Mocks, Stubs and Spies

Ease your testing pain with Sinon.js

Qafoo GmbH

June 4, 2013

What comes next?

Welcome

About Me

Jakob Westhoff

- More than 12 years of professional PHP experience
- More than 9 years of professional JavaScript experience
- Open source enthusiast
- Regular speaker at (inter)national conferences
- Consultant, Trainer and Author

Working with

Qafoo
passion for software quality

About Me

Jakob Westhoff

- More than 12 years of professional PHP experience
- More than 9 years of professional JavaScript experience
- Open source enthusiast
- Regular speaker at (inter)national conferences
- Consultant, Trainer and Author

Working with

Qafoo
passion for software quality

Helping people to create high-quality Applications

About Me

Jakob Westhoff

- More than 12 years of professional PHP experience
- More than 9 years of professional JavaScript experience
- Open source enthusiast
- Regular speaker at (inter)national conferences
- Consultant, Trainer and Author

Working with

Qafoo

passion for software quality

Helping people to create high-quality Applications

`http://qafoo.com`

Questions answered about Sinon.js

1. What is Sinon.JS?
2. What are Spys, Stubs and Mocks?
3. How do all of those things work in Sinon.JS?
4. How to control the timeflow during the tests?
5. How to fake XMLHttpRequests?

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

Sinon.JS about itself

http://sinonjs.org

Standalone test spies, stubs and mocks for JavaScript. No dependencies, works with any unit testing framework.

What comes next?

Mocks, Stubs and Spys

Mocks, Stubs and Spys in general

- Stubs
  - Simulation of behaviour from other units
  - Most stubs are simply returning fixed values

passion for software quality

Mocks, Stubs and Spys in general

- Stubs
  - Simulation of behaviour from other units
  - Most stubs are simply returning fixed values
- Spys
  - Augment certain methods/units with the abillity to track calls
  - The normal functionallity of the method is hereby not compromised

Mocks, Stubs and Spys in general

- Stubs
  - Simulation of behaviour from other units
  - Most stubs are simply returning fixed values
- Spys
  - Augment certain methods/units with the abillity to track calls
  - The normal functionallity of the method is hereby not compromised
- Mocks
  - A combination of Stubs and Spys
  - Override certain parts of a unit with stubs and automatically validate their calling structure against a predefined scheme

What comes next?

Spies

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Spys with Sinon.JS

http://sinonjs.org/docs/#spies

A test spy is a function that records arguments, return value, the value of this and exception thrown (if any) for all its calls. A test spy can be an anonymous function or it can wrap an existing function.

Anonymous spy

```
"testSubscriberCalledOnPublish": function () {

    var callback = sinon.spy();

    PubSub.subscribe("message", callback);
    PubSub.publishSync("message");

    assertTrue(callback.called);
}
```

Anonymous spy - `calledWith`

- Assertions based on given arguments is easy:

Anonymous spy - `calledWith`

- Assertions based on given arguments is easy:

```
"testPublishProvidesGivenPayload": function () {

    var payload = getSomeRandomPayload();
    var spy = sinon.spy();

    PubSub.subscribe("message", spy);
    PubSub.publishSync("message", payload);

    assert(spy.calledWith(payload));
}
```

Create partial Spies - `withArgs`

- Spies can be created in a partial manner
- Only certain argument combinations are spied on
- `.withArgs(arg1, arg2, ...)` initializes a partial Spy

Create partial Spies - `withArgs`

```
"testCalledWithTwoArgumentsOnce": function () {
    var spy = sinon.spy();

    spy.withArgs(42);
    spy.withArgs(1);

    spy(42);
    spy(1);

    assert(spy.withArgs(42).calledOnce);
    assert(spy.withArgs(1).calledOnce);
}
```

Augment existing functions and methods

- Existing functions can be augmented with spying functionallity
  - `var spy = sinon.spy(myFunc);`

Augment existing functions and methods

- Existing functions can be augmented with spying functionallity
  - `var spy = sinon.spy(myFunc);`
- Existing methods can be augmented as well
  - `var spy = sinon.spy(object, "method");`
  - The method inside the object will be replaced with the augmented one.

