# 概述

|  |
| --- |
| 本文所述内容在Windows系统下部署测试。  版本信息：  casperjs-1.1.4-2  phantomjs-2.1.1-windows |

CasperJS导航脚本和测试工具，开源，基于PhantomJS（WebKit）和SlimerJS（Gecko）。

注意：Casper测试是没有界面的。

其功能主要分为三大方面：

1. navigation & automation
2. testing：断言
3. scraping：抓取数据、检测

可以完成下面这些常见任务：

* 定义 & 排序浏览器导航步骤
* 填充 & 提交表单
* 点击 & 跟踪链接
* 捕获网页截图 (还可以截取某一区域)
* 在远程DOM上进行断言测试
* 记录事件
* 下载资源，包括二进制文件
* 编写功能测试套件，结果保存为JUnit XML文件
* 抓取网页内容

# 软件安装

|  |  |  |
| --- | --- | --- |
| **名称** | **网址** | **格式** |
| CasperJS | <http://casperjs.org/> | casperjs-1.1.4-1.zip |
| PhantomJS | <http://phantomjs.org/> | phantomjs-2.1.1-windows.zip |
| SlimerJS | <https://slimerjs.org/> | slimerjs-0.10.3.zip |
| Python | <https://www.python.org/> | exe, zip |

以上资源基本都有压缩包版本提供，下载解压即可使用，否则需要安装。

为了方便命令行调用，请将它们的命令目录（通常是bin目录）加入path中。

此时，就可以在命令行输入以下命令检查各软件的版本了：

|  |
| --- |
| casperjs --version  phantomjs --version  python --version |

|  |
| --- |
| CasperJS依赖于PhantomJS或SlimerJS，所以需要安装其中之一。目前使用基于PhantomJS。  CasperJS有使用Python编写的执行脚本，需要依赖Python，但在Windows系统下，也可以使用casperjs.exe来执行。 |

# 入门

## 简单的开始

CasperJS使用JS来编写，并将代码保存为\*.js文件即可。比如，将如下代码保存为baidu.js：

|  |
| --- |
| var casper = require('casper').create();  phantom.outputEncoding = 'GBK';  casper.start('http://www.baidu.com/', function() {  this.echo(this.getTitle());  this.echo('test');  });  casper.run(); |

这样，就可以使用CasperJS的命令来执行它了：

|  |
| --- |
| casperjs baidu.js |

## 测试流程

1. 创建CasperJS实例

|  |
| --- |
| var casper = require('casper').create(); |

1. 步骤

|  |
| --- |
| start()  then\*()  wait\*()  open() |

1. 执行

|  |
| --- |
| run() // 可指定onComplete()回调 |

1. 退出

|  |
| --- |
| exit()  die() |

# 测试环境

CasperJS也是一个测试框架，但测试需要在casperjs test子命令下执行，且测试脚本稍有不同：

|  |
| --- |
| casper.test.begin('Test baidu login', 1, function suite(test) {  casper.start('https://www.baidu.com/', function() {  this.echo(this.getTitle());  });  ……  casper.run(function() {  test.done();  });  }); |

注意：不需要创建casper实例；并且测试完成后，需要调用test.done()；且使用casperjs test a.js执行。

|  |
| --- |
| 1.1版本后，不可以在非测试环境中使用casper.test属性。  1.1-beta3版本后，在测试环境中不可以覆盖预定义的casper实例。 |

## 设置配置项

由于测试环境下，casper实例是预配置的，所以没有办法定义配置信息。但可以在脚本中更新：

|  |
| --- |
| casper.options.optionName = optionValue;  casper.options.clientScripts.push("new-script.js"); |

## setUp() & tearDown()

Tester.begin()不仅可接受一个函数，也可以使用一个配置对象来描述测试套。

配置对象中可设置setUp()和tearDown()函数：

|  |
| --- |
| casper.test.begin('Cow can moo', 2, {  setUp: function(test) {  …  },  tearDown: function(test) {  …  },  test: function(test) {  test.assertEquals(this.cow.moo(), 'moo!');  test.assert(this.cow.mowed);  test.done();  }  }); |

