

# 前言

在这段期间，我一直在找关于服务器的端测试方案，自动化工具等等，无意间我发现了 Selenium 这个工具。在试用一段时间后，觉得 Selenium 确实是一个很不错的 Web 测试工具。在和强大的 QTP 比较后，我最后还是选择了使用 Selenium，主要的原因是**工具使用灵活，简单，并且完全满足我的要求**。而 QTP 虽然强大，但它的使用让我觉得非常繁琐，而调试功能也让我郁闷.....鉴于种种这样的原因，我最后选择了 Selenium。

本系列文章主要是总结我在前一段时间了解到的关于 Selenium 的一些知识。

## QTP VS Selenium

下面我想先稍微讨论一下这两个工具，我主要从功能和文档资料这两方面对比 QTP 和 Selenium。

### 功能

从 Web 测试的角度，两个工具都能满足正常的测试需求，这个你无需怀疑！但 QTP 更强大些，因为：

- 1.它是商业软件，只要你有钱，什么都可以有。
- 2.它支持对操作 Windows 窗口，当你要从 Web 上下载一些东西，自然此时就会弹出一个“下载框”，由于那个框框是 Windows 窗口，Selenium 是处理不了的，所以必须通过第三方的脚本处理。

### 文档和参考资料资料

如果你这方面的新手，当你用 QTP/Selenium 时，相信书和文档是你第一样会找的东西。在这样面，QTP 使用在中国使用还是蛮广泛的，资料和书籍也相当丰富，而 Selenium 的资料就相对比较少一些，找资料最好的地方就是官方论坛。

### 为什么用 Selenium

既然上面两项都是 QTP 优胜一些，为什么我们还是用 Selenium，因为：

- 1.Selenium 使用灵活，简单，写出的测试案例非常简洁，优美，也易于维护。
- 2.Selenium RC 支持用多种语言编写测试案例，你可以用 VS2008写测试案例了:>。
- 3.如果你有一个测试平台，你会发现把 QTP 整合到平台是很麻烦的事情。
- 4.....

如果以上 Selenium 的优点不能打动你，又或者你是测试新手，对测试经验不多，对编程也不熟悉，那么你还是继续使用 QTP 吧，不过相信有一天，你会开始尝试 Selenium 的。

### Selenium 名字的来源

在这里，我还想说一下关于 Selenium 名字的来源，很有意思的:>: Selenium 的中文名为“硒”，是一种化学元素的名字，它对汞（Mercury）有天然的解毒作用，实验表明汞暴露水平越高，硒对汞毒性的拮抗作用越明显，所以说硒是汞的克星。大家应该知道 Mercury 测试工具系列吧（QTP，QC，LR，WR...），他们功能强大，但却价格不菲，大家对此又爱又恨！故 thoughtworks 特意把他们的 Web 开源测试工具命名为 Selenium，以此帮助大家脱离汞毒。

# 目录

- 1.Selenium 私房菜系列1 -- Selenium 简介**
- 2.Selenium 私房菜系列2 -- XPath 的使用【ZZ】**
- 3.Selenium 私房菜系列3 -- Selenium API 参考手册【ZZ】**
- 4.Selenium 私房菜系列4 -- Selenium IDE 的使用**
- 5.Selenium 私房菜系列5 -- 第一个 Selenium RC 测试案例**
- 6.Selenium 私房菜系列6 -- 深入了解 Selenium RC 工作原理(1)**
- 7.Selenium 私房菜系列7 -- 深入了解 Selenium RC 工作原理(2)**
- 8.Selenium 私房菜系列8 -- 玩转 Selenium Server**
- 9.Selenium 私房菜系列9 -- Selenium RC 服务器命令行参数列表【ZZ】**

## 参考资料

- [1].<http://seleniumhq.org/>: Selenium 官网。**
- [2].<http://openqa.org/>: Selenium 官方论坛，有很多参考资料: >**
- [3].[Selenium 中文](#): 记录大量 Selenium 中文资料的地方。**

作者: **[hyddd](#)**

日期: **2009-09-02**

# Selenium 私房菜系列1 -- Selenium 简介

## 一.Selenium 是什么？

Selenium 是 ThroughtWorks 公司一个强大的开源 Web 功能测试工具系列，本系列现在主要包括以下4款：

1.Selenium Core: 支持 DHTML 的测试案例（效果类似数据驱动测试），它是 Selenium IDE 和 Selenium RC 的引擎。

2.Selenium IDE: FireFox 的一个插件，支持脚本录制。

3.Selenium RC: Selenium Remote Control。后续的系列文章我会主要针对 Selenium RC 展开介绍。

4.Selenium Grid: 允许同时并行地、在不同的环境上运行多个测试任务，极大地加快 Web 应用的功能测试。

## 二.选择合适的 Selenium 工具

既然 Selenium 工具有4款这么多，那到底如何选择呢？？我从"[Selenium 官网](#)"这里找了一个表：

	Selenium IDE	Selenium Control	Remote Selenium Core	Selenium Core HTA
浏览器支持	仅 Firefox	很多	所有	仅 IE
需要远程安装	否	否	是	否
支持 HTTPS/SSL	是	是*	是	是
支持跨域	是	是*	否	是
需要 Java	否	是	否	否
将测试结果保存到磁盘	是	是	否	是
多语言支持	仅 Selenese	很多	仅 Selenese	仅 Selenese

这里没有介绍 Selenium Grid，但介绍了另外一个 Selenium Core HTA，Selenium Core HTA 其实是 Selenium Core 的额外模式，你只要 Selenium Core 配置稍加修改，即为 HTA 模式，Selenium Core HTA 可以在 IE 最高安全等级（特权）下工作，这意味着它仅能在 IE 下工作，由于限制较大，下面将排除对 Selenium Core HTA 的讨论。

### 1.浏览器支持:

(1).Selenium IDE 仅可以在 Firefox 中工作。

(2).Selenium Remote Control 支持很多浏览器，包括最常用的：firefox，ie，safari 等 N 款浏览器。

(3).Selenium Core 支持的浏览器是最广的，这点和它的实现有关。作为 IDE 和 RC 的引擎，Selenium Core 几乎可以在任何浏览器中工作。

### 2.需要远程安装: 是否需要在被测网站的服务端安装？

这里只有 Selenium Core 需要，这是出于同源策略的原因。这也是 Selenium Core 一个很大的限制，试问，如果你要测试 Google.com，还得在 google 的服务器上装一个 Selenium Core，那是多搞笑的一件事。

而 Selenium IDE 和 Selenium Core HTA 不会被同源策略所限制，因为他们对浏览器扩展了。

Selenium RC 提供一个代码服务器来保证 Selenium JS 文件看似来自相同的远程服务器，从而符合同源策略；代理服务器欺骗浏览器，让它认为这里的确有像 <http://www.google.com/selenium/>这样的目录。

### 3.支持 HTTPS/SSL:

这里不说了，都支持。Selenium RC 在“是”后面加\*因为它是在最近版本支持的，仅此而已。

### 4.需要 Java: 准确的说是需要 JRE

这项只有 Selenium RC 需要，上面2中所说的“代理服务器”是一个 Java 程序，需在跑测试案例前启动。

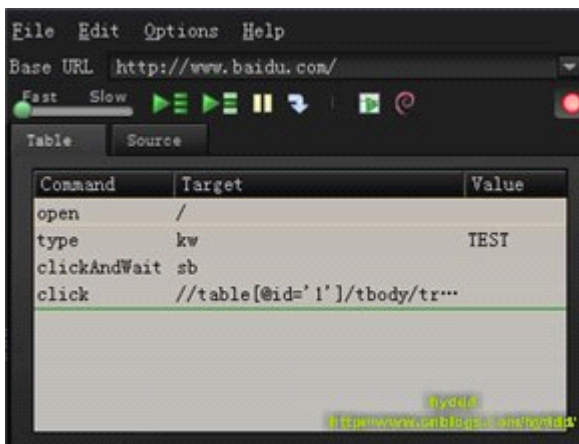
### 5.将测试结果保存到磁盘

只有 Selenium Core 不能将任何测试结果写到磁盘上（因为它用 javascript 写的，它不允许向磁盘写数据），其解决方案是当然你可以将测试结果发送到另外一台服务器保存。这也是 Selenium Core 的一大限制。

### 6.多语言支持

- (1).Selenium IDE 仅支持 Selenium 语言。
- (2).Selenium RC 支持很多语言，如：C#，Java，Python，Ruby 等。
- (3).Selenium Core 也是仅支持 Selenium 语言。

Selenium 语言的测试案例如下：



它的优点是：简单，用（Command，Target，Value）三种元素组成一个行为，并且有辅助录制脚本工具（如：Firefox IDE,Selenium Core 等），不懂程序的测试人员都可以轻松地编写测试案例。

它的缺点是：Selenese 有一些严格的限制：它没有条件（没有"if"表达式），并且它没有循环（没有"For"表达式）。这会编写复杂的测试变得困难甚至不可能。

OK，现在我们来研究下到底该使用哪款工具开展测试！

(1).Selenium IDE 支持并且只支持 Firefox 浏览器，支持的浏览器太少，而依附于 Firefox 也不便于日后开展自动化测试，但是，它的录制快捷好用！并且有代码 转换功能，可以把 Selenium 语言测试案例转为 C#,Java 等语言的测试案例，我建议使用 Selenium IDE + FireBug 进行测试案例的编写，然后转为其他语言的测试案例后，再调用 Selenium RC 运行测试案例。

(2).Selenium Core，它的优点是编写测试案例简单，并且支持绝大多数的浏览器，但缺点也同样明显，Selenium Core 需要远程安装，Selenese 语言也限制了复杂案例的可能性，并且没有良好的外部扩展，这些都会是致命的问题。因为一个款测试工具不可能 100%满足你测试需求的，当它不能满足你测试需求时候，它必须有一个扩展机制可以让你可以使用其他方式满足你需求，否则这款测试软件即使功能强大，也 请三思慎用，否则当投入大量资源后才发现某些问题不能解决，那时候已经晚了，这是我的切身体会。

(3).Selenium RC 是我推荐使用的工具，它支持很多浏览器，可以使用 C#, Java 等语言编写测试案例，易于维护，同时提供了很好的扩展性，所以后续的文档我会以 Selenium RC 作为默认的测试工具。

# Selenium 私房菜系列2 -- XPath 的使用【ZZ】

在编写 Selenium 案例时，少不免是要用到 XPath 的，现在外面关于 XPath 使用的参考资料很多，下面我直接转一篇关于 XPath 使用的文档。如果对 XPath 不熟悉请参考下文，你不需要去百度/Google 搜索关于 XPath 的资料，因为下面的内容已经足够你写测试时使用，如果你已熟悉 XPath，本章大可忽略跳过。

**xpath 的语法**(转载自: <http://www.cnblogs.com/jianjialin/archive/2009/02/01/1382056.html>)

XPath 是 XML 的查询语言，和 SQL 的角色很类似。以下面 XML 为例，介绍 XPath 的语法。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>

  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>

  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd></catalog>
</catalog>
```

## 定位节点

XML 是树状结构，类似档案系统内数据夹的结构，XPath 也类似档案系统的路径命名方式。不过 XPath 是一种模式 (Pattern)，可以选出 XML 档案中，路径符合某个模式的所有节点出来。例如要选 catalog 底下的 cd 中所有 price 元素可以用：

```
/catalog/cd/price
```

如果 XPath 的开头是一个斜线 (/) 代表这是绝对路径。如果开头是两个斜线 (//) 表示文件中所有符合模式的元素都会被选出来，即使是处于树中不同的层级也会被选出来。以下的语法会选出文件中所有叫做 cd 的元素（在树中的任何层级都会被选出来）：

```
//cd
```

## 选择未知的元素

使用星号 (Wildcards, \*) 可以选择未知的元素。下面这个语法会选出/catalog/cd 的所有子元素:

```
/catalog/cd/*
```

以下的语法会选出所有 catalog 的子元素中, 包含有 price 作为子元素的元素。

```
/catalog/*/price
```

以下的语法会选出有两层父节点, 叫做 price 的所有元素。

```
/**/*.price
```

以下的语法会选择出文件中的所有元素。

```
//*
```

要注意的是, 想要存取不分层级的元素, XPath 语法必须以两个斜线开头(/), 想要存取未知元素才用星号(\*), 星号只能代表未知名称的元素, 不能代表未知层级的元素。

## 选择分支

使用中括号可以选择分支。以下的语法从 catalog 的子元素中取出第一个叫做 cd 的元素。XPath 的定义中没有第0元素这种东西。

