# Web app系统测试方案

## Step by Step

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **发布日期** | **修订内容** | **主要修订人** |
| 1.0 | 2012-5-20 | 第一次初稿 | 莉霞 |

Change log

V 1.1

1.增加一键配置环境脚本

### STEP 1 web app系统测试环境配置

#### 1.1 web app系统测试方案配置

**1.1.1 配置系统环境**

本系统测试方案的测试方式是将待测web app嵌入测试页面中，测试代码对测试页面进行操作并进行结果的验证。考虑到web app渲染方式以及运行环境各不相同，这里采用的方式是将测试框架置于web app相同服务器下进行。

登陆到web app的测试机上，将fet框架所有代码拷贝到测试机上，运行代码清单中的“start.sh”文件即可进行环境配置。需要传入2个参数：web app运行的apache服务器的DocumentRoot的路径，即服务器根目录的路径；测试代码的文件夹的名称。这2个参数用于配置apache的rewrite rule（很多时候web app会在apache的配置文件中添加rewrite rule，用于对在该服务器下的请求进行转发），从而使得测试页面能够被访问。

**1.1.2 配置web app测试框架**

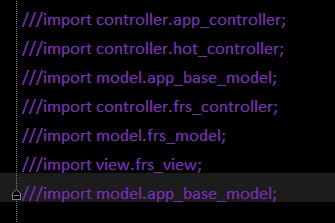
在“conf/config.php”中设置测试代码的目录，这里配置待测组件与测试代码分别在app和test-tieba目录下，如下图所示。$testdir中设置测试代码的路径。



**注：**config.php配置的路径是通过bin/list.php来读取的，因此也是相对于bin/list.php的路径。

**1.1.3 源文件加载**

考虑到测试文件运行时需要加载源文件，且经常还会需要额外加载一些lib，因此在$testdir下需要用户创建一个import.js文件，在文件中手动写下所有需要加载的源文件或lib库的路径（以“///import开头”，不需要以“.js”结尾）。例如下图中，引入了“controller.app\_controller.js”、“controller.hot\_controller.js”等。



**1.1.4自动化用例书写**

1）QUnit规范

一般来说，一个测试文件对应于一个JS组件，每一个测试文件中包含多个用例。自动化用例书写过程中应当遵循QUnit的规范，下面给出一些常用API接口以及功能描述，细则可以参考附录一的“QUnit API文档”。

**test：**QUnit中每一个用例用关键字“test”来定义，test的回调函数为用例主体。

**assertitons：**每一个test中可以包含若干断言，断言包括“ok”，“equal”，“same”，“notEqual”等，每一个用例至少包含一个断言，否则这个用例是没有意义的。

**expect：**expect方法可以设置预期的断言个数，如“expect(3)”表示预期这个用例中有3个断言，若断言数少于或多于3个，则用例失败。这个方法在异步操作中尤其有效。

**module：**多个test可以被一个module包含，这些test可以共用同一个moduleStart和moduleDone进行初始化和销毁工作，module在这里的作用类似于testsuite（用例套件）。

**testStart和testDone：**在用例执行前先执行testStart，当用例执行结束后再执行testDone，因此可以在testStart中添加用例初始化代码，在testDone中添加用例的环境清理代码。

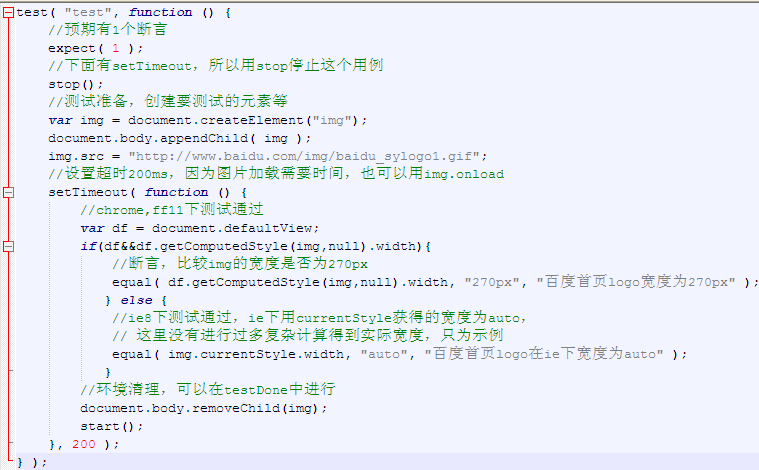
**moduleStart和moduleDone：**在module执行前先执行moduleStart，当module执行结束后再执行moduleDone，因此可以在moduleStart中添加module初始化代码，在moduleDone中添加module的环境清理代码。

