JSONP (mezidoménový požadavek), podle výrazu =? v adrese URL. Tento výraz není možné zadat do pole `params`.

```
var url = 'http://www.flickr.com/services/feeds/photos_public.gne?
  jsoncallback=?';
var params = {format: 'json'};
```

Potom zavoláme metodu `$.getJSON()` knihovny jQuery, které předáme proměnné `url`, `params` a naši funkci zpětného volání, jež přijímá objekt JSON. V naší funkci zpětného volání ověříme, že existuje pole s položkami a potom budeme iterovat přes první tři prvky prostřednictvím metody `$.each()`. Pro každý prvek vytvoříme odkaz, připojíme k němu obrázek a pak jej přidáme do elementu s identifikátorem `photos`. V naší funkci vrátíme při třetí iteraci hodnotu `false` (když má proměnná `i` hodnotu `2`), čímž ukončíme cyklus.

```
$.getJSON(url, params, function(json) {
  if (json.items) {
    $.each(json.items, function(i, n) {
      var item = json.items[i];
      $('<a href="' + item.link + '"></a>')
        .append('<img src="' + item.media.m + '" />')
        .appendTo('#photos');
      return i < 2;
    });
  }
});
```

Díky kombinaci datového formátu JSON s možností formátu JSONP pro přenos dat mezi doménami můžou vývojáři vytvářet nové aplikace, které shromažďují a převádí data novými inovativními způsoby a rozšiřují tak sémantický web.