```
/catalog/cd[1]
```

以下语法选择 catalog 中的最后一个 cd 元素: (XPath 并没有定义 first() 这种函式喔, 用上例的 [1]就可以取出第一个元素。

```
/catalog/cd[last()]
```

以下语法选出含有 price 子元素的所有/catalog/cd 元素。

```
/catalog/cd[price]
```

以下语法选出 price 元素的值等于10.90的所有/catalog/cd 元素

```
/catalog/cd[price=10.90]
```

以下语法选出 price 元素的值等于10.90的所有/catalog/cd 元素 的 price 元素

```
/catalog/cd[price=10.90]/price
```

## 选择一个以上的路径

使用 Or 操作数()就可以选择一个以上的路径。例如:

```
/catalog/cd/title | catalog/cd/artist
```

选择所有 title 以及 artist 元素

```
//title | //artist
```

选择所有 title 以及 artist 以及 price 元素

```
//title | //artist | //price
```

### 选择属性

在 XPath 中，除了选择元素以外，也可以选择属性。属性都是以@开头。例如选择文件中所有叫做 country 的属性。

```
//@country
```

选择所有含有 country 这个属性的 cd 元素：

```
//cd[@country]
```

以下语法选择出含有属性的所有 cd 元素

```
//cd[@*]
```

以下语法选择出 country 属性值为 UK 的 cd 元

```
//cd[@country='UK']
```



# Selenium 私房菜系列3 -- Selenium API 参考手册【ZZ】

大家在编写测试案例想查阅 Selenium API 说明时，可以查阅本文，否则请跳过！

(注：这里 API 版本比较老，新版本的 Selenium 的 API 在这里不一定能找到。)

**Selenium API 说明文档**(转载自：<http://wiki.javascud.org/display/springs/SeleniumRefrence>)

## Commands (命令)

- **Action**

对当前状态进行操作  
失败时，停止测试

- **Assertion**

校验是否有产生正确的值

- **Element Locators**

指定 HTML 中的某元素

- **Patterns**

用于模式匹配

## 1. Element Locators (元素定位器)

\* id=id

id locator 指定 HTML 中的唯一 id 的元素

\* name=name

name locator 指定 HTML 中相同 name 的元素中的第一个元素

\* identifier=id

identifier locator 首先查找 HTML 是否存在该 id 的元素，若不存在，查找第一个该 name 的元素

\* dom=javascriptExpression

dom locator 用 JavaScript 表达式来定位 HTML 中的元素,注意必须要以"document"开头

例如:

dom=document.forms['myForm'].myDropdown

dom=document.images[56]

\* xpath=xpathExpression

xpath locator 用 XPath 表达式来定位 HTML 中的元素,必须注意要以"//"开头

例如:

xpath=//img[@alt='The image alt text']

xpath=//table[@id='table1']//tr[4]/td[2]

\* link=textPattern

link locator 用 link 来选择 HTML 中的连接或锚元素

例如:

link=The link text

\* 在没有 locator 前序的情况下 Without a locator prefix, Selenium uses:

如果以 "document." 开头, 则默认是使用 dom locator, 如果是以 "/" 开头, 则默认使用 xpath locator, 其余情况均认作 identifier locator

## 2. String Matching Patterns (字符串匹配模式)

\* glob:pattern

glob 模式, 用通配符 "\*" 代表任意长度字符, "?" 代表一个字符

\* regexp:regexp

正则表达式模式, 用 JavaScript 正则表达式的形式匹配字符串

\* exact:string

精确匹配模式, 精确匹配整个字符串, 不能用通配符

\* 在没有指定字符串匹配前序的时候, selenium 默认使用 glob 匹配模式

## 3. Select Option Specifiers (Select 选项指定器)

\* label=labelPattern

通过匹配选项中的文本指定选项

例如: label=regexp:^(Oo)ther

\* value=valuePattern

通过匹配选项中的值指定选项

例如: value=other

\* id=id

通过匹配选项的 id 指定选项

例如: id=option1

\* index=index

通过匹配选项的序号指定选项, 序号从 0 开始

例如: index=2

\* 在没有选项选择前序的情况下, 默认是匹配选项的文本

## Actions

描述了用户所会作出的操作。

Action 有两种形式: action 和 actionAndWait, action 会立即执行, 而 actionAndWait 会假设需要较长时间才能得到该 action 的响应, 而作出等待, open 则是会自动处理等待时间。

\* click

click(elementLocator)

- 点击连接, 按钮, 复选和单选框

- 如果点击后需要等待响应, 则用 "clickAndWait"

- 如果是需要经过 JavaScript 的 alert 或 confirm 对话框后才能继续操作, 则需要调用 verify 或 assert 来告诉 Selenium 你希望对对话框进行什么操作。

click aCheckbox

clickAndWait submitButton

clickAndWait anyLink

\* open

open(url)

- 在浏览器中打开 URL,可以接受相对和绝对路径两种形式
- 注意: 该 URL 必须在与浏览器相同的安全限定范围之内

open /mypage

open http://localhost/

\* type

type(inputLocator, value)

- 模拟人手的输入过程, 往指定的 input 中输入值
- 也适合给复选和单选框赋值
- 在这个例子中, 则只是给勾选了的复选框赋值, 注意, 而不是改写其文本

type nameField John Smith

typeAndWait textBoxThatSubmitsOnChange newValue

\* select

select(dropDownLocator, optionSpecifier)

- 根据 optionSpecifier 选项选择器来选择一个下拉菜单选项
- 如果有多于一个选择器的时候, 如在用通配符模式, 如"f\*b\*", 或者超过一个选项有相同的文本或值, 则会选择第一个匹配到的值

select dropDown Australian Dollars

select dropDown index=0

selectAndWait currencySelector value=AUD

selectAndWait currencySelector label=Auslian D\*rs

\* goBack,close

goBack()

模拟点击浏览器的后退按钮

close()

模拟点击浏览器关闭按钮

\* selectWindow

select(windowId)

- 选择一个弹出窗口
- 当选中那个窗口的时候, 所有的命令将会转移到那窗口中执行

selectWindow myPopupWindow

selectWindow null

\* pause

pause(millisecnds)

- 根据指定时间暂停 Selenium 脚本执行
- 常用在调试脚本或等待服务器段响应时

pause 5000

pause 2000

\* fireEvent

fireEvent(elementLocatore,evenName)

模拟页面元素事件被激活的处理动作

fireEvent textField focus

fireEvent dropDown blur

\* waitForCondition

waitForCondition(JavaScriptSnippet,time)

- 在限定时间内, 等待一段 JavaScript 代码返回 true 值, 超时则停止等待

waitForCondition var value=selenium.getText("foo"); value.match(/bar/); 3000

\* waitForValue

waitForValue(inputLocator, value)

- 等待某 input(如 hidden input)被赋予某值,
- 会轮流检测该值, 所以要注意如果该值长时间一直不赋予该 input 该值的话, 可能会导致阻塞

waitForValue finishIndication isfinished

\* store, storeValue

store(valueToStore, variablename)

保存一个值到变量里。

该值可以由自其他变量组合而成或通过 JavaScript 表达式赋值给变量

store Mr John Smith fullname

store \$. {title} \$. {firstname} \$. {surname} fullname

store javascript. {Math.round(Math.PI\*100)/100} PI

storeValue inputLocator variableName

把指定的 input 中的值保存到变量中

storeValue userName userID

type userName \$. {userID}

\* storeText, storeAttribute

storeText(elementLocator, variablename)

把指定元素的文本值赋予给变量

storeText currentDate expectedStartDate

verifyValue startDate \$. {expectedStartDate}

storeAttribute( { } elementLocator@attributeName, variableName. { } )

把指定元素的属性的值赋予给变量

storeAttribute input1@class classOfInput1

verifyAttribute input2@class \$. {classOfInput1}

\* chooseCancel..., answer..

chooseCancelOnNextConfirmation()

- 当下次 JavaScript 弹出 confirm 对话框的时候, 让 selenium 选择 Cancel
- 如果没有该命令时, 遇到 confirm 对话框 Selenium 默认返回 true, 如手动选择 OK 按钮一样

chooseCancelOnNextConfirmation

- 如果已经运行过该命令, 当下一次又有 confirm 对话框出现时, 也会同样地再次选择 Cancel

answerOnNextPrompt(answerString)

- 在下次 JavaScript 弹出 prompt 提示框时, 赋予其 answerString 的值, 并选择确定

## Assertions

允许用户去检查当前状态。两种模式: Assert 和 Verify, 当 Assert 失败, 则退出测试; 当 Verify 失败, 测试会继续运行。

\* assertLocation, assertTitle

assertLocation(relativeLocation)

判断当前是在正确的页面

verifyLocation /mypage

assertLocation /mypage

\* assertTitle(titlePattern)

检查当前页面的 title 是否正确

```

verifyTitle My Page
assertTitle My Page
* assertValue
  assertValue(inputLocator, valuePattern)
  - 检查 input 的值
  - 对于 checkbox 或 radio, 如果已选择, 则值为"on",反之为"off"
  verifyValue    nameField    John Smith
  assertValue    document.forms[2].nameField    John Smith
* assertSelected, assertSelectedOptions
  assertSelected(selectLocator, optionSpecifier)
  检查 select 的下拉菜单中选中的选型是否和 optionSpecifier(Select 选择选项器)的选项相同
  verifySelected dropdown2    John Smith
  verifySelected dropdown2    value=js*123
  assertSelected document.forms[2].dropDown    label=J*Smith
  assertSelected document.forms[2].dropDown    index=0
* assertSelectOptions(selectLocator, optionLabelList)
  - 检查下拉菜单中的选项的文本是否和 optionLabelList 相同
  - optionLabelList 是以逗号分割的一个字符串
  verifySelectOptions dropdown2    John Smith,Dave Bird
  assertSelectOptions document.forms[2].dropdown    Smith,J,Bird,D
* assertText
  assertText(elementLocator,textPattern)
  - 检查指定元素的文本
  - 只对有包含文本的元素生效
  - 对于 Mozilla 类型的浏览器, 用 textContent 取元素的文本, 对于 IE 类型的浏览器, 用 innerText 取元素文本
  verifyText statusMessage    Successful
  assertText //div[@id='foo']/h1    Successful
* assertTextPresent, assertAttribute
  assertTextPresent(text)
  检查在当前给用户显示的页面上是否有出现指定的文本
  verifyTextPresent    You are now logged in
  assertTextPresent    You are now logged in
* assertAttribute(. { } elementLocator@attributeName. { } , ValuePattern)
  检查当前指定元素的属性的值
  verifyAttribute txt1@class    bigAndBlod
  assertAttribute document.images[0]@alt    alt-text
  verifyAttribute //img[@id='foo']/alt    alt-text
* assertTextPresent, etc.
  assertTextPresent(text)
  assertTextNotPresent(text)
  assertElementPresent(elementLocator)
  verifyElementPresent    submitButton
  assertElementPresent    //img[@alt='foo']    assertElementNotPresent(elementLocator)
* assertTable
  assertTable(cellAddress, valuePattern)
  - 检查 table 里的某个 cell 中的值
  - cellAddress 的语法是 tableName.row.column, 注意行列序号都是从 0 开始
  verifyTable    myTable.1.6    Submitted

```

assertTable results0.2 13

\* assertVisible, nonVisible

assertVisible(elementLocator)

- 检查指定的元素是否可视的

- 隐藏一个元素可以用设置 css 的'visibility'属性为'hidden', 也可以设置'display'属性为'none'

verifyVisible postcode

assertVisible postcode

\* assertNotVisible(elementLocator)

verifyNotVisible postcode

assertNotVisible postcode

\* Editable, non-editable

assertEditable(inputLocator)

检查指定的 input 是否可以编辑

verifyEditable shape

assertEditable colour

\* assertNotEditable(inputLocator)

检查指定的 input 是否不可以编辑

\* assertAlert

assertAlert(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 alert 对话框

- alert 产生的顺序必须与检查的顺序一致

- 检查 alert 时会产生与手动点击'OK'按钮一样的效果。如果一个 alert 产生了, 而你却没有去检查它, selenium 会在下个 action 中报错。

- 注意: Selenium 不支持 JavaScript 在 onload()事件时 调用 alert();在这种情况下, Selenium 需要你自己手动来点击 OK.

\* assertConfirmation

assertConfirmation(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 confirmation 对话框和 alert 情况一样, confirmation 对话框也必须在它们产生的时候进行检查

- 默认情况下, Selenium 会让 confirm() 返回 true, 相当于手动点击 Ok 按钮的效果。你能够通过 chooseCancelOnNextConfirmation 命令让 confirm()返回 false. 同样地, 如果一个 cofirmation 对话框出现了, 但你却没有检查的话, Selenium 将会在下个 action 中报错

- 注意: 在 Selenium 的环境下, confirmation 对话框框将不会再出现弹出显式对话框

- 注意: Selenium 不支持在 onload()事件时调用 confirmation 对话框, 在这种情况下, 会出现显示 confirmatioin 对话框, 并需要你自己手动点击。

\* assertPrompt

assertPrompt(messagePattern)

- 检查 JavaScript 是否有产生带指定 message 的 Prompt 对话框

- 你检查的 prompt 的顺序 Prompt 对话框产生的顺序必须相同

- 必须在 verifyPrompt 之前调用 answerOnNextPrompt 命令

- 如果 prompt 对话框出现了但你却没有检查, 则 Selenium 会在下个 action 中报错

answerOnNextPrompt Joe

click id=delegate

verifyPrompt Delegate to who?

## Parameters and Variables

参数和变量的声明范围由简单的赋值到 JavaScript 表达式赋值。

Store, storeValue 和 storeText 为下次访问保存值。

在 Selenium 内部是用一个叫 storeVars 的 map 来保存变量名。

\* Variable Substitution 变量替换

提供了一个简单的方法去访问变量,语法 \$. {xxx}

