KAPITOLA 10

JavaScript v systému Drupal

Použití JavaScriptu v systému Drupal je stejné jako v ostatních webových aplikacích. Je součástí mocných funkcí – například překrývání, automatického dokončování, přesouvání apod. V této kapitole se zaměříme na integraci JavaScriptu do Drupalu a jak používat pomocné funkce tohoto jazyka.

Mezi důležitá témata této kapitoly patří:

- Jak vkládat JavaScript do stránek pomocí Drupalu.
- Změna JavaScriptu přidaného do stránek prostřednictvím jádra Drupalu a ostatních modulů.
- Používání pomocných funkcí vestavěných do Drupalu.
- Práce se stylováním a překlady v jazyce JavaScript.
- Práce s technologií AJAX a systémem Drupal.

Na konci této kapitoly byste měli mít základní znalost práce s jazykem JavaScript v systému Drupal.

JavaScript uvnitř Drupalu

JavaScript je interní součástí Drupalu. Drupal poskytuje dynamické funkce, jedinečnou administrační zkušenost a knihovnu JavaScriptu pro vývojáře modulů. Systém Drupal nabízí knihovnu jQuery a několik dalších zásuvných modulů této knihovny. Tento systém zprostředkovává JavaScript pomocí rozhraní Library API, jež je dostupné pro moduly.

Drupal poskytuje spolu s knihovnou jQuery 1.4.4 rovněž níže uvedené zásuvné moduly:

- Knihovnu jQuery UI 1.8.6.
- Zásuvný modul jQuery Coookie – jednoduchý nástroj pro čtení, psaní a mazání cookies.
- Zásuvný modul jQuery Form pro snadné a nevtíravé vylepšení formulářů jazyka HTML pomocí technologie AJAX.
- Zásuvný modul jQuery Once filtruje pryč elementy, které měly dříve stejný filtr.
- Zásuvný modul jQuery BBQ: Back Button & Query Library.
- Zásuvný modul Farbtastic – kolečko pro výběr barvy.

Vkládání JavaScriptu

Většina JavaScriptu v systému Drupal používá knihovnu jQuery, ale není to požadavek. Když vkládáme JavaScript do stránky, měli bychom si dávat pozor na několik věcí.

Drupal nastavuje pro knihovnu jQuery bezkonfliktní režim. To znamená, že knihovna jQuery se vzdává své proměnné `$` tak, abychom ji mohli používat pro jiné knihovny JavaScriptu. Více informací najdete na adrese: *http://api.jquery.com/jQuery.noConflict/*.

Bez přítomnosti proměnné `$` můžeme použít dvě metody pro psaní kódu, v němž používáme knihovnu jQuery. První je psát `jQuery` všude tam, kde bychom normálně psali `$`. Například:

```
jQuery().ready(function() {
  ...
});
```

Druhým způsobem je obalit náš kód do anonymní funkce s parametrem `$`. Například:

```
(function($) {
  $().ready(function() {
    ...
  });
})(jQuery);
```

V tomto případě předáváme objekt `jQuery` jako argument `$`. V podstatě bychom mohli zvolit libovolný název místo `$`. To funguje jak u JavaScriptu uvnitř souboru, tak u vloženého přímo do stránky.

Drupal umí předzpracovávat soubory jazyka JavaScript tak, že převede více souborů na méně souborů. Předzpracování přináší koncovým uživatelům vylepšení výkonu, protože můžou stahovat méně souborů. Aby nástroj pro předzpracování vytvořil platný kód jazyka JavaScript, je vhodné, abychom psali středníky všude, kde jsou pouze volitelné. V předchozím příkladu je středník za výrazem `(jQuery)` ukázkou, kde používat volitelné středníky.

Přidávání souborů jazyka JavaScript a jazyka CSS do souborů .info

Nejjednodušší metodou přidávání souborů jazyků JavaScript a CSS je vkládat je do souborů *.info* pro modul. Když specifikujeme soubory jazyků JavaScript a CSS v souboru *.info*, přidávají se ke všem stránkám a používají předzpracování. Ukázka, v níž přidáváme skript a soubor jazyka CSS, vypadá takto:

```
scripts[] = foo.js
stylesheets[screen][] = bar.css
```

Jedná se o dva soubory a jejich cesta je relativní ke kořenu modulu. Proměnná `scripts` je pole souborů jazyka JavaScript. Proměnná `stylesheet` je polem typů médií, přičemž každý typ je pole souborů jazyka CSS.

Funkce drupal_add_js()

Nejběžnějším způsobem vkládání JavaScriptu do stránky je zavolat funkci `drupal_add_js()`. Tato pomocná funkce umožňuje přidávat soubory (jak externí, tak ze systému souborů), vložený kód JavaScriptu a předávat data mezi jazyky PHP a JavaScript.

Obyčejně používáme jazyk CSS spolu s jazykem JavaScript. Drupal poskytuje funkci pro přidávání souborů jazyka CSS do stránky. Tato funkce funguje podobně jako funkce `drupal_add_js()` a jmenuje se `drupal_add_css()`. Rozhraní obou těchto dvou funkcí je téměř shodné. Rozdílem je, že varianta pro jazyk CSS nepřijímá data z jazyka PHP a šablony stylů mají nastavení pro média a prohlížeč Internet Explorer.

V příkladech v této kapitole vytvoříme modul Hello World, který bude zobrazovat text `Ahoj světe` různými způsoby pomocí jazyka JavaScript. Definice funkce `drupal_add_js()` obsahuje dva parametry s proměnlivými hodnotami, a to v závislosti na tom, co chceme dělat. Tato definice vypadá následovně:

```
function drupal_add_js($data = NULL, $options = NULL)
```

Jak budeme postupovat tvorbou modulu Hello World, budeme se seznamovat s různými variantami a možnými hodnotami, které lze předat funkci `drupal_add_js()`.

Přidávání souborů jazyka JavaScript

Přidávání souboru je výchozím cílem funkcí `drupal_add_js()` a `drupal_add_css()`. Soubor jazyka JavaScript a soubor jazyka CSS vložíme do stránky takto:

```
drupal_add_js('cesta/k/hello_world.js');
drupal_add_css('cesta/k/hello_world.css');
```

Jedná se o nejjednodušší způsob přidání souboru. V tomto případě je první argument vždy cestou k danému souboru. Cesty k souborům jsou v instalaci Drupalu relativní k základní cestě webových stránek. Když Drupal zobrazuje tyto soubory ve webovém prohlížeči, vloží před ně základní cestu.

Protože moduly se mohou nacházet na více místech v systému souborů, pomocí funkce `drupal_get_path()` je možné najít cestu k modulu nabízejícímu daný soubor. Kdybychom chtěli přepsat výše uvedené příklady tak, aby se dynamicky odkazovaly na umístění modulu, napsali bychom:

```
$path = drupal_get_path('module', 'hello_world');
drupal_add_js($path . '/hello_world.js');
drupal_add_css($path . '/hello_world.css');
```

Tip

Více informací o používání funkce `drupal_get_path()` v kombinaci s moduly, tématy vzhledu a jinými systému v Drupalu najdete v dokumentaci rozhraní API na adrese *http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_get_path/7*.