Augment existing functions and methods

- Existing functions can be augmented with spying functionallity
  - `var spy = sinon.spy(myFunc);`
- Existing methods can be augmented as well
  - `var spy = sinon.spy(object, "method");`
  - The method inside the object will be replaced with the augmented one.
  - A call to `restore()` on the spy will unwrap the augmented method again

Augment existing methods - Example

```
"testjQueryUsesAjaxFunction": function () {
```

Augment existing methods - Example

```
"testjQueryUsesAjaxFunction": function () {
    sinon.spy(jQuery, "ajax");
```

Augment existing methods - Example

```
"testjQueryUsesAjaxFunction": function () {
    sinon.spy(jQuery, "ajax");
    jQuery.getJSON("/some/resource");
```

passion for software quality

Augment existing methods - Example

```
"testjQueryUsesAjaxFunction": function () {
    sinon.spy(jQuery, "ajax");
    jQuery.getJSON("/some/resource");
    assert(jQuery.ajax.calledOnce);
```

Augment existing methods - Example

```
"testjQueryUsesAjaxFunction": function () {
    sinon.spy(jQuery, "ajax");
    jQuery.getJSON("/some/resource");
    assert(jQuery.ajax.calledOnce);
    jQuery.ajax.restore(); // Unwraps the spy
}
```

Spys - Area of application

- Use Spys, whenever…

Spys - Area of application

- Use Spys, whenever…
  - you need to check for the invocation of a callback

Spys - Area of application

- Use Spys, whenever…
  - you need to check for the invocation of a callback
  - you want to validate callbacks are executed with certain arguments

Spys - Area of application

- Use Spys, whenever…
  - you need to check for the invocation of a callback
  - you want to validate callbacks are executed with certain arguments
  - you want to validate internal functions provide the correct return value

Spys - Area of application

- Use Spys, whenever. . .
  - you need to check for the invocation of a callback
  - you want to validate callbacks are executed with certain arguments
  - you want to validate internal functions provide the correct return value
  - you want to validate a certain simple calling behaviour

Spys - Area of application

- Use Spys, whenever...
  - you need to check for the invocation of a callback
  - you want to validate callbacks are executed with certain arguments
  - you want to validate internal functions provide the correct return value
  - you want to validate a certain simple calling behaviour
    - You will most likely want to use a Mock for this

What comes next?

Stubs

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Stubs with Sinon.JS

http://sinonjs.org/docs/#stubs

Test stubs are functions (spies) with pre-programmed behavior. They support the full test spy API in addition to methods which can be used to alter the stub's behavior. As spies, stubs can be either anonymous, or wrap existing functions.

Create a Stub

- Stubs are created the same way as Spies

Create a Stub

- Stubs are created the same way as Spies
- Create an anonymous Stub
  - `var stub = sinon.stub();`

Create a Stub

- Stubs are created the same way as Spies
- Create an anonymous Stub
  - `var stub = sinon.stub();`
- Replace an objects method with a Stub
  - `var stub = sinon.stub(object, "method");`

Create a Stub

- Stubs are created the same way as Spies
- Create an anonymous Stub
  - `var stub = sinon.stub();`
- Replace an objects method with a Stub
  - `var stub = sinon.stub(object, "method");`
- Replace all methods of one object with stubs
  - `var stub = sinon.stub(obj);`

Stubs are Spies

- Stubs implement the full feature set of Spies
  - `.called`, `.calledOnce`, `.calledTwice`, `.calledThrice`
  - `.calledBefore(anotherSpy)`, `.calledAfter(anotherSpy)`
  - `.calledOn(obj)`
  - `.calledWith(arg1, arg2, ...)`
  - `.threw()`, `.threw("TypeError")`, `.threw(e)`
  - `...`