```
store Mr title
```

```
storeValue nameField surname
```

```
store $. {title} $. {surname} fullname
```

```
type textElement Full name is: $. {fullname}
```

\* JavaScript Evaluation JavaScript 赋值

你能用 JavaScript 来构建任何你所需要的值。

这个参数是以 javascript 开头,语法是 javascript. {'with a trailing'}。

可以通过 JavaScript 表达式给某元素赋值。

```
store javascript. {'merchant'+(new Date()).getTime()} merchantId
```

```
type textElement javascript. {storedVars['merchantId'].toUpperCase() }
```

# Selenium 私房菜系列4 -- Selenium IDE 的使用

前面说过，Selenium IDE 是 Firefox 的一个插件，是可以进行脚本录制以及案例转换，所以 Selenium IDE+Firebug 会成为你日后写测试案例的两大助手(IE 下可以使用 Selenium Core+IEDeveloperToolBar)。

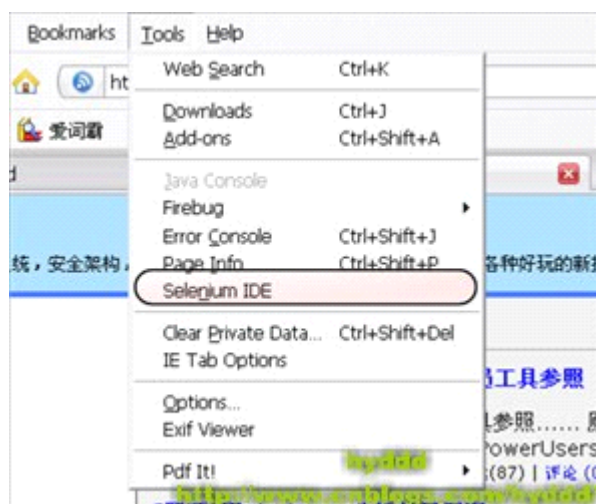
Selenium IDE 下载: <http://seleniumhq.org/download/>

Firebug 下载: <https://addons.mozilla.org/firefox/addon/1843>

下面将演示 **Selenium** 的使用:

## 1.安装 Selenium IDE, Firebug。

## 2.启动 Selenium IDE:



IDE 启动后，弹出如下对话框:





上图标明了一些 Selenium IDE 的主要功能。其中，由 Command, Target, Value 组成的表格就是脚本，每个脚本都是由一条一条的 Action(行为)组成，而每个 Action 又由(Command, Target, Value)三者组成。Command 就是上文《API 参考手册》提到的内容，Target 指的是 Web 中的某个对象，比如：文字，输入框等等，如果选取对象呢？呵呵，这里就用到了 XPath，不熟悉可以参考《XPath 的使用》，而 Value 就是这个对象的值。

### 3.脚本的录制及运行

当弹出上面的 IDE 窗口后，我们就可以开始 Selenium 的脚本录制了，右上角有个红色的圆点，当它下按时(如上图)就表示 IDE 正在进行脚本录制。OK，开始录制，录制的时候，直接操作 Firefox 浏览器窗口就可以了，IDE 会自动记录你的操作的，下面我演示一个例子：

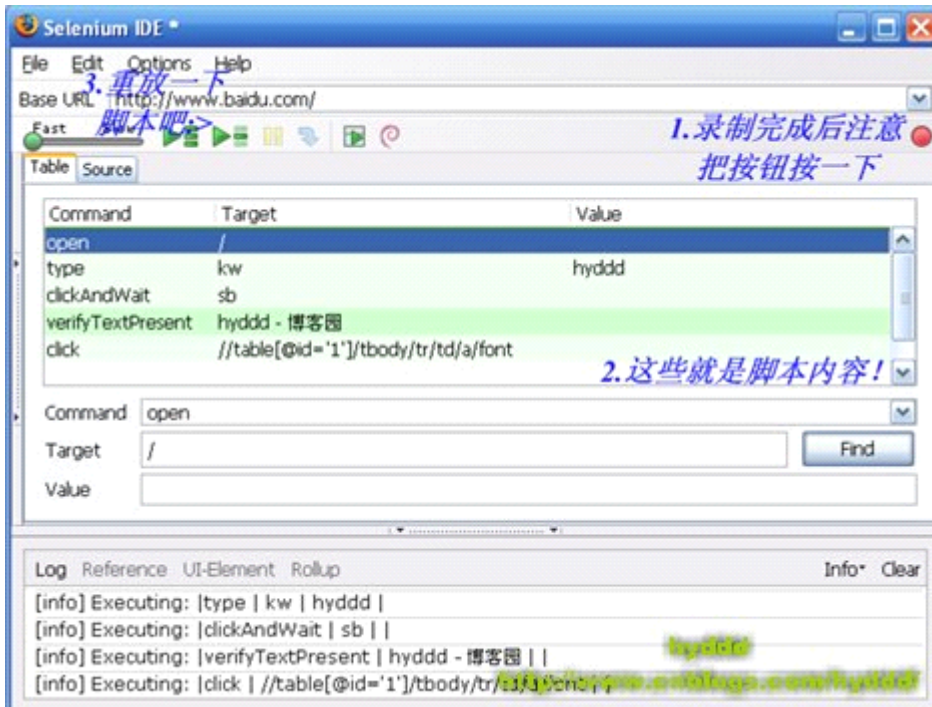




上图例子中，我的操作步骤如下：

- (1) .在地址栏输入：<http://www.baidu.com/>
- (2) .登陆百度首页后，在查询框输入“hyddd”。
- (3) .按“百度一下”按钮
- (4) .进入搜索结果页面后，右键单击第一条记录（即：hyddd - 博客园），在右键弹出菜单中，单击“Verify TestPersent hyddd - 博客园”。
- (5) .单击第一条记录（即：进入 hyddd - 博客园）
- (6) .Firefox 弹出一个新 Tab 页面，并进入了我的博客。

OK，现在看看我们的 Selenium IDE 录制的结果吧：>

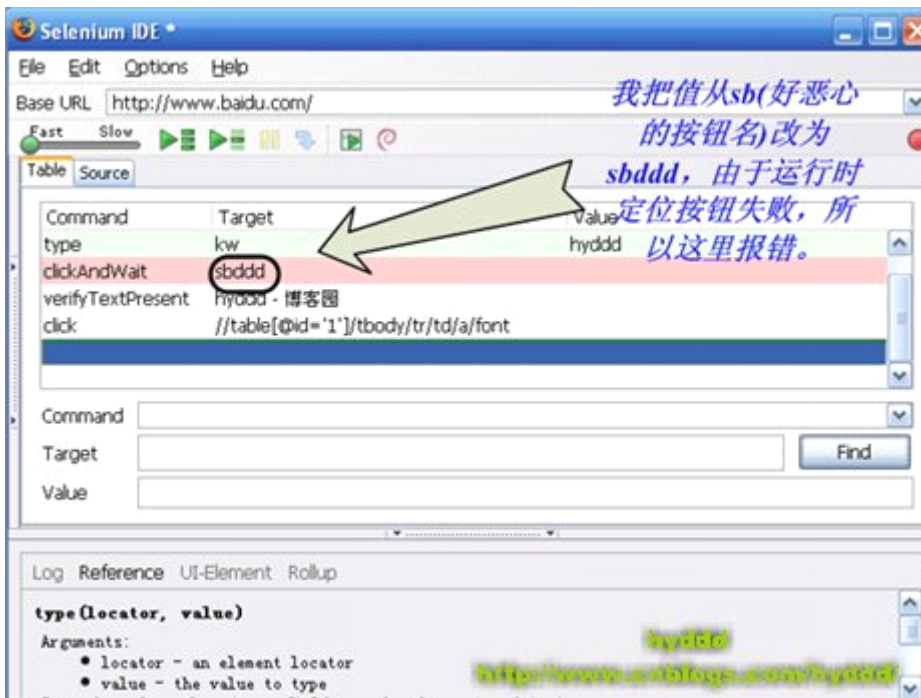


上图中，中间的表格就是录制的结果，你可以按“运行脚本”重新回放脚本看看，值得注意的是，在运行时，Firefox可能会认为脚本中最后一个操作（即：步骤6）为非法弹出框，浏览器会自动阻止其弹出，这个需要设置一下 Firefox，具体位置是：Firefox->Menubar->Tools->options->content->Block pop-up Window，你可以把钩去掉或者在 Exceptions 里面添加相应的网址。

恩，到此为止，脚本录制圆满完成:>

在运行脚本后，你会发现 IDE 表格的颜色发生了变化，运行前，脚本表格为白色，成功运行完毕后，表格为青色，其中还分为深青色和浅青色两种，浅青色表示：动作成功，如：打开网页成功，点击按钮成功等等，而深青色表示：判断正确，如：“hyddd - 博客园”这段文字在页面中存在等等。

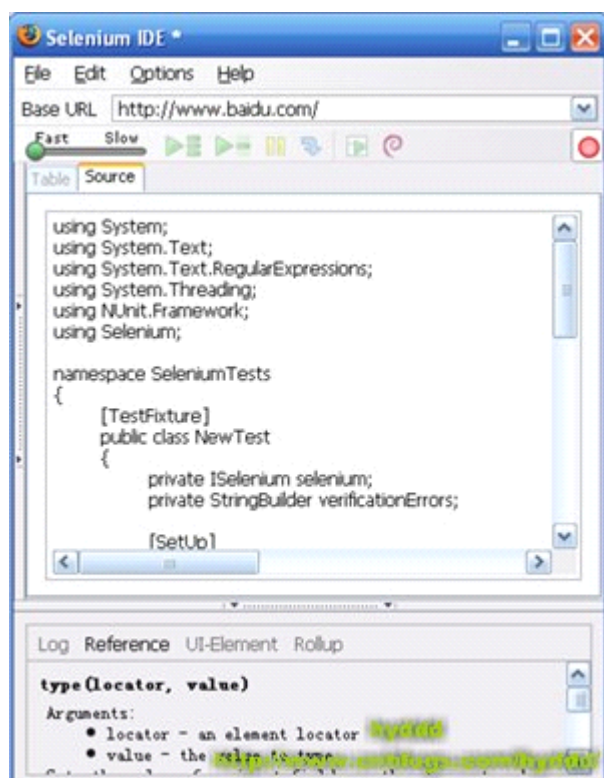
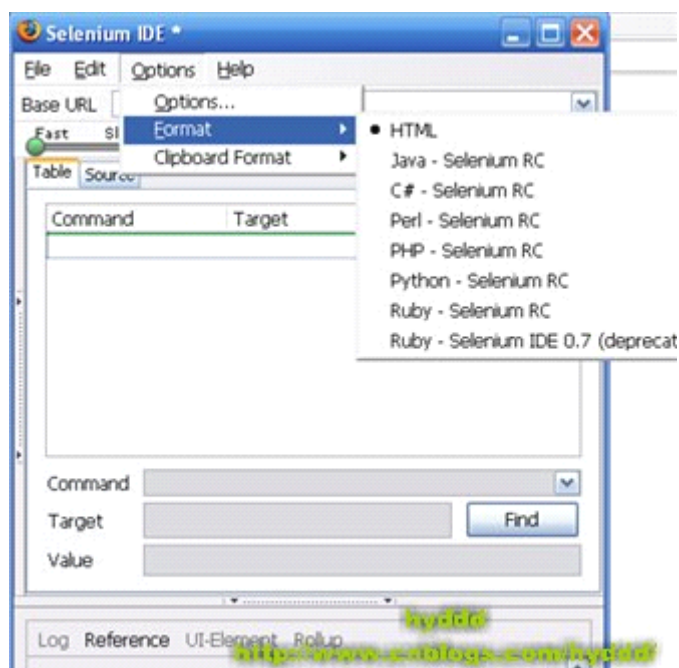
看完正确，现在我们看看出错时的情况吧。



出错时，表格可能会出现两种颜色，一种是浅粉红色，一种是深粉红色。浅粉红色表示判断结果为 false，这种情况案例还是会继续执行下去，判断的失败不会影响案例的运行，深粉红色表示动作失败，如：没有找到按钮等（如上图），这种情况下案例会停止运行。

## 4.Selenium IDE 其他的重要功能

本文开始时提到了，Selenium IDE 还有一个重要的功能就是把脚本的转换，一起来看看吧：>



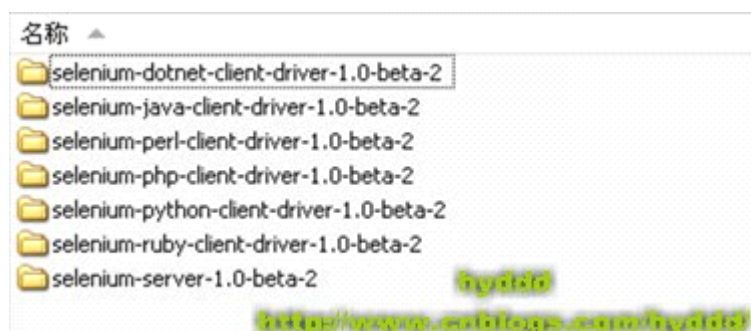
Selenium IDE 可以把 HTML 的脚本转为 C#, JAVA 等等其他语言的脚本, 为我们日后写 Selenium RC 的测试案例提供了极大的方便。

# Selenium 私房菜系列5 -- 第一个 Selenium RC

## 测试案例

《[Selenium 简介](#)》中讲过，Selenium RC 支持多种语言编写测试案例，如：C#，Python。在工作中，我倾向于是用 Python 这类动态语言编写测试案例，因为这样的测试案例无需编译: >，试想如果你有1000个测试案例，每个都要编译，那会给编译服务器很大的压力，而且案例修改后，还得重新编译才能运行:<。但在本系列的文章中，我还是打算使用 C#编写示范例子。

Selenium RC 下载: <http://seleniumhq.org/download/>



## 写 Selenium RC 的测试案例

上一篇《[Selenium IDE 的使用](#)》中，提到了 Selenium IDE 可以把录制的脚本转为其他语言的脚本，所以我继续用上一篇的脚本为例子，下面是把脚本语言转换为 C#后的代码：

