# 测试框架使用常见问题Q&A

1. **环境搭建**

Php+apache即可，也可使用easyphp这个工具，自身已经集成了php和apache。

1. **测试文件组织方式：**

源文件与测试文件目录结构应当相同，例如src目录下的文件目录为“src/core/dom/domUtils.js”，那么test目录下的结构为“test/core/dom/domUtils.js”

1. **如何引入我所需要的源文件**

通过Tools/br/impor.php服务器端去分析import.js文件完成这一功能，在根目录下的import.js:中将测试页面中需要的文件引入进来，例如测试文件中依赖源文件中的“core.browser”，则在import.js中只需要添加“///import core.browser;”即可。

1. **测试框架根目录下index.php这个文件是干嘛用的**

由于测试框架的测试页面目录较深，为了避免很长的服务器地址，利用index.php直接跳转到页面

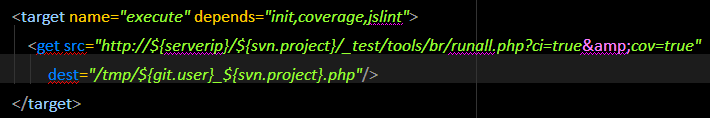
1. **Build\_run.xml这个文件是干嘛的**

Ant脚本，用于CI构建，功能包括：从1）CI服务器从svn拉取最新的代码；2）计算覆盖率 3）进行js静态代码检查（用jslint） 4）启动所有浏览器运行用例，并产生结果

其中，

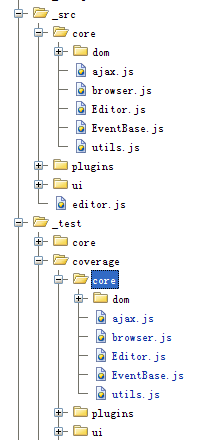
1）代码静态检查的jar包在“\tools\lib”下

2）关于build\_run.xml这个文件中最后一个target元素中“get src”进行的操作就是在CI服务器端请求测试服务器上的一个资源，从而使得所有测试机上的浏览器执行用例。runall.php会调用staf，从而唤起多台测试机上的多个浏览器。“Ci=true”这个参数好像是个坑，没有太大意义（目前当带有“ci”这个参数会不发送测试结果的邮件）……带有参数“cov=true”时会计算当前代码的覆盖率。