V tomto jednoduchém příkladu předáváme pouze argument `$data` (konkrétně název souboru), protože Drupal standardně vkládá soubory. Druhý argument se jmenuje `$options` a lze pro něj použít buď textový řetězec s typem přidávaného JavaScriptu nebo CSS, nebo pole možností nastavení. Výše uvedený kód je možné přepsat následovně:

```
$path = drupal_get_path('module', 'hello_world');
drupal_add_js($path . '/hello_world.js', 'file');
drupal_add_css($path . '/hello_world.css', 'file');
```

Když pracujeme se soubory, můžeme nastavit jako druhý argument hodnotu `'file'`, jestliže se jedná o soubor v systému souborů Drupalu nebo na relativní adrese URL, nebo hodnotu `'external'`, pokud se jedná o soubory vně instalace Drupalu.

Pomocí argumentu `$options` můžeme nastavovat několik dalších možností nastavení pro každý soubor, a to včetně vlastností `weight`, `group`, `every_page`, `scope`, `defer`, `preprocess` a `caching`.

Soubory jazyka JavaScript se zobrazují podle hodnoty `group` (ať už jsou na každé stránce, nebo ne) a potom podle hodnoty `weight`. JavaScriptovými skupinami jsou skupiny `JS_LIBRARY`, `JS_DEFAULT` a `JS_THEME`. Každá skupina se řadí podle toho, zda má vlastnost `every_page` hodnotu `true`. Skripty pro všechny stránky se vypisují před soubory, které jsou jen na některých stránkách. Nakonec se jednotlivé soubory uspořádají podle hodnoty `weight`.

Pokud přidáváme knihovnu JavaScriptu nebo zásuvný modul, měli bychom jej přidat se skupinou `JS_LIBRARY`, aby se tyto soubory vložily do stránky před kódem JavaScriptu, ve kterém je používáme. Když přidáváme do stránky dvě knihovny, které spolu vzájemně souvisejí, lze je vložit do stránky se stejnou skupinou a s pořadím, v němž by se měly vkládat, nebo s různými šířkami. Abychom si to předvedli, v níže uvedeném zdrojovém kódu přidáváme soubor *mylibrary.js* do stránky jako knihovnu před souborem *hello_world.js*.

```
$path = drupal_get_path('module', 'hello_world');
$options = array(
  'group' => JS_LIBRARY,
);
drupal_add_js($path . '/mylibrary.js', $options);
drupal_add_js($path . '/hello_world.js');
```

V tomto příkladu si povšimněte, že vynecháváte výchozí nastavení. Protože hodnota `'file'` je výchozí, nemusíme ji psát u žádného z těchto volání.

Aby se kód JavaScriptu vložil do každé stránky, můžeme nastavit příznak `every_page` na hodnotu `true`. Jakmile nastavíme tuto hodnotu `true`, ovlivníme předzpracování (více si popíšeme později) a pořadí, v jakém se skript vkládá. Uvnitř skupiny se všechny soubory označené příznakem `every_page` vkládají před soubory, které se nevkládají na všech stránkách.

Pokud máme skripty ve skupině rozdělené podle toho, zda se vkládají do každé stránky, uspořádávají se dále podle šířky. Výchozí šířkou je `0`. Soubory s nižší šířkou se vkládají před soubory s vyšší šířkou.

Tip

Knihovny spoléhající se na soubor *drupal.js* musí znát šířku nastavenou tomuto souboru. Soubor *jquery.js* má nastavenou šířku `-20` a soubor *drupal.js* šířku `-1`.

Soubor lze přidat do oblasti (`scope`) `'header'` nebo `'footer'`. Výchozí hodnotou je `'header'`, přičemž umístí kód JavaScriptu do záhlaví stránky. Nejtypičtějšími místy, kam lze vložit kód JavaScriptu, jsou záhlaví nebo zápatí. Vlastní oblasti pro JavaScript je možné definovat v tématu vzhledu nebo v modulu. Jestliže definujeme vlastní oblasti, oblasti z tématu vzhledu nebo modulu můžeme použít navíc k hodnotám `'header'` a `'footer'`.

Pomocí nastavení `Defer` konfigurujeme element `script` jazyka HTML podporovaný prohlížečem Internet Explorer. Říkáme tak prohlížeči, že by měl odložit načtení skriptu, dokud se nezobrazí celá stránka. To je velmi užitečné pro skripty, které nemusí být dostupné, a nemusíme je spouštět při zobrazování stránky. V systému Drupal můžeme této vlastnosti nastavit hodnotu `true` nebo `false`.

Předzpracování a ukládání vlastností do mězipaměti jde ruku v ruce. Předzpracování je funkce, kterou systém Drupal poskytuje pro spojování souboru přidaných ke stránce do menšího množství souborů. Předzpracování se řídí skupinami a příznakem `every_page`. Kupříkladu soubory ze skupiny `JS_LIBRARY` vkládané do každé stránky se shlukují do jednoho předzpracovaného souboru. Soubory seskupené do skupiny JS_LIBRARY, které se nevkládají do každé stránky, spadají do jiného předzpracovaného souboru. Každá skupina a podskupina určená příznakem `every_page` tvoří samostatný předzpracovaný soubor. Předzpracování má za účel minimalizovat množství kódu JavaScriptu odesílaného uživateli a využít výhody ukládání do mezipaměti ve webovém prohlížeči. Když nastavíme vlastnosti `cache` hodnotu `false`, soubory se nebudou předzpracovávat, jelikož předzpracované soubory se ukládají do mezipaměti.

Když dáme vše dohromady – soubor jazyka JavaScript s odkladem a vypnutým ukládáním do mězipaměti a předzpracováním a se skupinou nastavenou tak, aby se vkládal po souboru *drupal.js*, by vypadal takto:

```
$path = drupal_get_path('module', 'hello_world');
$options = array(
  'group' => JS_LIBRARY,
  'cache' => FALSE,
  'preprocess' => FALSE,
  'defer' => TRUE,
);
drupal_add_js($path . '/mylibrary.js', $options);
```

Vkládání souborů jazyka CSS

Soubory jazyka CSS vkládáme stejně jako soubory jazyka JavaScript. Definice funkce `drupal_add_css()` se liší pouze v možnostech nastavení, které lze předávat druhým argumentem. Možnostmi pro soubory jazyka CSS jsou `weight`, `group`, `every_page`, `media`, `basename`, `browsers` a `preprocess`. Podobně jako u funkce `drupal_add_js()` používáme hodnotu `'file'` pro interní soubory Drupalu nebo relativní adresu URL a hodnotu `'external'` pro externí soubory jazyka CSS s úplnou adresou URL.

Drupal definuje tři konstanty pro skupiny:

- Skupina `CSS_SYSTEM` je pro systémové soubory a knihovny.
- Skupina `CSS_DEFAULT` jsou soubory jazyka CSS, které bychom měli používat v modulech.
- Skupina `CSS_THEME` slouží pro kaskádové styly tématu vzhledu.

