Document for s-tea Automation Framework

About S-tea 关于 s-tea

S-tea 是一个开源的自动化测试框架，它提供的是更加优化的 api，封装的 selenium2 的框架。对于 selenium 的面向过程性，我们封装了一些特性，使 s-tea 可以更加好的支持面向对象的方式来使用。这样对于我们后期维护脚本提供了更多的可能性。
S-tea 的 http 地址：www..github.com/celeskyking/s-tea
在 github 上我们可以找到最新的源码，并且一些配置文件我们都可以及时的获取到。下面会讲述 s-tea 的环境搭建。

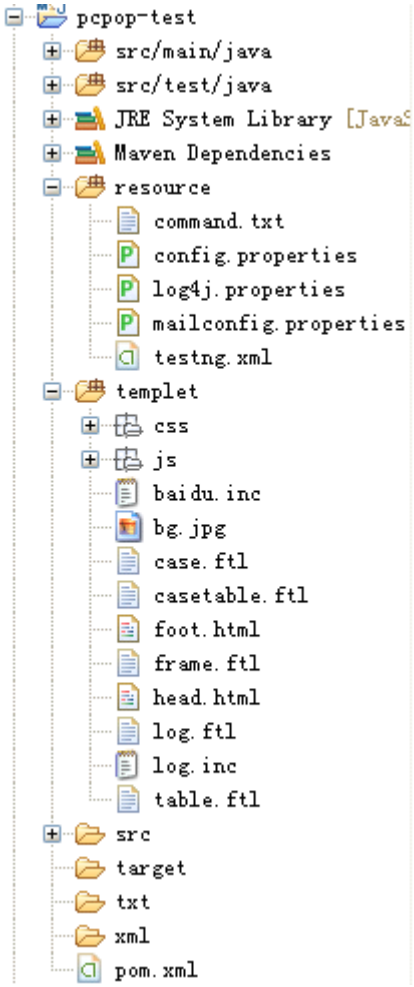
Install And Environment 安装和环境搭建

编程环境搭建

（先参考：ppt 文件 jdk 和 maven 的安装）
S-tea 现在已经被上传到 maven 的中央仓库，现在我们如果会使用 maven 的话直接在本地构建项目就可以。下面会给出指定的 maven 依赖代码。

```
<dependency>
  <groupId>com.github.celeskyking</groupId>
  <artifactId>s-tea</artifactId>
  <version>1.0.0</version>
  <scope>test</scope>
</dependency>
```

通过引用这些引用我们就可以加入 s-tea 的 jar 的引用了，但是这个时候我们还不能够正确的使用它，我们需要加入一些配置文件，这些文件我们能够在 www.github.com/celeskyking/s-tea 这个网址内下载到，需要的配置文件就是 resource，templet 加入到项目中，并且需要我们创建 xml 和 txt 两个文件夹。配置文件暂时使用默认的就行。创建完整的项目就是大概下面的结构。



自动化实施环境搭建

在上面的环境中我们只能够用来在本地编写自动化的脚本，但是我们的环境是需要构建在 jenkins 上面的，并且通过 git 来进行版本控制和源码管理，所以我们需要把本地通过 git 来和服务器端进行构建实现。

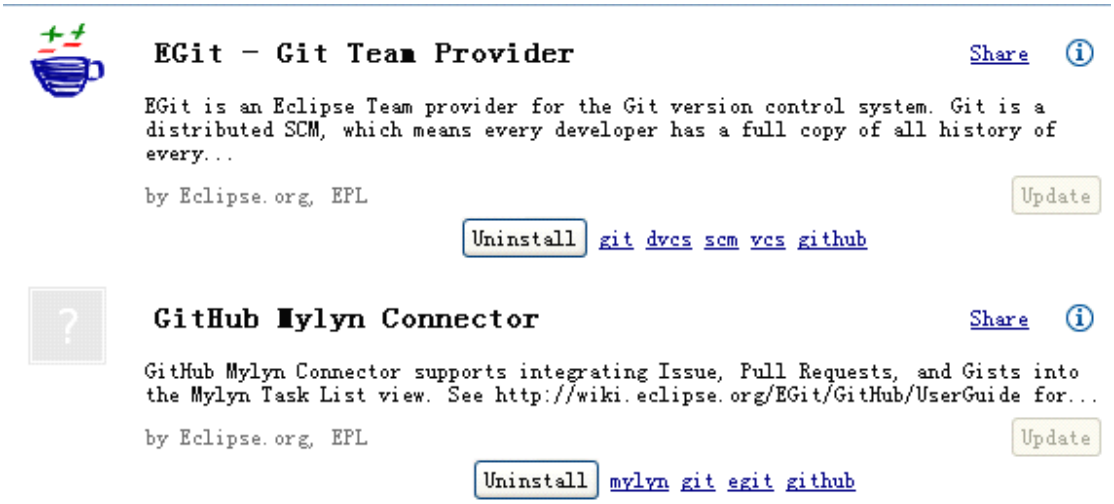
- Git 的安装

因为我们的工作环境都在 eclipse 中实现，所以我们不再提供 pc 端的 git 环境搭建，我们只需要在 eclipse 构建 git 客户端就可以了。

在 Eclipse 中选择 help->eclipse marketplace 然后等待打开 eclipse 市场。

在搜索框中输入 git 搜索，会出现很多的插件，我们选择下图中标出的两个进行安装

注意： 在安装过程中可能只安装一个会出现错误，因为依赖不够的原因，我们只需要 back，选择第二个插件，点击 install 就可以了，这样就会加载两个插件了，然后一直 next 就可以安装完成。



- Git 的使用

在我们使用 git 之前，我们必须通过管理员提供足够的权限，这个时候需要我们在本地创建 ssh 的密钥。为了方便我们创建密钥，我们需要装 ssh 的客户端，或者安装 git 的客户端。具体安装过程我们不再提供，百度或者 google 有很多的教程。我提供一种 git 客户端的方式，下载地址：http://msysgit.github.io/

我们只需要默认安装就可以了，它的使用很简单，如果使用过 linux 系统的话就会明白它的使用方式，它只需要打开一个 git bash 就可以在命令行中进行密钥的生成。生成密钥之前，我们需要确保${user.home}的目录下面存在.ssh 文件夹，如果不存在，我们可以在 cmd 中存在，打开 cmd，然后输入 mkdir .ssh 就可以创建了。然后我们在 git bash 中输入如下命令：

```
ssh-keygen -t rsa
```