Stubs are Spies

- Stubs implement the full feature set of Spies . . .
  - `.called`, `.calledOnce`, `.calledTwice`, `.calledThrice`
  - `.calledBefore(anotherSpy)`, `.calledAfter(anotherSpy)`
  - `.calledOn(obj)`
  - `.calledWith(arg1, arg2, ...)`
  - `.threw()`, `.threw("TypeError")`, `.threw(e)`
  - `...`
- . . . in conjunction with their own API

Anonymous Stubs - `returns`

- Anonymous stubs are frequently used

Anonymous Stubs - `returns`

- Anonymous stubs are frequently used
- Creating a stub with a fixed return value is extremely simple

Anonymous Stubs - `returns`

- Anonymous stubs are frequently used
- Creating a stub with a fixed return value is extremely simple

```
"testUselessStubDemo": function () {
    var callback = sinon.stub();

    callback.returns(42)

    assertEquals(
        callback(),
        42
    );
}
```

Qafoo
passion for software quality

Anonymous Stubs - `throws`

- … or throwing an exception

Anonymous Stubs - `throws`

- … or throwing an exception

```
"testUselessStubThrowsDemo": function () {
    var callback = sinon.stub();

    callback.throws("Some␣Error")

    try{
        callback();
    } catch( e ) {
        // Expected
    }

    assert(callback.threw('Some␣Error'));
}
```

passion for software quality

Partial Stubs - `withArgs`

- As with Spies, partial Stubs can be created as well

Partial Stubs - `withArgs`

- As with Spies, partial Stubs can be created as well
- Partials can react differently based on their arguments

Partial Stubs - `withArgs`

- As with Spies, partial Stubs can be created as well
- Partials can react differently based on their arguments
- As with Spys the `withArgs` method is used to define a partial
  - `.withArgs(arg1, arg2, ...)`

Partial Stubs - `withArgs`

```
"testPartialStubBehaviour": function () {
    var callback = sinon.stub();
```

Partial Stubs - `withArgs`

```
"testPartialStubBehaviour": function () {
    var callback = sinon.stub();

    callback.withArgs(42).returns(1);
    callback.withArgs(1).throws("TypeError");
```

Partial Stubs - `withArgs`

```
"testPartialStubBehaviour": function () {
    var callback = sinon.stub();

    callback.withArgs(42).returns(1);
    callback.withArgs(1).throws("TypeError");

    callback(42); // Returns 1
```

Partial Stubs - `withArgs`

```
"testPartialStubBehaviour": function () {
    var callback = sinon.stub();

    callback.withArgs(42).returns(1);
    callback.withArgs(1).throws("TypeError");

    callback(42); // Returns 1

    callback(1); // Throws TypeError
```

passion for software quality

Partial Stubs - `withArgs`

```
"testPartialStubBehaviour": function () {
    var callback = sinon.stub();

    callback.withArgs(42).returns(1);
    callback.withArgs(1).throws("TypeError");

    callback(42); // Returns 1

    callback(1); // Throws TypeError

    callback(); // No return value, no exception
}
```

passion for software quality

Creating a stubbed instance

- In object-oriented JavaScript applications (prototyping) you often need to create a stub instance of a certain *”class”*

Creating a stubbed instance

- In object-oriented JavaScript applications (prototyping) you often need to create a stub instance of a certain *”class”*
- That’s what `createStubInstance` is for

Creating a stubbed instance

- In object-oriented JavaScript applications (prototyping) you often need to create a stub instance of a certain *"class"*
- That's what `createStubInstance` is for

```
var Game = function() {/* ... */}
var Game.prototype.newRound = function() {/* ... */};

var stubbedGame = sinon.createStubInstance(
    Game
);

stubbedGame.newRound.retuns(true);
```

Stubs - Area of application

- Use Stubs, whenever. . .