# 命令行

## 参数解析

CasperJS自带有内置的命令行解析器，可以自动解析命令行参数。

可以有两种参数：位置参数（positional argument）和命名选项（named option）。

它们可以从cli模块中分别取得，如：casper.cli.args和casper.cli.options。

传参形式分别为：arg1 arg2和--foo=bar。

下面的脚本可以输出命令行参数：

|  |
| --- |
| var casper = require("casper").create();  casper.echo("Casper CLI passed args:");  require("utils").dump(casper.cli.args);  casper.echo("Casper CLI passed options:");  require("utils").dump(casper.cli.options);  casper.exit(); |

注意：在Windows环境下，如果参数中有空格，需要使用转义的双引号包围，如：–foo=\”space bar\”。

## 原生（raw）参数值

参数会自动转换为探测类型，这可能不是我们想要的，比如命令行执行如下：

|  |
| --- |
| casperjs test.js --foo=01234567  casper.cli.get(‘foo’) // 该脚本会返回1234567 |

如果我们想要得到原始的字符串，则需要：

|  |
| --- |
| casper.cli.raw.get(‘foo’) |

## 测试环境命令行参数

测试命令执行时，命令行除了“--”的命名参数外，其他参数都被认为是脚本文件或目录，否则将会因找不到路径而报错。

# 常用配置

## 中文乱码

|  |
| --- |
| phantom.outputEncoding = "utf-8"; |

比如，当我们在命令行中执行脚本，获取百度首页的标题时，通常会是中文乱码。这是因为，百度网页是UTF-8，但cmd命令行是GBK编码，所以会出现中文乱码，则需要我们指定编码为GBK。

## 引入插件

有时，可能我们想在脚本中使用第三方js库，则可以通过clientScripts配置项引入：

|  |
| --- |
| var casper = require('casper').create({  clientScripts: ["jquery.js"] // 本地文件  }); |

但是，在testing脚本中，casper实例不是我们创建，则使用下面的方式引入：

|  |
| --- |
| casper.options.clientScripts.push("new-script.js"); |

需要注意的是，这种方式引入的时机是在页面加载完成后，所以，在页面加载时不能使用到这些引入的库。但是，有的情况下我们需要在页面加载时，让页面使用引入的库。比如，由于phantomjs的浏览器内核版本较老，可能不支持es6，这时，需要使用另一种方式引入es6的依赖库：

|  |
| --- |
| // 注入ES6支持库  casper.options.onPageInitialized = function() {  casper.page.injectJs("es6-shim.js");  }; |

以上是在page对象初始化完成时进行注入的，在页面加载之前。

## 日志输出

### 脚本日志

默认日志不打印到标准输出，若需要，应开启verbose选项。

默认日志级别是error，用logLevel选项设置。

因此，如果在命令行参数中指定日志等级及输出，那么命令通常是下面这样的：

|  |
| --- |
| casperjs [test] xxx.js --verbose --logLevel=debug |

很明显，这样比较繁琐，我们也可以在脚本中覆盖这些配置项，如：

|  |
| --- |
| casper.options.verbose=true;  casper.options.logLevel="debug"; |

### 页面错误日志

默认情况下，页面出错信息也不会出现在日志中，除非监听“page.error”事件：

|  |
| --- |
| casper.on("page.error", function(msg, trace) {  this.echo("Error: " + msg, "ERROR");  }); |

## 读入外部文件内容

理论上来说，测试数据不应该混在脚本中。因此，我们可以把测试数据放在外部的一个文件里，再在脚本中读入数据。

|  |
| --- |
| var fs = require("fs");  var filePath = "readfile-data.txt";  var content = fs.read(filePath); |

注意：将会一次性读入文件所有内容，并得到其字符串表示。需要自行处理这些文本数据，因此，以JSON格式提供数据是一种比较方便的形式。

## 视图大小

截图的图片大小是根据视图（类似浏览器窗口）大小而定的，默认大小应该是400\*300，这显然不是我们想要的。这就需要设置视图大小：

|  |
| --- |
| casper.start(data.page.url).viewport(800, 600).then(function init() {  casper.capture("test-internal-demo-" + (captureIdx++) + "-init.png");  }); |