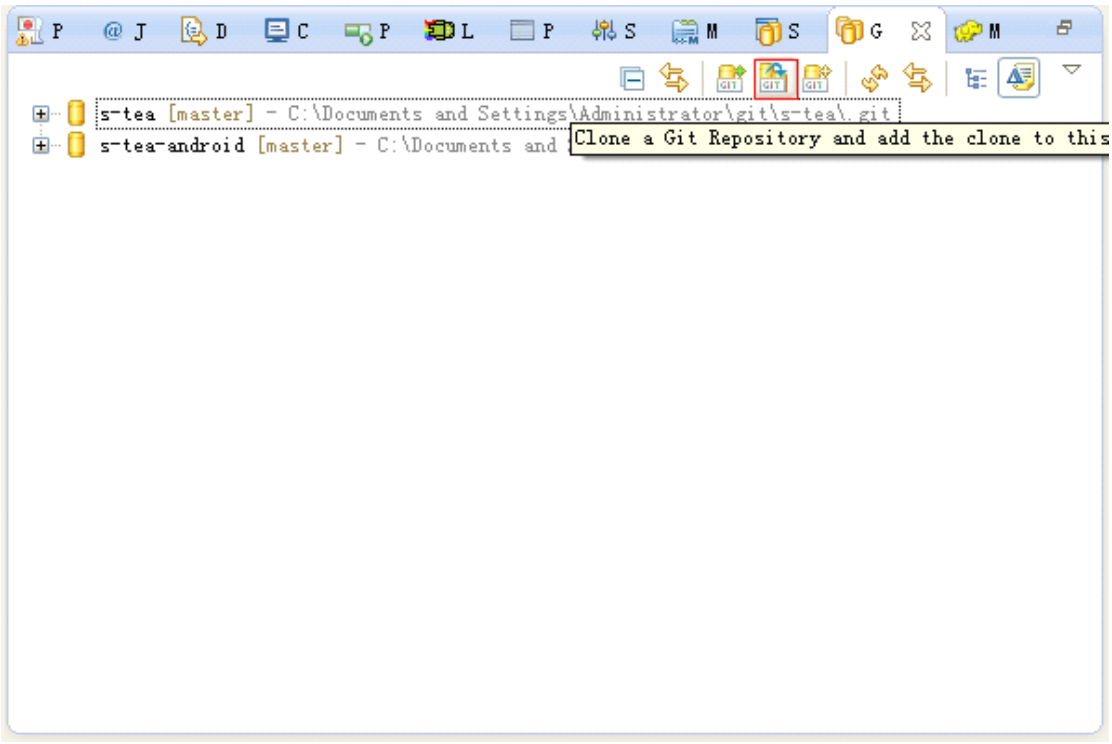
输入完命令之后，它会提示你指定 id_rsa 的目录，我们指定在当前或者直接指定~/.ssh 目录都可以的。然后可能需要输入密码，如果我们直接回车就是不使用。

最后确保 id_rsa 和 id_rsa.pub 在用户目录下面就可以了，然后我们把 id_rsa.pub 文件复制并且改成自己的名字发给管理员。这样管理员就会提供服务器的 git 权限给你，可以给你开通使用仓库的权限。