Stubs - Area of application

- Use Stubs, whenever. . .
  - enforcement of control flow is needed
    - Throwing an exception to test error behaviour
    - Return a specific value to test a certain `if`/`else` branch

Stubs - Area of application

- Use Stubs, whenever. . .
  - enforcement of control flow is needed
    - Throwing an exception to test error behaviour
    - Return a specific value to test a certain `if`/`else` branch
  - a certain methods behaviour should be suppressed
    - Suppress the invocation of other modules/units
    - Suppress asynchronous requests (`XMLHttpRequest`)

Qafoo
passion for software quality

What comes next?

Mocks

Mocks with Sinon.JS

http://sinonjs.org/docs/#mocks

Mocks (and mock expectations) are fake methods (like spies) with pre-programmed behavior (like stubs) as well as pre-programmed expectations. A mock will fail your test if it is not used as expected.

passion for software quality

Mocks with Sinon.JS

http://sinonjs.org/docs/#mocks

Mocks (and mock expectations) are fake methods (like spies) with pre-programmed behavior (like stubs) as well as pre-programmed expectations. A mock will fail your test if it is not used as expected.

Mocks with Sinon.JS

http://sinonjs.org/docs/#mocks

Mocks (and mock expectations) are fake methods (like spies) with pre-programmed behavior (like stubs) as well as pre-programmed expectations. A mock will fail your test if it is not used as expected.

Mocks with Sinon.JS

http://sinonjs.org/docs/#mocks

Mocks (and mock expectations) are fake methods (like spies) with pre-programmed behavior (like stubs) as well as pre-programmed expectations. A mock will fail your test if it is not used as expected.

Mocks with Sinon.JS

http://sinonjs.org/docs/#mocks

Mocks (and mock expectations) are fake methods (like spies) with pre-programmed behavior (like stubs) as well as pre-programmed expectations. A mock will fail your test if it is not used as expected.

Mocks specialities

- Mocks can only augment objects
    - in contrast to single functions

Mocks specialities

- Mocks can only augment objects
  - in contrast to single functions
- Mock expectations have to be stated before, not after executing the relevant methods

Mocks specialities

- Mocks can only augment objects
  - in contrast to single functions
- Mock expectations have to be stated before, not after executing the relevant methods
- Mocks implement the Stub as well as the Spy API

Mocking an objects method

```
"testMockAnObject": function() {
    var obj = {
        someMethod: function(arg) {...}
    };
```

Mocking an objects method

```
"testMockAnObject": function() {
    var obj = {
        someMethod: function(arg) {...}
    };

    var mock = sinon.mock(obj);
```

Mocking an objects method

```
"testMockAnObject": function() {
    var obj = {
        someMethod: function(arg) {...}
    };

    var mock = sinon.mock(obj);

    mock
        .expects("someMethod")
        .atLeast(2)
        .withArgs(42);
```

Mocking an objects method

```
"testMockAnObject": function() {
    var obj = {
        someMethod: function(arg) {...}
    };

    var mock = sinon.mock(obj);

    mock
        .expects("someMethod")
        .atLeast(2)
        .withArgs(42);

    obj.someMethod(42);
    obj.someMethod(42);
```

Qafoo
passion for software quality

Mocking an objects method

```
"testMockAnObject": function() {
    var obj = {
        someMethod: function(arg) {...}
    };

    var mock = sinon.mock(obj);

    mock
        .expects("someMethod")
        .atLeast(2)
        .withArgs(42);

    obj.someMethod(42);
    obj.someMethod(42);

    mock.verify();
}
```

Qafoo
passion for software quality

Stacking expectations

```
"testStackExpectations": function() {
    ...
    var mock = sinon.mock(obj);
```

Stacking expectations

```
"testStackExpectations": function() {
    ...
    var mock = sinon.mock(obj);

    mock
        .expects("someMethod")
        .once()
        .withArgs(42);
```

