



1

ПАТТЕРНЫ JAVASCRIPT

Простые решения стандартных задач



ЧТО ТАКОЕ ПАТТЕРН?

1. Выверенное решение
2. Легко используются повторно
3. Формируют структуру

В JS ПАТТЕРНЫ НЕ ПРИЖИЛИСЬ?

6

ПОЧЕМУ?

1. У меня простые задачи, зачем усложнять?
2. Мы ищем оптимальное решение для каждой задачи
3. ООП в JS и паттерны дают более ресурсоемкие решения

7

НУ ПОЧЕМУ ЖЕ?

4. За паттерны надо наказывать!
Для начинающих это «взрыв мозга», а
опытным – не нужно.
5. Мы позволяем нашим разработчикам
самим решать: использовать или не
использовать паттерны
6. Паттерны? Какие паттерны?

8



9

НУ И ЗАЧЕМ МНЕ ИСПОЛЬЗОВАТЬ ПАТТЕРНЫ?

10

1. Проверены множеством людей, что может уберечь от ошибок
2. Не требуют дополнительного времени на написание велосипеда
3. Формируют четкую и узнаваемую структуру кода

4. Обеспечивают предсказуемость кода
5. Упрощают поддержку и расширяемость кода
6. Упрощают подключение новых людей к проекту

А ПОЧЕМУ ВЫ,
ДА-ДА, ВОТ ВЫ,
НЕ ИСПОЛЬЗУЕТЕ

ПАТТЕРНЫ?

13



14

БАЗОВЫЕ ПОНЯТИЯ

15

ТИПЫ ДАННЫХ:

1. Number
2. String
3. Boolean
4. null
5. undefined
6. Object

16

ФУНКЦИИ:

1. Декларативные

```
function foo(){};
```

2. Выражения

```
var foo = function(){};
```

3. Созданные конструктором function

```
var foo = new Function();
```

17

ФУНКЦИИ МОГУТ БЫТЬ:

1. Самовызывающимися

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

2. Конструкторами

```
function Constructor(){}
var newObj = new Constructor;
```

КОНСТРУКТОР

Все объекты порождаются конструкторами

ПРОТОТИПЫ

1. Объект, служащий прообразом для других объектов.
2. Есть у всех объектов.
3. Прототипом может быть либо объект, либо null.

20

```
function Constructor() {
    this.firstMethod = function() {};
}
Constructor.prototype.secondMethod = function() {};
var newObj = new Constructor;
```

21

ССЫЛКИ ДЛЯ ИЗУЧЕНИЯ

1. ECMA-262 серия статей <http://goo.gl/5ymVJ>
2. ECMA-262 спецификация
<http://goo.gl/H2bBe>
3. ECMA-357 спецификация
<http://goo.gl/CmHX2>
4. Wikipedia <http://goo.gl/Oyljc>

22

ПАТТЕРНЫ (НАКОНЕЦ-ТО)

23

МОДУЛЬ

24

ЗАДАЧИ:

1. Инкапсуляция
2. Создание четкой структуры из подключаемых модулей
3. Прекратить засорять глобальный контекст

25

```
(function() {  
/*Наш код*/  
})();
```

26

```
(function(  
    global,  
    window,  
    document,  
    undefined){  
    function Module(){  
        /*Наш код*/  
        console.log(global); //window  
        console.log(window); //window  
        console.log(document); //document  
        console.log(undefined); //undefined  
    }  
})()
```

```
global.ourModule = new Module;  
})(this, window, window.document);
```

27

```
(function(global){  
    function Module(){  
        var privateVar = 0;  
        return {  
            getPrivateVar: function(){  
                return privateVar;  
            }  
        }  
    }  
    global.ourModule = new Module;  
})(this);
```

28

```
console.log(typeof window.ourModule.privateVar);  
//undefined  
console.log(typeof window.ourModule.getPrivateVar);
```

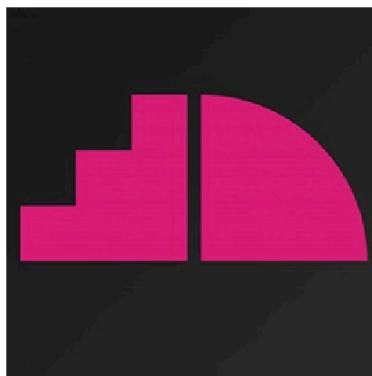
```
//function  
console.log(window.ourModule.getPrivateVar());  
//0
```