Kaskádové styly lze aplikovat na různá média. Například šablony stylů s médiem obrazovka se aplikují pouze tehdy, když se daná stránka zobrazuje na obrazovkách. Ostatní média – jako kupříkladu tiskárna ignorují tuto šablonu stylů. Výchozí hodnotou je `'all'`.

V následujícím kódu přidáváme systémový soubor jazyka CSS, který nepředzpracováváme a použijeme jej jen na obrazovkách:

```
$path = drupal_get_path('module', 'hello_world');
$options = array(
  'group' => CSS_SYSTEM,
  'media' => 'screen',
  'preprocess' => FALSE,
);
drupal_add_css($path . '/hello_world.css', $options);
```

Předávání proměnných z jazyka PHP do jazyka JavaScript

Drupal nabízí prostředky pro předávání proměnných z jazyka PHP do jazyka JavaScript pomocí funkce `drupal_add_js()`. Spousta aplikací chce předat konfigurační informaci kódu JavaScriptu, který se nachází ve stránce. Tato funkce je právě těmi prostředky, s nimiž je možné tuto informaci lehce předat.

V terminologii Drupalu se proměnné předávané z jazyka PHP do jazyka JavaScript nazývají nastaveními. Zde je jednoduchý příklad nastavení, které předává text „Ahoj světe!“ z jazyka PHP do jazyka JavaScript:

```
drupal_add_js(array('helloWorld' => "Ahoj světe!"), 'setting');
```

Poznámka

Názvy proměnných v kódu jazyka PHP zapisujeme v Drupalu malými písmeny a jednotlivá slova oddělujeme podtržítkem. V jazyce JavaScript bychom je měli zapisovat zápisem **camelCase** s počátečním malým písmenem. Více informací o těchto programovacích standardech najdete na adrese *http://drupal.org/coding-standards*.

V kódu JavaScriptu poté najdeme tento text ve vlastnosti `Drupal.settings.helloWorld`. Například následujícím kódem JavaScriptu bychom zobrazili text „Ahoj světe!“ v dialogovém oknu:

```
alert(Drupal.settings.helloWorld);
```

Nastavení jsou odlišná od ostatních způsobů použití funkce `drupal_add_js()`. Přidávají se do záhlaví stránky s šířkou skupiny `JS_LIBRARY`. Nejsou zde žádná další běžná nastavení kromě toho, že specifikujeme, že se jedná o nastavení, jak je patrné z tohoto příkladu:

```
drupal_add_js(array('helloWorld' => "Ahoj světe!"), 'setting');
```

Nastavení bychom měli přidávat způsobem, jenž respektuje jmenné prostory ostatních nastavení vkládaných do stránky. Více nastavení bychom měli vkládat do vnořeného pole. Například takto:

```
$settings = array(
  'helloWorld' => array(
    'display' => 'alert',
    'message' => 'Ahoj světe!',
  ),
);
drupal_add_js($settings, 'setting');
```

V tomto příkladu se zpráva nachází ve vlastnosti `Drupal.settings.helloWorld.message`. Jakmile uchováme všechna nastavení v poli `Drupal.settings.helloWorld`, oddělíme nastavení pro tento modul od nastavení přidávaných ostatními moduly.

Přidávání vloženého kódu JavaScriptu

Kód jazyka JavaScript je možné vkládat přímo do stránky pomocí možnosti `inline`. Příklad oznamující zprávu „Ahoj Světe!“ by vypadal takto:

```
drupal_add_js('alert("Ahoj světe!");', 'inline');
```

Možnosti nastavení pro vložený kód jsou `defer`, `group`, `every_page`, `weight` a `scope`. Vložený kód JavaScript se neukládá do mezipaměti a není možné jej předzpracovat. Vložený kód JavaScriptu s odkladem do nahrání stránky a šířkou skupiny `JS_THEME` by vypadal takto:

```
$options = array(
  'type' => 'inline',
  'group' => JS_THEME,
  'defer' => TRUE,
);
drupal_add_js("alert('Ahoj světe!')", $options);
```

Poznámka

Dokumentace rozhraní API pro funkci `drupal_add_js()` je k dispozici na adrese *http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_js/7*.

Přidávání vloženého kódu jazyka CSS

Kód jazyka CSS lze taktéž vložit přímo do stránky a zápis je podobný jako u verze pro JavaScript. Kód jazyka CSS bychom vložili přímo do stránky takto:

```
drupal_add_css("body { color: #ffffff; }", 'inline');
```

Druhý parametr může obsahovat pole možností nastavení – včetně skupiny, oblasti a předzpracování. Kód jazyka CSS se skupinou `CSS_THEME` a bez předzpracování by vypadal následovně:

```
$options = array(
  'type' => 'inline',
  'group' => CSS_THEME,
  'preprocess' => FALSE,
);
drupal_add_css("body { color: #ffffff; }", $options);
```

Poznámka

Dokumentaci rozhraní API pro funkci `drupal_add_css()` najdete na adrese *http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_css/7*.

Používání rozhraní Library API

Systém Drupal 7 poskytuje rozhraní Library API, kde lze definovat knihovny a zásuvné moduly jazyků JavaScript a CSS a přidávat je programově později. Drupal definuje knihovnu jQuery a ostatní knihovny pomocí implementace háčku `hook_library()` ze systémových modulů. Jako příklad můžeme použít zásuvný modul Farbtastic – nástroj pro výběr barvy z knihovny jQuery. Tento zásuvný modul obsahuje soubor jazyka JavaScript a jazyka CSS. Do stránky bychom jej přidali následovně:

```
drupal_add_library('system', 'farbtastic');
```

Takto nepřidáme jen soubor jazyka JavaScript a jazyka CSS, ale také případné závislé knihovny. Příkladem může být modul Overlay, v němž funkce `drupal_add_library()` přidává překryvnou knihovnu. Následujícím voláním přidává rodičovský překryvný kód JavaScriptu:

```
drupal_add_library('overlay', 'parent');
```

Funkce `overlay_library()` nastavuje rodiče tak, že závisí na zásuvném modulu jQuery BBQ a jádru knihovny jQuery UI. To znamená, že tyto dvě knihovny se přidají do stránky před překryvnou knihovnou. Systém Drupal zná závislostní řetěz knihoven, takže si jej nemusíte pamatovat.

Poznámka

Dokumentace rozhraní API pro funkci `drupal_add_library()` je k dispozici na adrese *http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_library/7*.