Stacking expectations

```
"testStackExpectations": function() {
        ...
        var mock = sinon.mock(obj);

        mock
                .expects("someMethod")
                .once()
                .withArgs(42);

        mock
                .expects("someMethod")
                .atLeast(2)
                .atMost(4)
                .withArgs(23);
```

Qafoo
passion for software quality

Stacking expectations

```
"testStackExpectations": function() {
        ...
        var mock = sinon.mock(obj);

        mock
                .expects("someMethod")
                .once()
                .withArgs(42);

        mock
                .expects("someMethod")
                .atLeast(2)
                .atMost(4)
                .withArgs(23);

        ...
        mock.verify();
}
```

passion for software quality

Mocks - Area of application

- Use Mocks, whenever…
  - you want to validate the calling behaviour of the unit under test

Mocks - Area of application

- Use Mocks, whenever. . .
  - you want to validate the calling behaviour of the unit under test
  - you want to test a unit in isolation, but still be sure other units are called correclty.

Mocks - Area of application

- Use Mocks, whenever. . .
  - you want to validate the calling behaviour of the unit under test
  - you want to test a unit in isolation, but still be sure other units are called correclty.
  - you want to state expectations upfront instead of asserting afterwards.

What comes next?

The Flow of Time

Controling the flow of time

The Problem:

- Timers (`setTimeout`, `setInterval`) are often used in JavaScript for various applications

Controling the flow of time

The Problem:

- Timers (`setTimeout`, `setInterval`) are often used in JavaScript for various applications
- Testing units using them implies waiting for those timers to finish

Controling the flow of time

The Problem:

- Timers (`setTimeout`, `setInterval`) are often used in JavaScript for various applications
- Testing units using them implies waiting for those timers to finish
- Unit tests should run as fast as possible, to be easily executable during development cycles

Controling the flow of time

The Problem:

- Timers (`setTimeout`, `setInterval`) are often used in JavaScript for various applications
- Testing units using them implies waiting for those timers to finish
- Unit tests should run as fast as possible, to be easily executable during development cycles
- Testing code using the `Date` object may be tricky as it is producing uncontrolable results

Controling the flow of time

The Problem:

- Timers (`setTimeout`, `setInterval`) are often used in JavaScript for various applications
- Testing units using them implies waiting for those timers to finish
- Unit tests should run as fast as possible, to be easily executable during development cycles
- Testing code using the `Date` object may be tricky as it is producing uncontrolable results

All those tests are asynchronous, which makes them complex

Controling the flow of time

The Solution:

Controling the flow of time

The Solution:

- Taking control over the flow of time

Controling the flow of time

The Solution:

- Taking control over the flow of time

→ Sinon.JS Fake Timers

Fake timers

- Sinon.JS provides API to override all global time related functions with sophisticated stubs
- Flow of time can be controlled inside your tests at will

Fake timers usage

- Call `useFakeTimers()` to initialize
- Invoke `tick(ms)` to advance time an arbitrary amount in an instant
- Call `restore()` to return to usual time flow again

Test animation with fake timers

```
"testAnimateOver5000ms" : function() {
```

Test animation with fake timers

```
"testAnimateOver5000ms" : function() {
    var clock = sinon.useFakeTimers();
```

Test animation with fake timers

```
"testAnimateOver5000ms" : function(){

    var clock = sinon.useFakeTimers();

    var el = jQuery("#someElement");
    el.animate(
        {width: "200px"},
        5000
    );
```

Test animation with fake timers

```
"testAnimateOver5000ms" : function() {

    var clock = sinon.useFakeTimers();

    var el = jQuery("#someElement");
    el.animate(
        {width: "200px"},
        5000
    );

    clock.tick(5010);
```