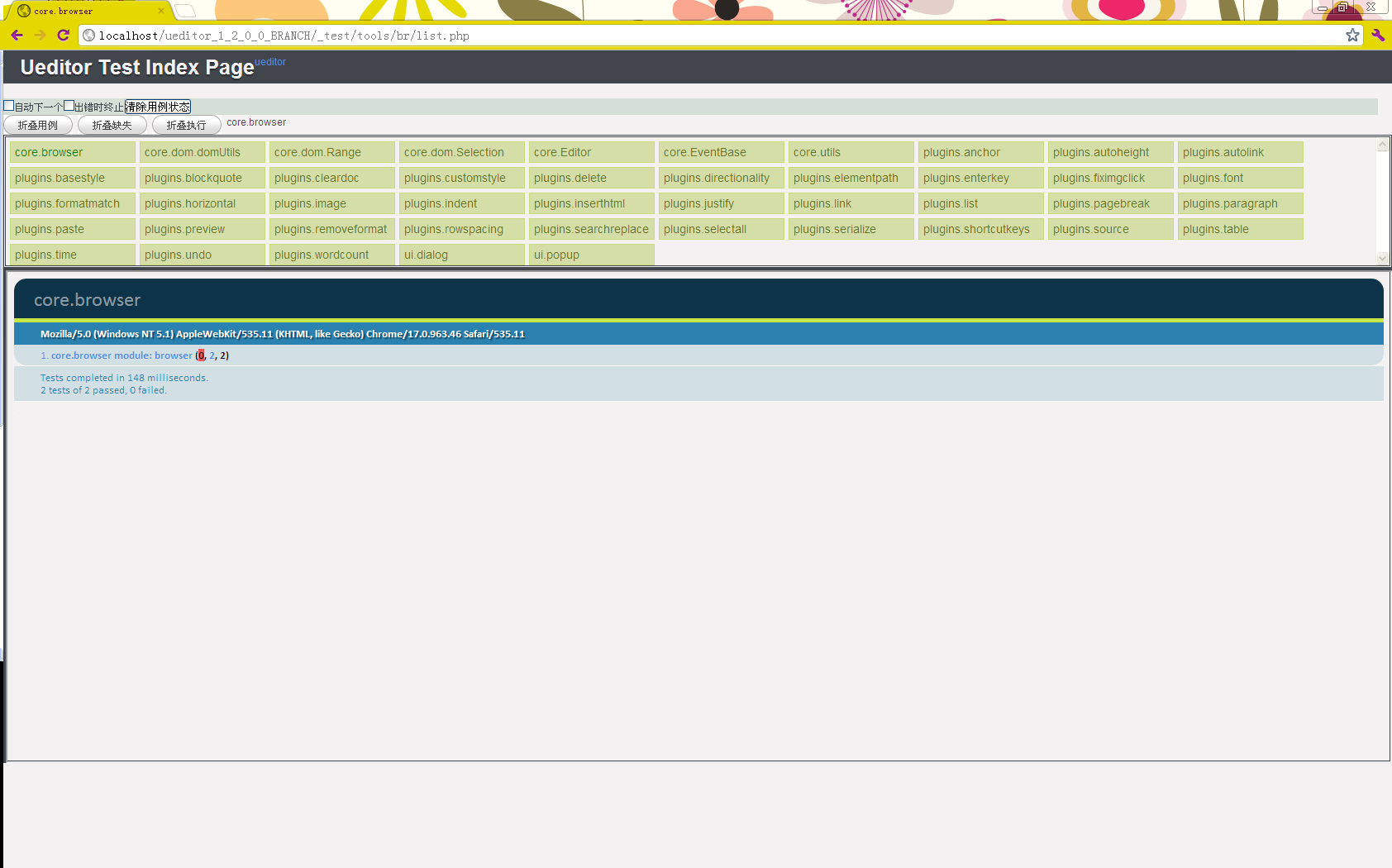
1. **覆盖率是如何计算的**

通过“tools\coverage”文件夹下的“jscoverage.exe”生成，命令行为“jscoverage.exe --encoding=UTF-8 ../../../\_src ../../coverage”，该命令会自动产生一个coverage文件夹，该文件夹下产生与src同样的文件结构，所有文件与src下文件同名。所不同的是coverage下的文件对src下的所有文件进行了重写，对任意一个文件中的任意一行产生一个计数器，没调用一次加1，从而可以计算出该文件的覆盖率。计算覆盖率时，Tools/br/impor.php会自动去加载coverage下的文件，而不是去访问src下的文件。

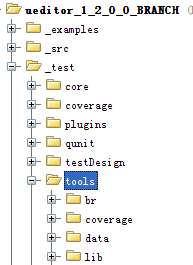


1. **如何运行测试框架**

测试代码与源代码存放于同一个根目录下（也可以不放在同一路径），测试框架置于同源的目录下，在浏览器中打开形如“<http://localhost/ueditor_1_2_0_0_BRANCH/_test/tools/br/list.php>”的页面即可看到所有测试用例。



如下图中“tools”目录即对应于测试框架。



1. **如何配置src、test、coverage的目录**

Tools/br/Config.php中“$SOURCE\_PATH”，“$test\_PATH”、“$COVERAGE\_PATH”分别对应src、test和coverage的路径。

1. **如何运行用例**

本测试框架有3种方法可以运行用例:

1. 一种形如“<http://localhost/ueditor_1_2_0_0_BRANCH/_test/tools/br/list.php>”，且支持参数配置，如“<http://localhost/ueditor_1_2_0_0_BRANCH/_test/tools/br/list.php>？batchrun=true&filter=core”,batchrun=ture表示批量运行，打开这个链接时会自动运行所有的用例。而”filter”则支持过滤用例，如只想运行”core”目录下的所有用例，用”filter=core”即可达到目的。
2. 如果只想查看单个用例，则在浏览器中输入“<http://localhost/ueditor_1_2_0_0_BRANCH/_test/tools/br/lrun.php>?case=core.dom.domUtils”，那么就会自动运行“core.dom.domUtils”这个用例，用例对应于通常对应于一个测试文件，如“core/dom/domUtils.js”.
3. 如果想同时运行多台机器上的多个浏览器，则在浏览器中输入形如“<http://localhost/ueditor_1_2_0_0_BRANCH/_test/tools/br/runall.php>”，这个在前面也提过。
4. **如何在多台测试机中运行用例**

基于staf，tools/br/lib中的staf相关文件提供staf库的支持，在Tools/br/runall.php中调用staf命令，会起浏览器。需要在服务器和每一台测试机上安装staf，设置staf开机启动，并在**windows测试机上**staf的配置文件”bin/staf.cfg”设置为：

