JS Libraries

Exadel®

О чем эта лекция?

Обо всем понемногу:

- Работа с DOM
- Обработка событий
- Работа с AJAX
- Наборы утилит
- Наборы готовых компонент
- Прогрессивное улучшение: инструменты, polyfills
- Организация кода
- Single Page Applications и MV* фреймворки
- Шаблонизаторы
- Тестирование
- Полезные ссылки

Работа с DOM

jQuery, Zepto

Изменение/удаление/добавление атрибутов

```
.attr( attributeName )
.removeAttr( attributeName )
.hasClass( className )
.toggleClass( className )
```

и др.

Изменение/удаление/добавление элементов

```
.replaceWith( newContent )
.remove( [selector ] )
.append( content [, content ] )
.html()
.prepend( content [, content ] )
```

Изменение/удаление/добавление стилей

```
.css( propertyName [, value] )
.innerWidth( [value] )
.innerHeight()
.width( [value] )
.height( [value] )
```

Передвижение по дереву

```
.siblings( [selector ] )
.children( [selector ] )
.first()
.last()
.parent()
```

Обработка событий

jQuery, Zepto

Кроссбраузерный объект события

Объект event отличается от браузера к браузеру.
jQuery обеспечивает кроссбраузерность, добавляя/изменяя некоторые свойства этого объекта. Некоторые из них будут иметь значение undefined.

Обработка событий

jQuery, Zepto

События браузера:
`.resize( handler ), .scroll( handler )`

Загрузка документа:
`.ready( handler )`

Добавление обработчиков событий:
`var handler = function() { … };`
`$( “#el" ).on( "hover", handler );` – сработает только на существующих элементах

Делегирование:
`var foo = function() { … };`
`$( "body" ).on( "click", "p", foo );` – сработает даже на элементах, добавленных после привязки

Удаление обработчика:
`$( "body" ).off( "click", "p", foo );`

Обработка событий

jQuery, Zepto

Передача данных в обработчик:

```
$("p").each(function(i){
    $(this).on("click",{x:i},function(event){
      alert("The " + $(this).index() + " paragraph has data: " +
event.data.x);
    });
  });
```

События мыши:

`.click(), .dblclick() …`

События клавиш клавиатуры:

`.keydown(), .keypress() …`

События формы:

`.select(), .submit()`

Обработка событий

jQuery, Zepto

Вызов всех обработчиков события для указанных элементов:

```
$( "form:first" ).trigger( "submit" );

$( "body" ).trigger({
  type:"logged",
  user:"foo",
  pass:"bar"
});

$( "#foo").trigger( "custom", [ "Custom", "Event" ] );

$( "#foo" ).on( "custom", function( event, param1, param2 ) {
  alert( param1 + "\n" + param2 );
});
```

AJAX

jQuery, Zepto

Библиотеки, предоставляющие набор методов для работы с AJAX

Глобальные обработчики событий:

```
.ajaxComplete()
.ajaxError()
.ajaxSend()
.ajaxStart()
.ajaxStop()
.ajaxSuccess()
```

```
$( document ).ajaxComplete(function() {
  $( ".log" ).text( "Triggered ajaxComplete handler." );
});
```

Вспомогательные функции:

```
$( "form" ).serialize();
// input1=val1&select1=option1&input2=val3
```

и др.

AJAX

jQuery, Zepto

Низкоуровневый интерфейс:
`jQuery.ajax()`

```
$.ajax({ options });
```
Возможные параметры: `type, contentType, success, error, complete, url, data, timeout…`

Shorthand Methods:
```
jQuery.get()
jQuery.getJSON()
jQuery.getScript()
jQuery.post()
$( el ).load()
```

Deferred Object

Deferred объекты упрощают работы с отложенными вызовами обработчиков.

Они позволяют отделить логику, которая зависит от результатов выполнения действия от самого действия.

