# Web app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 |
| 1.1 | 2012-5-31 | 第二次修改稿 | 莉霞 |
| 1.2 | 2012-06-08 | 第三次修改稿 | 莉霞 |

Change log

V 1.1

1.为了支持fet在mtc中被调用，将url中的“&”改为了“--\_\_--”，即url中的参数通过“--\_\_--”分隔，该参数在config.php中可配，不建议使用类似“&”的特殊字符

V1.2

1. 修复jquery与zepto冲突的问题，将jquery的命名空间改为“$J”
2. 修改源文件与测试文件之间的关系，将v1.1中要求源文件与测试文件一一对应的规则去除，测试目录下除了“import.js”其他均会被认为是测试用例并被展示在list.php中。

### 0 准备工作

如果你已经熟悉Apache+php的安装及配置，可以跳过该章节。由于JS组件手工测试和自动化测试方案都需要依赖服务器和php环境，因此需要首先安装和配置这2个工具。

考虑到apache和php环境配置比较繁琐，且二者某些版本存在不兼容的问题，因此这里推荐大家使用EasyPHP，它是一个Windows下的Apache+Mysql+Perl/PHP/Python开发包，包中集成了PHP、Apache、MySQL，同时也集成了一些辅助的开发工具。详细信息可以参考百科中的“easyphp”词条：<http://baike.baidu.com/view/2376281.htm>。

JS组件的开发和编译都需要依赖FIS，因此也需要安装FIS。当然如果后续JS组件不依赖FIS开发和编译，测试方案也可以进行微调从而适应不同需求。

##### 0.1 EasyPHP安装

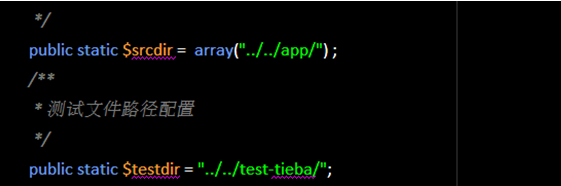
EasyPHP的安装很简单，不需要做任何特别设置，安装时一路next即可。官方下载地址：<http://sourceforge.net/projects/quickeasyphp/files/EasyPHP/5.3.9/EasyPHP-5.3.9-setup.exe/download>（EasyPHP 5.3.9 (#PHP 5.3.9, #Apache 2.2.21, #MySQL 5.5.20, #PhpMyAdmin 3.4.9, #Xdebug 2.1.3）。安装好后打开目录“C:\Program Files\EasyPHP-5.3.9\www”，将工程文件拷贝到这个目录下即完成部署。

### STEP 1 web app测试环境配置

#### 1.1 web app测试方案配置

**1.1.1 配置web app测试框架**

首先下载fet框架所有代码，并将所有代码放在服务器目录下。在“conf/config.php”中设置待测组件、测试代码以及覆盖率的目录，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。



$srcdir设置源文件的路径，支持传入多个源文件，$testdir中设置测试代码的路径。

$covdir设置覆盖率的文件夹（计算覆盖率时需要用jscoverage.exe——js的覆盖率计算工具，对$srcdir下的文件进行操作，生成一个可以计算覆盖率的与$srcdir功能完全一致的文件夹，存放在$covdir下）。



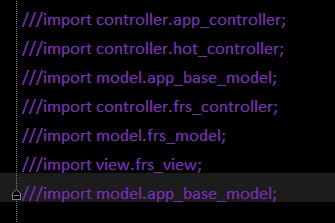
**注：**config.php配置的路径是通过bin/list.php来读取的，因此也是相对于bin/list.php的路径。

**1.1.2 目录结构**

~~所有测试文件与源文件必须一一对应（v1.1旧功能），即若源文件下存在文件“model/frs\_model.js”，那么如果需要对该文件进行测试，那么测试文件夹下也应当存在”model/frs\_model.js”这个文件。~~ V1.2中不要求测试文件与源文件一一对应，测试目录下除了import.js其他均会被当做用例在list.php中展示。测试框架常用的文件为“list.php”和“run.php”。“list.php”用于浏览当前所有的待测组件的用例。