**begin和done：**所有用例之前前执行begin，所有用例执行结束后执行done，因此可以在begin中添加所有用例的初始化代码，在done中添加所有用例的环境清理代码。

**stop和start：**同步操作。在用例中经常会遇到异步操作，如setTimeout，setInterval等，如果不做特殊操作，则用例顺序执行到用例结束，有可能setTimeout的回调函数还没有执行，则相应的功能没有检测到，这时就需要进行同步操作。

在存在异步操作的用例中，首先执行stop方法，它将停止当前正在运行的用例，等待异步操作执行，只有当start方法执行时，当前用例才会结束，下面的用例才会被执行。下面以一个简单的例子做一些说明，这个例子中集成了“test”，”expect”，“stop”，“start”，“equal”等API。

Example:



该用例在Chrome下的运行结果如下图所示。



4）总结

以上的示例是一个比较常见的例子，这里进行总结，在以后的用例编写过程中的常用要素是：

①test：声明一个测试用例，并定义用例主体内容

②同步操作：调用“stop”和“start”方法实现

2）系统测试方案的书写

正如在前面提到的系统测试方案的方法是将待测应用的页面嵌入到一个iframe中，测试代码对测试页面进行操作并进行结果的验证。下面给出一个测试代码的样例，用例中需要传入3个参数，第一个参数为用例的描述，第二个参数为待测页面的地址，可以用相对地址，也可以用绝对地址，最后一个为用例的执行函数，函数中将暴露2个参数：win和doc，分别为待测应用所在窗口的window对象和documeng对象，可以直接使用。通过这2个参数可以直接操作待测页面中的对象。

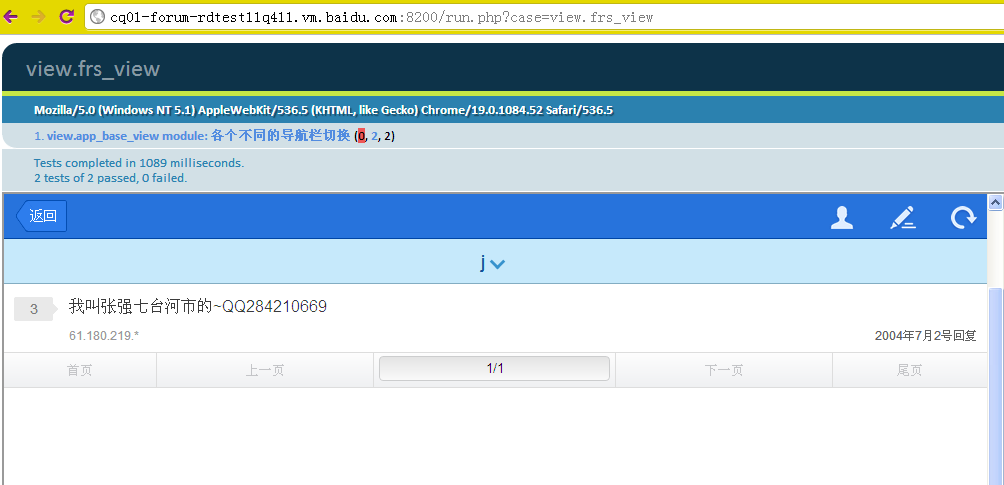
### STEP 2 执行用例

#### 2.1 执行自动化用例

自动化用例编写完毕后，就可以执行自动化用例了。在前面也提到过fet测试框架与web app放在同一个服务器下，执行start.sh脚本后，会将fet的测试框架拷贝到服务器根目录下，并添加rewrite rule，使得只需要访问web app的服务器地址+端口号+list.php/run.php就可以进行用例的执行。