Definování knihovny pomocí háčku hook_library()

Jestliže chce modul používat knihovnu (případně zásuvný modul) nebo ji zpřístupnit ostatním knihovnám, měl by ji definovat jako knihovnu pomocí háčku `hook_library()`. Protože v našem modulu Hello World máme soubor jazyka JavaScript a soubor jazyka CSS, můžeme jej přidat jako knihovnu tímto způsobem:

```
/**
 * Implementuje háček hook_library().
```

```
 */
function hello_world_library() {
  $path = drupal_get_path('module', 'hello_world');
  $libraries = array();
  $libraries['hello_world_library'] = array(
    'title' => 'Ahoj světe',
    'website' => 'http://example.com',
    'version' => '1.0',
    'js' => array(
      $path . '/hello_world.js' => array(),
    ),
    'css' => array(
      $path . '/hello_world.css' => array(),
    ),
    'dependencies' => array(
      array('system', 'ui.dialog'),
    ),
  );
  return $libraries;
}
```

Pomocí vlastností `title`, `website` a `version` specifikujeme metadata o knihovnách. To je důležité, jestliže hledáme informace, dokumentaci nebo akualizace pro knihovnu.

Veškerou práci odvedou vlastnosti `js`, `css` a `dependencies`. V případě, že definujeme nějaké závislosti, vloží se uvedené soubory před zde definovanými soubory jazyků JavaScript a CSS. Potom se přidají námi definované soubory s klíči jako prvními argumenty pro funkci `drupal_add_js()` nebo funkci `drupal_add_css()` a hodnotami jako argumenty `options` pro příslušné funkce.

Drupal má tři speciální závislosti, které se přidávají, aniž bychom je museli uvádět. Jedná se o soubory *jquery.js*, *jquery.once.js* a *drupal.js*. Tyto soubory vloží do stránky, jakmile Drupal prvně zavolá funkci `drupal_add_js()` nebo funkci `drupal_add_library()`.

Kdybychom teď chtěli použít knihovnu Hello World, přidali bychom ji prostřednictvím následujícího volání:

```
drupal_add_library('hello_world', 'hello_world_library');
```

Prvním argumentem je modul definující danou knihovnu a druhým argumentem je klíč pro definovanou knihovnu.

Poznámka

Dokumentaci rozhraní API pro háček `hook_library()` najdete na adrese *http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_library/7*.

Změna dat háčku hook_library()

Drupal poskytuje funkci `hook_library_alter()`, ve které mohou moduly zachytávat knihovny definované háčkem `hook_library()` a buď podle nich jednat, nebo je změnit. Jednoduchým příkladem by mohl být další modul nabízející novější verzi skriptu *hello_world.js*. Tento modul nazve-

me kupříkladu Hello World Update. Dejme tomu, že v souboru *hello_world_update.module* máme níže uvedenou implementaci háčku `hook_library_alter()`:

```
/**
 * Implementuje háček hook_library_alter().
 */
function hello_world_library_alter(&$libraries, $module) {
  if ($module == 'hello_world' &&
      isset($libraries['hello_world_library'])) {
    // Ověříme, že současná verze je starší než ta, na níž aktualizujeme.
    if (version_compare($libraries['hello_world_library']['version'],
        '2.0', '<')) {
      // Aktualizujeme současnou knihovnu Hello World na verzi 2.0.
      $libraries['hello_world_library']['version'] = '2.0';
      $libraries['hello_world_library']['js'] = array(
        drupal_get_path('module', 'hello_world_update') .
          '/hello_world_2.0.js' => array(),
      );
    }
  }
}
```

Jako dva argumenty předáváme knihovny definované modulem a jméno modulu. V tomto případě kontrolujeme, zda je již definovaná verze starší než verze poskytovaná tímto modulem. Pokud tomu tak je, nahradíme volání JavaScript novým voláním.

Poznámka

Dokumentaci rozhraní API pro háček `hook_library_alter()` najdete na adrese *http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_library_alter/7*.

Používání zobrazitelných polí

Zobrazitelné pole Drupalu představuje způsob, jakým Drupal reprezentuje převážnou část výstupu, než ji převede do kódu jazyka HTML. Funkce témat vzhledu z kapitoly 3, „Témata vzhledu Drupal," vracejí buď zobrazitelná pole, nebo kód jazyka HTML. Pokud vrátí zobrazitelné pole, Drupal jej převede do kódu jazyka HTML pomocí funkce `drupal_render()`. Pokud znáte rozhraní Form API ze starších verzí Drupalu, viděli jste určitě i zobrazitelná pole. Jedná se totiž o pole rozhraní Form API vyčleněná k dalším účelům. Zobrazitelné pole pro formulářový prvek může vypadat takto:

```
$form['options'] = array(
  '#type' => 'textfield',
  '#title' => t('Jméno autora'),
  '#maxlength' => 25,
  '#attached' => array(
    'css' => array(
      drupal_get_path('module', 'hello_world') . '/example.css',
    ),
    'js' => array(
```

```
      "alert('Ahoj světe!')" => array('type' => 'inline'),
    ),
  ),
);
```

Pomocí vlastnosti `#attached` u zobrazitelného nebo formulářového pole můžeme přidávat kód jazyka JavaScript, jazyka CSS a knihovny. Klíče jsou následující:

- Klíč `js` pro kód jazyka JavaScript.
- Klíč `css` pro kód jazyka CSS.
- Klíč `library` pro přidávání knihoven.

Pro každý prvek v dílčích polích (jako je například pole `js`) platí, že klíč označuje data a hodnota představuje možnosti nastavení. V případě, že vynecháme hodnotu, Drupal použije výchozí možnosti nastavení.

Používání zobrazitelných polí a připojování kódu jazyků JavaScript a CSS a knihoven je důležité na různých místech Drupal, v nichž se ukládají do mezipaměti jednotlivé elementy. Jako příklad můžou posloužit bloky, které lze ukládat do mezipaměti. Běžnou situací je, že k bloku přidáváme kód jazyka JavaScript. Jestliže přidáváme kód jazyka JavaScript uvnitř obsahu daného bloku funkcí `drupal_add_js()`, přidá se pouze za předpokladu, že se tento blok neukládá do mezipaměti. Bloky uložené v mezipaměti nesestavují znovu svůj obsah, takže funkce `drupal_add_js()` se nezavolá.

Jestliže místo toho použijeme zobrazitelné pole s připojeným kódem jazyka JavaScript, uloží se do mezipaměti toto zobrazitelné pole, které obsahuje volání daného kódu JavaScriptu. Jakmile Drupal zobrazuje obsah uložený v mezipaměti, vloží do stránky rovněž připojený kód jazyka JavaScript, jazyka CSS a knihovny.

Příklad přidávání souborů JavaScriptu a jazyka CSS k výstupu bloku by vypadal následovně:

```
$output['content'] = array(
  '#value' => 'Obsah bloku.',
  '#attached' => array(
    'css' => array(
      drupal_get_path('module', 'hello_world') . '/example.css',
    ),
    'js' => array(
      "alert('Ahoj světe!')" => array('type' => 'inline'),
    ),
  ),
);
```

Změna kódu JavaScriptu

Poslední šance na změnu kódu JavaScriptu nastává před zobrazením výstupní stránky. Těsně předtím, než se kód JavaScriptu zobrazí do kódu jazyka HTML, prochází přes háček `hook_js_alter()`. Modul implementující háček `hook_js_alter()` tak získává poslední šanci, jak jednat podle kódu JavaScriptu nebo jej změnit.