## 下载文件

需要关闭web security。

### 命令行关闭

|  |
| --- |
| casperjs --web-security=no myscript.js |

### 配置项关闭

|  |
| --- |
| var casper = require('casper').create({  pageSettings: {  webSecurityEnabled: false  }  }); |

### 代码关闭

|  |
| --- |
| casper.page.settings.webSecurityEnabled = false; |

# 事件

## 监听获取log

|  |
| --- |
| casper.then(function() {  this.page.onConsoleMessage = function(e) {  console.log(e);  }  search\_result\_titles = this.evaluate(getTitles)  this.echo(search\_result\_titles.join('\n'))  }) |

假设getTitles方法中有访问页面的console.log事件，则可以侦听并导出到CasperJS中。

## 获取Alert提示信息

|  |
| --- |
| // 读取Alert消息  var alertMessage;  casper.on("remote.alert", function(msg) {  alertMessage = msg;  this.echo("Alert: " + msg, "WARNING");  }); |

## 输出错误信息

|  |
| --- |
| // 输出错误消息  casper.on("page.error", function(msg, trace) {  this.echo("Error: " + msg, "ERROR");  }); |

# 常用操作

|  |  |  |
| --- | --- | --- |
| **操作** | **函数** | **示例** |
| 控制台输出 | echo | casper.start('http://www.baidu.com/', function() {  this.echo(this.getTitle());  }); |
| 获取页面标题 | getTitle | 见上 |
| 获取页面截图 | capture | casper.then(function() {  this.capture('baidu-homepage.png');  }); |
| 区域截图 | captureSelector | casper.start('http://domain.tld/page.html', function() {  this.captureSelector('capture.png', '.article-content'); //给页面中'.article-content'选择器匹配的元素截图,输出图片文件名为cpature.png,目录为当前目录  }); |
| 填写表单 | fill | casper.then(function() {  this.fill('form[action="/s"]', { wd: 'thoughtworks' }, true);//填入form，进行搜索  }); |
| 注入函数 | evaluate | casper.then(function() {  search\_result\_titles = this.evaluate(getTitles)  this.echo(search\_result\_titles.join('\n'))  });  function getTitles() {  …  } |
| 等待 | waitForSelector | casper.start('https://twitter.com/casperjs\_org', function() {  this.waitForSelector('.tweet-row', function() { //等到'.tweet-row'选择器匹配的元素出现时再执行回调函数  this.captureSelector('twitter.png', 'html'); //成功时调用的函数,给整个页面截图  }, function() {  this.die('Timeout reached. Fail whale?').exit(); //失败时调用的函数,输出一个消息,并退出  }, 2000); //超时时间,两秒钟后指定的选择器还没出现,就算失败  }); |
| 键盘输入 | sendKeys | this.sendKeys('input#login\_user', 'zwq'); |
| 鼠标点击 | click | this.click(".btn"); |

# 选择器

可以使用CSS选择器，也可以使用XPath。

## 使用CSS3选择器

默认使用。

## 使用XPath

|  |
| --- |
| var x = require('casper').selectXPath;  casper.then(function() {  this.click(x('/html/body/div[1]/div[1]/div/div[3]/a[7]'));  }); |

# 参考

## 命令行选项

|  |  |  |  |
| --- | --- | --- | --- |
| **选项** | **说明** | **候选值** | **备注** |
| --direct \* | 打印日志信息到控制台 |  | 1.1 已废弃 |
| --verbose | 替代--direct |  |  |
| --log-level | 日志级别 | debug  info  warning  error |  |
| --engine | 浏览器引擎 | phantomjs (default)  slimerjs |  |
| --xunit | 导出测试套结果到XUnit XML文件 | <filename> | 仅测试环境 |
| --auto-exit | 自动退出 | yes  no | 仅测试环境 |
| --includes | 每个测试文件执行前置包含 | <js filenames> | 仅测试环境。概念类似“测试环境”setUp() |
| --pre | 整个测试套执行前包含 | <js filename> | 仅测试环境。概念类似“测试集”setUp() |
| --post | 整个测试套执行后包含 | <js filename> | 仅测试环境。概念类似“测试集”tearDown() |
| --fail-fast | 当首个失败时终止测试套 |  | 仅测试环境 |
| --concise | 简明输出 |  | 仅测试环境 |
| --no-colors | 无颜色输出 |  | 仅测试环境 |

