Zrąb JavascriptMVC

Krzysztof Płocharz

Uniwersytet Warszawski

6 kwiecień 2009

Spis Treści

MVC

Wzorzec architektoniczny mający na celu wydzielenie trzech podstawowych części aplikacji:

- Model - reprezentacja danych. Pozwala abstrahować od sposobu przechowywania i pobierania danych.
- Widok - interfejs użytkownika, pozwala na komunikację z użytkownikiem.
- Kontroler - logika aplikacji. Łączy model z widokiem.

MVC w JavascriptMVC

Javascript jest językiem używanym głównie po stronie klienta w przeglądarkach. Chcąc mieć przyzwoity zrąb część aplikacji musi znajdować się jednak po stronie serwera. Za tę część odpowiedzialny jest Rhino.

Prawie jak klasy

W Javascript występują tylko obiekty, bez klas. Klasy są symulowane przez specjalne funkcje. W JavascriptMVC podstawową "klasą" jest Class:

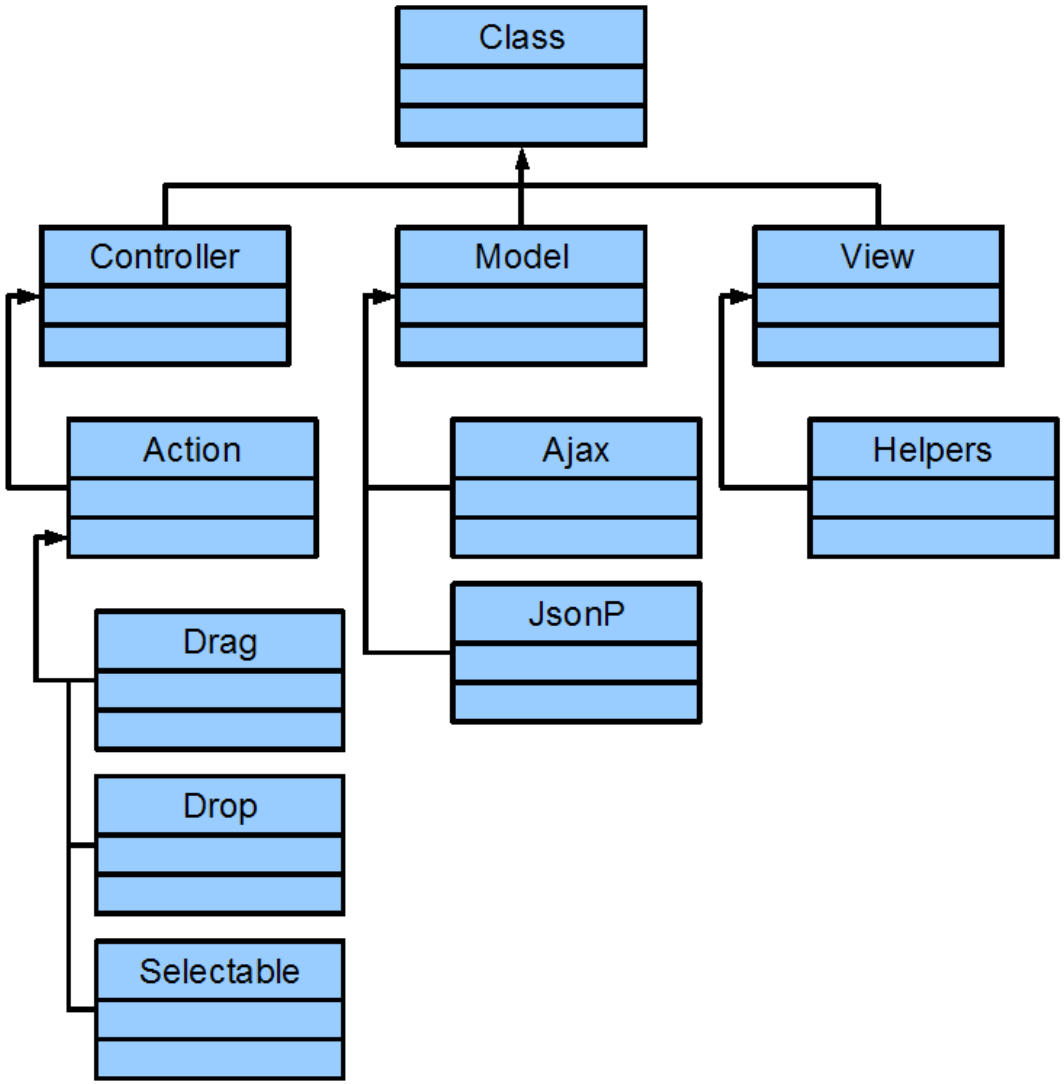
- `extend` - Tworzy nową klasę dziedziczącą po innej klasie.
- `init` - Konstruktor, funkcja wywoływana gdy instancja klasy jest tworzona.
- `_super` - Konstruktor klasy nadrzędnej.
- `className` - Nazwa klasy obiektu.