Deferred объекты

- хранят состояние выполнения задачи: «еще не выполнено», «выполнено», «ошибка»
- имеют методы для изменения этого состояния
- имеют методы для установки обработчиков, реагирующих на переход объекта из состояния «еще не выполнено» в состояние «выполнения»/«ошибки»

Deferred Object

- Если обработчик «выполнения»/«ошибки» добавляется к уже «выполненному»/«отменённому» объекту, то он будет вызван немедленно.

- Прикрепленные к Deferred объектам обработчики всегда вызываются в том порядке, в котором они были установлены.

Deferred Object

Пример использования:

```
function testDeferred(){
  var d = $.Deferred();
  setTimeout(function(){
    // some code here
    d.resolve();
  }, 3000);
  return d;
}

var t = testDeferred().done(function(){ alert("done!"); });

// add callback AFTER deferred object was resolved
setTimeout(function() {
  t.done(function() { alert("done as well!"); });
}, 5000);
```

Promises & AJAX

Deferred объект обладает важным методом .promise().

Этот метод возвращает объект с практически тем же самым интерфейсом, что и deferred, то есть позволяет добавлять обработчики, просматривать состояние оригинального объекта, но не даёт возможности изменить состояние оригинального объекта (вызвать resolve() или reject()).

$.ajax(), $.get(), $.post(), $.getScript(), $.getJSON() возвращают promises.

Promises & AJAX

Стандартный AJAX запрос:

```
$.ajax({
  url: "/ServerResource.txt",
  success: successFunction,
  error: errorFunction
});
```

Используя promise:

```
var promise = $.ajax({
  url: "/ServerResource.txt"
});

promise.done(successFunction1);
promise.done(successFunction2);
promise.fail(errorFunction);
promise.always(alwaysFunction[, alwaysFunctions]);
```

Promises & AJAX

Пример. Вызов функции после нескольких одновременных запросов AJAX.

```
function doAjax(){
 return $.get(‘foo.htm');
}

function doMoreAjax(){
 return $.get('bar.htm');
}

$.when( doAjax(), doMoreAjax() )
 .then(function(){
  console.log( 'I fire once BOTH ajax requests have completed!' );
  }, function(){
  console.log( 'I fire if one or more requests failed.' );
  });
```

Утилиты

Lo-Dash, Underscore

Lo-Dash Underscore — библиотеки, содержащие

методы для работы с массивами

```
_.intersection([1, 2, 3], [5, 2, 1, 4], [2, 1]);
// → [1, 2],
```

методы для работы с объектами

```
_.invert({ 'first': 'fred', 'second': 'barney' });
// → { 'fred': 'first', 'barney': 'second' },
```

методы для работы с коллекциями

```
var evens = _.filter([1, 2, 3, 4],
    function(num) {
        return num % 2 == 0;
    });
// → [2, 4],
```

Утилиты

jQuery

`jQuery.noop()` — пустая функция

```
$.each([ 52, 97 ], function( index, value ) {
  alert( index + ": " + value );
});
```

```
$.inArray( 5 + 5, [ "8", "9", "10", 10 + "" ] );
// -1
```

```
$.merge( [ 3, 2, 1 ], [ 4, 3, 2 ] )
// [ 3, 2, 1, 4, 3, 2 ]
```

и др.

UI компоненты

JQuery UI

Библиотека предоставляет собой набор виджетов

- Accordion — «Аккордеон»
- Autocomplete — Поле ввода с автодополнением
- Button — улучшенная кнопка, может также быть флажком (check box) или радио-кнопкой (radio button); все виды кнопки могут располагаться на панели инструментов (toolbar)
- Datepicker — виджет для выбора даты или диапазона дат
- Dialog — диалоговое окно, которое может иметь любое содержимое
- Progressbar — полоса прогресса
- Slider — слайдер
- Tabs — вкладки

UI компоненты

Bootstrap

Инструментарий:

- Сетки — Заранее заданные размеры колонок.
- Шаблоны — Фиксированный или резиновый шаблон документа.
- Типографика — Описания шрифтов, определение некоторых классов для шрифтов, таких как код, цитаты и т. п.
- Медиа — Представляет некоторое управление изображениями и Видео.
- Таблицы — Средства оформления таблиц, вплоть до добавления функциональности сортировки.
- Формы — Классы для оформления не только форм, но и некоторых событий происходящих с ними.
- Навигация — Классы оформления для Табов, Вкладок, Страничности, Меню и Тулбара.
- Алерты — Оформление диалоговых окон, Подсказок и Всплывающих окон.

Прогрессивное улучшение

Прогрессивное улучшение предполагает, что веб-интерфейсы должны создаваться поэтапно, циклически, от простого к сложному. На каждом из этапов должен получаться законченный веб-интерфейс, который будет лучше, красивее и удобнее предыдущего.

Можно выделить следующие этапы:

- «Старый-добрый-HTML» этап (смысл документа, логическая разметка)
- CSS этап (внешний вид)
- CSS3 этап (безупречный внешний вид)
- JavaScript этап (взаимодействие, интерактивность, удобство)

Прогрессивное улучшение

Browser sniffing

~~navigator.userAgent~~

Feature detection

Modernizr – JS библиотека, которая определяет, поддерживает ли конкретный браузер конкретную фичу.

Прогрессивное улучшение

Modernizr: CSS

```
<html class="js flexbox flexbox-legacy canvas canvastext no-webgl no-touch
geolocation postmessage websqldatabase no-indexeddb hashchange history
draganddrop websockets rgba hsla multiplebgs backgroundsize borderimage
borderradius boxshadow textshadow opacity cssanimations csscolumns cssgradients
cssreflections csstransforms csstransforms3d csstransitions fontface
generatedcontent video audio localstorage sessionstorage webworkers
applicationcache svg inlinesvg smil svgclippaths">
```

```
.multiplebgs selector {
    background-image: url('img.png'), url('background.png');
}
```

Прогрессивное улучшение

Modernizr: JS

```
Modernizr
▼ Object {touch: false, postmessage: true, history: true, multiplebgs: true, boxshadow: true…}
  ▶ _cssomPrefixes: Array[4]
  ▶ _domPrefixes: Array[4]
  ▶ _prefixes: Array[6]
    _version: "2.6.1"
  ▶ addTest: function (a,b){if(typeof a=="object")for(var d in a)A(a,d)&&e.addTest(d,a[d]);else{a=
    blob: true
    blobbuilder: false
    bloburls: true
    boxshadow: true
    cssanimations: true
    csscolumns: true
    cssgradients: true
    csstransforms: true
    csstransitions: true
    download: true
    fontface: true
    formdata: true
    history: true
    inlinesvg: true
```

```
if (Modernizr.geolocation) {
    // функции
}
```

Polyfills

Polyfill — JS замена стандартного API для старых браузеров

Пример: браузер не поддерживает веб сокеты. Будет создана глобальная переменная window.WebSocket с такими же свойствами и методами, как и в нативной имплементации.

```
Modernizr.load([
  // Presentational polyfills
  {
    // Logical list of things we would normally need
    test : Modernizr.fontface && Modernizr.canvas && Modernizr.cssgradients,
    yep : ‘presentational.js',
    // Modernizr.load loads css and javascript by default
    nope : ['presentational-polyfill.js', 'presentational.css']
  },
  // Functional polyfills
  {
    // This just has to be truthy
    test : Modernizr.websockets && window.JSON,
    // socket-io.js and json2.js
    nope : 'functional-polyfills.js',
    // You can also give arrays of resources to load.
    both : [ 'app.js', 'extra.js' ]
  }
]);
```

Организация кода

Module pattern

Модуль — логический блок кода. Идея модульной разработки приложения состоит в инкапсуляции кода. При таком подходе глобальная область видимости не засоряется, реализация скрыта, доступно только публичное API.