**2.1.1 执行单个用例**

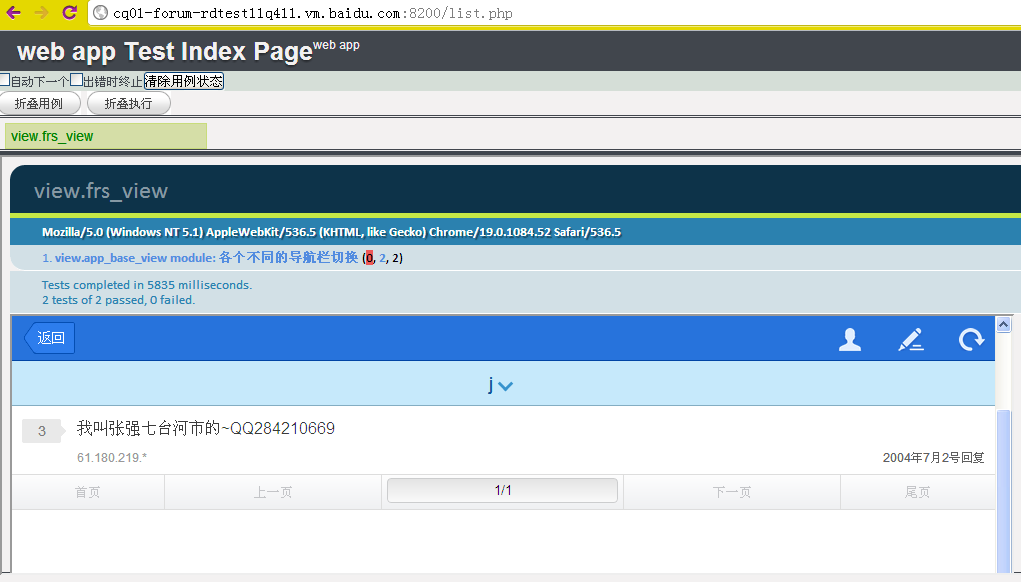
如果想执行用例frs\_view，打开页面web app的服务器地址和端口号”[http://cq01-forum-rdtest11q411.vm.baidu.com:8200”再直接加上run.php](http://cq01-forum-rdtest11q411.vm.baidu.com:8200\”再直接加上run.php)以及参数即“<http://cq01-forum-rdtest11q411.vm.baidu.com:8200/run.php?case=view.frs_view>”就可以直接运行这个用例，其中参数“case”的值表示frs\_view这个用例对应的路径“static/common/ui/pic/Dialog/Dialog.js”，case参数中不需要加js后缀。运行结果截图如下：



**2.1.2 浏览用例**

1）浏览所有用例以及运行用例

打开页面“<http://cq01-forum-rdtest11q411.vm.baidu.com:8200>/list.php”即可浏览当前所有的用例文件，如下图所示，list.php页面上显示了所有的待测用例：“frs\_view”，点击用例对应的色块将运行这个用例并展现用例运行结果。



**2.1.3 批量执行多个用例**

当用例特别多的时候，如果手动去点每一个用例的色块去运行相应的用例，那就会很低效。这里提供了一种批量运行用例的方案，利用batchrun参数可以批量运行所有的用例。例如打开“<http://cq01-forum-rdtest11q411.vm.baidu.com:8200/list.php&batchrun=true>”页面后，所有的用例将会顺序自动运行。此外，还可以在运行用例前勾选“自动下一个”复选框，效果等同于“batchrun=true”，需要在运行用例前勾选。

**2.1.4 其他参数**

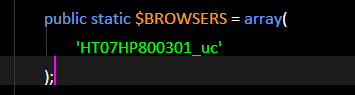
list.php还提供其他参数选项，用于满足不同的需求。

1. breakonerror，用例错误时是否停下，例如JS报错，setTimeout的回调函数没有正确执行，这些都可以看做是用例错误，当breakonerror为true时，用例执行失败时会停止，不执行下面的用例。反之，等待一段超时（默认30s），若超时结束时用例还没有结束，则设置用例fail，并继续执行下面的用例。默认breakonerror为false。
2. cov，是否计算覆盖率，cov为true时会计算代码覆盖率，当然这个需要先用jscoverage生成带有覆盖率信息的源文件。