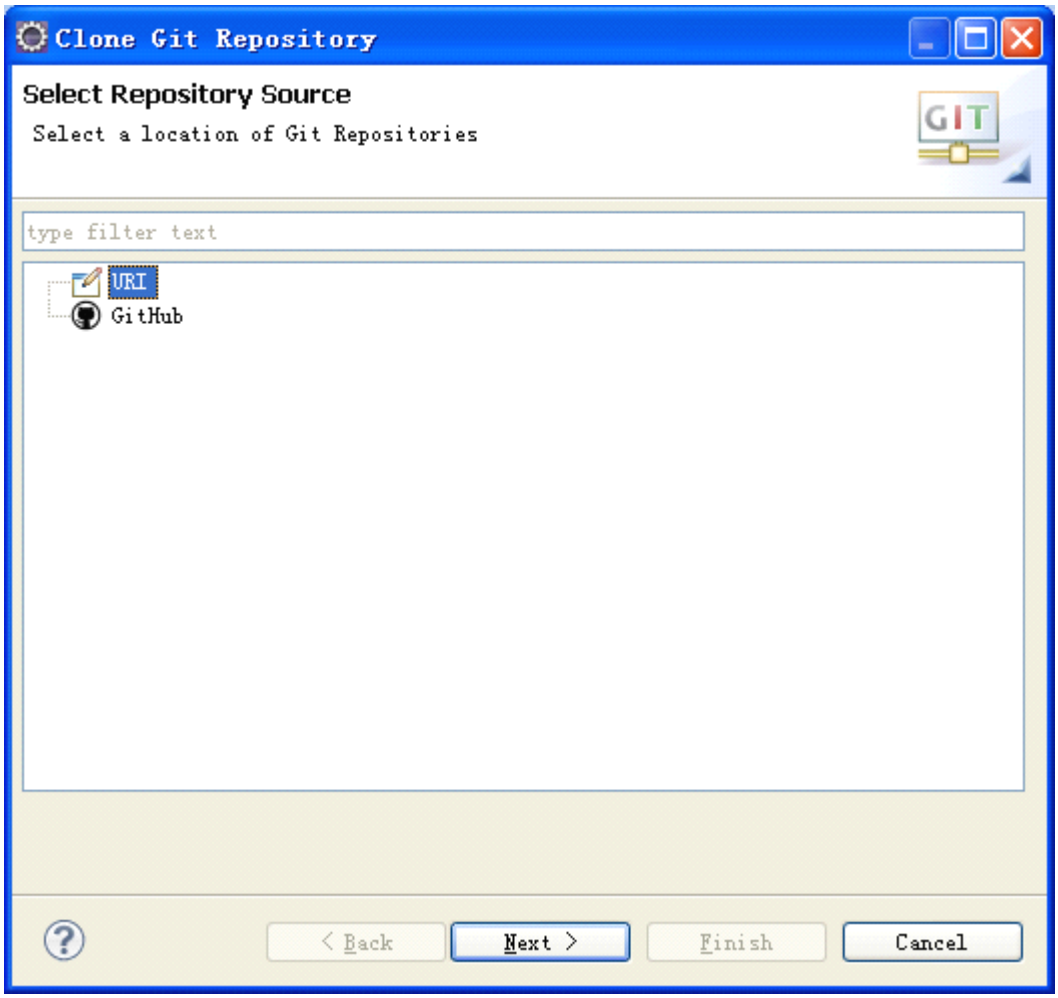
- Git 插件的使用

我们已经有了插件，这个时候我们需要去服务器 clone 下我们的仓库。举个例子，假如我们的仓库有个 auto-work.git 的仓库，我们按照截图去进行操作。

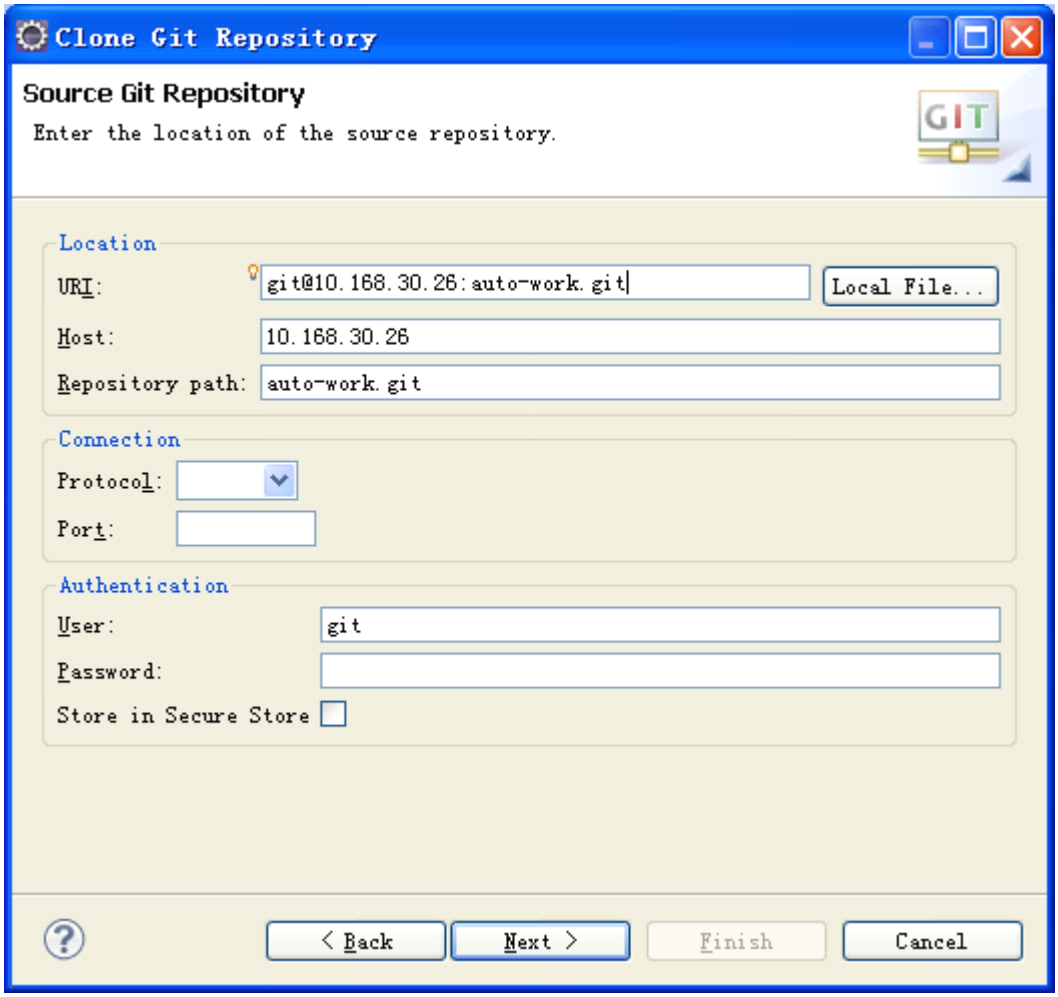
提示：在 eclipse 中不知道 git 插件的图标在哪里的话，我们去 window--->show view-->other 中选择 git 的目录就能够找到仓库的图标，点选的话就会在 eclipse 中显示出来，在控制台一起的位置。