Пример:

```
var testModule = (function () {
  var counter = 0;
  return {
    incrementCounter: function () {
      return counter++;
    },
    resetCounter: function () {
      console.log( "counter value prior to reset: " + counter );
      counter = 0;
    }
  };
})();

testModule.incrementCounter(); // Increment our counter
testModule.resetCounter(); // Check the counter value and reset; Outputs: 1
```

Организация кода

AMD и RequireJS

AMD (Asynchronous Module Definition) API предоставляет механизм определения модулей таким образом, что модули и их зависимости могут быть загружены асинхронно.

RequireJS — загрузчик модулей и файлов, реализует AMD API

```
define("<module_name>",
        ["dependency1", "dependency2],
        function(dependency1, dependency2) {
       // some code here
        }
    );
```

Организация кода

Пример использования RequireJS (взят отсюда)

messages.js

```
define(function () {
    return {
        getHello: function () { return 'Hello World’; }
    };
});
```

main.js

```
define(function (require) {
    // Load any app-specific modules with a relative require call, like:
    var messages = require('./messages');
    // Load library/vendor modules using full IDs, like:
    var print = require('print');
    print(messages.getHello());
});
```

Single Page Applications

SPA-приложение — приложение, выполняющееся на стороне клиента.

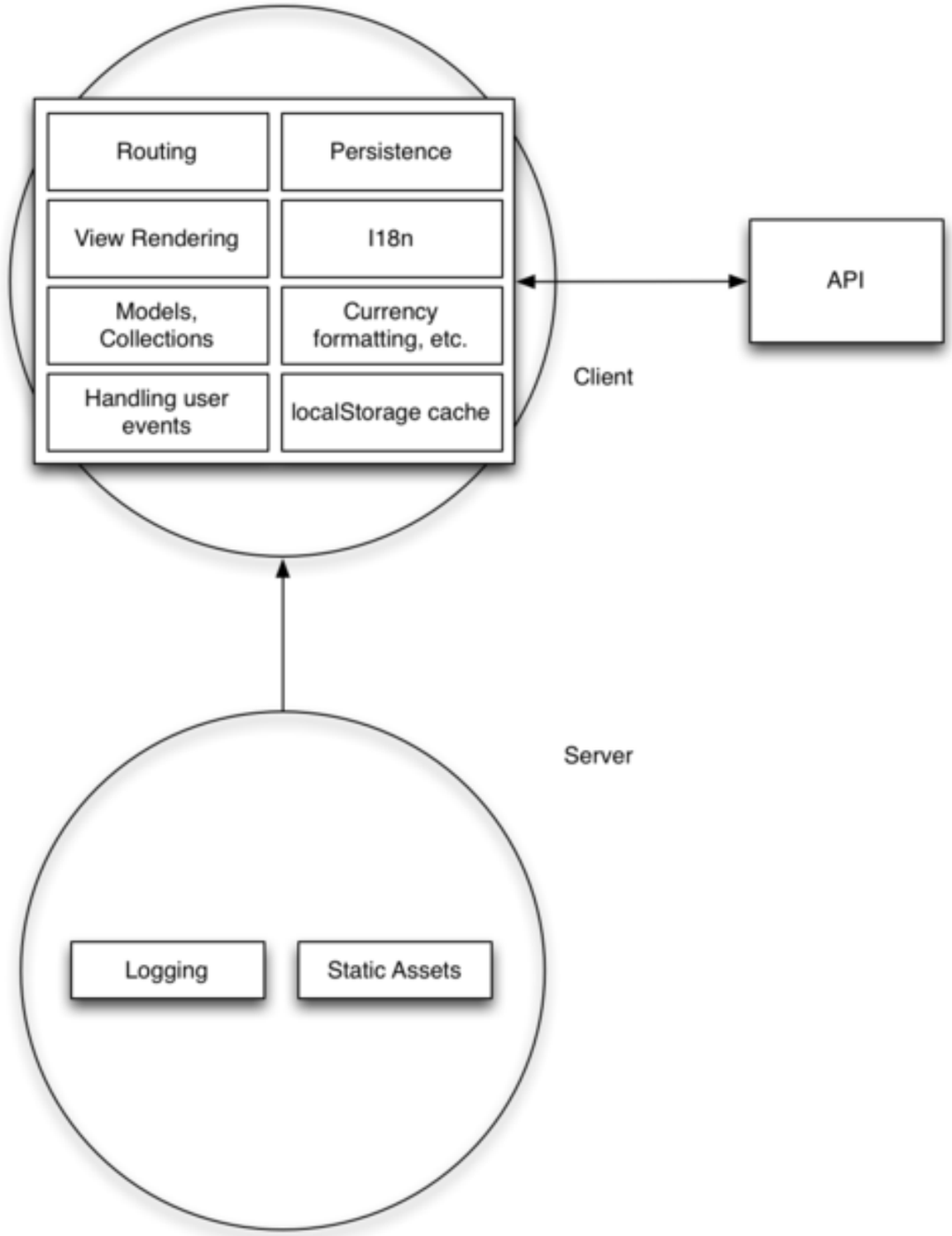
Основные характеристики SPA:
Страница разбита на отдельные HTML **фрагменты**; данные приходят в JSON формате; при запросах отрисовываются только отдельные фрагменты, а не страница целиком.