```
using System;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading;
using NUnit.Framework;
using Selenium;

namespace SeleniumTests
{
    [TestFixture]
    public class NewTest
    {
        private ISelenium selenium;
        private StringBuilder verificationErrors;

        [SetUp]
        public void SetupTest ()
        {
            selenium = new DefaultSelenium("localhost", 4444, "*chrome", "http://change-this-to-the-site-you-are-testing/");
```



```

        selenium.Start();
        verificationErrors = new StringBuilder();
    }

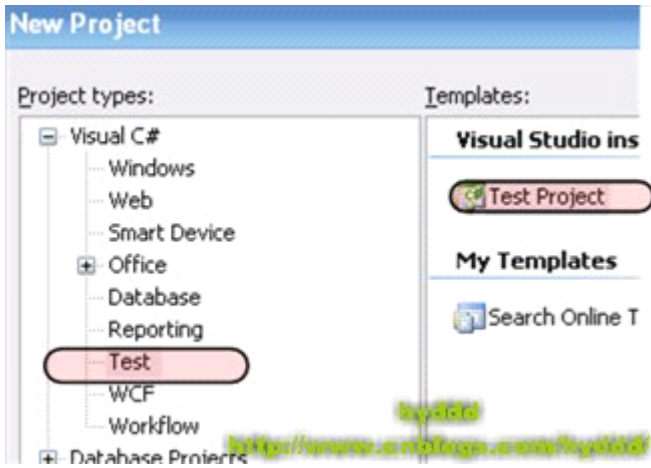
    [TearDown]
    public void TeardownTest()
    {
        try
        {
            selenium.Stop();
        }
        catch (Exception)
        {
            // Ignore errors if unable to close the browser
        }
        Assert.AreEqual("", verificationErrors.ToString());
    }

    [Test]
    public void TheNewTest()
    {
        selenium.Open("/");
        selenium.Type("kw", "hyddd");
        selenium.Click("sb");
        selenium.WaitForPageToLoad("30000");
        try
        {
            Assert.IsTrue(selenium.IsTextPresent("hyddd - 博客园"));
        }
        catch (AssertionException e)
        {
            verificationErrors.Append(e.Message);
        }
        selenium.Click("//table[@id='1']/tbody/tr/td/a/font");
    }
}
}

```

在这里，转换后的脚本使用了 NUnit 测试框架，为了简化，我用 VS 的 Test Project 代替（当然你也可以用 Console Application 建立测试工程的），步骤如下：

## 1. 建立 Test Project



## 2. 导入 DLL 引用

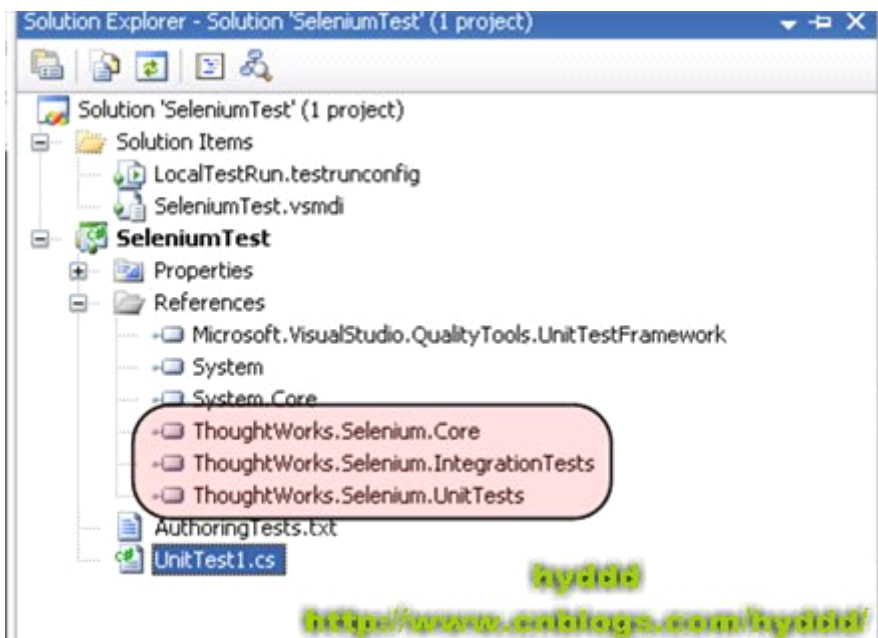
把 selenium-dotnet-client-driver-1.0-beta-2 目录中的

[ThoughtWorks.Selenium.Core.dll](#),

[ThoughtWorks.Selenium.IntegrationTests.dll](#),

[ThoughtWorks.Selenium.UnitTests.dll](#)

加入项目。



## 3. 把上面自动生成的代码再改一下

```
using System;
```

```
using System.Text;
```

```
using System.Collections.Generic;
```



```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Selenium;

namespace SeleniumTest
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Test()
        {
            //127.0.0.1为Selenium 测试服务器位置。
            //4444为Selenium 测试服务器监听端口。
            //*iexplore 为启动浏览器类型，我把它改为了IE浏览器。
            //http://www.baidu.com为源地址。
            ISelenium selenium = new DefaultSelenium("127.0.0.1", 4444, "*iexplore", "http://www.baidu.com");
            selenium.Start();
            selenium.Open("/");
            selenium.Type("kw", "hyddd");
            selenium.Click("sb");
            selenium.WaitForPageToLoad("30000");
            Assert.IsTrue(selenium.IsTextPresent("hyddd - 博客园"));
            selenium.Click("//table[@id='1']/tbody/tr/td/a/font");
            selenium.Close();
            selenium.Stop();
        }
    }
}

```

## 4. 启动 Selenium 测试服务器

打开 cmd 进入 selenium-server-1.0-beta-2 目录，输入“java -jar selenium-server.jar”(需要先安装 JRE)，启动 Selenium 测试服务器。