Test animation with fake timers

```
"testAnimateOver5000ms" : function() {

        var clock = sinon.useFakeTimers();

        var el = jQuery("#someElement");
        el.animate(
                {width: "200px"},
                5000
        );

        clock.tick(5010);

        assertEquals("200px", el.css("width"));
```

passion for software quality

Test animation with fake timers

```
"testAnimateOver5000ms" : function() {

    var clock = sinon.useFakeTimers();

    var el = jQuery("#someElement");
    el.animate(
        {width: "200px"},
        5000
    );

    clock.tick(5010);

    assertEquals("200px", el.css("width"));

    clock.restore();
}
```

passion for software quality

Controling `Date` with fake timers

- FakeTimers control the `Date` object as well
- The object is automatically mocked and follows a controllable flow of time as well

Controling `Date` with fake timers

```
"testControlTheDate": function() {
```

Controling `Date` with fake timers

```
"testControlTheDate": function() {
    var clock = sinon.useFakeTimers();
```

Controling `Date` with fake timers

```
"testControlTheDate": function() {

    var clock = sinon.useFakeTimers();

    var now = new Date();
    clock.tick(5 * 60 * 1000);
    var in5Minutes = new Date();
```

passion for software quality

Controling `Date` with fake timers

```
"testControlTheDate": function() {

    var clock = sinon.useFakeTimers();

    var now = new Date();
    clock.tick(5 * 60 * 1000);
    var in5Minutes = new Date();

    assertEquals(
        in5Minutes.getTime(),
        now.getTime() + (5 * 60 * 1000)
    );
```

Qafoo
passion for software quality

Controling `Date` with fake timers

```
"testControlTheDate": function() {

    var clock = sinon.useFakeTimers();

    var now = new Date();
    clock.tick(5 * 60 * 1000);
    var in5Minutes = new Date();

    assertEquals(
        in5Minutes.getTime(),
        now.getTime() + (5 * 60 * 1000)
    );

    clock.restore();
}
```

Qafoo
passion for software quality

Fake timers - Area of application

- Use Fake timers, whenever…

Fake timers - Area of application

- Use Fake timers, whenever…
  - you want to test anything which uses `setTimeout` or `setInterval`

Fake timers - Area of application

- Use Fake timers, whenever…
  - you want to test anything which uses `setTimeout` or `setInterval`
  - you want to test anything which uses the `Date` object to determine the current date/time

Fake timers - Area of application

- Use Fake timers, whenever…
  - you want to test anything which uses `setTimeout` or `setInterval`
  - you want to test anything which uses the `Date` object to determine the current date/time
- Fake timers exist as a standalone package: `sinon-timers.js`, `sinon-timers-ie.js`

What comes next?

Decoupling XmlHttpRequests

Faking XMLHttpRequests

The Problem:

- `XMLHttpRequests` are used commonly throughout modern JavaScript applications

Faking XMLHttpRequests

The Problem:

- `XMLHttpRequests` are used commonly throughout modern JavaScript applications
- Unit tests are supposed to be isolated

Faking XMLHttpRequests

The Problem:

- `XMLHttpRequests` are used commonly throughout modern JavaScript applications
- Unit tests are supposed to be isolated
- The last thing a unit test should do is to rely on an external resource

Faking XMLHttpRequests

The Problem:

- `XMLHttpRequests` are used commonly throughout modern JavaScript applications
- Unit tests are supposed to be isolated
- The last thing a unit test should do is to rely on an external resource

The Solution:

- Intercept `XMLHttpRequest` calls and return a stubbed response based on the request

Faking XMLHttpRequests with Sinon.JS

- Sinon.JS provides two different ways of intercepting `XMLHttpRequest` calls
- Low-Level: The `FakeXMLHttpRequest` interface
- High-Level: The Fake server

Faking XMLHttpRequests with Sinon.JS

- Sinon.JS provides two different ways of intercepting `XMLHttpRequest` calls
- Low-Level: The `FakeXMLHttpRequest` interface
- High-Level: The Fake server

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();
```

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();

    server.respondWith('{"some":"json"}');
```