Архитектурное разделение кода на модели данных (model) представления (view) и логику (controller).

Использование **шаблонов** для построения UI. Данные для шаблонов предоставляет модель.

Поддержка **маршрутизации** и **навигации** между представлениями (в рамках одной страницы). Позволяет помещать страницы в закладки, передвигаться по истории.

Хранение данных в браузере в **Local Storage**



MV*

Классическое понятие MVC (серверная сторона)

Модель предоставляет данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Представление отвечает за отображение информации (визуализацию).

Контроллер обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

MVC на стороне клиента

JavaScript фреймворки по-совему интерпретируют понятие шаблона MVC. Компонент, который отличается от аналогичного на серверной части — контроллер. В качестве названия паттерна часто фигурирует аббревиатура **MV***.

MV* фреймворки:
http://backbonejs.org/
http://emberjs.com/
https://angularjs.org/
http://knockoutjs.com/

Шаблонизаторы

Handlebars

Шаблон:

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Компиляция:

```
var source = $(“#entry-template").html();
var template = Handlebars.compile(source);
```

Подключение:

```
<script id="entry-template" type="text/x-
handlebars-template">
  template content
</script>
```

В контексте:

```
var context = {title: "My New Post", body:
"Test!"}
var html = template(context);

<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    Test!
  </div>
</div>
```

Шаблонизаторы

Mustache

```
<html>
<body onload="loadUser">
<div id="target">Loading...</div>
<script id="template" type="x-tmpl-mustache">
Hello {{ name }}!
</script>
</body>
</html>

function loadUser() {
  var template = $('#template').html();
  Mustache.parse(template);   // optional, speeds up future uses
  var rendered = Mustache.render(template, {name: "Luke"});
  $('#target').html(rendered);
}
```

Шаблонизаторы

Underscore

```
<script type="text/html" id='table-data'>
    <% _.each(items,function(item,key,list){ %>
        <tr>
            <td><%= key %></td>
            <td><%= item.name %></td>
        </tr>
    <% }) %>
</script>

var items = [
    {name:"Nick"},
    {name:"Lee"},
    {name:"Jenny"},
    {name:"Julie"}
]

var tableTemplate = $("#table-data").html();
$(“table tbody").html(_.template(tableTemplate,{items:items}));
```

Модульное тестирование

Введение

Модульное тестирование, или **юнит-тестирование** (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Тестирование