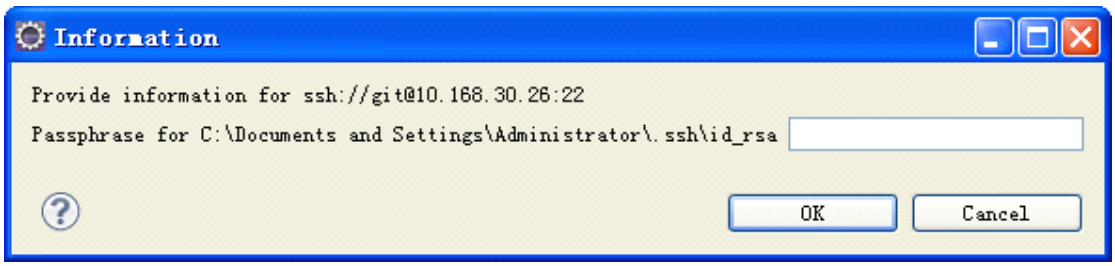
1、点击红色框内的按钮。



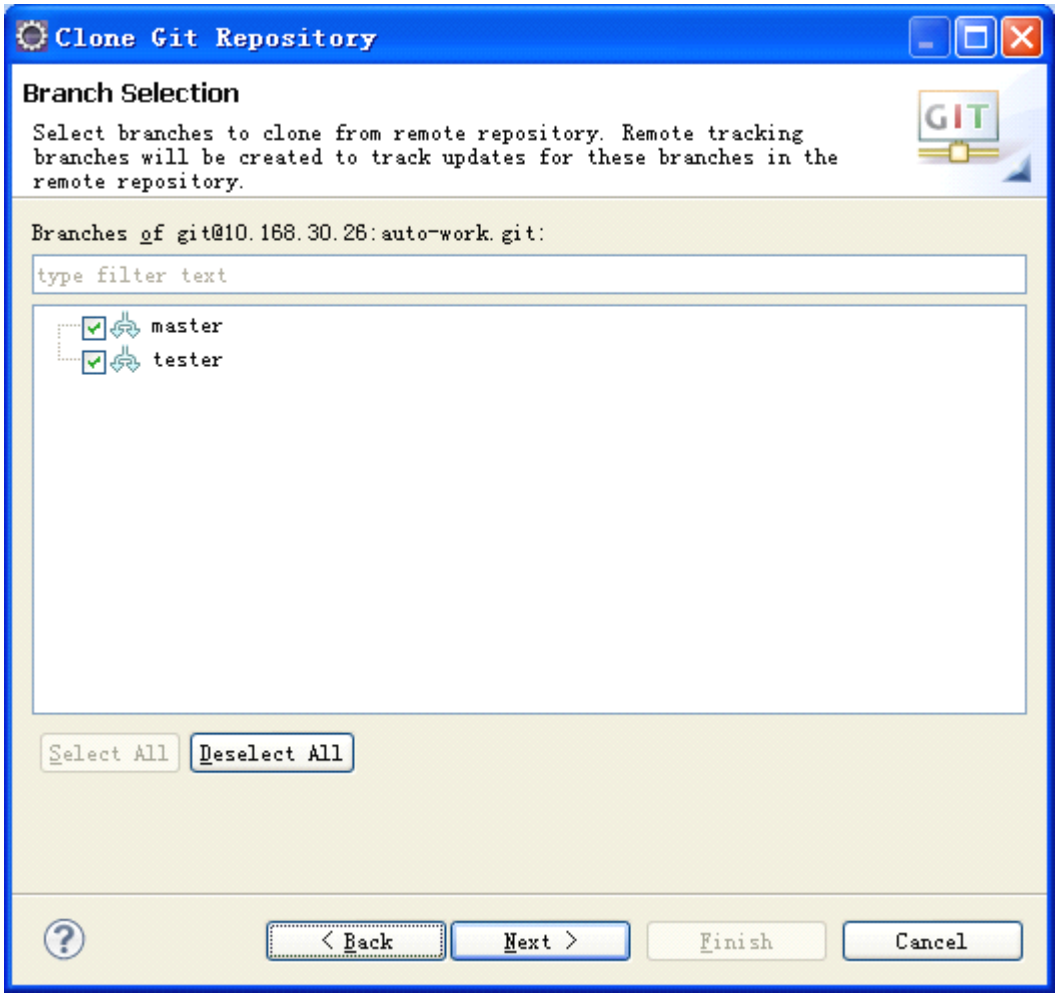
2、选择 URI，然后 next。



3、输入仓库地址，输入规格，开头要用 git 协议，git@，@后面是服务器的地址，冒号后面是自己的 git 仓库的名字，所有的服务器仓库都是.git 结尾的。输入正确后然后 next。



4、这个时候会出现一个这样框框，这个东西就是输入我们本地的密钥，就是我们创建 id_rsa 的时候输入的密码，如果我们没有密码的话，这东西是不会弹出的。



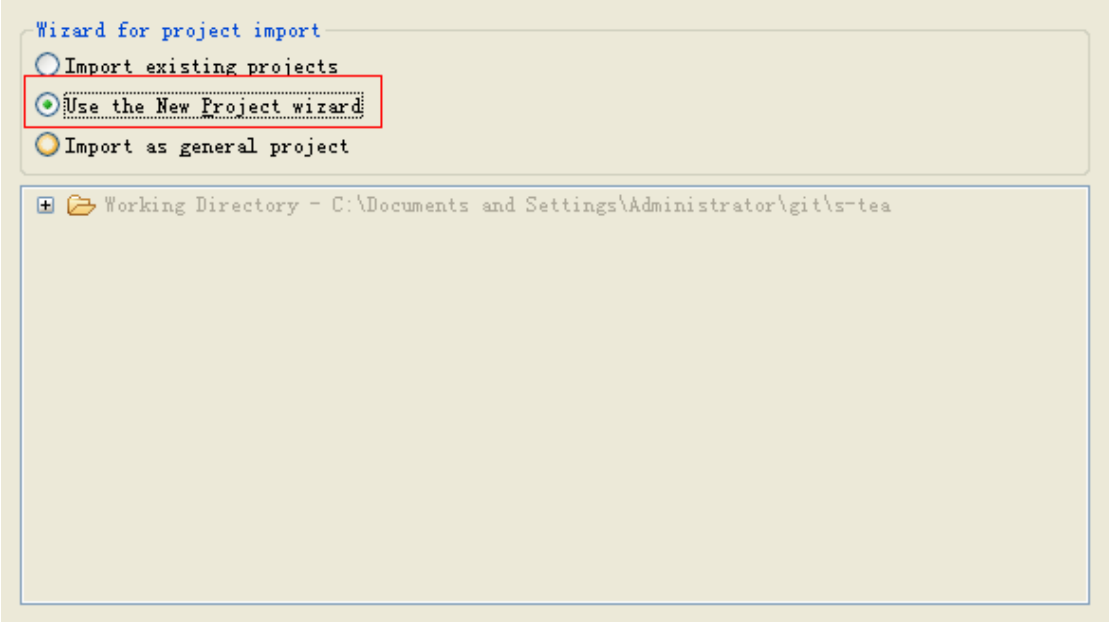
5、出现这个框的时候，我们都选中直接 NEXT 就行了，如果我们项目由管理员分配了分支的话，我们需要选择自己的分支，不过我们的总环境中直接 clone 主分支 master 就可以了。

6、然后在随后的框中直接点击 finish 就可以了。它会自动的 clone 源码到你本地的。

- 创建服务器端的项目

  在我们 clone 的仓库处点击右键，选择 import project。

  如果我们的仓库已经有项目了，直接 import exist project 就行了，如果没有的话，我们选择创建新的项目



然后下一步就可以了，下面出现的就和我们创建 maven 项目的情况一下，不过我们在选择 location 的时候最好是把项目放在我们的仓库目录下面，否则我们无法管理这个项目到远程服务器。