**2.1.5 测试结果收集**

1. 配置浏览器信息

若要批量运行大量浏览器，并生成最后的测试结果，首先需要在“conf/config.php”中配置所有需要运行的浏览器的信息。例如下图中，在“$BROWSERS”变量中配置待测浏览器的名称，格式为“机器名\_浏览器名”，机器名指待运行用例的手机的机器名，如下图中“HT07HP800301”表示机器名，“uc”表示uc浏览器。



2. 生成覆盖率

如果想统计测试文件的覆盖率信息，那么首先需要利用tools目录下的jscoverage.exe首先对待测文件生成带有覆盖率信息的文件，具体用法如下：

**“jscoverage.exe --encoding=UTF-8 ../../../\_src ../../coverage”**

其中“jscoverage.exe”为生成覆盖率文件的可执行文件的名称，encoding用于设置编码。后面2个字符串分别表示源文件和生成覆盖率文件的路径。

当所有需要测试的浏览器都在变量$BROWSERS中设置完毕后，在想要获取结果的浏览器中输入“<http://cq01-forum-rdtest11q411.vm.baidu.com:8200/list.php?>cov=true--\_\_--browser=chrome--\_\_--batchrun=true”，这里采用“--\_\_--”作为分隔符的原因是因为在linux下“&”会被截断，导致后面的参数不能正确解析，导致不能正常在andriod的脚本环境下跑通。其中localhost为服务器名，这里是本地服务器，参数“cov”表示会计算覆盖率信息，参数“browser”用于设置当前的浏览器，如这里设置为chrome，则所有用例执行结束后将会在测试文件夹下生成report/chrome.xml文件。当config.php中配置的浏览器均运行完毕时，所有的”$browser$.xml”将会合并为一个文件——report.xml。该文件包含所有浏览器的运行结果以及覆盖率信息。

### 附录一：QUnit API文档

**setup：**

1）test( name, expected, test ) 添加一个测试。

**Arguments:**

|  |  |
| --- | --- |
| Name | [String](http://docs.jquery.com/Types#String) |
| test的名称，用于描述test的用途 | |
| expected(Optional) | [Number](http://docs.jquery.com/Types#Number) |
| 可选参数，预期断言的个数 | |
| test | [Function](http://docs.jquery.com/Types#Function) |
| 实际运行的代码，至少包含一个断言 | |

**Examples:**

添加一个用例，包含一个永远为true的断言

test("a test", function() {

ok(true, "always fine");

});

2）asyncTest( name, expected, test ) 添加一个异步测试，这个测试必须回调start()。

**Arguments:**

|  |  |
| --- | --- |
| Name | [String](http://docs.jquery.com/Types#String) |
| test的名称，用于描述test的用途 | |
| expected(Optional) | [Number](http://docs.jquery.com/Types#Number) |
| 可选参数，预期断言的个数 | |
| test | [Function](http://docs.jquery.com/Types#Function) |
| 实际运行的代码，至少包含一个断言 | |

3）expect( amount ) 特别指出在一个测试中预期的多个断言。

**Arguments:**

|  |  |
| --- | --- |
| **Amount** | [Integer](http://docs.jquery.com/Types#Integer) |
| 预期断言的个数 | |

1. module( name, lifecycle )：对测试用例进行分组，[lifecycle] 用于初始化测试和清理测试。

**Arguments:**

|  |  |
| --- | --- |
| **Name** | [String](http://docs.jquery.com/Types#String) |
| Module的名称 | |
| **lifecycle**(Optional) | [Options](http://docs.jquery.com/Types#Options) |
|  | |