29

ССЫЛКИ ДЛЯ ИЗУЧЕНИЯ

1. JavaScript Module Pattern: In-Depth
<http://goo.gl/cL9P8>
2. Learning JavaScript Design Patterns
<http://goo.gl/cL9P8>
3. Show love to the module pattern
<http://goo.gl/0fLpb>
4. Again with the module pattern – reveal something to the world
<http://goo.gl/WkWnE>

30



31

ОДИНОЧКА

32

ЗАДАЧИ:

1. Ограничить количество экземпляров класса одним
2. При повторном вызове конструктора возвращать ссылку на созданный ранее экземпляр класса

33



НО В JS НЕТ
КЛАССОВ...

```
var a = {};  
var b = {};  
console.log(a==b); //false  
console.log(a===b); //false
```

ПРИМЕНИМ СИНГЛТОН
К МОДУЛЮ

```
(function(global){  
    function Singleton(){  
        /*Наш код*/  
    }  
    firstSingleton = new Singleton;  
    secondSingleton = new Singleton;  
})(this);
```

```
console.log(firstSingleton==secondSingleton); //false
```

ГДЕ СОХРАНИТЬ ССЫЛКУ НА ЭКЗЕМПЛЯР КЛАССА?

1. В статическом свойстве
2. В замыкании

```
(function(global){  
    function Singleton(){  
        var instance = this;  
        Singleton = function () {  
            return instance;  
        }  
    }  
    firstSingleton = new Singleton;  
    secondSingleton = new Singleton;  
})(this);
```

40

```
console.log(firstSingleton===secondSingleton); //true
```

41

```
(function(global){  
    function Singleton(){  
        var instance = this;  
        Singleton = function () {  
            return instance;  
        }  
    }  
    firstSingleton = new Singleton;  
    Singleton.prototype.newProperty = true;  
    secondSingleton = new Singleton;  
})(this);
```

42

```
console.log(firstSingleton==secondSingleton); //true  
console.log(firstSingleton.newProperty); //undefined  
console.log(secondSingleton.newProperty); //undefined
```

```
(function(global){
    function Singleton(){
        var instance = this,
            prototype = Singleton.prototype;
        Singleton = function () {
            return instance;
        }
        Singleton.prototype = prototype;
        Singleton.constructor = Singleton;
        instance.constructor = Singleton;
        return instance;
    }
    firstSingleton = new Singleton();
    Singleton.prototype.newProperty = true;
    secondSingleton = new Singleton();
})(this);
```

```
console.log(firstSingleton===secondSingleton); //true
console.log(firstSingleton.newProperty); //true
console.log(secondSingleton.newProperty); //true
```

ССЫЛКИ ДЛЯ ИЗУЧЕНИЯ

1. Wikipedia <http://goo.gl/YgEuL>
2. Learning JavaScript Design Patterns
<http://goo.gl/VbLvM>
3. JavaScript Design Patterns: Singleton
<http://goo.gl/3ICcd>



LIVEJOURNAL



47

НАБЛЮДАТЕЛЬ

48

ЗАДАЧИ:

При изменении состояния одного из объектов оповещать об этом «подписанные» на это событие объекты

```
(function(global){  
    function Observable() {}  
    function Observer() {}  
})(this);
```

```
function Observer(){}
Observer.prototype.tweet = function(data){};
```

```
function Observable(url){
  this.subscribers = {};
  this.dataUrl = url;
}
Observable.prototype.subscribe = function(event, observer, callback){};
Observable.prototype.unsubscribe = function(event, observer){};
Observable.prototype.publish = function(data, event){};
Observable.prototype.load = function(){};
```

```
Observable.prototype.loaded = function(data){};
```

!

```
Observable.prototype.load = function(){
  $.ajax({
    url: this.dataUrl,
    success: $.proxy(this.loaded,this)
  });
};
```

53

```
Observable.prototype.loaded = function(data) {  
    this.publish(data, 'dataLoaded')  
};
```

54

```
Observable.prototype.subscribe = function(event, callback){  
    if(this.subscribers.hasOwnProperty(event)){  
        var index = this.subscribers[event].length;  
        while(index--){  
            if(this.subscribers[event][index] == callback){  
                return false;  
            }  
        }  
        this.subscribers[event].push(callback);  
    }else{  
        this.subscribers[event]=[callback]  
    }  
    return true;  
};
```