# Turn on tracing of internal errors and deprecated options

TRACE ENABLE ALL

TRACE SET DESTINATION TO FILE {STAF/Config/STAFRoot}\STAF.trc

# Enable TCP/IP connections

interface ssl library STAFTCP option Secure=Yes option Port=8550

interface tcp library STAFTCP option Secure=No option Port=8500

# Set default local trust

trust default level 5

# Add default service loader

serviceloader library STAFDSLS

# Set default options

SET DEFAULTSAMECONSOLE

**Linux服务器下**staf安装目录下”bin/STAF.cfg”设置为：

# Turn on tracing of internal errors and deprecated options

trace enable all services

trace enable all tracepoints

trace set destination to file {STAF/Config/STAFRoot}/staf.trc

trace set MAXSERVICERESULTSIZE 50M

# Enable TCP/IP connections

interface ssl library STAFTCP option Secure=Yes option Port=6550

interface tcp library STAFTCP option Secure=No option Port=8500

#Set default local trust

trust default level 5

#trust machine local level 5

# Add default service loader

serviceloader library STAFDSLS

最后通过“**staf ping**”命令，如果服务器和测试机能互相ping通的话那就说明设置成功了。

1. **如何查看测试结果**

如果是自己本地调试，那执行第9步就可以查看测试结果。QUnit采用可视化的展示方式，测试成功与失败一目了然，如下图所示：



如果是在CI上批量运行，那么每一个浏览器运行结束以后都会在“tools/br/report”文件夹下生成一个以当前浏览器明明的xml文件，如“chrome.xml”。服务器会不断查询当前生成的xml文件是否与tools/br/config.php中配置的浏览器的数目一致，如果不一致，则等待其他浏览器执行完毕。当所有浏览器运行结束后，tools/br/report.php将所有xml文件整合为1个xml文件，存储为“tools/br/report.xml”。测试结果中整合了用例成功、失败的个数以及覆盖率信息。

1. **如何配置测试机和浏览器**

Tools/br/Config.php文件中$BROWSERS变量中设置测试机的ip和相应浏览器在测试机中的安装路径即可。如

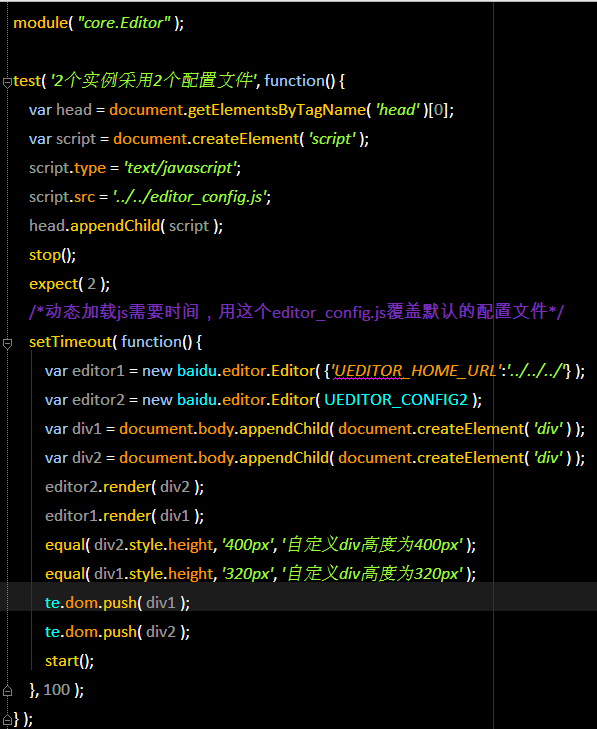


1. **如何加载外部js文件**

在”tools/br/case.class.php”中设置，在”print\_js”函数中添加相应的代码即可。如我想引用一个”tools\br\js”下的tools.js，那么只需要添加代码“print '<script type="text/javascript" src="js/tools.js"></script>' . "\n";”即可，原理就是在测试页面创建一个script标签，从而引入该js文件。

1. **如何利用QUnit写用例**

一般来说，一个源文件对应一个测试文件。测试文件通过module和test来组织用例，一个module下可以有多个test，每一个test可认为是测试用例。每个测试用例里可以有多个断言，断言通常包含以下几种：equal，ok，notEqual等。QUnit还支持异步操作，当涉及到“setTimeout”之类的操作时，在执行setTimeout前执行方法“stop()”，在setTimeout的回调函数末尾调用start()方法即可保证在用例一定会被执行。而“expect”方法则会判断当前的断言个数是否与预期的断言个数一致，这个函数在异步操作过程中尤其有效。下图是一个典型的例子。



1. **tools\br\log.php是干什么的**

由于在CI上运行的过程中总是会出现在线上运行失败但是本地运行却没有问题的用例，log.php用于记录一些更详细的log信息，如是哪一个test的哪一个断言失败了。

1. **QUnit可以mock吗**

可以。例如在“tools/br/js/ext\_qunit.js”文件中就有对qunit自身的函数进行重写，从而进行一些定制。如对QUnit.done方法进行了修改，使得用例执行结束会将用例成功、失败的个数以及覆盖率信息挂载到一个全局变量上。