TDD / BDD

1) написание теста до внесения желаемого изменения → тест падает с ошибками
2) написание кода → тест отрабатывает без ошибок
3) рефакторинг кода

– Уменьшение количества ошибок
– Улучшение поддерживаемости кода
– Улучшение дизайна кода

TDD = Test Driven Development
ориентирован на код

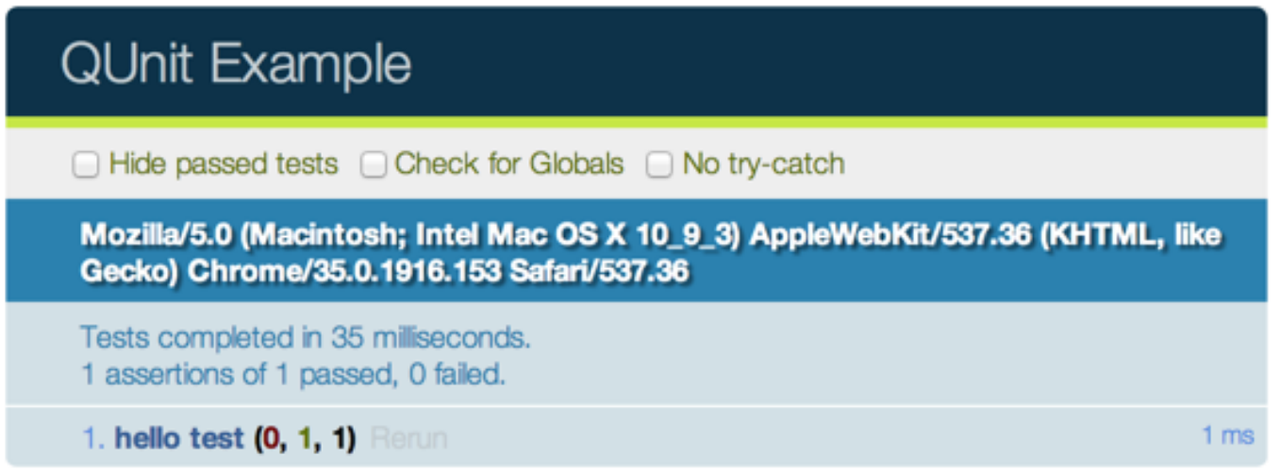
BDD = Behavior Driven Development
Ориентирован на поведение (думаем не функциями и возвращаемыми значениями, а поведением). Данный подход помогает:
– понять с чего начать
– понять что тестировать
– сколько тестов нужно написать за один подход
– как структурировать тесты

Наиболее популярные библиотеки: QUnit, Jasmine, Mocha

Тестирование

QUnit

```
QUnit.test( "hello test", function( assert ) {
  assert.ok( 1 == "1", "Passed!" );
});
```




```
QUnit.module( "group a" );
QUnit.test( "a basic test example", function( assert ) {
  assert.ok( true, "this test is fine" );
});
QUnit.test( "a basic test example 2", function( assert ) {
  assert.ok( true, "this test is fine" );
});
```

Тестирование

Jasmine

BDD фреймворк

определение набора тестов,
наборы могут быть вложенными

```
describe("A spec", function() {
  it("is just a function, so it can contain any code", function() {
    var foo = 0;
    foo += 1;

    expect(foo).toEqual(1);
  });

  it("can have more than one expectation", function() {
    var foo = 0;
    foo += 1;

    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

определение теста

```
A spec
  is just a function, so it can contain any code
  can have more than one expectation
```

Тестирование

Sinon

Крайне полезная в процессе тестирования библиотека.

Шпионы (Spies)
Spy — функция, которая, при каждом своем вызове, фиксирует аргументы, возвращаемое значение, значение `this`, ошибки (если таковые имеются).

Пример использования: проверка работы функции, имеющей callback

```
"test should call subscribers on publish": function () {
    var callback = sinon.spy();
    PubSub.subscribe("message", callback);
    PubSub.publishSync("message");
    assertTrue(callback.called);
}
```

Тестирование

Sinon

Шпионы (Spies)

Шпионы могут также оборачивать существующие функции.