```

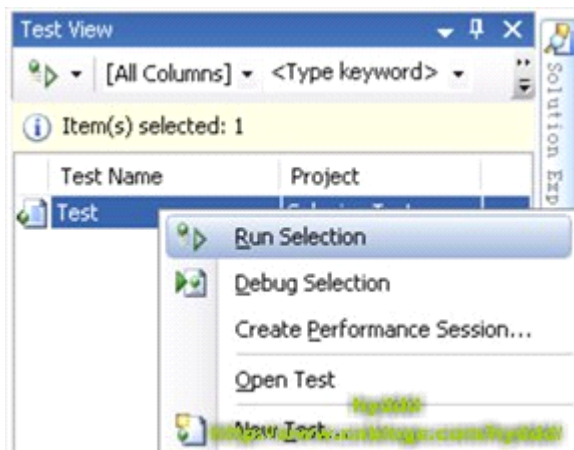
F:\DevTools\Test\Selenium\selenium-remote-control-1.0-beta-2\selenium-server-1.0
-beta-2>java -jar selenium-server.jar
8:20:15.718 INFO - Java: Sun Microsystems Inc. 10.0-b23
8:20:15.734 INFO - OS: Windows XP 5.1 x86
8:20:15.734 INFO - v1.0-beta-2 [2571], with Core v1.0-beta-2 [2330]
8:20:15.921 INFO - Version Jetty/5.1.x
8:20:15.921 INFO - Started HttpContext[/selenium-server/driver,/selenium-server
/driver]
8:20:15.937 INFO - Started HttpContext[/selenium-server,/selenium-server]
8:20:15.937 INFO - Started HttpContext[/,/]
8:20:15.953 INFO - Started SocketListener on 0.0.0.0:4444
8:20:15.953 INFO - Started org.morthay.jetty.Server@52fe85

```

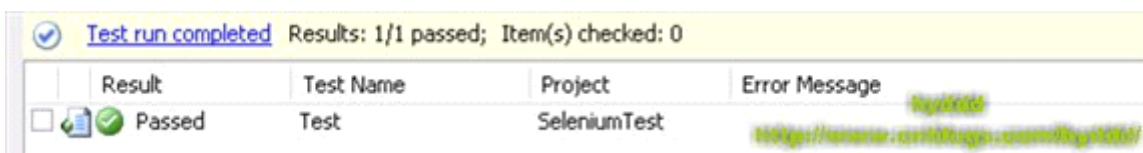
**hyddd**  
<http://www.cnblogs.com/hyddd/>

## 5. 运行测试案例

(1). 运行测试案例:



(2). 测试结果:



恩，案例 Pass 了，如果案例失败的话，Error Message 会说明失败的原因。

(注意:和 Firefox 一样，IE 下也有屏蔽弹出网页功能，修改设置方法: MenuBar->Tools->Popup Blocker->Turn off Popup Blocker，或者在 Popup Blocker Settings 里面配置。)

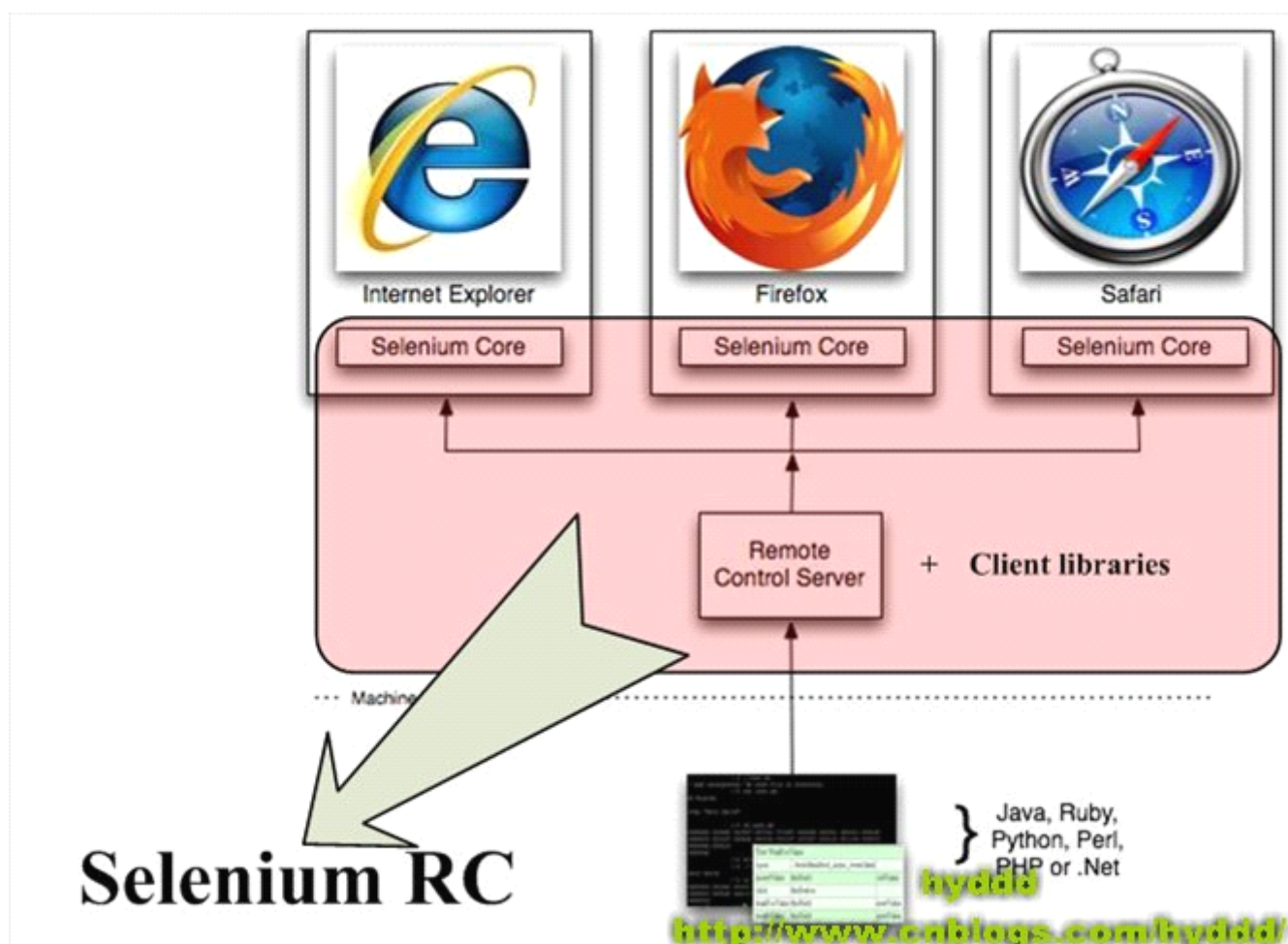
# Selenium 私房菜系列6 -- 深入了解 Selenium

## RC 工作原理(1)

前一篇已经比较详细讲述了如何使用 Selenium RC 进行 Web 测试，但到底 Selenium RC 是什么?或者它由哪几部分组成呢??

### 一.Selenium RC 的组成:

关于这个问题，我拿了官网上的一幅图来说明这个问题。



Selenium RC 主要由两部分组成:

#### (1) .Selenium Server:



Selenium Server 负责控制浏览器行为，总的来说，Selenium Server 主要包括3个部分：**Launcher**、**Http Proxy**、**Selenium Core**。其中 Selenium Core 是被 Selenium Server 嵌入到浏览器页面中的。其实 Selenium Core 就是一堆 JS 函数的集合，就是通过这些 JS 函数，我们才可以实现用程序对浏览器进行操作。

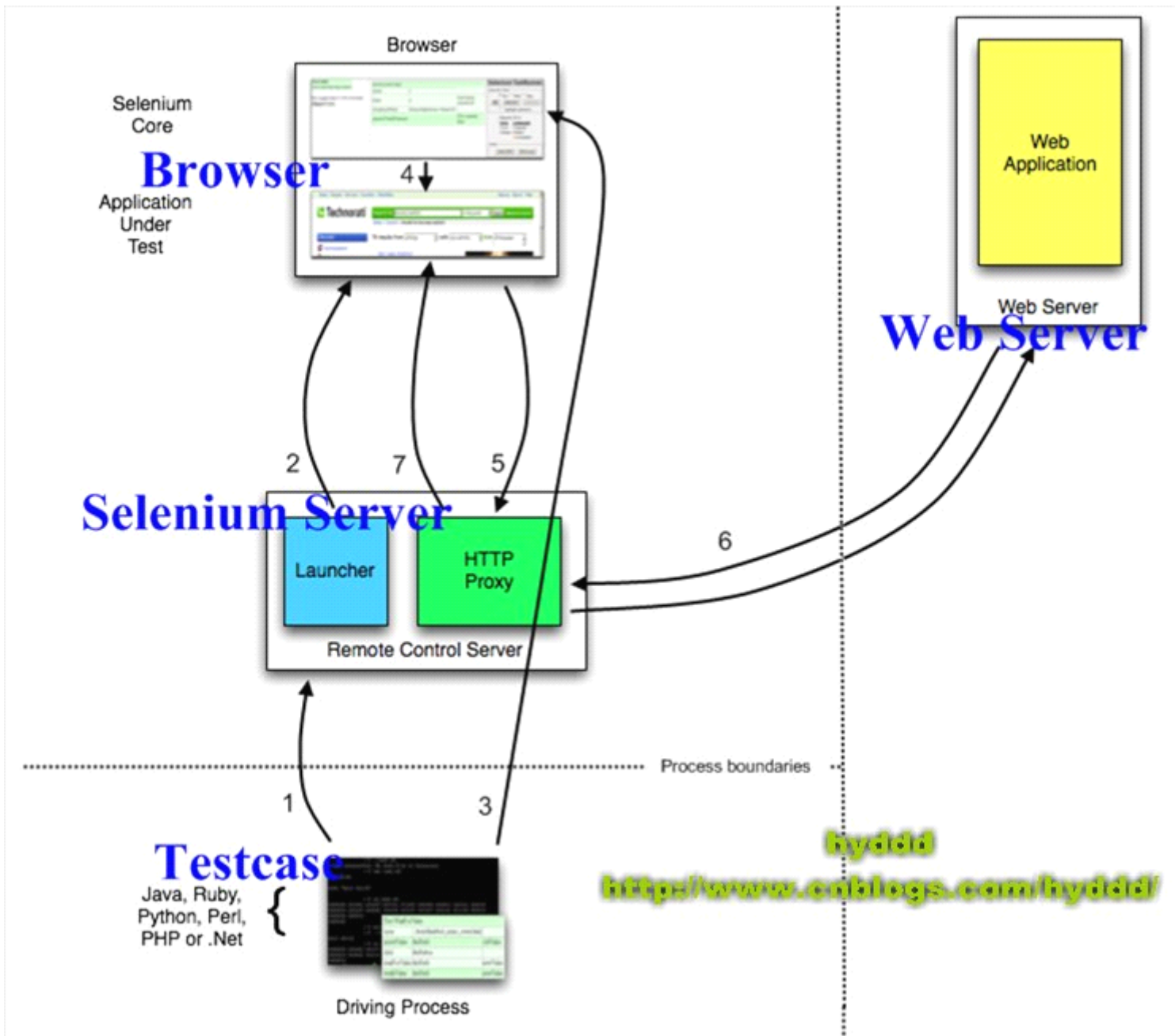
## (2) .Client Libraries:

写测试案例时用来控制 Selenium Server 的库。



## 二.Selenium RC 与 Testcase 的关系

先看下图：



(1).测试案例（Testcase）通过 Client Lib 的接口向 Selenium Server 发送 Http 请求，要求和 Selenium Server 建立连接。

为什么要通过发送 Http 请求控制 Selenium Server 而不采用其他方式呢？从上文可以看出，Selenium Server 是一个独立的中间服务器（确切地说是代理服务器），它可以架设在其他机器上！所以测试案例通过发送 HTTP 请求去控制 Selenium Server 是很正常的。

(2).Selenium Server 的 Launcher 启动浏览器，把 Selenium Core 加载入浏览器页面当中，并把浏览器的代理设置为 Selenium Server 的 Http Proxy。

(3).测试案例通过 Client Lib 的接口向 Selenium Server 发送 Http 请求，Selenium Server 对请求进行解析，然后通过 Http Proxy 发送 JS 命令通知 Selenium Core 执行操作浏览器的动作。

(4).Selenium Core 接收到指令后，执行操作。

(5).浏览器收到新的页面请求信息（因为在(4)中，Selenium Core 的操作可能引发新的页面请求），于是发送 Http 请求，请求新的 Web 页面。