Intercept an XMLHttpRequest invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();

    server.respondWith('{"some":"json"}');

    jQuery.getJSON( '/foo/bar', spy );
```

Qafoo
passion for software quality

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();

    server.respondWith('{"some":"json"}');

    jQuery.getJSON( '/foo/bar', spy );

    server.respond();
```

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();

    server.respondWith('{"some":"json"}');

    jQuery.getJSON( '/foo/bar', spy );

    server.respond();

    assert(
        spy.calledWith({ 'some': 'json' })
    );
```

passion for software quality

Intercept an `XMLHttpRequest` invocation

```javascript
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
    var spy = sinon.spy();

    server.respondWith('{"some":"json"}');

    jQuery.getJSON( '/foo/bar', spy );

    server.respond();

    assert(
        spy.calledWith({ 'some': 'json' })
    );

    server.restore();
}
```

passion for software quality

Intercept an `XMLHttpRequest` invocation

- Stubbing all `XmlHttpRequests` with the same response does not always fit the usecase
- What if a unit needs to be tested, which fires multiple requests?

Intercept an `XMLHttpRequest` invocation

- Stubbing all `XmlHttpRequests` with the same response does not always fit the usecase
- What if a unit needs to be tested, which fires multiple requests?
- Sinon.js allows for route based responses to be defined

Intercept an `XMLHttpRequest` invocation

- Stubbing all `XmlHttpRequests` with the same response does not always fit the usecase
- What if a unit needs to be tested, which fires multiple requests?
- Sinon.js allows for route based responses to be defined
- Even responses based on different HTTP verbs are possible

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();
```

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();

    server.respondWith(
        "GET", "/some/resource",
        '{"some":"json"}'
    );
```

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();

    server.respondWith(
        "GET", "/some/resource",
        '{"some":"json"}'
    );

    server.respondWith(
        "GET", "/another/resource",
        '{"another":"json"}'
    );
```

passion for software quality

Intercept an `XMLHttpRequest` invocation

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();

    server.respondWith(
        "GET", "/some/resource",
        '{"some":"json"}'
    );

    server.respondWith(
        "GET", "/another/resource",
        '{"another":"json"}'
    );

    ...
    jQuery.getJSON( '/some/resource', spy1 );
    jQuery.getJSON( '/another/resource', spy2 );
    ...
}
```

Highly sophisticated interception

- Talking to a REST service might be even harder to mock

Highly sophisticated interception

- Talking to a REST service might be even harder to mock
- The service might answer with special Status-Codes and/or Headers

Highly sophisticated interception

```
"testInterceptAnXMLHttpRequest": function() {
    var server = sinon.fakeServer.create();

    server.respondWith(
        "GET", "/some/resource",
        [
            201,
            {
                'Location': '/some/newly/created/resource',
            },
            JSON.stringify({
                uris: ["/some/newly/created/resource"]
            })
        ]
    );

    ...
}
```

passion for software quality

Fake XMLHttpRequest - Area of application

- Use Fake XMLHttpRequest, whenever. . .
  - you want to test anything contacting the outside world using `XMLHttpRequest`

Fake XMLHttpRequest - Area of application

- Use Fake XMLHttpRequest, whenever. . .
  - you want to test anything contacting the outside world using `XMLHttpRequest`
- Fake XMLHttpRequests exist as a standalone package: `sinon-server.js`, `sinon-ie.js`

What comes next?

Sinon.js Conclusion

What you have learned today about Sinon.js

1. What Sinon.JS is
2. What Mocks, Stubs and Spies are
3. When to use them
4. Why you want to control the flow of time
5. Why you want to intercept `XMLHttpRequest` calls
6. . . . and how to do it

Want to learn more?

- The Sinon.JS documentation is excellent
    - ...with lots of code examples
- `http://sinonjs.org/docs`

Thanks for listening

Questions, comments or annotations?

Slides: http://talks.qafoo.com

Contact: Jakob Westhoff <`jakob@qafoo.com`>
Follow Me: @jakobwesthoff
Hire us: `http://qafoo.com`