我们选择的是创建的 maven 项目，如果我们想要 maven 项目在远程服务器运行的话，需要对 pom 文件进行一定的配置，下面我给出一个完整 pom 配置，可以根据具体的项目自己进行更改环境。

```
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.st.test</groupId>
  <artifactId>pcpop-test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>pcpop-test</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
```

```
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.github.celeskyking</groupId>
    <artifactId>s-tea</artifactId>
    <version>1.0.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
   <plugins>
       <plugin>
           <groupId>org.apache.maven.plugins</groupId>
           <artifactId>maven-compiler-plugin</artifactId>
           <version>3.0</version>
           <configuration>
               <forkMode>once</forkMode>
              <argLine>-Dfile.encoding=UTF-8</argLine>
              <encoding>UTF-8</encoding>
           </configuration>
       </plugin>
        <plugin>
             <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-surefire-plugin</artifactId>
             <version>2.14</version>
             <dependencies>
                 <dependency>
                 <groupId>org.apache.maven.surefire</groupId>
                      <artifactId>surefire-junit4</artifactId>
                 <version>2.14</version>
                      </dependency>
         </dependencies>
             <configuration>
                 <parallel>methods</parallel>
              <!--  threadCount>10</threadCount-->
                 <!--  argLine>-Dfile.encoding=UTF-8</argLine-->
                 <!-- <skip>true</skip> -->
            <!-- <testFailureIgnore>true</testFailureIgnore> -->
             <forkMode>once</forkMode>
              <argLine>-Dfile.encoding=UTF-8</argLine>
              <encoding>UTF-8</encoding>
             </configuration>
        </plugin>
```

```
        <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-source-plugin</artifactId>
        <version>2.2.1</version>
        <configuration>
            <encoding>UTF-8</encoding>
        </configuration>
        <executions>
                 <execution>
                 <id>attach-sources</id>
                 <phase>verify</phase>
                 <goals>
                          <goal>jar-no-fork</goal>
                 </goals>
                 </execution>
        </executions>
        </plugin>
        <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>2.9</version>
        <configuration>
            <stylesheet>maven</stylesheet>
            <aggregate>true</aggregate>
               <charset>UTF-8</charset>
               <encoding>UTF-8</encoding>
               <docencoding>UTF-8</docencoding>
        </configuration>
        <executions>
                 <execution>
                 <id>attach-javadocs</id>
                      <goals>
                      <goal>jar</goal>
                      </goals>
                 </execution>
        </executions>
        </plugin>
   </plugins>
   <defaultGoal>compile</defaultGoal>
</build>
   <reporting>
    <plugins>
        <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

```
    <artifactId>maven-surefire-report-plugin</artifactId>
    <version>2.14.1</version>
    </plugin>
  </plugins>
</reporting>
</project>
```

上面给的就是一个完整的配置文件的信息，我们需要增加的部分就是<plugins></plugins>内的内容。

这个时候我们通过 git 插件就可以把这个初始项目进行上传到服务器了。在项目处右键单击，然后选择 term，然后点击 add to index,然后 commit，然后弹出提交本地的一个框，我们需要在里面添加一些注释，我推荐大家一定要写清楚注释，然后我们点击 commit 就可以了，最后我们在 term 中点击 push to upstream ，这样它就会自动上传到我们服务器仓库进行同步了。

这样子我们本地的实施环境就全部搭建完成了，剩下的就是去服务器 jenkins 上创建我们的构建项目。我们需要访问 http://10.168.30.26:8080/jenkins 这个页面，这个页面没有任何的管理员，所以我们不需要登录。

点击新 JOB，然后我们在新打开的页面里面选择 maven 项目，然后给自己的项目起个名字，这个时候我们会进入这个项目的构建页面。

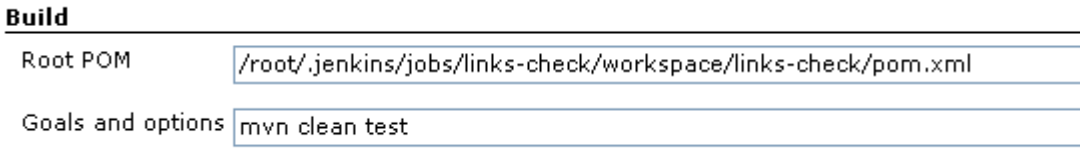
我们会根据具体的说明一下服务器端的基本目录。先看一个完整的项目配置，其中会有一些地方已经做了标注。

这个配置模块中是我们需要指定 git 的位置。一般的路径都是:

/home/git/repositories/${your.projectGit}.git

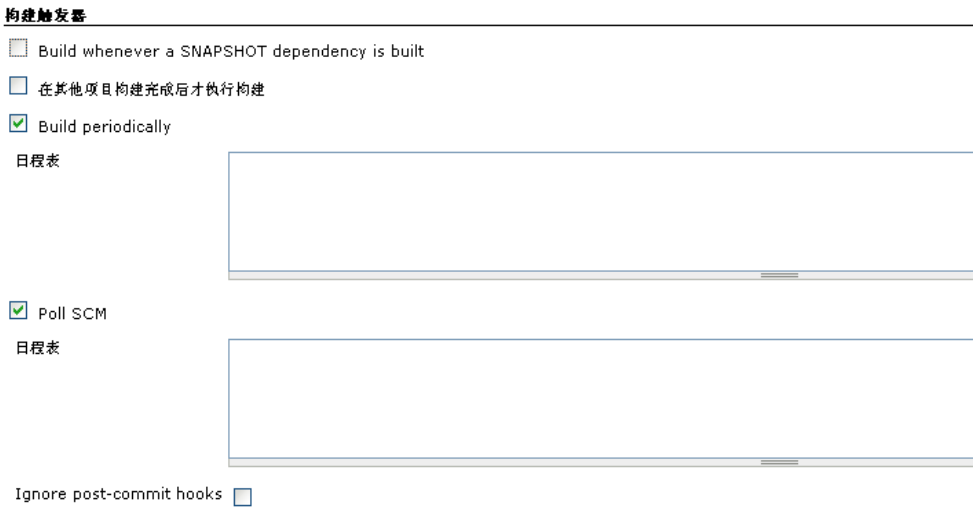
${your.projectGit}.git 就是你这个项目的服务器的仓库名字，需要配置一下子。

下面是构建的过程



这个是构建的配置，里面的 Root POM 指定的是你的 pom 在服务器里面的位置。具体的写法：
/root/.jenkins/job/${jenkins.project}/workspace/${maven.project}/pom.xml

持续构建规则



这个地方由我们来构建规则， 意思就是定义什么时候来自动的构建这些项目来运行来编译。
构建规则： Schedule 的配置规则是有 5 个空格隔开的字符组成，从左到右分别代表：分 时 天 月 年。*代表所有，0 12,20 * * * 表示在任何年任何月的任何天的 12 和 20 点的 0 分 进行构建。
上面的是一个构建规则的例子，所有的构建都可以这样子来进行。
Poll SCM 选项的意思是自动的更新 git 仓库。这个一般都是在 builder 之前来运行更新。这个时候我们的基本配置就结束了，如果需要了别的功能，我们可能需要装一些的插件来实现，配置需要我们自行学习。然后我们点击 save 就能够保存了。这样就会按照我们定制的构建过程来实现自动和持续构建了。

通过上面的总体介绍我们这些就是整个服务器实施环境的搭建。大家如果有新项目一定要找管理员开通权限。

S-TEA Getting started 快速开始

XML 说明

说明：我们的项目是自动化的脚本，因为我们的所有脚本都是通过 maven 来自动构建和运行的，所以对于脚本的命名和放置有规定，我们所有的 class 类必须是以 Test 来结尾命名的，并且所有的@Test 的类都需要放在 src/test 下面的包里面

为了我们方便开发，我们应该硬性的规定我们的包名防止冲突。所以我们用测试的页面的 url 的反转形式来定义自己的包名，比如我测试的是 it168 的产品库，那么我们需要命名自己的名为 com.it168.product 然后在基础的包名下进行脚本编写。

首先我们使用我们推荐的方式来编写脚本，推荐的方式就是使用 xml 来管理元素，然后通过 JUnit 来编写脚本。不过脚本也可以通过 page-object 模式来编写，现在我们先来编写基础的脚本。我们已百度首页为例子。假如目前定义的元素可以做任何的操作。

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:elements xmlns:tns="http://www.example.org/page" xmlns:xsi="http://www.w3.
    <tns:element value="kw" by="id" id="百度首页-搜索框"/>
    <tns:element value="su" by="id" id="百度首页-搜索按钮"/>
</tns:elements>
```