**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。

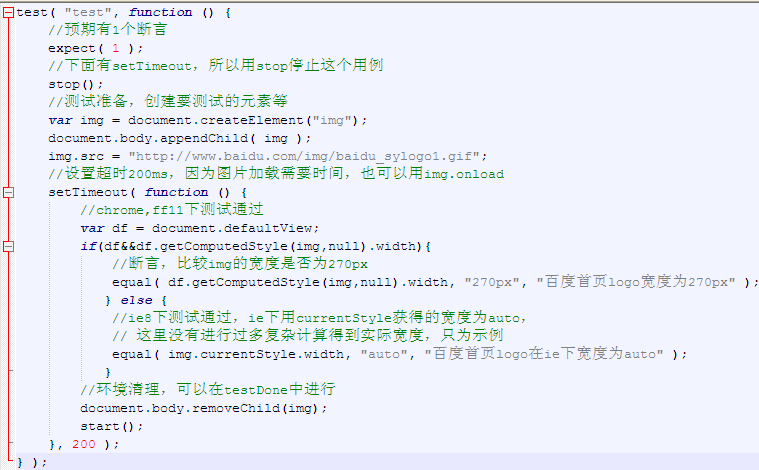
**moduleStart和moduleDone：**在module执行前先执行moduleStart，当module执行结束后再执行moduleDone，因此可以在moduleStart中添加module初始化代码，在moduleDone中添加module的环境清理代码。

**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

### 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。

**2.1.1 执行单个用例**

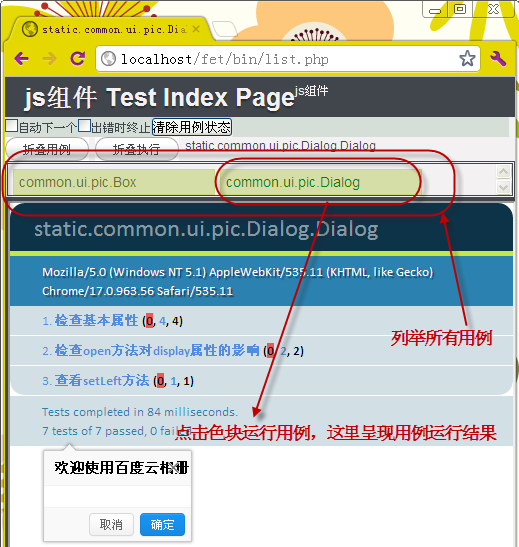
如果想执行用例Dialog，打开页面“<http://localhost/fet/bin/run.php?case=static.common.ui.pic.Dialog.Dialog>”就可以直接运行这个用例，其中参数“case”的值表示Dialog这个用例对应的路径“static/common/ui/pic/Dialog/Dialog.js”，case参数中不需要加js后缀。运行结果截图如下：



**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://localhost/fet/bin/list.php”即可浏览当前所有的用例文件](http://localhost/componentDemo/test/tools/br/run.php?case=test.testcase.common_Dialog)，如下图所示，list.php页面上显示了所有的待测用例：“Dialog”和“Box”组件，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）只浏览某些用例

如果用例特别多，只想浏览某一类的用例，那么可以在list.php后面添加filter参数，参数值为ConfigTest::$srcdir目录下（不包括）该文件的全路径，以“.”连接每一层目录。例如“<http://localhost/componentDemo/test/tools/br/list.php?filter=static.common.ui.pic.Dialog>”将只显示以“Dialog”的用例。



**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://localhost/componentDemo/test/tools/br/list.php?batchrun=true>”页面后，所有的用例将会顺序自动运行。下图为所有用例执行自动执行完毕的效果图。

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**

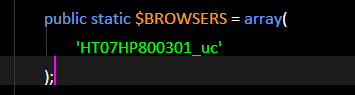
list.php还提供其他参数选项，用于满足不同的需求。

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$BROWSERS中设置完毕后，在想要获取结果的浏览器中输入“<http://localhost/fet/bin/list.php?cov=true--__--browser=chrome--__--batchrun=true>”，其中localhost为服务器名，这里是本地服务器，”fet”为fet框架的工程名，参数“cov”表示会计算覆盖率信息，参数“browser”用于设置当前的浏览器，如这里设置为chrome，则所有用例执行结束后将会在测试文件夹下生成report/chrome.xml文件。当config.php中配置的浏览器均运行完毕时，所有的”$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。

到这里，web app的测试方案的介绍就结束了，附录是QUnit的API文档以及JS框架的规范。

### 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。