由于 Selenium Server 在启动浏览器时做了手脚，所以 Selenium Server 会接收到所有由它启动的浏览器发送的请求。

(6).Selenium Server 接收到浏览器的发送的 Http 请求后，自己重组 Http 请求，获取对应的 Web 页面。

(7).Selenium Server 的 Http Proxy 把接收的 Web 页面返回给浏览器。

为什么 Selenium RC 中的 Selenium Server 需要以这种代理服务器的形式存在呢？下一篇继续介绍: >

# Selenium 私房菜系列7 -- 深入了解 Selenium

## RC 工作原理(2)

继续上一篇的问题，为什么 Selenium RC 中的 Selenium Server 需要以这种代理服务器的形式存在?其实，这和浏览器的“同源策略”(The Same Origin Policy)有关。

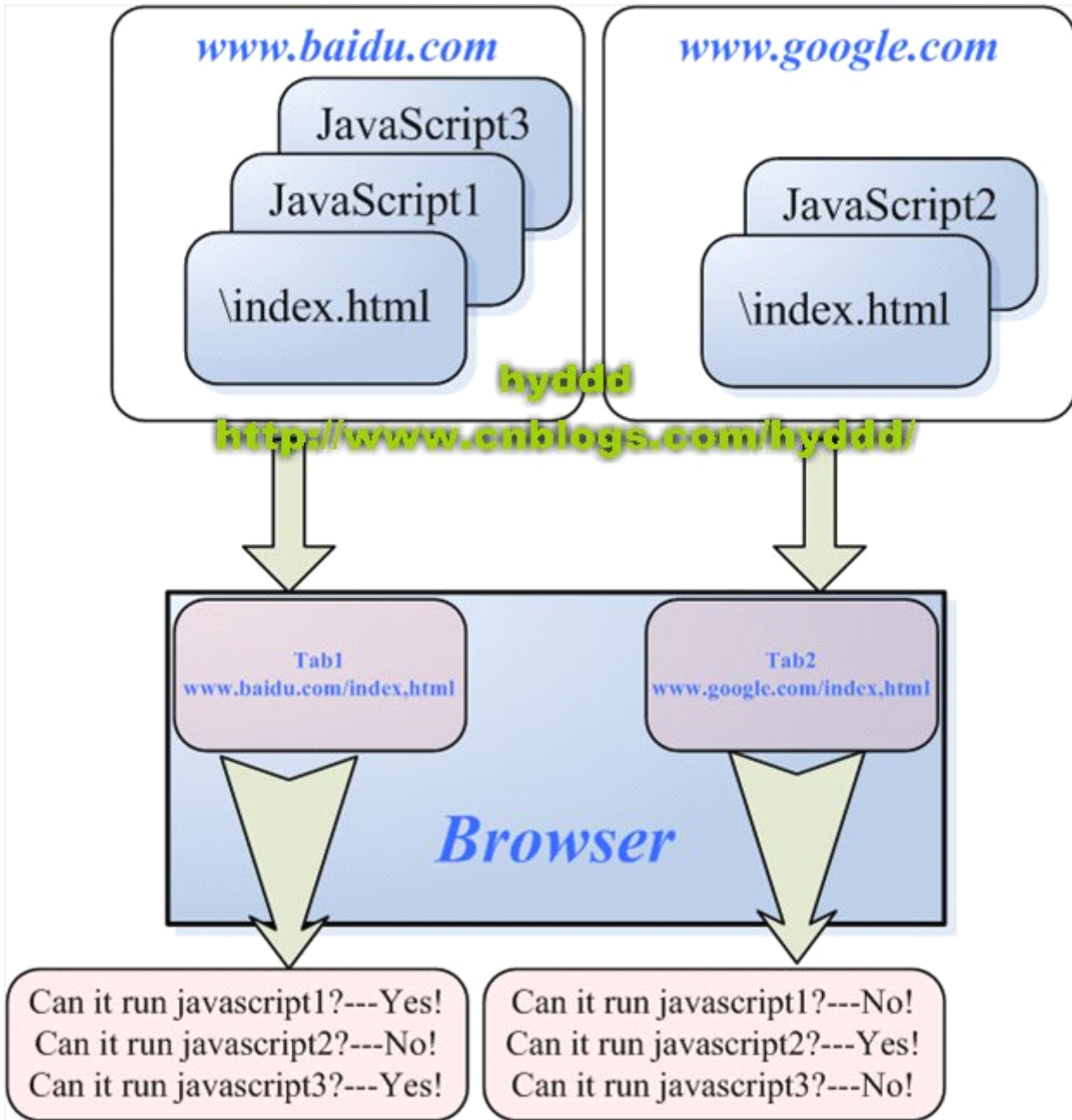
### 一.什么是同源策略

**同源策略**，它是由 Netscape 提出的一个著名的安全策略，现在所有的可支持 javascript 的浏览器都会使用这个策略。

**为什么需要同源策略**，这里举个例子：

假设现在没有同源策略，会发生什么事情呢？大家知道，JavaScript 可以做很多东西，比如：读取/修改网页中某个值。恩，你现在打开了浏览器，在一个 tab 窗口中打开了银行网站，在另外一个 tab 窗口中打开了一个恶意网站，而那个恶意网站挂了一个的专门修改银行信息的 JavaScript，当你访问 这个恶意网站并且执行它 JavaScript 时，你的银行页面就会被这个 JavaScript 修改，后果会非常严重！而同源策略就为了防止这种事情发生， 看下图：



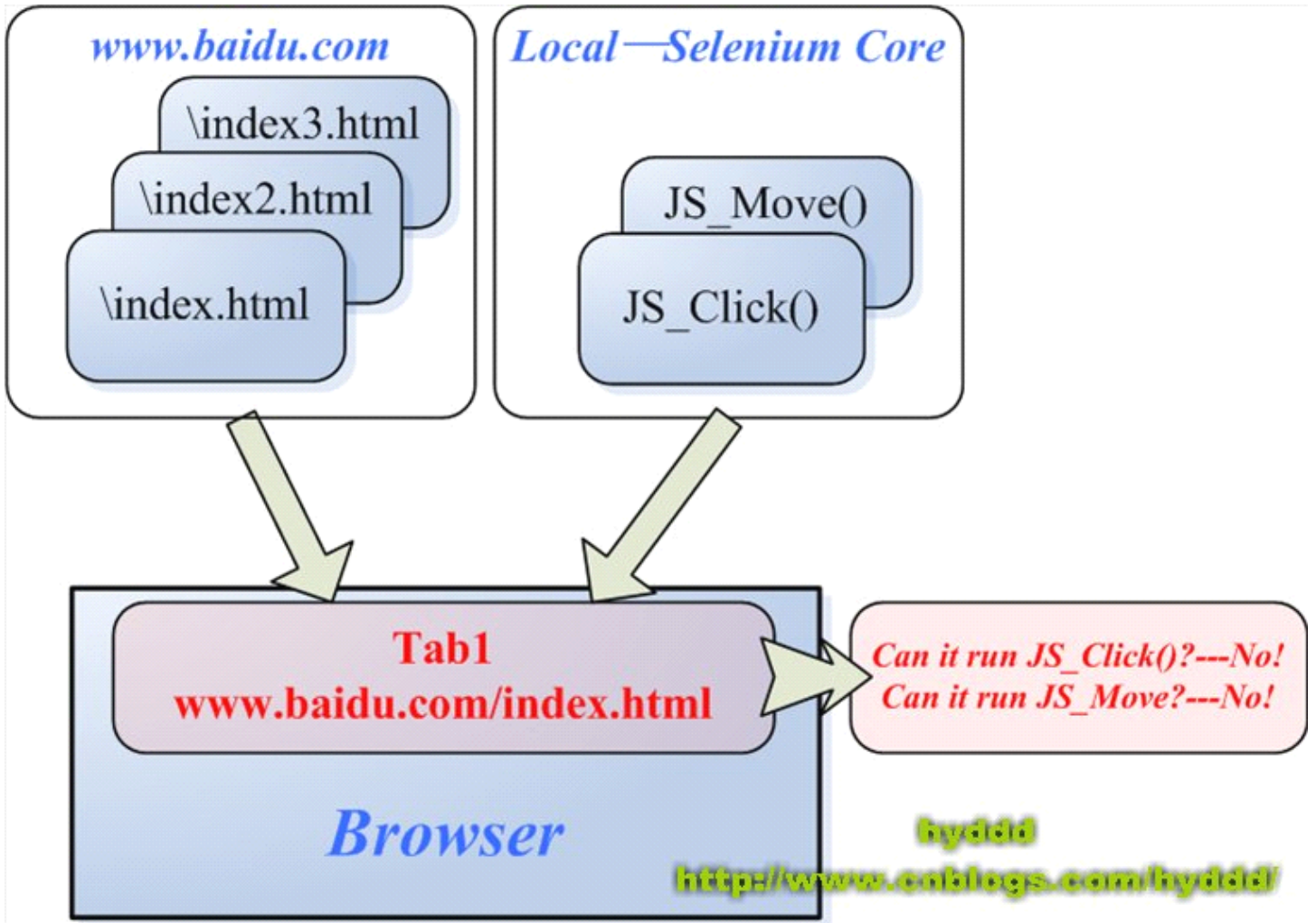


比如说，浏览器的两个 tab 页中分别打开了 <http://www.baidu.com/index.html> 和 <http://www.google.com/index.html>，其中，JavaScript1和 JavaScript3是属于百度的脚本，而 JavaScript2是属于谷歌的脚本，当浏览器的 tab1要运行一个脚本时，便会进行同源检查，只有和 `www.baidu.com` 同源的脚本才能被执行，所谓同源，就是指域名、协议、端口相同。所以，tab1只能执行 JavaScript1和 JavaScript3脚本，而 JavaScript2不能执行，从而防止其他网页对本网页的非法篡改。

## 二.Selenium Server 为什么以这种代理服务器的形式存在

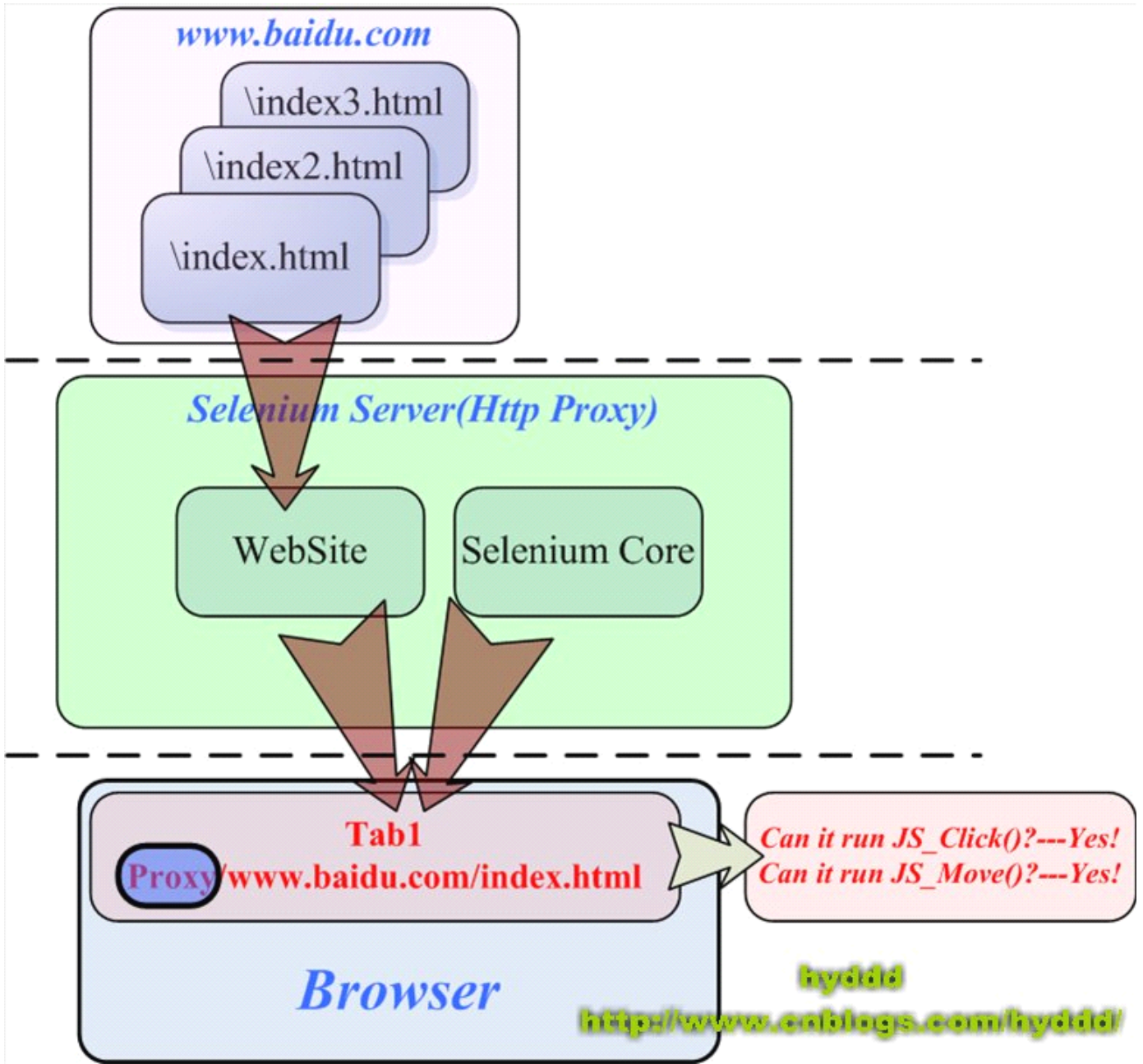
上面说了同源策略，那同源策略的 Selenium Server 有什么关系呢？呵呵，上一篇说过，Selenium Core 是一堆 JS 函数的集合，它是我们操作浏览器的基础。当存在同源策略时，便出现一些问题，看下图：





因为 Selenium Core 的 JS 脚本的“源”是 localhost，所以浏览器会阻止 Selenium Core 的 JS 脚本在测试页面上执行，这就是为什么在本系列第一篇中说，如果只使用 Selenium Core 进行测试，需要把 Selenium Core 安装到远程服务器上。

为了解决上面这个问题，Selenium RC 中的 Selenium Server 就以代理服务器的形式出现了，下图说明它是如何借助代理的身份蒙骗浏览器的: >



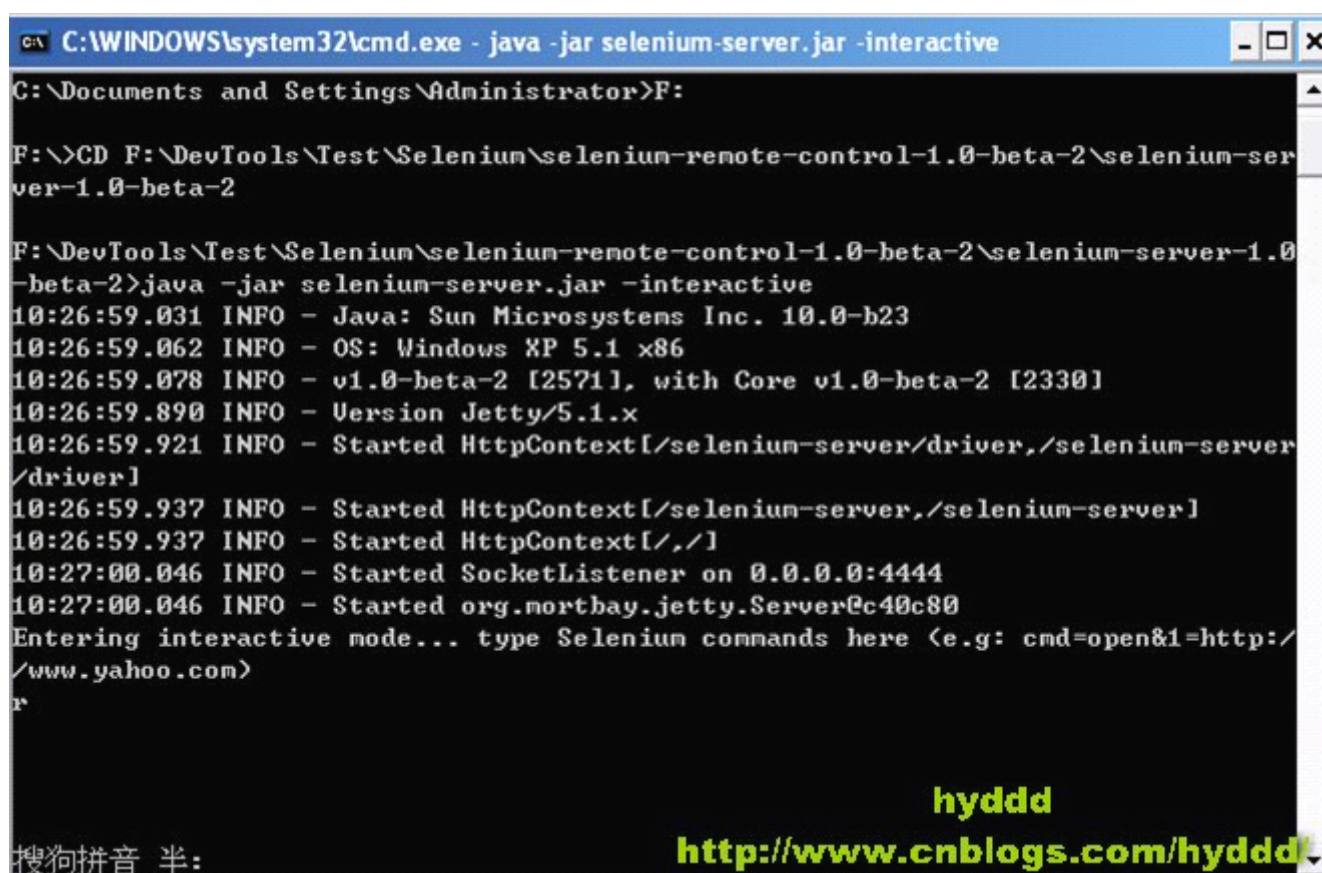
Selenium Server 以代理的形式存在，通过修改 WebSite 的源信息，从而达到欺骗浏览器的目的，就这样，Selenium RC 就轻松绕过了同源策略。在上图中，浏览器会认为 WebSite 和 Selenium Core 来自同一个“源”----代理服务器！

# Selenium 私房菜系列8 -- 玩转 Selenium Server

本篇主要是想更进一步介绍 Selenium Server 的工作原理，这次我们从 Selenium Server 的交互模式开始。

在《[第一个 Selenium RC 测试案例](#)》中，我们以命令“java -jar selenium-server.jar”启动了 Selenium Server，其实在启动 Selenium Server 时，我们还可以加上各种参数（具体的参数请参考《[Selenium RC 服务器命令行参数列表](#)》），而开启 Selenium Server 交互模式的命令为“**java -jar selenium-server.jar -interactive**”。交互模式，是 Selenium Server 提供了一种快速的测试方法，你可以对 Selenium Server 输入命令从而直接启动测试。

## 1.启动 Selenium Server 交互模式