55

```
Observable.prototype.unsubscribe = function(event, callback){  
    if(this.subscribers.hasOwnProperty(event)){  
        var index = this.subscribers[event].length;  
        while(index--){  
            if(this.subscribers[event][index] == callback){  
                this.subscribers[event].splice(index,1);  
                return true;  
            }  
        }  
    }  
    return false;  
};
```

56

```
Observable.prototype.publish = function(data, event){  
    if(this.subscribers.hasOwnProperty(event)){  
        var index = this.subscribers[event].length;  
        while(index--){  
            this.subscribers[event][index](data);  
        }  
        return true;  
    }  
    return false;  
};
```

57

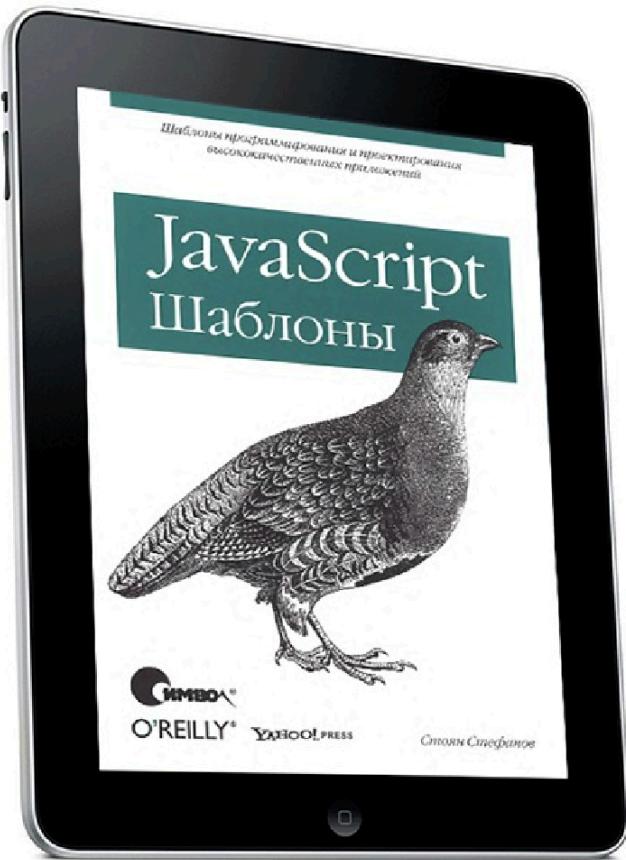
```
global.observer = new Observer;
global.observable = new Observable('http://test.com/data/');
global.observable.subscribe('dataLoaded', global.observer.tweet);
global.observable.load();
```

ССЫЛКИ ДЛЯ ИЗУЧЕНИЯ

1. Wikipedia <http://goo.gl/yWznH>
2. Learning JavaScript Design Patterns
<http://goo.gl/VbLvM>
3. JavaScript Design Patterns: Observer
<http://goo.gl/vt0Ob>
4. Understanding the Publish/Subscribe Pattern for Greater JavaScript Scalability
<http://goo.gl/oH2Mg>



ПАТТЕРНЫ JAVASCRIPT
РЕШАЮТ МНОЖЕСТВО
ЗАДАЧ. ЧТО ДАЛЬШЕ?