注意：可以使用PhantomJS和SlimerJS的命令行选项。

# 注意事项

## 选择器中的字符转义

有时，在不太规范的页面中，标签属性可能存在存在特殊字符，这时，选择器就需要对这些字符进行转义。

转义字符是反斜杠“\”，但需要两个反斜杠。比如：

|  |
| --- |
| // <input id="#submit" …>  casper.mouse.click("#\\#submit"); |

## 提示框

以alert提示框为例，当某个操作引起弹出提示框时，脚本中需要使用waitForAlert()函数进行等待。如：

|  |
| --- |
| casper.waitForAlert(function (response) {  test.assert(response.data == data.page.noneAlert);  }); |

我们可以看到，waitForAlert()回调函数中有个参数，它是一个响应对象，其data属性即是弹出提示框上的提示信息。

需要注意的是，由于没有界面，所以并没有真实的提示框弹出，截图也不会截到。因此，也不必进行任何提示框操作来关闭它。

另外，也可以通过监听remote.alert事件来获取消息信息，该事件的参数即是提示信息。

|  |
| --- |
| var alertMessage;  casper.on("remote.alert", function(msg) {  alertMessage = msg;  }); |

另外，说明下，响应对象大致内容如下：

|  |
| --- |
| {  "contentType":"text/html",  "headers":  [  {  "name":"Accept-Ranges",  "value":"bytes"  },  {  "name":"ETag",  "value":"W/\"1142-1513051232130\""  },  {  "name":"Last-Modified",  "value":"Tue, 12 Dec 2017 04:00:32 GMT"  },  {  "name":"Content-Type",  "value":"text/html"  },  {  "name":"Content-Length",  "value":"1142"  },  {  "name":"Date",  "value":"Tue, 12 Dec 2017 05:48:57 GMT"  }  ],  "id":2,  "redirectURL":null,  "stage":"end",  "status":200,  "statusText":"",  "time":"2017-12-12T05:48:58.118Z",  "url":"http://localhost:8080/demo/?",  "data":"账户不能为空"  } |

## 测试react

如果使用CasperJS以PhantomJS为基础测试ReactJS编写的程序，那么，可能会发现一些问题，典型的情况是截图一片空白，页面没有实质内容。

这是因为ReactJS中通常使用到了ES5甚至是ES6的特性，而PhantomJS使用的浏览器内核版本比较低，并不支持这些特性。

我们可以通过输出页面错误消息的方式来验证这一点：

|  |
| --- |
| // 输出错误消息  casper.on("page.error", function(msg, trace) {  this.echo("Error: " + msg, "ERROR");  }); |

因此，为了能在低版本的内核上运行，需要引入相关支持库，esX-shim.js。比如：

|  |
| --- |
| // 注入ES6支持库  casper.options.onPageInitialized = function() {  casper.page.injectJs("es6-shim.js");  }; |

注意：我们是在page.initialized事件中注入依赖库的，该事件在页面对象初始化后触发的。不能使用clientScripts配置项，因为它发生在页面加载完成后，此时已经晚了。

其他类似的情况也可以这样处理。

# 技巧

## 属性映射

|  |
| --- |
| var links = [];  function getLinks() {  var links = document.querySelectorAll('h3.r a');  return Array.prototype.map.call(links, function(e) {  return e.getAttribute('href');  });  } |

# FAQ

## 错误：Maximum call stack size exceeded.

在测试reactjs的页面时，当使用sendKeys、click等模拟操作时，几乎总是会报 Maximum call stack size exceeded.

从现象看，很可能是模拟操作函数里触发的事件与前端框架生成的控件事件存在递归调用引起的。

目前尚未定位到问题源，待跟踪解决。