Příkladem může být, kdyby modul chtěl prohodit komprimovanou verzi knihovny jQuery za její nekomprimovanou verzi. To by bylo užitečné pro účely ladění. V tomto příkladu zavoláme modul jQuery Uncompressed. Implementace háčku `hook_js_alter()` v souboru *jquery_uncompressed.module* by vypadala následovně:

```
/**
 * Implementuje háček hook_js_alter().
 */
function jquery_uncompressed_js_alter(&$javascript) {
  $path = drupal_get_path('module', 'jquery_uncompressed');
  $javascript['misc/jquery.js']['data'] =
    $path . '/jquery.uncompressed.js';
}
```

Předané pole `$javascript` obsahuje veškerý kód JavaScriptu vkládaný do aktuální stránky. Drupal volá tento háček těsně předtím, než zobrazí kód JavaScriptu do stránky. Jedná se o poslední bod, v němž jej můžeme změnit.

Klíče tohoto pole závisí na typu přidávaného kódu.

- Soubor a externí kód JavaScriptu identifikujeme pomocí cesty k jeho umístění.
- Nastavení jsou k dispozici v poli `$javascript['settings']`.
- Vložený kód JavaScript nelze snadno rozpoznat. Kdykoliv přidáme vložený skript do stránky, získá číselnou hodnotu (počínaje nulou). Kdybychom do stránky přidali dva vložené skripty, našli bychom je pod výrazy `$javascript[0]` a `$javascript[1]`. Tato čísla závisí na pořadí, v jakém vkládáme kód do stránky, což je nežádoucí.

Poznámka

Dokumentace rozhraní API pro háček `hook_js_alter()` je k dispozici na adrese *http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_js_alter/7* .

Změna kódu jazyka CSS

Kód jazyka CSS můžeme změnit podobným způsobem, a to pomocí háčku `hook_css_alter()`. Drupal předává pole s veškerým kódem jazyka CSS háčku `hook_css_alter()` – to je poslední šance, jak jej změnit před zobrazením do kódu jazyka HTML a přidáním do stránky. Jako příklad použití háčku `hook_css_alter()` si ukážeme, jak odstranit soubor *system.css* zavedený jádrem Drupalu:

```
/**
 * Implementuje háček hook_css_alter().
 */
function example_css_alter(&$css) {
  unset($css[drupal_get_path('module', 'system') . '/system.css']);
}
```

Rozpoznávání prvků tohoto pole je stejné jako pro kód JavaScriptu. Klíčem pro interní a externí soubory je cesta k souboru, kterou jsme použili při jeho vkládání. Vložené kaskádové styly mají číselné klíče s pořadím, v jakém jsme je přidávali.

Poznámka

Dokumentaci rozhraní API pro háček `hook_css_alter()` najdete na adrese *http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_css_alter/7*.

Kód JavaScriptu určený pro systém Drupal

V Drupal existuje několik pomocných funkcí. Existuje celá řada těchto funkcí – od funkcí pro témata vzhledu a překlad až po pomocné funkce, které analyzují formát JSON.

Prezentace s možností použití témat vzhledu

Celou prezentaci v Drupalu je možné upravit prostřednictvím systému témat vzhledu a kód jazyka JavaScript není žádnou výjimkou. Drupal poskytuje systém pro stylování prezentace generované kódem JavaScriptu, který lze přepsat uvnitř kódu JavaScriptu v tématu vzhledu. Můžeme začít příkladem *hello_world.js*, jejž vkládá modul vypadající takto:

```
(function($) {
  $().ready(function() {
    $('#hello-world').html('<h2>Ahoj světe!</h2>');
  });
})(jQuery);
```

Jakmile se nahraje stránka, výše uvedený kód nahradí kód jazyka HTML uvnitř elementu s identifikátorem `hello-world` kódem `<h2>Ahoj světe!</h2>`. Co kdybychom chtěli v tématu vzhledu nahradit obalující element `h2` elementem `h3`? V takovém případě přijde vhod stylování kódu JavaScriptu.

Modul by měl nabízet funkci tématu vzhledu uvnitř jmenného prostoru `Drupal.theme.prototype`. Potom bychom k této funkci tématu vzhledu přistupovali v modulu voláním `Drupal.theme()`. Díky tomuto systému může téma vzhledu poskytovat přepisující funkci v jmenném prostoru `Drupal.theme`. Níže uvedený příklad demonstruje toto řešení.

```
(function($) {
  Drupal.theme.prototype.hello = function(text) {
    return '<h2>' + text + '</h2>';
  }
  $().ready(function() {
    $('#hello-world').html(Drupal.theme('hello', 'Ahoj světe!'));
  });
})(jQuery);
```

Funkce `Drupal.theme()` zavolá funkci `Drupal.theme.prototype.hello()` a předá jí všechny argumenty, přestože v tomto případě jen jeden. To je užitečné, protože téma vzhledu může definovat přepisující funkci. Kdyby modul obsahoval výše uvedený soubor *hello_world.js* a téma vzhledu obsahovalo kód JavaScriptu s následující funkcí, změnil by se výsledný výstup.

```
(function($) {
  Drupal.theme.hello = function(text) {
    return '<h3>' + text + '</h3>';
```

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

Funkce `Drupal.theme()` zavolá funkci `Drupal.theme.hello()` místo funkce `Drupal.theme.prototype.hello()`. Tímto způsobem může téma vzhledu přepisovat prezentaci generovanou JavaScriptem.

Přeložitelné textové řetězce

Veškerý text uvnitř rozhraní Drupalu je možné přeložit. Text v kódu JavaScriptu lze přeložit stejně tak. V kódu jazyka PHP můžeme volat funkci `t()` pro překlad. V kódu JavaScriptu se o tuto úlohu stará funkce `Drupal.t()`.

Budeme pokračovat v našem modul Hello World a rozšíříme jej tak, abychom pozdravili různá města, přičemž část s textem „ahoj" budeme moct přeložit.

```
(function($) {
  Drupal.theme.prototype.hello = function(text) {
    return '<h2>' + Drupal.t('Ahoj @city', {'@city': text}) + '</h2>';
  }
  $().ready(function() {
    $('#hello-world').html(Drupal.theme('hello', 'Brno'));
  });
})(jQuery);
```

Zde nás zajímá především tato část:

```
Drupal.t('Ahoj @city', {'@city': text})
```

Textový řetězec `'Ahoj @city'` je přeložitelným textovým řetězcem. Část `@city` je dynamická a řídí se hodnotou parametru `text`. Takto umožňujeme překladům změnit daný textový řetězec, který si přitom zachovává svou dynamickou část.

Existují tři způsoby, jak předávat proměnné funkci `Drupal.t()`, které se liší v prvním znaku této proměnné:

- Proměnné začínající znakem `!` se vloží beze změny.
- Když proměnná začíná znakem `@`, volá se na její hodnotu funkce `Drupal.checkPlain()` – funkce, jež převádí textový řetězec na prostý text. Tato funkce převádí kód jazyka HTML na text, který lze zobrazit.
- Znak `%` na začátku proměnné způsobí, že Drupal zavolá na její hodnotu funkci `Drupal.checkPlain()` a `Drupal.theme('placeholder')`.