```
C:\WINDOWS\system32\cmd.exe - java -jar selenium-server.jar -interactive
C:\Documents and Settings\Administrator>F:
F:\>CD F:\DevTools\Test\Selenium\selenium-remote-control-1.0-beta-2\selenium-server-1.0-beta-2
F:\DevTools\Test\Selenium\selenium-remote-control-1.0-beta-2\selenium-server-1.0-beta-2>java -jar selenium-server.jar -interactive
10:26:59.031 INFO - Java: Sun Microsystems Inc. 10.0-b23
10:26:59.062 INFO - OS: Windows XP 5.1 x86
10:26:59.078 INFO - v1.0-beta-2 [2571], with Core v1.0-beta-2 [2330]
10:26:59.890 INFO - Version Jetty/5.1.x
10:26:59.921 INFO - Started HttpContext[/selenium-server/driver,/selenium-server/driver]
10:26:59.937 INFO - Started HttpContext[/selenium-server,/selenium-server]
10:26:59.937 INFO - Started HttpContext[/,/]
10:27:00.046 INFO - Started SocketListener on 0.0.0.0:4444
10:27:00.046 INFO - Started org.mortbay.jetty.Server@c40c80
Entering interactive mode... type Selenium commands here (e.g: cmd=open&1=http://www.yahoo.com)
r
hyddd
http://www.cnblogs.com/hydd
```

2.在命令行中输入：**cmd=getNewBrowserSession&1=\*iexplore&2=http://www.google.com**。控制 Selenium Server 启动浏览器，以及创建 Session。



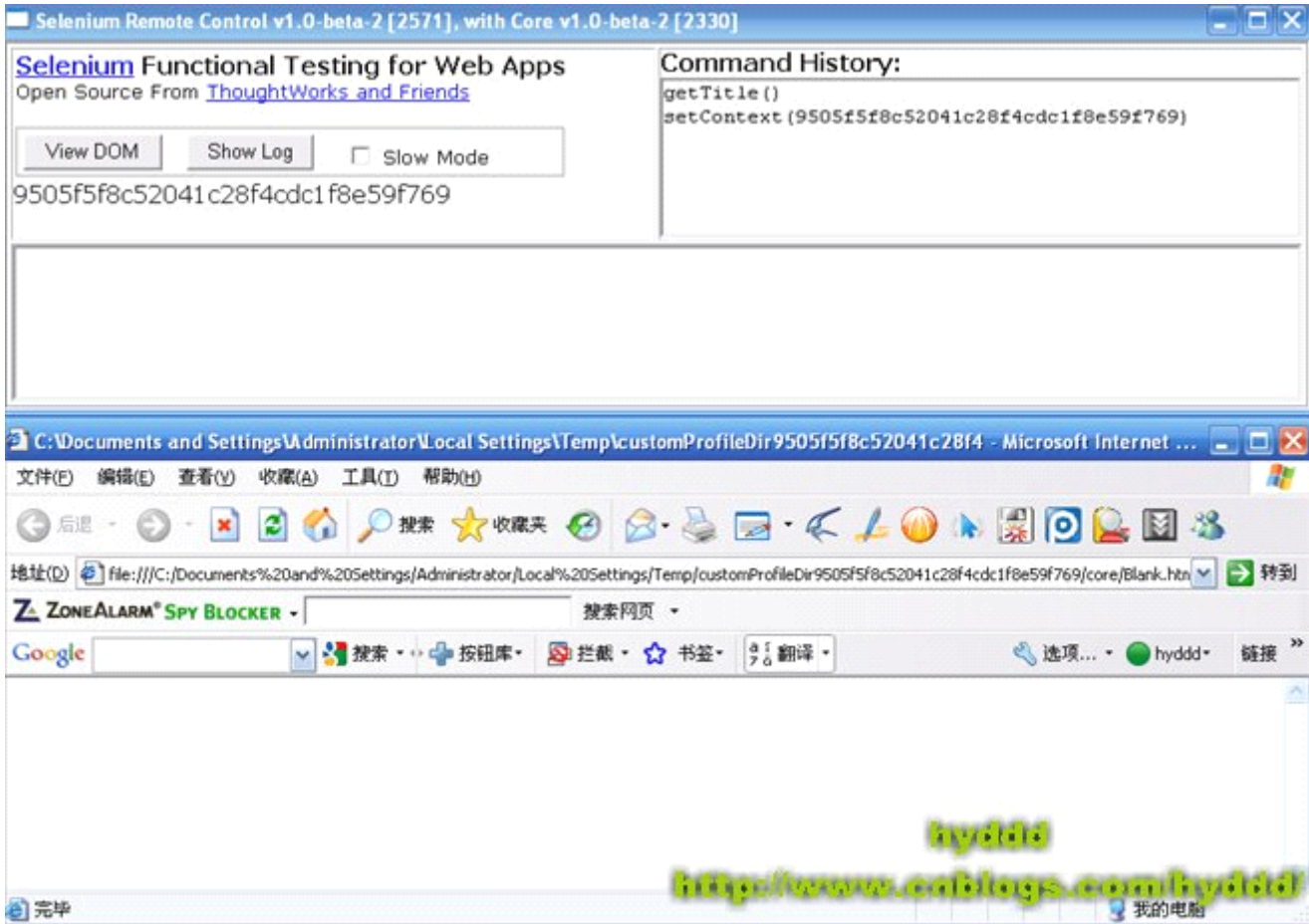
```
C:\WINDOWS\system32\cmd.exe - java -jar selenium-server.jar -interactive
10:33:42.500 INFO - v1.0-beta-2 [2571], with Core v1.0-beta-2 [2330]
10:33:42.625 INFO - Version Jetty/5.1.x
10:33:42.640 INFO - Started HttpContext[/selenium-server/driver]
10:33:42.640 INFO - Started HttpContext[/selenium-server,/selenium-server]
10:33:42.640 INFO - Started HttpContext[/,/]
10:33:42.640 INFO - Started SocketListener on 0.0.0.0:4444
10:33:42.656 INFO - Started org.mortbay.jetty.Server@52fe85
Entering interactive mode... type Selenium commands here (e.g: cmd=open&1=http://www.yahoo.com)
cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com
10:33:54.671 INFO - ---> Requesting http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com
10:33:54.796 INFO - Checking Resource aliases
10:33:54.796 INFO - Command request: getNewBrowserSession[*iexplore, http://www.google.com] on session null
10:33:54.796 INFO - creating new remote session
10:33:54.968 INFO - Allocated session cf1a0a202e84468c9b63e76f621f80da for http://www.google.com, launching...
10:33:55.750 INFO - Launching Embedded Internet Explorer...
10:33:56.906 INFO - Launching Internet Explorer HTA...
10:34:02.781 INFO - Got result: OK,cf1a0a202e84468c9b63e76f621f80da on session cf1a0a202e84468c9b63e76f621f80da
搜狗拼音 半:
```

( 1 ) .---> Requesting [http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=\\*iexplore&2=http://www.google.com](http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com)

看过《[深入了解 Selenium RC 工作原理\(1\)](#)》的应该了解：我们所编写的测试案例，其实是通过发送 Http 请求实现对 Selenium Server 的控制，而测试案例所发送的请求就正是： ---> Requesting [http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=\\*iexplore&2=http://www.google.com](http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com)。我们可以再打开一个 IE 浏览器，在地址栏输入：[http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=\\*iexplore&2=http://www.google.com](http://localhost:4444/selenium-server/driver?cmd=getNewBrowserSession&1=*iexplore&2=http://www.google.com)，回车！看，Selenium Server 又为此产生了一个 Session 了！呵呵：>

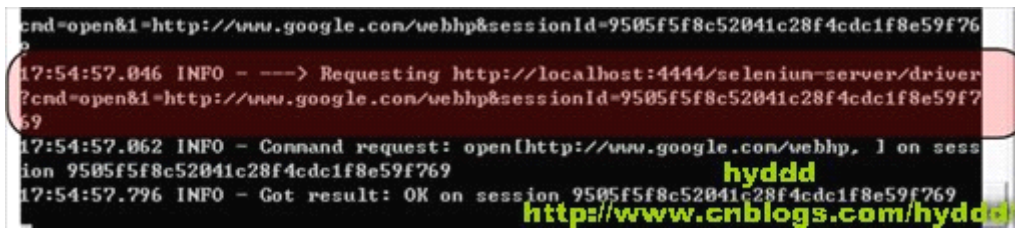
(2) .这里，Selenium Server 为上面的请求随机生成了一个 Session ID：9505f5f8c52041c28f4cdc1f8e59f769（由于写这篇文章的时候中途重启了 Selenium Server，所以这里和上图的 Session ID 不同，并且下文会继续使用 Session ID：9505f5f8c52041c28f4cdc1f8e59f769）。

(3) .如果一切正常，Selenium Server 最后会出现 Get Result Ok 的字样，并出现如下两个框框：



3.控制浏览器访问 [www.google.com/webhp](http://www.google.com/webhp)，输入：

`cmd=open&1=http://www.google.com/webhp&sessionId=9505f5f8c52041c28f4cdc1f8e59f769`



噢，浏览器成功访问 <http://www.google.com/webhp> 了:>。

总结一下：

(1) .在 Selenium Server 中输入命令的格式为：`cmd=Command&1=Target&2=Value&SessionID=...`，这和 Selenium IDE 的案例语句很像。

(2) .在输入命令后，Selenium Server 会发条 Http 请求给自己，请求的 URL 格式也是固定的：`http://localhost:4444/selenium-server/driver?cmd=Command&1=Target&2=Value&SessionID=...`，我们完全可以用浏览器发送请求控制 Selenium Server 进行测试。

(3) .另外，sessionId 是很重要的一个参数，当一个 Selenium Server 同时运行多个测试案例时，Selenium Server 就是通

过 sessionId 判断到底该操作哪个浏览器窗口。而在下面的 C#代码中:

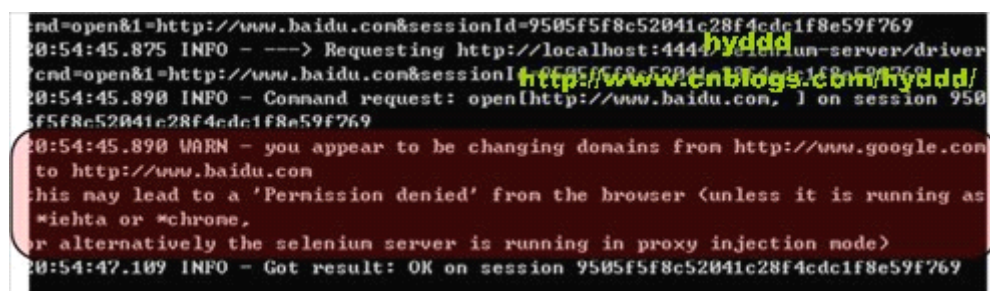
```
ISelenium selenium = new DefaultSelenium("127.0.0.1", 4444, "*iexplore",  
"http://www.google.com");  
selenium.Start();  
selenium.Open("/webhp");
```

selenium 就相当于上文中的 sessionId。

(4) 在 Selenium Server 启动一个 Session 时, 必须先指定一个“源”(原因见《[深入了解 Selenium RC 工作原理\(2\)](#)》)。在上面的代码中 <http://www.google.com> 就是“源”了, 然而这是可能出现问题, 请看下面代码:

```
ISelenium selenium = new DefaultSelenium("127.0.0.1", 4444, "*iexplore",  
"http://www.google.com");  
selenium.Start();  
selenium.Open(http://www.baidu.com);
```

我们在启动 Session 时, 定义了源为 <http://www.google.com>, 但在后来的操作中, 我们打开的却是 <http://www.baidu.com>. 由于二者非同源, 所以接下来的操作就可能会出现各种问题, 故此 Selenium Server 会给出以下警告:



Selenium Server 提示说: 如果测试案例是运行在\*iehta 或者\*chrome 上, 或者改变 Selenium Server 的运行模式为 proxy injection mode 即可避免问题出现。

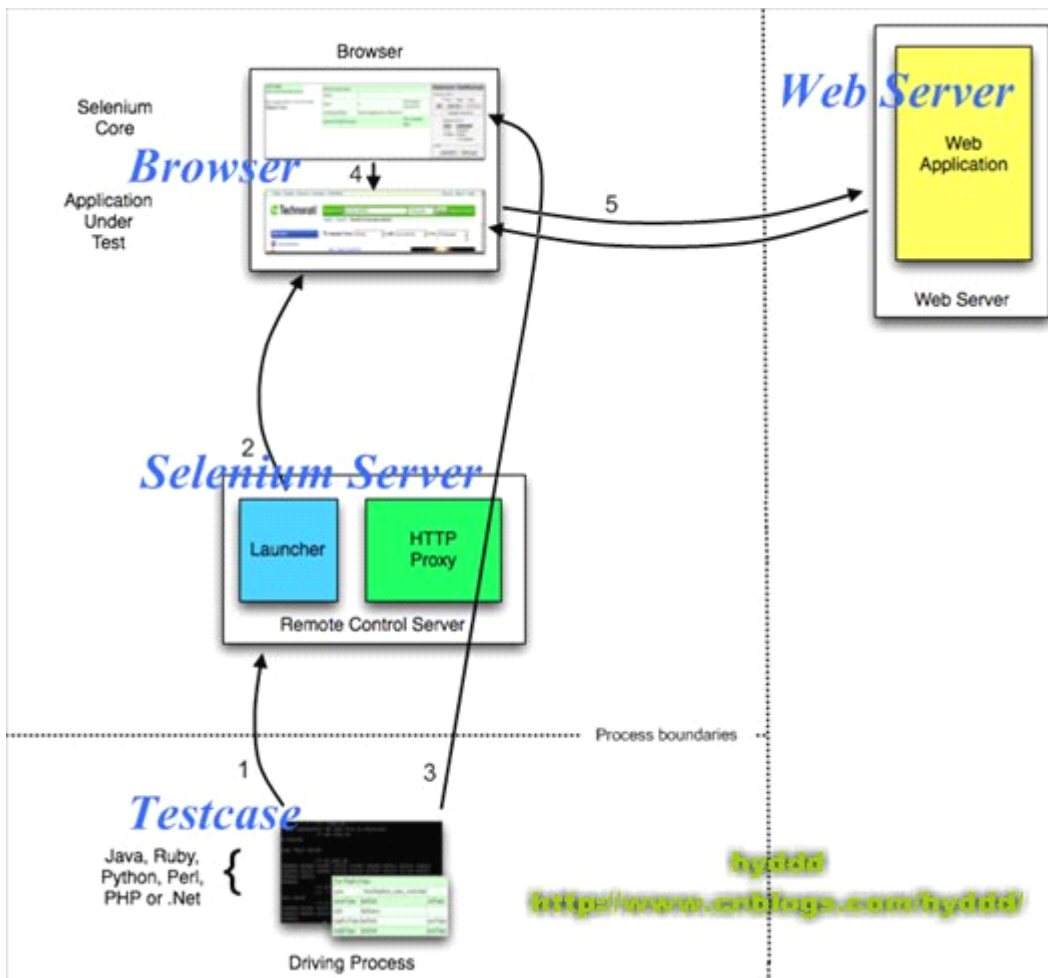
恩, 在这里, 我不得不承认之前在《[深入了解 Selenium RC 工作原理\(1\)](#)》中, 为了简化问题, 我故意少写了一些东西!

其实, Selenium Server 其实有2种运行模式:

### (1). Heightened Privileges Browsers

### (2). Proxy Injection

现在 Selenium Server 启动的默认模式为: Heightened Privileges Browsers。如果要启动 Proxy Injection 模式, 可以加参数“-proxyInjectionMode”。而之前在《[深入了解 Selenium RC 工作原理\(1\)](#)》中介绍 Selenium RC 与 Testcase 关系, 其实就是在描述 **Proxy Injection** 的工作模式, 因为我个人认为 Proxy Injection 设计模式更为合理, 所以只对 Proxy Injection 模式作介绍。在这里我补充说明一下, 为什么 Heightened Privileges Browsers 模式不能避免上面的问题。先看看 Selenium Server 在 **Heightened Privileges Browsers** 模式下的工作流程图:



和 Proxy Injection 模式不一样，在 Heightened Privileges Browsers 模式下，Browser 对 Web 的请求没有经过 Http Proxy，所以返回的 Web 页面就有可能和 Selenium Core 不同源了(一般的情况下，Open 都应该只获取“源”下的某个子页面，用 Open 获取其他“源”的页面在测试中应该是极少数的，因为在这种情况下，我们还应该再 new 一个新的 ISelenium selenium 进行处理，虽然用 Open 获取其他“源”的页面出现的机会极少，但如果真的需要这样的话，也只能启动 - proxyInjectionMode 模式了，虽然这样的效率会低一些)。



# Selenium 私房菜系列9 -- Selenium RC 服务器命

## 令行参数列表【ZZ】

本文转载自：<http://wiki.javascud.org/display/SEL/Selenium+Remote+Control+-+options>

使用示例：`java -jar selenium-server.jar [-interactive] [options]`

- **-port <nnnn>**: selenium 服务器使用的端口号（默认 4444）
- **-timeout <nnnn>**: 我们放弃前（超时）所等待的秒数
- **-interactive**: 进入交互模式。参考[教程](#)获取更多信息
- **-multiWindow**: 进入被测试网站都在单独窗口打开的模式，并且 selenium 支持 frame
- **-forcedBrowserMode <browser>**: 设置浏览器模式（例如，所有的会话都使用 `"*iexplore"`，不管给 `getNewBrowserSession` 传递什么参数）
- **-userExtensions <file>**: 指定一个被载入到 selenium 的 JavaScript 文件
- **-browserSessionReuse**: 停止在测试间重新初始化和替换浏览器。
- **-alwaysProxy**: 默认情况下，我们尽量少的进行代理；设置这个标志将会强制所有的浏览器通讯都通过代理
- **-firefoxProfileTemplate <dir>**: 一般情况，我们在每次启动之前都生成一个干净的 Firefox 设置。您可以指定一个目录来让我们将您的设置拷贝过来，代替我们生成的。
- **-debug**: 进入 debug 模式，会有更多的跟踪调试信息
- **-htmlSuite <browser> <startURL> <suiteFile> <resultFile>**: 使用指定的浏览器（例如 `"*firefox"`）在指定的 URL（例如 `"http://www.google.com"`），运行一个单独的 HTML Selenese (Selenium Core) 测试套件然后立即退出。您需要指定 HTML 测试套件的绝对路径还有我们将会生成的 HTML 测试结果文件的路径。
- **-proxyInjectionMode**: 进入代理注入模式，这个模式中 selenium 服务器作为进入测试程序的所有内容的代理服务器。在这个模式下，可以跨多个域访问，并且还支持如下附加参数：
  - **-dontInjectRegex <regex>**: 附加的正则表达式，代理注入模式能够使用它决定是否进行注入
  - **-userJsInjection <file>**: 指定一个 JavaScript 文件，将它注入到所有页面中
  - **-userContentTransformation <regex> <replacement>**: 一个正则表达式，对所有被测 HTML 内容进行匹配；第二个 string 将会对替换所有匹配的内容。这个标志能够使用多次。一个简单的适合使用这个参数的例子：如果你添加 `"-userContentTransformation https http"` 那么测试应用程序的 HTML 中的所有 `"https"` 字符串都会被替换为 `"http"`。

我们还支持两种 Java 系统属性：**-Dhttp.proxyHost** 和 **-Dhttp.proxyPort**。使用 Selenium 服务器作为代理服务器，Selenium RC 一般重载你的代理服务器配置。使用这个参数适合在使用 Selenium 服务器代理的同时使用你自己的代理服务器。使用代理服务器时这样配置：

```
java -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=1234 -jar selenium-server.jar
```

如果你的 HTTP 代理服务器需要验证，你还可以在 `http.proxyHost` 和 `http.proxyPort` 后面设置 **-Dhttp.proxyUser** 和 **-Dhttp.proxyPassword**。

```
java -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=1234 -Dhttp.proxyUser=joe -Dhttp.proxyPassword=example -jar selenium-server.jar
```