AJAX/JS OOP

Kamil Frydel

20 marca 2009

Spis treści

Co to jest?

JS

JavaScript (JS) – obiektowy skryptowy język programowania, stworzony przez firmę Netscape, najczęściej stosowany na stronach WWW. Pod koniec lat 90. XX wieku organizacja ECMA wydała na podstawie JavaScriptu standard języka skryptowego o nazwie ECMAScript. Głównym autorem JavaScriptu jest Brendan Eich. (wikipedia)

Co to jest? c.d.

JS - możliwości

1. dostarcza obiekty reprezentujące drzewo dokumentu
2. tworzenie ciasteczek
3. manipulowanie oknami przeglądarki
4. wyświetlanie okien dialogowych
5. dostarcza informacje o przeglądarce
6. zarządzanie pluginami przeglądarki
7. zarządzanie arkuszami stylów
8. reaguje na zdarzenia użytkownika
9. AJAX!

Co to jest? c.d.

JS - problemy

1. brak standardu
2. wiele szczegółów związanych ze specyfiką środowiska jakim jest przeglądarka
3. stosunkowo trudne rysowanie
4. niezbyt przyjemne debugowanie

Obiekt w JS

Obiekty

Można tworzyć pojedyncze obiekty poprzez zdefiniowanie jego atrybutów i metod. Przykład obiektu:

```
var cat = {
  color: 'black',
  run: function() {}
}
```

Obiekty - tablice asocjacyjne

Obiekty w uproszczeniu są tablicami asocjacyjnymi (przydatne gdy nazwy atrybutów mamy w postaci napisów):

```
cat.color == cat['color']
cat.run == cat['run']
```

Obiekt w JS c.d.

Dynamiczne rozszerzanie

Istniejące obiekty można dynamicznie rozszerzać o atrybuty / funkcje (można podmieniać metody):

```
cat.tail = 'very long'
```

Zagnieżdżenia

Obiekty można zagnieżdżać:

```
var rectangle = {
  upperLeft : { x : 2, y : 2 },
  lowerRight : { x : 4, y : 4}
}
```

Obiekt w JS c.d.

Problemy?

Nie możemu re-użyć napisanego kodu.
tzn. nie możemy zainicjować (w prosty sposób) kolejnego obiektu mającego takie same metody/atrybuty.

Klasy

Definicja

Klasy definiuje się za pomocą słowa kluczowego 'function':

```
function cat(name) {
  this.name = name;
  this.talk = function() {
    alert( this.name + " say meeow!" )
  }
}
```

Wyjaśnienie

- cat() - konstruktor
- w konstruktorze definicje atrybutów i metod muszą być poprzedzone słowem this

Klasy c.d.

Tworzenie instancji

Słowo kluczowe 'new'.

```
cat1 = new cat("felix")
cat1.talk() //alerts "felix says meeow!"
```

Uwagi

Dla każdej instancji utworzonej w ten sposób atrybuty i metody przechowywane są w pamięci osobno (prawdopodobnie).
Rozwiązanie - prototyp.

Prototyp

Rozszerzenie poprzedniej klasy

'prototype' szczególnym atrybutem instancji klasy.

```
cat.prototype.changeName = function(name)
    {
  this.name = name;
}

firstCat = new cat("pursur")
firstCat.changeName("Bill")
firstCat.talk() //alerts "Bill says meeow
    !"
```

Prototyp c.d.

Hint!

Rozszerzać można również wbudowane typy danych, np: Array.

```
// o ile metoda nie istnieje...
if(!Array.prototype.shift) {
  Array.prototype.shift = function(){
    firstElement = this[0];
    this.reverse();
    this.length=Math.max(this.length-1,0);
    this.reverse();
    return firstElement;
  }
}
```

Dziedziczenie

InheritFrom

Dziedziczenie przez wywołanie konstruktora nadklasy.

```
function superClass() {
  this.supertest = function(){
    return "superTest";
  }
}
function subClass() {
  this.inheritFrom = superClass;
  this.inheritFrom();
  this.subtest = function(){
    return "subTest";
  }
}
```

Kopiowanie obiektu

```
Object.prototype.begetObject = function(){
  function F() {}
  F.prototype = this;
  return new F();
}
oldObject = {
  attr: 8,
  fun: function(x){
    alert(x + this.attr)
  }
}
newObject = oldObject.begetObject();
```