Przykład

Dziedziczenie

```
Monster = MVC.Class.extend('monster',
 /* @static */
 {
   count: 0
 },
 /* @prototype */
 {
   init : function(name){
     this.name = name;
     this.Class.count++
   }
 })
 hydra = new Monster('hydra')
 dragon = new Monster('dragon')
 hydra.name          // -> hydra
 Monster.count       // -> 2
 Monster.className   // -> 'monster'
```

Hierarchia


Class
Controller
Model
View
Action
Ajax
Helpers
JsonP
Drag
Drop
Selectable

Widoki

Pod tym względem JavascriptMVC jest podobny do Django. Widoki to głównie pliki html z dodatkowymi znacznikami. Plik html jest pobierany z serwera, a następnie parsowany. To co znajduje się w specjalnych tagach jest interpretowane jako JavaScript. Pliki html mogą być przechowywane w pamięci podręcznej, i nie muszą być za każdym razem pobierane z serwera.

Przykład

Kod HTML

```
  <h1><%= title %></h1>
<ul>
    <% for(var i=0; i<supplies.length; i++) { %>
      <li>
        <%= link_to(supplies[i], 'supplies/'+supplies[i]) %>
      </li>
    <% } %>
</ul>
```

Przykład

Kod HTML

```
  <h1><%= title %></h1>
<ul>
    <% for(var i=0; i<supplies.length; i++) { %>
      <li>
        <%= link_to(supplies[i], 'supplies/'+supplies[i]) %>
      </li>
    <% } %>
</ul>
```

Kod Javascript

```
var html = new View({
  url: 'views/cleaning.ejs'
}).render(data);
View({url: 'comments.ejs'}).update('element_id',
  '/comments.json')
```

Kontrolery

Kontrolery pozwalają przypisać kod Javascript do elementów strony. Elementy mogą być wybierane za pomocą id lub selektorów CSS. Nazwy funkcji to nazwy zdarzeń, którym mają odpowiadać.

Rozpoznawane zdarzenia:

`change, click, contextmenu, dblclick, mousedown, mousemove, mouseout, mouseover, mouseup, reset, resize, select, submit, dblclick, focus, blur, load, unload, keypress`

Przykład

Przykład kontrolera

```
TodosController = Controller.extend('todos',{
  // the onclick event handler
  click: function(){
    alert('clicked todo');
  },
  'ul.foo click': function(params){
    params.element.style.class = 'clicked';
  }
});
```

Modele

Modele dziedziczą po klasie Model udostępniającej kilka podstawowych metod do manipulacji poszczególnymi obiektami. Niestety większość z tych metod należy samemu napisać, jeżeli chcemy przykładowo korzystać z bazy danych znajdującej się na serwerze.

Modele

Modele dziedziczą po klasie Model udostępniającej kilka podstawowych metod do manipulacji poszczególnymi obiektami. Niestety większość z tych metod należy samemu napisać, jeżeli chcemy przykładowo korzystać z bazy danych znajdującej się na serwerze.

Trzeba zaimplementować:

- `find_one(params, callbacks)`
- `find_all(params, callbacks)`
- `create(attributes, callbacks)`
- `update(id, attributes, callbacks)`
- `destroy(id, callbacks)`

Modele

Modele dziedziczą po klasie Model udostępniającej kilka podstawowych metod do manipulacji poszczególnymi obiektami. Niestety większość z tych metod należy samemu napisać, jeżeli chcemy przykładowo korzystać z bazy danych znajdującej się na serwerze.

Trzeba zaimplementować:

- `find_one(params, callbacks)`
- `find_all(params, callbacks)`
- `create(attributes, callbacks)`
- `update(id, attributes, callbacks)`
- `destroy(id, callbacks)`

Powinniśmy zaimplementować:

- `find`
- `publish`

Przykład

Dobre rady

```
//Zamiast:
new Ajax('/tasks.json', {
    onComplete: find_tasks_next_week
  })
//powinniśmy pisać:
Task.find('all', find_tasks_next_week)

//Zamiast
new Ajax('/tasks/'+id+'/complete.json',{
    onComplete: task_completed
  })
//powinniśmy pisać:
task.complete(task\_completed)
```

Poszczególne typy elementów aplikacji trzymane są w różnych katalogach. Podstawowe katalogi to:

- `apps` - katalog aplikacji
- `controllers` - katalog kontrolerów
- `docs` - katalog docelowy dla dokumentacji. docelowy bo nie ma tutaj żadnej gotowej dokumentacji dla zrębu. Tutaj będzie utworzona dokumentacja na podstawie plików naszej aplikacji.
- `engines` - silniki
- `jmvc` - Pliki dostarczane przez zrąb, jak biblioteka standardowa, generatory itd.
- `models` - modele
- `tests` - Testy jednostkowe
- `views` - widoki

Generatory kodu

JavascriptMVC podobnie jak Django zawiera generatory kodu dla większości podstawowych elementów aplikacji:

- `app` Generator aplikacji
- `page` Generator stron
- `controller` Generator kontrolerów
- `model` Generator modeli
- są jeszcze inne generatory, niestety mało udokumentowane

Przykładowo:

```
js jmvc/generate/app
```

Dokumentacja

Ciekawy i przydatny dodatek. Automatycznie generowana dokumentacja na podstawie komentarzy (składnia taka sama jak JavaDoc). Generacja dokumentacji odbywa się za pomocą tego samego polecenia co kompresja.

Jak to połączyć?

JavascriptMVC udostępnia funkcję include odpowiedzialną za ładowanie elementów aplikacji. Funkcja automatycznie zajmuje się kolejnością ładowania elementów. Pozwala także na spakowanie wszystkich ładowanych plików w jedną całość, aby przyspieszyć ładowanie strony.

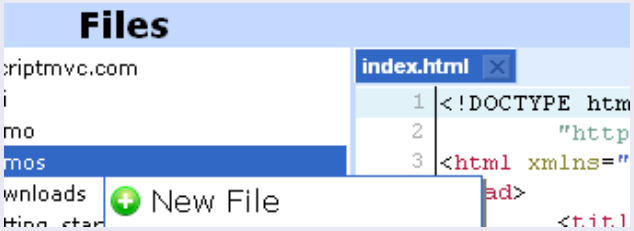
Kompresja strony

```
js apps/your_application_name/compress.js
```

Kto tego używa

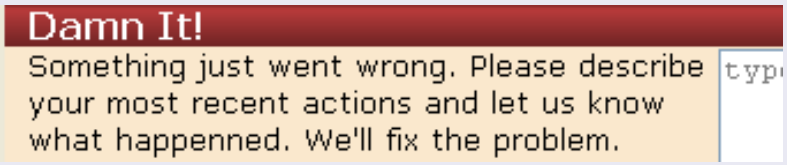
FIT

Klient
FTP z wbudowanym edytorem tekstu i podświetlaniem składni.



DamnIt

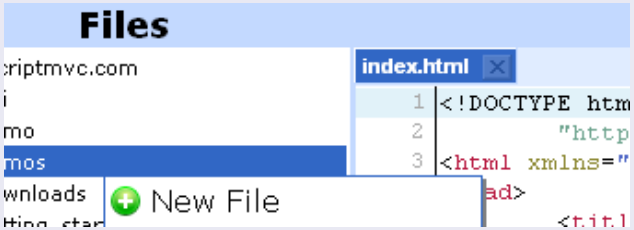
Serwis wysyłający
emaile, gdy użytkownik napotka na błąd Javascript na stronie.



Kto tego używa

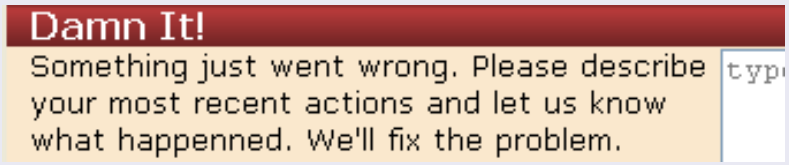
FIT

Klient
FTP z wbudowanym edytorem
tekstu i podświetlaniem składni.



DamnIt

Serwis wysyłający
emaile, gdy użytkownik napotka
na błąd Javascript na stronie.



Trafiik

Serwis łączący Google
Maps i Calendar - informacje
o miejscu i czasie wydarzeń.

Zalety i Wady

Zalety

- Pełny zrąb dla fanów Javascriptu
- sprawdzona idea
- automatyczne pakowanie całej aplikacji do jednego pliku
- testy jednostkowe
- generacja dokumentacji

Zalety i Wady

Zalety

- Pełny zrąb dla fanów Javascriptu
- sprawdzona idea
- automatyczne pakowanie całej aplikacji do jednego pliku
- testy jednostkowe
- generacja dokumentacji

Wady

- skromna dokumentacja z błędami
- część aplikacji po stronie serwera musi być samodzielnie napisana
- bardzo uproszczone modele
- niewiele przykładów działających aplikacji

Materiały

- http://javascriptmvc.com - strona główna projektu
- http://wikipedia.org
- http://groups.google.com/group/javascriptmvc - lista dyskusyjna