我们定义好了两个元素，元素的定义方式：
1、elemnets 标签是根元素，没有任何的意义；
2、Element 是元素元素，它代表了具体的一个元素；
3、Id 属性代表了你定义的这个 element 的名字，这个名字是自己起的，一般我们都有自己的起名规范。自己名字缩写-页面名字-元素命名。
4、By 属性是通过什么方式来定位元素，定位方式就是 webdriver 提供的几种方式。
5、Value 值就是通过定位方式来定义的时候的值。

6、Frame 元素，这个元素定义的是一个 frame，如果元素处于一个 frame 中，我们需要通过这种方式来定义。Frame 元素是可以嵌套的，如果一个 frame 中还有另外一个 frame 我们可以嵌套的写，他会自动的去查找里面的元素。
7、Childelement 元素，这个元素的意思是我们找的 element 中有子元素，需要通过层级定位的方式来查找，所以我们需要通过 childelement 的方式来解决。它需要用在 element 元素下面，在 s-tea 的 github 页面，我们可以在源码中找到 page.xsd 文件，里面做了 xml 的约束，我们可以通过它来创建 xml，这样用 eclipse 就能够很方便的进行编写元素了。
8、所有的定义的元素都需要放在 xml 文件中，命名不限，不过为了方便自己的管理，最好是通过模块或者页面的方式来进行定义，这样我们能够更明确的维护这些 xml 元素了。它会自动的扫描 xml 目录下面的所有的 xml 文件。

脚本编写

我们通过 eclipse 的代码编写方式来讲述这些元素的功能，这些功能都是我们有可能用到的方法。代码中我会直接给出注释，方法只为了讲述使用方式，而不一定能够正确运行，因为元素可能不匹配，所以大家主要是参考这些方法的使用。

```
/**打开浏览器操作*/
    AutoBase.open(Browser.Firefox, "http://baidu.com");
    /**当前页面的百度输入框元素进行点击操作
     * 下面的代码中我们的currentpage()也可以省略掉。
     * */
    AutoBase.currentpage().sElement("百度首页-输入框").sendKeys("北京");
    /**这个写法是上面的写法的缩略形式，它默认的就是当前页面的操作。*/
    AutoBase.sElement("百度首页-搜索框").sendKeys("北京");
    /**点击操作*/
    AutoBase.sElement("百度首页-搜索按钮").click();
    /**双击操作*/
    AutoBase.sElement("百度首页-搜索按钮").doubleClick();
    /**拖拽操作*/
    AutoBase.sElement("百度首页-拖拽1").dragAndDrop(AutoBase.sElement("百度
    首页-拖拽2"));
    /**滚动到当前元素的视角*/
    AutoBase.sElement("百度首页-搜索框").scroll();
    /**将焦点放在当前的元素上面，一般指在input输入框内指定焦点输入*/
    AutoBase.sElement("百度首页-搜索框").focus();
    /**清理输入框内容为空*/
    AutoBase.sElement("百度首页-搜索框").clear();
    /**得到元素的位置，返回一个Point的对象*/
    AutoBase.sElement("百度首页-搜索框").getLocation();
    /**鼠标左键按下操作*/
    AutoBase.sElement("百度首页-搜索按钮").leftDown();
    /**在当前元素上按下键盘回车键*/
```

```
AutoBase.sElement("百度首页-搜索按钮").keyDown(Keys.ENTER);
/**在当前元素上面松开键盘回车键*/
AutoBase.sElement("百度首页-搜索按钮").keyUp(Keys.ENTER);
/**获取元素的value属性值*/
AutoBase.sElement("百度首页-搜索框").getAttribute("value");
/**获取元素的文本值*/
AutoBase.sElement("百度首页-搜索框").getText();
/***获取元素的标签名*/
AutoBase.sElement("百度首页-搜索框").getTagName();
/**获取元素的dom子元素中所有的a元素的，返回list,通过这个方法来模仿我们查找list元素*/
AutoBase.sElement("百度首页-搜索框").getOptions("a").get(0).click();
/**校验当前元素的value属性值为空*/
AutoBase.sElement("百度首页-搜索框").assertAttribute("value", "");
/**校验当前元素的可编辑性，错误的话会中断测试*/
AutoBase.sElement("百度首页-搜索框").assertEditable();

/**校验当前元素的文本值是否为text*/
AutoBase.sElement("百度首页-搜索按钮").assertText("GOOD");
/**校验当前元素的value属性是否存在或者为空，如果不存在或者为空，程序中断**/
AutoBase.sElement("百度首页-搜索按钮").assertValue();

/**不通过xml元素来定义元素*/
AutoBase.sElement().addLocator(Locator.Id, "kw").sendKeys("北京");
AutoBase.sElement().addLocator(By.id("kw")).sendKeys("北京");
/**如果元素在一个元素的list中，我们可以通过索引值的方式来定义*/
AutoBase.sElement().addLocator(By.id("kw"), 0).sendKeys("北京");
AutoBase.sElement().addLocator(Locator.Id, "kw",0).sendKeys("北京");
/**如果直接定位不到，我们需要层级定位*/
AutoBase.sElement().addLocator(By.id("kw")).childElement(By.id("su"))
.sendKeys("北京");
/**ajax的元素，会自动的等待默认30s的时间直到元素出现后才会继续进行操作*/
AutoBase.ajaxElement("百度首页-搜索框").sendKeys("北京");
```