Dziedziczenie za pomocą prototype

```
function Bazowa() {
  this.metodaA = function() {
    alert("Bazowa::A()");
  }
  this.metodaB = function() {
    alert("Bazowa::B()");
  }
}
function Pochodna() {
  this.metodaB = function () {
    alert("Pochodna::B()");
  }
}
Pochodna.prototype = new Bazowa();
```

Dziedziczenie za pomocą prototype - c.d.

```
x = new Bazowa();
y = new Pochodna();
x.metodaA(); // wyswietla: Bazowa::A()
y.metodaA(); // wyswietla: Bazowa::A()
x.metodaB(); // wyswietla: Bazowa::B()
y.metodaB(); // wyswietla: Pochodna::B()
```

Dziedziczenie przez kopiowanie atrybutów

Parent

Dodanie metody do wszystkich obiektów...

```
Object.prototype.parent = function (
    supClass) {
  tempObj = new supClass();
  for (property in tempObj) {
    this[property] = tempObj[property];
  }
}
```

Dziedziczenie przez kopiowanie atrybutów - c.d.

Parent

...aby móc kopiować atrybuty nadklasy w konstr. podklasy.

```
function Bazowa() {
  this.metoda1 = function () {
    alert("Bazowa::metoda1()");
  }
  this.metoda2 = function () {
    alert("Bazowa::metoda2()");
  }
}
function Pochodna() {
  this.parent(Bazowa);
  this.metoda1 = function () {
   alert("Pochodna::metoda1()");
  }
}
```

Dziedziczenie - podsumowanie

Wniosek

W JS nie dziedziczy się - dziedziczenie się uzyskuje stosując różne technologie.

Wyjątki

Wbudowane klasy wyjątków

Nazwa	**Opis**
EvalError	Błąd w wywołaniu eval.
RangeError	w = [1, 2], w[-1], w[3]
ReferenceError	Odwołanie do niezdefiniowanej zmiennej.
SyntaxError	Błąd składniowy.
TypeError	np. null.innerHTML
URIError	Błąd przy kodowaniu/odkodowywaniu URI (encodeURI())

Wyjątki - c.d.

Zgłaszanie

Standardowa składnia (pod warunkiem, że mydiv nie istnieje, zostanie zgłoszony wyjątek):

```
try{
  document.getElementById("mydiv").
     innerHTML='Success'
}catch(e){
  alert(e.name.toString())
} finally {
  alert("Finaly");
}
```

Wyjątki - c.d.

Zgłaszanie własnych wyjątków

Istnieje kilka sposobów zgłoszenia własnego wyjątku.
Nie ma właściwie żadnych ograniczeń co do tego, czym jest obiekt wyjątku.