1. init( ):初始化测试，不常用

**Assertions:**

|  |
| --- |
| [**ok(state,** message**)**](http://docs.jquery.com/QUnit/ok#statemessage) |
| 布尔型断言，如果第一个参数的值是true则断言通过。 |
| [**equal( actual, expected,** message **)**](http://docs.jquery.com/QUnit/equal#actualexpectedmessage) |
| 比较actual和expected是否相同，如果相同，则断言通过，message为断言的描述语句。 |
| [**same( actual, expected,** message **)**](http://docs.jquery.com/QUnit/equal#actualexpectedmessage) |
| 比较actual和expected是否相同，强比较 |
| [**notEqual( actual, expected,** message **)**](http://docs.jquery.com/QUnit/notEqual#actualexpectedmessage) |
| 比较actual和expected是否不同，如果不同，则断言通过，message为断言的描述语句。 |
| [**deepEqual( actual, expected,** message **)**](http://docs.jquery.com/QUnit/deepEqual#actualexpectedmessage) |
| 深递归比较断言，适用于原型类型、数组、对象 |
| [**notDeepEqual( actual, expected,** message **)**](http://docs.jquery.com/QUnit/notDeepEqual#actualexpectedmessage) |
| 深递归比较断言，适用于原型类型、数组、对象，actual和expected不相等则用例通过 |
| [**strictEqual( actual, expected,** message **)**](http://docs.jquery.com/QUnit/strictEqual#actualexpectedmessage) |
| 绝对相等断言，actual===expected时断言通过 |
| [**notStrictEqual( actual, expected,** message **)**](http://docs.jquery.com/QUnit/notStrictEqual#actualexpectedmessage) |
| 绝对不相等断言，actual!==expected时通过 |

**异步测试：**

start( ) 当测试停止的时候，启动测试。

**Arguments:**

|  |  |
| --- | --- |
| **decrement**(Optional) | [Integer](http://docs.jquery.com/Types#Integer) |
| 可选参数，减少给定的信号量（不常用） | |

stop() 停止测试，等待异步测试的开始。

**Arguments:**

|  |  |
| --- | --- |
| **increment**(Optional) | [Integer](http://docs.jquery.com/Types#Integer) |
| 可选参数，增加stop信号量 increment the stop-semaphore the given times. | |

**环境处理API:**

**QUnit.log**({ result, actual, expected, message}) is called whenever an assertion is completed. result is a boolean (true for passing,false for failing) and message is a string description provided by the assertion.

**QUnit.testStart**({ name }) : 用例执行前执行，“name”为用例的名称。

**QUnit.testDone**({ name, failed, passed, total }): 用例执行结束后执行，”failed”表示断言的个数，”total”是断言的总个数，“passed”为成功的断言个数。

**QUnit.moduleStart**({ name }) : module中的所有用例执行前执行，“name”为module的名称。

**QUnit.moduleDone**({ name, failed, passed, total}) : module运行结束后执行，“name”为module的名称，”failed”表示断言的个数，”total”是断言的总个数，“passed”为成功的断言个数。

**QUnit.begin**() : 所有用例运行前执行

**QUnit.done**({ failed, passed, total, runtime }):所有用例运行结束时被调用，”failed”表示断言的个数，”total”是断言的总个数，“passed”为成功的断言个数，“runtime”为所有用例运行的毫秒数。

### 附录二：TMock文档

域名空间：baidu

使用说明：mock框架中的断言基于qunit，因此在使用TMock的时候需要引入qunit。创建mock对象是挂载在tangram base上的，因此也需要加载tangram。

**API**

**Mock(template)**:template是mock方法挂载的对象，这里一般用“baidu“，即tangram的命名空间。

**expect(mockObj,{method:””,args:[],run:fn})**:mockObj为Mock方法创建出来的mock对象，method用于设置预期的被调用的方法名称，args为预期被调用的方法的参数，fn为执行预期执行的method后执行的方法，可以当做“onsuccess”或“onerror”之类的回调来使用。

**verify(mockObj)**：调用该方法后会验证结果，验证expect中设置的预期值，如预期掉要调用的方法名，方法的个数，方法的参数，参数的个数等。下图为一个简单示例。