Chování

Pokud umíte pracovat s knihovnou jQuery, víte, že kód, jejž chcete spustit bezprostředně po načtení stránky, obalujete do takového kódu:

```
$(document).ready(function() {
  ...
});
```

Jakmile bude dokument připravený, kód uvnitř této funkce se provede. Jedná se o obvyklý vzor pro kód knihovny jQuery. Systém Drupal nabízí systém **chování**, který obaluje a rozšiřuje tuto koncepci. Chování připojujeme a odpojujeme od různých částí obsahu. Nejtypičtějším případem je připojit chování k celé stránce.

Náš dřívější příklad Hello World bychom mohli přepsat tak, aby používal chování, následujícím způsobem:

```
(function($) {
  Drupal.theme.prototype.hello = function(text) {
    return '<h2>' + Drupal.t('Ahoj @city', {'@city': text}) + '</h2>';
  }
  Drupal.behaviors.helloWorld = {
    attach: function(context, settings) {
      $('#hello-world', context).html(Drupal.theme('hello', 'Brno'));
    }
  }
})(jQuery);
```

Naše chování se skrývá pod názvem `Drupal.behaviors.helloWorld` a má funkci `attach()`, která připojuje toto chování k stránce. Když je dokument připravený, Drupal volá funkce `attach()` pro všechna chování. Předává jim kontext (aktuální dokument) a nastavení, což je v případě načtení stránky hodnota `Drupal.settings`.

Na první pohled se může tato metoda zdát neintuitivní v porovnání s běžnějším vzorem používaným v knihovně jQuery. Vzor chování ale nabízí mnohem více. Když kupříkladu vkládáme do stránky obsah načtený AJAXem, předává se funkcím `attach()` pro všechna chování. Argumentem `context` je v tomto případě obsah získaný ajaxovým požadavkem a argumentem `settings` jsou buď nastavení vrácená ajaxovým požadavkem, nebo nastavení z vlastnosti `Drupal.settings`.

Poznámka

Více informací o kontextu najdete na adrese *http://api.jquery.com/jQuery/*.

Ruku v ruce se schopností připojovat chování jde schopnost odpojovat chování. Typická struktura chování vypadá takto:

```
(function($) {
  Drupal.behaviors.example = {
    attach: function(context, settings) {
      ...
    }
    detach: function(context, settings, trigger) {
      ...
    }
  }
})(jQuery);
```

Když kupříkladu v kódu JavaScriptu založeném na technologii AJAX odstraňujeme obsah části stránky předtím, než jej nahradíme novým obsahem, zavolají se na odstraňovaný obsah funkce `dettach()` pro chování. Díky tomu získávají chování poslední příležitost reagovat na tuto akci.

Příkladem může být samotný systém AJAXu v Drupalu. Než odešleme Drupalu formulář pomocí technologie AJAX, zavolají se funkce `dettach()` pro všechna chování. Takto získávají chování příležitost k tomu, aby odstranila všechny změny ve formuláři, které by se neměly odeslat Drupalu. Jakmile vložíme do stránky nový obsah, tato chování se znovu připojí.

Pomocné funkce technologie AJAX

V Drupalu se nachází knihovna AJAX, jež propojuje Drupal s knihovnou jQuery a technologií AJAX. Tento systém poskytuje sadu vlastností pro formuláře a funkce, které lze používat ve funkcích zpětného volání, jež rychle a jednoduše zavádí technologii AJAX do modulů Drupalu. Prostřednictvím pomocných funkcí a vlastností je možné zabudovat technologii AJAX do stránek Drupalu rychlostí blesku.

Přidávání technologie AJAX k formulářům

Běžně používáme technologii AJAX tak, že dynamicky aktualizujeme formuláře; počínaje elementy `input` až po jiné části formuláře. Jakmile aktualizujeme jeden formulářový prvek, ostatní prvky se změní nebo naplní daty podle provedené změny. Například si můžeme popsat formulářový prvek se seznamem možností, který aktualizuje element stránky.

Začneme implementací háčku `hook_menu()`, v němž definujeme stránku s formulářem:

```
/**
 * Implementuje háček hook_menu().
 */
function hello_world_menu() {
  $items = array();

  $items['hello_world/simple_form_example'] = array(
    'title' => 'Ahoj světe: jednoduchý příklad AJAXu',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('hello_world_simple_form'),
    'access callback' => TRUE,
  );

  return $items;
}
```

Pokračujeme tvorbou funkce zpětného volání s názvem `hello_world_simple_form()` pro formulář. Tím vytvoříme formulář, který lze vložit do stránky.

```
function hello_world_simple_form($form, &$form_state) {
  $form = array();
  $form['hello_city'] = array(
    '#title' => t("Vyberte město"),
    '#type' => 'select',
    '#options' => array(
      'světe' => 'světe',
      'Brno' => 'Brno',
      'Ostravo' => 'Ostravo',
```

```
      'Praho' => 'Praho',
    ),
    '#ajax' => array(
      'callback' => 'hello_world_simple_form_callback',
      'wrapper' => 'ajax_markup_div',
     ),
  );

  $form['ajax_markup'] = array(
    '#prefix' => '<div id="ajax_markup_div">',
    '#suffix' => '</div>',
    '#markup' => 'Ahoj světe',
  );
  if (!empty($form_state['values']['hello_city'])) {
    $form['ajax_markup']['#markup'] =
      t("Ahoj") .  " {$form_state['values']['hello_city']}";
  }

  return $form;
}
```

Zde máme dva formulářové prvky – `hello_city` a `ajax_markup`. Prvek `hello_city` má vlastnost `#ajax`, která je nová v systému Drupal 7. V této vlastnosti definujeme funkci zpětného volání, která je interní funkcí zpětného volání Drupalu, a obalující element. Jedná se o obalující element na stránce, jehož obsah aktualizujeme odpovědí na ajaxový požadavek.

Druhý prvek s názvem `ajax_markup` je formulářovým prvkem, která ukládá kód jazyka HTML. Zpočátku vyplníme tento prvek textem „Ahoj světe.“ Abychom mohli tento prvek aktualizovat, musíme k němu přidat obalující element. V tomto případě je tímto obalujícím elementem element `div` s identifikátorem, který jsme nastavili ve vlastnosti `#ajax` prvku `hello_city`.

Jakmile nastavíme formulář, uvádíme příkaz `if` pro případ, že by pole `$form_state` obsahovalo hodnotu pro prvek `hello_city`. Při úvodním vzniku tohoto formuláře zde žádná hodnota nebude. Jestliže uděláme ajaxový požadavek, projde přes tento formulář spolu s hodnotami z daného formuláře. Až se to stane, provede se tělo daného příkazu `if` a aktualizuje obsah prvku `ajax_markup`.