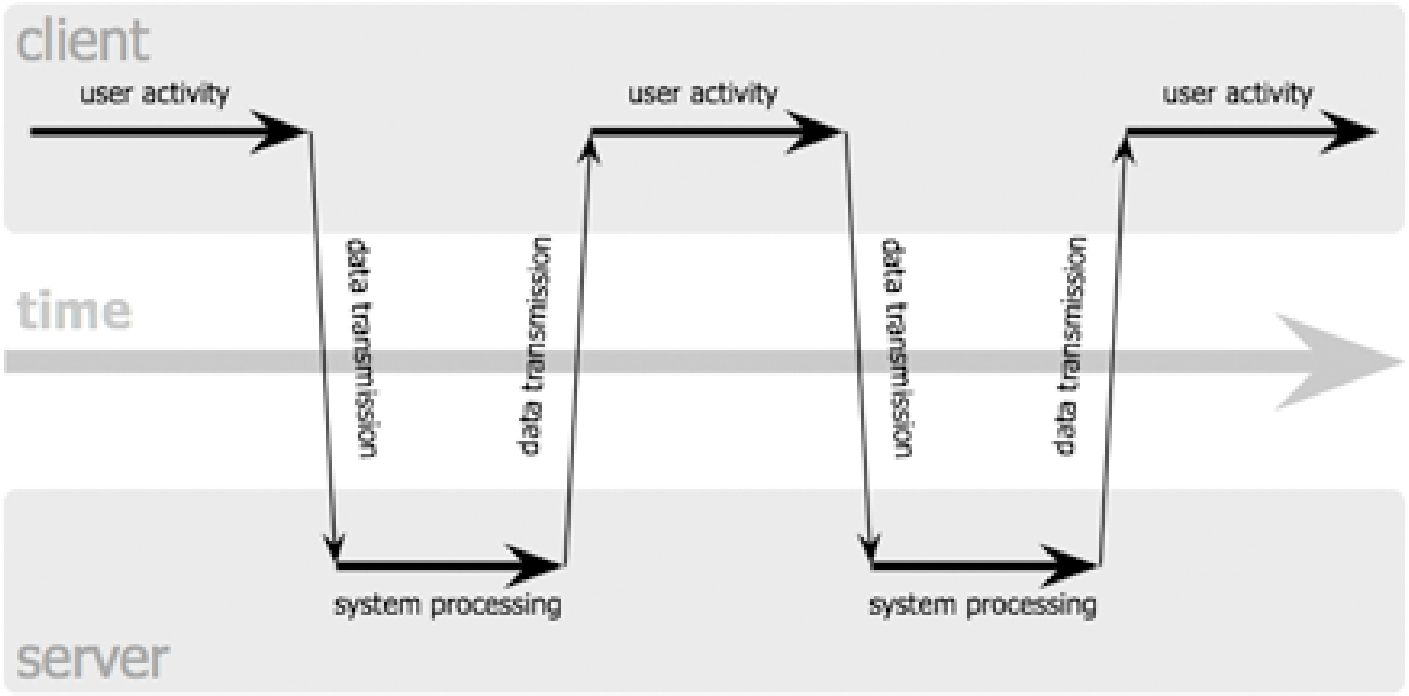
```
myerror = "Error"
throw myerror
throw new Error("An error has occured")
throw{
 name: "JavaScriptKit Error",
 message: "Error detected."
}
```

AJAX - Co to jest?

AJAX

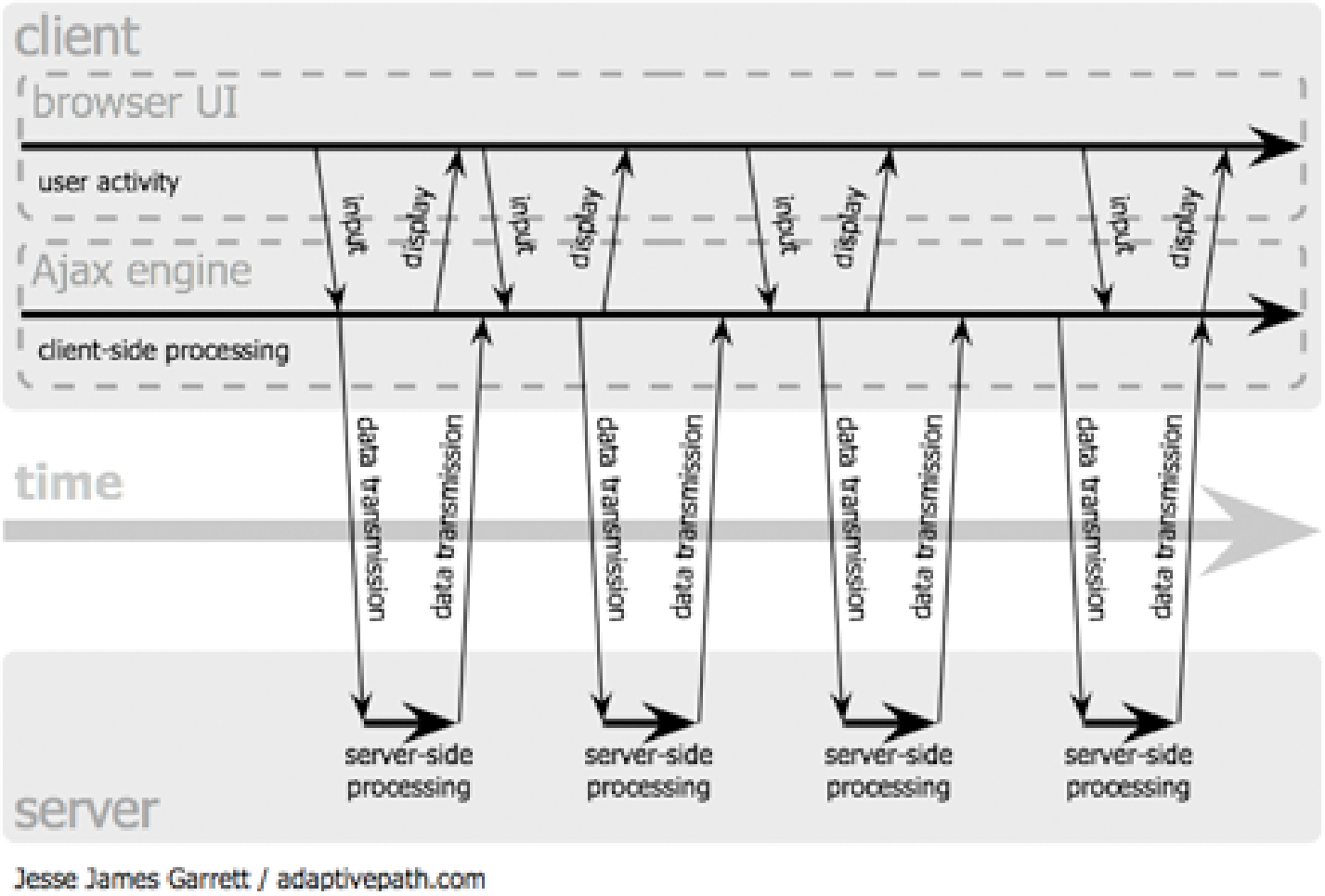
AJAX (ang. Asynchronous JavaScript and XML, Asynchroniczny JavaScript i XML) – technologia tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się bez przeładowywania całego dokumentu, w sposób asynchroniczny. Ma to umożliwiać bardziej dynamiczną interakcję z użytkownikiem niż w tradycyjnym modelu, w którym każde żądanie nowych danych wiąże się z przesłaniem całej strony HTML. (wikipedia)