上面是我们用的到的元素操作，这不是全部，但是使我们一般可能会使用的到的，下面我们讲一下当前页面的操作类，主要是 CurrentPage 类的操作。

```
/**当前页面的ajax元素*/
AutoBase.currentpage().ajaxElement("百度首页-搜索框").click();
/**CurrentPage类的使用说明*/
/**断言是否出现了alert*/
AutoBase.currentpage().assertAlert();
/**获取当前页的头信息*/
AutoBase.currentpage().getHeaders();
```

```
/**通过头信息的名字来获取头信息的值*/
AutoBase.currentpage().getHeaderValue("Context-Type");
/**页面是否使用了gzip压缩的方式*/
AutoBase.currentpage().isGzip();
/**获取当前页面的源码*/
AutoBase.currentpage().getPageSource();
/**获取当前页面的响应值*/
AutoBase.currentpage().getStatusCode();
/**获取页面的所有的js的引用地址，返回list*/
AutoBase.currentpage().getJavaScriptURL();
/**获取当前页面的title值*/
AutoBase.currentpage().getTitle();
/**获取当前页面的url*/
AutoBase.currentpage().getCurrentUrl();
/**获取当前页面的cookie值*/
AutoBase.currentpage().getAllCookies();
/**通过cookie给定的名字返回值*/
AutoBase.currentpage().getCookieByName("username");
```

上面的所有方法并没有设计元素的操作，因为所有的元素操作都是建立于页面的，所以我们的 currentpage 是支持所有的提到的 sElement()方法的。此处不在列举。

我们对页面和元素都进行了操作，下面我们对浏览器对应的 Window 对象进行解析，还是通过方式和注释的方式来解释。

```
/**浏览器后退操作*/
Window.back();
/**浏览器前进操作*/
Window.forward();
/**浏览器刷新操作*/
Window.refresh();
/**浏览器最大化，这个方法暂时不需要我们使用了，因为我在打开浏览器的时候已经默认的最大化了*/
Window.maxWindow();
/**处理alert框*/
Window.dealAlert();
/**处理confirm，true为确认，false为取消*/
Window.dealConfirm(true);
/**处理prompt，同理和confirm，第一个参数是要输入的文本内容*/
Window.dealPrompt("example", true);
/**切换页面到最新打开的窗口处，此方法只能够在新打开一个页面的情况下有效*/
Window.selectNewWindow();
/**返回到默认的窗口session下面,比如我们进入了frame，在框架中的xml元素中做过封装了，已经不需要了*/
```

```
Window.selectDefaultWindow();
/**执行js命令，可以带有参数*/
Window.runJS("alert(\"hello world\");");
/**截屏操作，此操作需要在配置文件中指定截屏路径*/
Window.takeScreenShot();
/**selectFrame是一个系列的方法，可通过各种定位方式来进入到frame内部，在xml元素定义中已经封装
 * 了此方法，不需要在进入进出frame
 * */
    Window.selectFrame(By.id("kw"));
```

AutoBase 类的特殊方法介绍。在这个方法内提供了一些特殊的方法，有可能我们在使用过程中会需要。

```
/**这些是一个系列的方法，里面都是封装的单个元素，就像SElement元素一样，不过它们都是继承的SElement
         * 它们有自己的特性，所有我单独做了一些封装，里面的方法都很简单，我们可以通过使用sElement的方式来使
         * 用它们，只需要继续.方法就方式就可以了，具体的方法我们需要自己查看API，不再一一列举，不过即使我们
         * 不使用它们依然可以做脚本，因为所有的封装都是基于SElement来做的分析。
         * */
        AutoBase.button("");
        AutoBase.checkBox("");
        AutoBase.image("");
        AutoBase.table("");
        AutoBase.textField("");
        AutoBase.link("");
        AutoBase.radioButton("");
        AutoBase.richTextField("");
        AutoBase.comoboBox("");

        /**这个方法是在当前浏览器里面重新开启一个window，但是它不会脱离driver对象，是在当前的进程中开启的
         * 打开的是一个空白页面，并且会把焦点放在新打开的页面中，它返回当前页面对象。
         * */
        AutoBase.openNew().open("http://www.baidu.com");
        /**这个方法来判断我们的浏览器是否被关闭了，如果为true说明关闭了，如果为false说明没有关闭*/
        AutoBase.isClose_Status();
        /**这个方法可以作为和WebDriver交互的通道*/
        AutoBase.driver();
```

上面的所有介绍的方法就是暴露给我们的所有的功能方法，但是 s-tea 的功能不知仅限于此，它还提供了监听器的功能，它提供了两种监听器，一种是动作监听器，一种是运行时的监听器。运行监听器的话需要我们在定义的 class 类中指定运行器。我们推荐使用 BaseJUnitAutoRunner.class 的运行器，它提供了参数化，多线程和运行时监听器的功能。

如下面的示例来使用我们自己的运行器

```
@RunWith(BaseJUnitAutoRunner.class)
@ThreadRunner(threads=1)
public class AppTest{
    //your methods
}
```

我们依旧通过代码的形式来介绍。

- 动作监听器

动作监听器的使用有两种方式，一种是通过注解的方式，一种是通过代码注册的方式，我们先讲解注解的方式。

我们需要先创建一个类 MyListener，这个类需要继承 RunnerListener 接口，然后我们需要编写我们需要定制的方法。这些方法在接口处一目了然，我们可以通过 javadoc 来查看这些方法，不再例举。我们定义好了自己的方法之后，只需要在这个类上面加上类级别的监听器@Register

```
@Register
public class MyListener extends AutoRunnerListener{

    @Override
    public void afterClickOn() {
        System.out.println("点击了");
    }
}
```