```
{
    setUp: function () { sinon.spy(jQuery, “ajax"); },
    tearDown: function () { jQuery.ajax.restore();},
    "test should inspect jQuery.getJSON's usage of jQuery.ajax":
function () {
        jQuery.getJSON("/some/resource");
        assert(jQuery.ajax.calledOnce);
        assertEquals("/some/resource",
jQuery.ajax.getCall(0).args[0].url);
    }
}
```

Тестирование

Sinon

Заглушки (Stubs)
Stubs — функции (шпионы) с определенным поведением.

Пример использования:
1) Контроль над выполнением кода метода (например, бросить исключение для проверки механизма обработки ошибок)
2) Замена реализации метода (например, в случае, когда нежелательно выполнение AJAX запроса)

Тестирование

Sinon

Заглушки (Stubs)

```
"test should stub method differently on consecutive calls":
function () {
    var callback = sinon.stub();
    callback.onCall(0).returns(1);
    callback.onCall(1).returns(2);
    callback.returns(3);

    callback(); // Returns 1
    callback(); // Returns 2
    callback(); // All following calls return 3
}
```

Тестирование

Sinon

Mock-объекты
Mock-объекты — функции (шпионы) с определенным поведением и определенными ожиданиями.

Пример использования:

```
sinon.mock(jQuery).expects("ajax").atLeast(2).atMost(5);
jQuery.ajax.verify();
```

Тестирование

Fake XMLHttpRequest

Sinon предоставляет поддельную имплементацию XMLHttpRequest.

```
{
    setUp: function () {
        this.xhr = sinon.useFakeXMLHttpRequest();
        var requests = this.requests = [];
        this.xhr.onCreate = function (xhr) { requests.push(xhr); };
    },

    tearDown: function () { this.xhr.restore(); },

    "test should fetch comments from server" : function () {
        var callback = sinon.spy();
        myLib.getCommentsFor("/some/article", callback);
        assertEquals(1, this.requests.length);