Synchroniczna komunikacja z serwerem



Rysunek: Schemat zapytań http.

Asynchroniczna komunikacja z serwerem



Rysunek: Schemat zapytań Ajax.

AJAX - tworzenie obiektu XMLHTTP

```
var xmlHttp;
try{
  xmlHttp=new XMLHttpRequest();
} catch(e) {
  try {
    xmlHttp=new ActiveXObject("Msxml2.
        XMLHTTP");
  } catch(e) {
    try {
      xmlHttp=new ActiveXObject("Microsoft
          .XMLHTTP");
    } catch(e) {
      alert("NO AJAX!");
    }
  }
}
```

AJAX - wysyłanie/odbieranie

Wysyłanie/odbieranie

Po utworzeniu obiektu XMLHTTP określamy funkcję, która ma reagować na zmiany stanu tego obiektu. Ta funkcja odbierze wiadomość od serwera.

```
xmlHttp.onreadystatechange  = function(){
  if(xmlHttp.readyState  == 4){
    if(xmlHttp.status == 200)
      alert("Received:"  + xmlHttp.
          responseText);
    else
      alert("Error: " + xmlHttp.status);
  }
}
xmlHttp.open(GET, "data.txt",  true);
xmlHttp.send(null);
```

AJAX - wysyłanie/odbieranie

Funkcje send/open

Open przyjmuje następujące agrumenty:

- metoda: GET/POST
- URL dokumentu który ma być wywołany
- true jeśli zapytanie ma być asynchroniczne

Argumenty send:

- dane POST (jeśli wybraną metodą jest POST)

AJAX - nośnik danych

JSON vs XML

Treść komunikatu xmlHttp.responseText może być dowolnym ciągiem znaków. W użyciu są dwa standardy: XML, JSON. Przykład JSON:

```
{
  "isbn": "99-99999-99-9",
  "title": "AJAX",
  "authors": [
    {"first_name": "John", "last_name": "
       Smith"},
    {"first_name": "James", "last_name": "
       White"}
  ]
}
```

AJAX - nośnik danych c.d.

Zalety XML

1. powszechnie znany format
2. wiele narzędzi po stronie serwera
3. możliwość przetworzenia za pomocą XSLT
4. “silne wsparcie ze strony biznesu”?

Zalety JSON

1. łatwy w parsowaniu (proste reguły poprawności) za pomocą JS
2. coraz częściej natywne parsery w przeglądarkach => szybie parsowanie
3. czytelny dla człowieka i przeglądarki

AJAX c.d.