Potom bude následovat spuštění funkce zpětného volání definované ve vlastnosti `#ajax` na prvku `hello_city`. Tato funkce vypadá následovně:

```
function hello_world_simple_form_callback($form, $form_state) {
  return $form['ajax_markup'];
}
```

Funkce zpětného volání `hello_world_simple_form_callback()` přijímá argumenty `$form` a `$form_state` poté, co jsme se zpracovali ve funkci `hello_world_simple_form()`. V tomto případě vracíme nahrazovaný formulářový prvek.

Drupal ví, že se jedná o zobrazitelné pole a zobrazí jej do příslušné hodnoty. Drupal odešle aktualizovaný kód jazyka HTML zpět stránce, v níž obsluhující ajaxové funkce získají změny a nahradí obalující element.

Automatické aplikovaní AJAXu

Technologii AJAX je možné automaticky aplikovat na elementy na stránce. Toho dosáhneme přiřazením třídy `use-ajax` na stránce. Běžné použití třídy `use-ajax` je přidat ji odkazu ve stránce za účelem spuštění ajaxové akce. Odkazy jsou častou volbou, protože odkaz může být zálohou v případě, že je jazyk JavaScript vypnutý.

V následujícím příkladu vytvoříme odkaz, přičemž klepnutí na něj způsobí, že se do elementu `div` přidá text „Ahoj světe.“ Pro začátek přidáme dvě funkce zpětného volání nabídky do háčku `hello_world_menu()`. Jednu položku nabídky použijeme pro generovanou stránku a druhou jako adresu URL zpětného volání pro AJAX nebo v případě, že je jazyk JavaScript zakázaný.

```
$items['hello_world/link'] = array(
  'title' => 'Ahoj světe: odkaz',
  'page callback' => 'hello_world_link',
  'access callback' => 'user_access',
  'access arguments' => array('access content'),
);
$items['hello_world_link_callback'] = array(
  'page callback' => 'hello_world_link_response',
  'access callback' => 'user_access',
  'access arguments' => array('access content'),
);
```

První položkou nabídky se odkazujeme na stránku, v níž se nachází náš odkaz a do které budeme přidávat obsah AJAXem. Druhou položkou nabídky je adresa zpětného volání, která vyřídí ajaxový požadavek nebo požadavek na stránku, pokud není jazyk JavaScript dostupný.

```
/**
 * Používá technologii AJAX pomocí třídy use-ajax.
 *
 * V následujícím příkladu aplikujeme třídu use-ajax na odkaz, abychom
 * přidali AJAX do stránky. Na konci adresy zpětného volání uvádíme
 * speciální textový řetězec /nojs/. Pokud je jazyk JavaScript povolený,
 * zmizí z adresy URL. Jestliže je jazyk JavaScript vypnutý, informuje
 * zobrazovací funkci o této skutečnosti.
 *
 * @see hello_world_link_response
 */
function hello_world_link() {
  drupal_add_js('misc/ajax.js');
  $link = l(t('Pozdravit'), 'hello_world_link_callback/nojs/',
    array('attributes' => array('class' => array('use-ajax'))));
  return '<div>' . $link . '</div><div id="saying-hello"></div>';
}
```

V této funkci začínáme přidáním souboru *misc/ajax.js* pomocí volání `drupal_add_js()`. Tento kód JavaScriptu automaticky zprovozňuje technologii AJAX. Posléze vytváříme odkaz s adresou zpětného volání *hello_world_link_callback/nojs/*. První část této adresy představuje skript, který zpracovává ajaxový požadavek. Výraz */nojs/* na konci je speciální. Jestliže není jazyk JavaScript k dispozici, získá jej odpovídající funkce, aby věděla, že se musí postarat o celou stránku. Pokud je

jazyk JavaScript povolený, nahradí se výrazem */ajax/*. Tento výraz obdrží funkce zpětného volání, aby věděla, že ji zavolal ajaxový požadavek.

Co z tohoto odkazu dělá ajaxový odkaz, je jemu přidělená třída `use-ajax`. Soubor jazyka JavaScript, jejž jsme přidali na začátku (*ajax.js*), vyhledává odkazy s touto třídou a převádí je na ajaxové odkazy.

```
/**
 * Funkce zpětného volání pro ukázkový odkaz.
 *
 * @param $type
 *   'ajax' nebo 'nojs'. Typ se nachází na konci adresy zpětného volání.
 *   Pokud je jazyk JavaScript povolený, nahradí se výraz /nojs/ výrazem
 *   /ajax/, a ten se předá jako argument $type.
 */
function hello_world_link_response($type = 'ajax') {
  if ($type == 'ajax') {
    $output = t("Ahoj světe!");
    $commands = array();
    $commands[] = ajax_command_append('#saying-hello', $output);
    $page = array('#type' => 'ajax', '#commands' => $commands);
    ajax_deliver($page);
  }
  else {
    return t("Ahoj světe v nové stránce.");
  }
}
```

Jakmile Drupal zavolá funkci zpětného volání pro zpracování požadavku, předá mu typ akce. Jestliže není jazyk JavaScript dostupný, předá mu typ `'nojs'`. V případě, že se jedná o ajaxový požadavek, předá mu typ `'ajax'`. Díky tomuto typu můžeme v kódu funkcí zpětného volání reagovat správně na jednotlivé situace.

V tomto případě používáme příkaz `if`. Když se jedná o ajaxový požadavek, reagujeme jedním způsobem; pokud není jazyk JavaScript k dispozici, reagujeme jinak.

V případě ajaxového požadavku vytváříme nejprve text „Ahoj světe!“ Potom vytváříme pole pro uchování příkazů, které chceme, aby Drupal provedl, a vkládáme do něj příkaz.

Příkaz je akce, kterou má provést kód JavaScriptu, jakmile obdrží ajaxovou odpověď. Příkazy umožňují kódu knihovny jQuery manipulovat s obsahem stránky obsahujícím data odpovědi. V tomto případě používáme příkaz `ajax_command_append()`. Tento příkaz přijímá selektor knihovny jQuery a obsah. Výsledkem je přidání daného obsahu k elementům specifikovaným daným selektorem. Tato funkce systému Drupal používá funkci `jQuery.append()`.

Jakmile nastavíme odpověď, vkládáme ji do zobrazitelného pole. Typem je `ajax`, aby Drupal věděl, jak má zobrazit ajaxový příkaz vytvořený v této funkci zpětného volání. Abychom nastavili ajaxovou odpověď správně, voláme funkci `ajax_deliver()`. Touto funkcí formátujeme odpoveď pro kód JavaScriptu na přijímající straně.

Systém Drupal navíc sleduje soubory jazyků JavaScript a CSS ve stránce. Jestliže ajaxová odpověď přidává nový soubor, který zatím prohlížeč nenahrál do stránky, tento nový soubor se odešle jako součást odpovědi a přidá se do stránky spolu se zbytkem odpovědi.