JavaScript Шаблоны

Стоян Стефанов

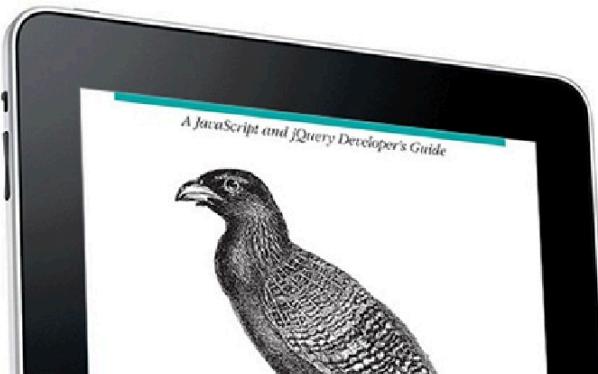
ISBN 978-5-93286-208-7 русский
бумажная

ISBN 978-0-596-80675-0 английский
бумажная

ISBN 978-0-596-80677-4 английский
электронная

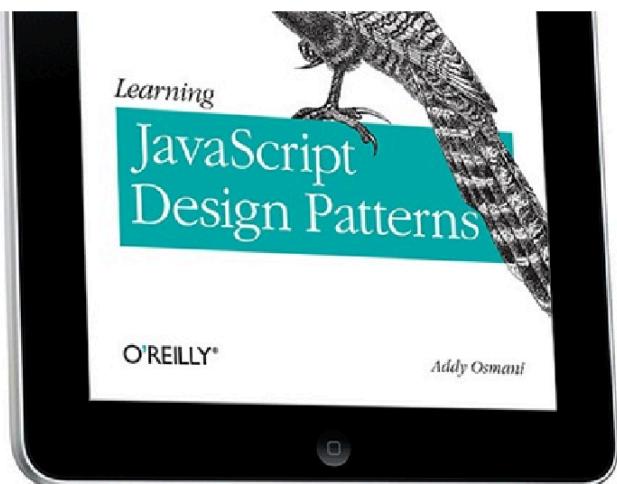
Market.ya.ru: <http://goo.gl/bXuSk>

Amazon.com: <http://goo.gl/FI5Eg>



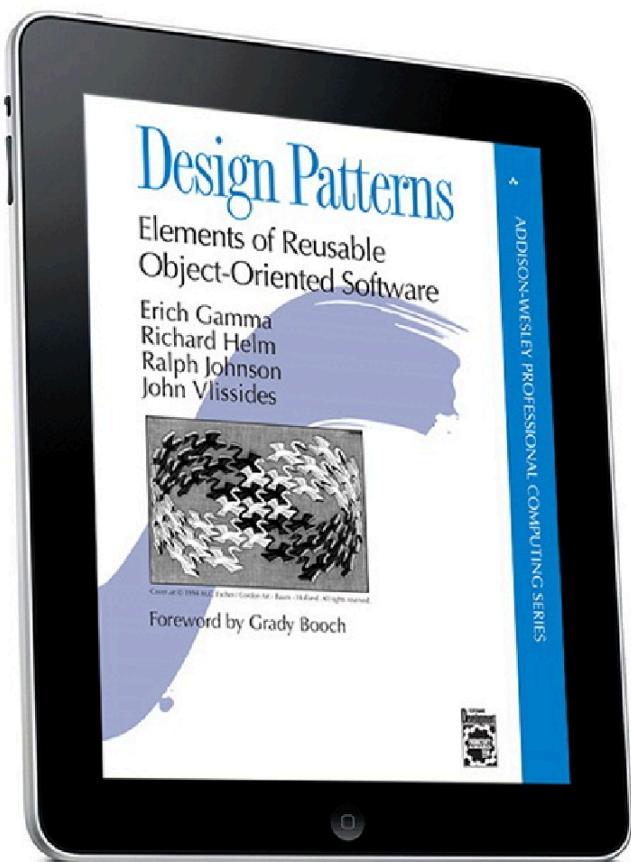
Learning JavaScript Design Patterns

Addy Osmani



ISBN 978-1-4493-3181-8 английский
бумажная
ISBN 978-1-4493-3486-4 английский
электронная
Litres.ru <http://goo.gl/4E1De>
Amazon.com: <http://goo.gl/gVH6X>
Читать OnLine: <http://goo.gl/wEKPU>

63



Design Patterns: Elements of Reusable Object-Oriented Software

Erich Gamma, Richard Helm, Ralph
Johnson, John Vlissides

ISBN 978-0-201-63361-0 английский
бумажная
ISBN 978-5-469-01136-1 русский
бумажная
Market.ya.ru: <http://goo.gl/G1du7>
Amazon.com: <http://goo.gl/PUKLB>
Читать OnLine: <http://goo.gl/lfRgk>

64





ИЛЛЮСТРАЦИИ
БЕССОВЕСТНО
ОДОЛЖЕНЫ У
MCBESS

ВЫ МОЖЕТЕ
НАЙТИ ЕЩЕ НА
MCBESS.COM

65



НЕМЕДЛЕННО
СПРОСИТЕ МЕНЯ
О ЧЕМ-НИБУДЬ

66



Антон Немцев
anton@websaints.net
@silentimp
skype: ravencry

Скачать презентацию
<http://goo.gl/iUxZX>
Смотреть online
<http://goo.gl/ixnK1>