Wady i ograniczenia

- ograniczenie dostępu dla osób korzystających ze specjalnych klientów
- utrudnione automatyczne pobieranie stron
- utudnione indeksowanie stron
- w przód / w tył - nie działa / działa nieintuicyjnie
- JS sam w sobie niezbyt odpowiedni dla dużych aplikacji
- wrażliwość na opóźnienia w komunikacji sieciowej
- sposunkowo trudna implementacja, debugowanie
- nie można zrobić zakładki do jakiejś dynamicznie wygenerowanej “strony”
- wypada powiadamiać użytkownika o postępie operacji => dodatkowa praca programistyczna

AJAX c.d.

Zalety

- rozszerzenie możliwości interfejsu użytkownika, większa responsywność
- zmniejszenie ruchu klient - serwer
- odciążenie serwera
- zwiększenie odczuwalnej szybkości działania aplikacji
- istnieje dużo bibliotek usprawniających pracę z AJAX

Prototype

Wstęp

- Strona domowa: http://www.prototypejs.org/
- Całkiem przyjemna ściągawka: http://www.sergiopereira.com/articles/prototype.js.html
- Zawiera wiele ułatwień, skrótów, warto jednak rozejrzeć się za czymś lepszym.

Prototype - c.d.

Definiowanie klasy - stara metoda

Bardziej konwencjonalna metoda. Najpierw “tworzymy” klasę, potem ustalamy jej prototyp.

```
NowaKlasa = Class.create();
NowaKlasa.prototype = {
  atrybut: 8,
  metoda: function(){
    alert(this.atrybut)
  }
}
```

Prototype - c.d.

Dziedziczenie - stara metoda

Dziedziczenie odbywa się poprzez rozszerzenie prototypu o atrybuty i metody prototypu nadklasy.

```
PochodnaKlasa = Class.create();
Object.extend(
  PochodnaKlasa.prototype,
  NowaKlasa.prototype)
Object.extend(
  PochodnaKlasa.prototype, {
    metoda: function(){
      alert("Pochodna: " + this.atrybut)
    }
})
```

Prototype - c.d.

Definiowanie klasy - nowa metoda

Od razu podajemy prototyp klasy.

```
NowaKlasa = Class.create({
  atrybut: 8,
  metoda: function(){
    alert(this.atrybut)
  }
})
```

Prototype - c.d.

Dziedziczenie - nowa metoda

Podajemy klasę z której dziedziczymy i o co rozszerzamy prototyp. Dodatkowo do dyspozycji mamy argument $super - metodę z nadklasy.

```
PochodnaKlasa = Class.create(NowaKlasa, {
  metoda: function($super){
    $super()
    alert("Pochodna: " + this.atrybut)
  }
})
```

Prototype - c.d. - AJAX

```
request = new Ajax.Request(
  '/przepisy', {
    method:'get',
    parameters: {table: 'przepisy'},
    asynchronous: true,
    onSuccess: function(transport){
      alert("Success:" + transport.
          responseText);
    },
    onFailure: function(){alert('Error')}
 });
```

Prototype - c.d. - AJAX

Updater

Jeżeli okresowo zmieniamy zawartość fragmentu strony (bannery, ...).

```
<h2>Our fantastic products</h2>
<div id="products">
  (fetching product list ...)
</div>

new Ajax.PeriodicalUpdater('products', '/
    spam', {
  method: 'get',
  insertion: Insertion.Top,
  frequency: 1,
  decay: 2
});
```

Kato

Co to takiego?

- http://code.google.com/p/kato/
- synchroniczna komunikacja
- obiektowa implementacja
- używa JSON-a

Kato

Jak działa?

- w treści strony deklaracje kontrolek (formularze, tabele, ...)
- po stronie serwera deklaracje tabelek (które, komu i na jakich prawach udostępniamy)
- na akcje użytkownika ładowanie zawartości odpowiednich tabelek (ich fragmentów) / wysyłanie danych do zapisania

Źródła

Źródła

Przykłady kodów źródłowych oraz obrazki pochodzą z:

- http://www.adaptivepath.com/ideas/essays/archives/000385.php
- http://kurs.browsehappy.pl/JavaScript/Dziedziczenie
- http://www.javascriptkit.com/javatutors/oopjs.shtml
- http://pl.wikipedia.org/wiki/AJAX
- http://www.prototypejs.org/learn/introduction-to-ajax