Pokud není jazyk JavaScript k dispozici ani při úvodním prohlížení stránky, přejde prohlížeč na cíl odkazu a zachází s ním jako s kompletním požadavkem na stránku, přičemž přesměruje uživatele na novou stránku. Tato stránka se nachází na stejné adrese zpětného volání, jakou používáme pro ajaxovou odpověď. Rozdílem je, že příslušná funkce zpětného volání získá typ `'nojs'`, aby věděla, že se nejedná o ajaxovou odpověď. V tomto případě se provede větev `else` generující jinou zprávu pro novou stránku.

Ajaxové příkazy

Drupal nabízí několik ajaxových příkazů, které umí přidávat a měnit obsah stránky prostřednictvím metod knihovny jQuery. V předchozí části této kapitoly jsme se zabývali příkazem `ajax_command_append()`. Zde si uvedeme všechny možné příkazy, které můžeme používat.

Příkaz ajax_command_after()

Jestliže použijeme příkaz `ajax_command_after()`, Drupal přidá obsah do stránky metodou `jQuery.after()`. Tento příkaz má parametry `$selector`, `$content` a `$settings`. Parametrem `$selector` je selektor knihovny jQuery a parametr `$content` představuje obsah, který přidáváme za elementy vybrané zadaným selektorem. Posledním parametrem `$settings` je sada nastavení, která používají chování pro tento jediný příkaz.

Příkaz ajax_command_alert()

Funkce `alert()` je standardní funkce jazyka JavaScript. Pomocí příkazu `ajax_command_alert($text)` prezentujeme vrácený text ve výstražném dialogovém okně.

Příkaz ajax_command_append()

Tento příkaz se podobá příkazu `ajax_command_after()`. Akorát nepřidává obsah za elementy vybrané selektorem, ale na jejich konec. Tato funkce má stejné rozhraní jako funkce `ajax_command_after()` – parametry `$selector`, `$content` a `$settings`. Parametrem `$selector` je selektor knihovny jQuery a parametr `$content` představuje obsah, který přidáváme na konec elementů vybraných zadaným selektorem. Sadu nastavení `$settings` používá chování pouze pro tento jeden příkaz.

Příkaz ajax_command_before()

Obsah přidáme před element prostřednictvím příkazu `ajax_command_before()`. Tento příkaz vkládá obsah před elementy definované selektorem funkcí `jQuery.before()`. Opět se zde setkáváme s parametry `$selector`, `$content` a `$settings`.

Příkaz ajax_command_changed()

Oznámit, že se něco na stránce změnilo, můžeme pomocí volání `ajax_command_changed($selector, $asterisk)`. Drupal vyhledá na stránce všechny elementy odpovídající danému selektoru knihovny jQuery a přidělí jim třídu `ajax-changed`. Parametr `$asterisk` je volitelný selektor jazyka CSS, s nímž můžeme hledat uvnitř skupiny vymezené selektorem `$selector`.

Příkaz ajax_command_css()

Příkaz `ajax_command_css()` používá metodu `jQuery.css()` k aktualizaci kaskádových stylů uvnitř stránky. Tento příkaz má parametry `$selector` a `$argument`. Barvu pozadí stránky bychom kupříkladu změnili následovně:

```
$commands[] = ajax_command_css('body',
  array('background-color' => '#FFFFFF'));
```

Příkaz ajax_command_data()

Knihovna jQuery poskytuje metodu `data()` pro ukládání dat do stránky mimo atributy daného elementu. Funkce `ajax_command_data()` umožňuje technologii AJAX v Drupalu přidávat a aktualizovat data uvnitř mezipaměti dat knihovny jQuery. Jejími třemi parametry jsou:

- Parametr `$selector` – selektor knihovny jQuery.
- Parametr `$name` – jméno datového prvku, k němuž přistupujeme.
- Parametr `$value` – hodnota pro daný prvek.

Příkaz ajax_command_html()

Příkaz `ajax_command_html()` používá metodu `jQuery.html()` k aktualizaci kódu jazyka HTML pro elementy určené daným selektorem. Parametry tohoto příkazu jsou `$selector` (selektor knihovny jQuery), `$html` (aktualizovaný kód jazyka HTML) a `$settings` (volitelná nastavení pro tento příkaz).

Příkaz ajax_command_prepend()

Obsah přidáme na začátek elementu pomocí příkazu `ajax_command_prepend()`. Tento příkaz přidává obsah prostřednictvím funkce `jQuery.prepend()`. Parametry příkazu `ajax_command_prepend()` jsou `$selector`, `$content` a `$settings`.

Příkaz ajax_command_remove()

Příkaz `ajax_command_remove()` odstraňuje elementy ze stránky. Jediným parametrem je selektor pro výběr odstraňovaných elementů. Tento příkaz odstraňuje elementy ze stránky pomocí metody `jQuery.remove()`.

Příkaz ajax_command_replace()

Příkaz `ajax_command_html()` nahrazuje kód jazyka HTML uvnitř elementu. Jestliže chceme nahradit celý element, měli bychom použít příkaz `ajax_command_replace()`. Tento příkaz používá metodu `jQuery.replaceWith()` pro nahrazování celého elementu. Třemi jeho parametry jsou `$selector`, `$html` a `$settings`. Jako argument `$html` předáváme kompletní kód jazyka HTML pro nahrazení. Mějme například tento kód jazyka HTML:

```
<div class="container">
  <div class="inner">Ahoj světe!</div>
</div>
```

Příkaz `ajax_command_replace()` vypadá následovně:

```
$commands[] = ajax_command_replace('.inner', '<h2>Nashledanou světe!</h2>';
```

Tento příkaz změní výše uvedený kód jazyka HTML následujícím způsobem:

```
<div class="container">
  <h2>Nashledanou světe!</h2>
</div>
```

Příkaz ajax_command_restripe()

Příkazem `ajax_command_restripe()` říkáme Drupalu, že bychom chtěli znovu proužkovat tabulku. To je užitečné, když se změní obsah tabulky. Jediným parametrem je selektor knihovny jQuery pro danou tabulku.

Příkaz ajax_command_settings()

Pomocí příkazu `ajax_command_settings()` přidáváme nastavení k odpovědi. Prvním parametrem jsou nastavení odesílaná spolu s odpovědí. Jestliže předáme pouze první argument, druhým argumentem bude hodnota `false`, a v tom případě se tato nastavení použijí pouze pro danou odpověď. Pokud předáme jako druhý argument (`$merge`) hodnotu `true`, tato nastavení se spojí s nastaveními `Drupal.settings`.

Poznámka

Více informací o rozhraní API knihovny jQuery najdete na adrese *http://api.jquery.com/*.

Shrnutí

Popsali jsme si základní principy a příkazy pro používání jazyka JavaScript a knihovny jQuery v kontextu systému Drupal. Začali jsme přidáváním JavaScriptu do stránky v podobě souboru, vloženého kódu a jako nastavení. Pokračovali jsme přidáváním kompletních knihoven se závislostmi a změnou kódu jazyka JavaScript bezprostředně před zobrazením stránky.

Systém Drupal nabízí pomocné funkce a knihovny, které můžeme používat při tvorbě jeho modulů. Rozebrali jsme si, jak tyto knihovny a některé nejčastěji používané prvky fungují.