        this.requests[0].respond(200, { "Content-Type": "application/json" },
                                 '[{ "id": 12, "comment": "Hey there" }]');
        assert(callback.calledWith([{ id: 12, comment: "Hey there" }]));
    }
}
```

Тестирование

Fake server

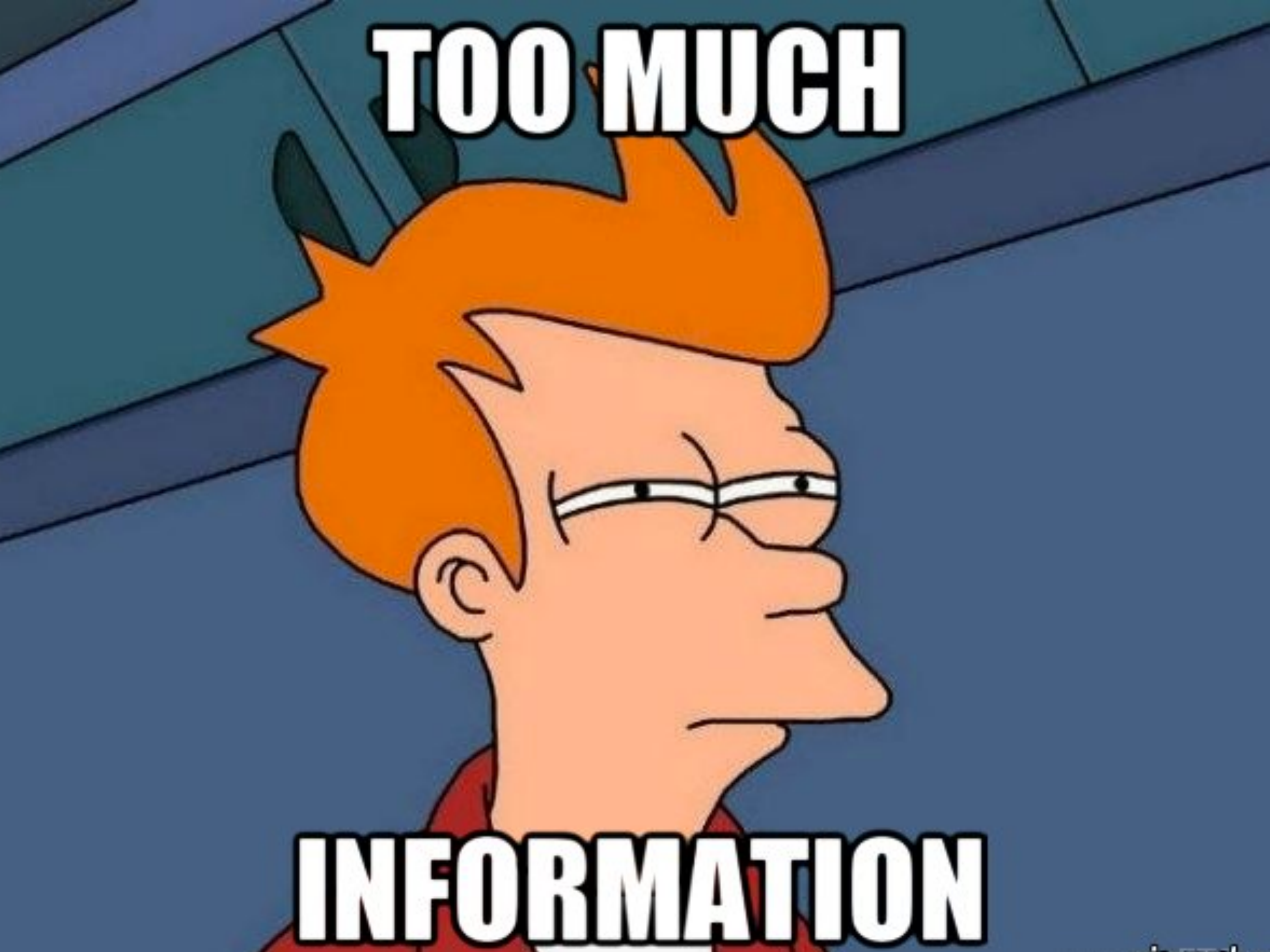
Высокоуровневое API для манипулирования XHR запросами.

```
{
    setUp: function () { this.server = sinon.fakeServer.create(); },

    tearDown: function () { this.server.restore(); },

    "test should fetch comments from server" : function () {
        this.server.respondWith("GET", "/some/article/comments.json",
            [200, { "Content-Type": "application/json" },
             '[{ "id": 12, "comment": "Hey there" }]']);

        var callback = sinon.spy();
        myLib.getCommentsFor("/some/article", callback);
        this.server.respond();

        sinon.assert.calledWith(callback, [{ id: 12, comment: "Hey" }]));
    }
}
```

Полезная информация

- Делать ли выбор в пользу Vanilla JS?
- Подборка легковесных библиотек для разных целей
- Коллекция HTML5 Cross Browser Polyfills
- Статистика использования JS библиотек
- React — библиотека для построения интерфейсов (V в MVC)
- Большая подборка библиотек
- MV* фреймворки
  - Addy Osmani «Learning JavaScript Design Patterns» http://addyosmani.com/resources/essentialjsdesignpatterns/book/#detailmvcmvp
  - Сравнение MV* фреймворков http://todomvc.com/ , а также обзор расширений (Marionette, Chaplin) к фреймворкам
  - Обзор MV* фреймворков: часть 1 и часть 2

TOO MUCH



INFORMATION

The End

Exadel®