就是通过这种方式来定义一个动作监听器。AutoRunnerListener 是 RunnerListener 接口的一个空实现，为了让我们编写更少的代码，所以提供了一个空类。

还有一种方式是通过注册的方式来使用，下面我们用代码的形式来讲解。前提是我已经定义了 MyListener 这个监听器，但是并没有加注解。

```
ProxyRunnerListener.register(MyListener.class);
ProxyRunnerListener.unregister(MyListener.class);
```

这两个方法是注册监听器和注销监听器，使用方法是我们在需要这个监听器的时候去用第一个 register 的方法去注册一下，然后在这个方法之后的所有代码中如果出现了点击操作，就会激活 MyListener 中的注册的点击之后打印”点击了”这个方法。在使用了 unregister 方法之后我们定义的监听器就会被取消掉，不再对点击操作进行监听了。

- 运行时监听器

运行时的监听器其实是对 JUnit 的监听器做的处理，我们定义自己的运行时的监听需要继承 RunListener 这个类，它是 JUnit 提供的类，它的注册方式只介绍我们推荐的一种，就是我们创建一个类，叫做 MyJUnitListener 它也继承 RunListener ，然后我

们编写自己的逻辑，具体的方法参考 JUnit 的 API,它的方法只需要通过方法名就能够清楚含义。不再介绍。

然后我们在这个类上面加上类级别的注解@RunListenerRegister 就可以了，这样我们的 case 运行的时候，运行器会自动的去扫描这些监听器。

```
@RunListenerRegister
public class MyJUnitListener extends RunListener{
    //your methods
}
```

\

服务器和节点服务器

我们的脚本最终需要上传到服务器来进行编写，所以我们必须要通过服务器来进行开启 selenium 的 server 来进行远程操作，目前有两种方式来进行服务器的 selenium server 和本地的 server 的开启。

一、第一种就是我们最传统的手动开启的方式,但是这种方式并不能在持续集成的过程中完成的特别良好，下面我们主要讲一下本地 server 和服务器 server 的开启命令。

我们在服务器开启的服务器叫做主机服务器，它是用来掌控所有节点服务器的。我们需要在服务器端开启 server，假如这个 selenium-server 在路径/server 下面，那么我们开启的命令是：java -jar /server/selenium-server-standalone-xxx.jar -role hub

这个时候服务器端会默认端口 4444 来开启一个主服务器的功能。下面我们通过在本地或者测试机上面执行命令(假如我们的 selenium-server 的路径为/node 下面)：

Java -jar /node/selenium-server-standalone-xxx.jar -role node -hub http:服务器 ip/grid/Register (-p 5555)

-p 命令是可选的，它可以指定端口，如果不指定的话，它默认的为 5555，这样我们的程序就可以通过远程服务器来进行运行了。

二、第二种方式是通过程序的方式来进行 server 的开启。我们的 resource 目录下面有一个 command.txt 的文件，这个文件是我们需要执行的远程服务器的命令，我们需要进行配置一下。这里先提一下，我会讲一个例子来具体指定要做的操作。

因为我们需要在运行前打开远程服务器，所以我们需要通过运行时监听器来进行定义，在运行测试之前先打开远程或者本地的服务器。

提供的功能在 org.sky.auto.server 包内，里面有几个方法，这些方法的实现是通过 ssh2 和 telnet 来实现的，所以要求我们的节点服务器必须开启这些服务。现在我贴一个开启服务的监听器。

```
@RunListenerRegister
public class MyJUnitListener extends RunListener{
    @Override
    public void testRunStarted(Description description) throws Exception {
        RemoteServer.start("/server/selenium-server-standalone-2.32.0.jar");
        /**因为我们的大多数环境都是window的环境，所以我们提供的是telnet的默认连接方式，在
         * server包内我们提供了运行ssh2的方式。但是没有做远程封装，所以需要我们手动封装或者直接调用
         * API来使用。我们在使用NodeServer的时候，在resource目录下面的command.txt文件中需要定义我们开启
         * 服务的命令，比如我们的server在路径D:\下面，那么我们的command.txt文件中需要定义
         * java -jar d:\selenium-server-standalone-2.32.0.jar -role node -hub http://服务器ip
         *  /grid/register
         *  这样我们的服务就能够开启了，睡眠十秒是为了给服务器开启服务的缓冲时间
         * */
        AutoBase.sleep(10);
        NodeServer.start("10.168.x.xx", 23, "sky", "123456");
    }

    @Override
    public void testFinished(Description description) throws Exception {
        NodeServer.stop();
        RemoteServer.close();
    }
}
```

现在这个开启服务器的方法对于环境和网络的依赖特别大，很不稳定。需要我们做好测试机或者节点服务器的配置工作，并且 RemoteServer 这个类只能在 selenium-server 主机的机器上运行。

测试报告插件

s-tea 提供了测试报告的一个功能，它需要的配置文件内容都能够在 s-tea 的 github 主页上找到。我们需要实时更新就可以了。开启测试报告只需要使用运行器 ReportJUnitRunner 就可以了。

```
@RunWith(ReportJUnitRunner.class)
@ThreadRunner(threads=1)
public class AppTest{
```

这样就能够开启了，前提是我们的配置文件都已经搭建好了，它必须基于 templet 文件夹的内容，这个文件夹里面是一些模版内容，需要通过它们来生成测试报告，测试报告的内容由三部分提供，它能够提供一份总的报告，总报告会罗列出所有的 case，绿色为成功，红色为失败了。然后又没个 case 有一个报告页面，这个报告页面中会罗列出一些信息和必要的日志信息，然后我们还有一份全部的日志信息。里面是所有的日志内容，不过可能比较乱....就是我们平时在控制台看到的日志。

断言

目前提供了一个断言类，这个断言类暂时不提供断言之后继续运行程序的功能，我们可以通过打印错误日志的方式来实现这个功能。我们的 SElement 和 currentPage 类里面已经提供了

一个简单的断言方法，我们提供的这个断言实现是 JUnit 的断言类。在 org.sky.auto.verify 这个类中 Assert，它的使用和 JUnit 的断言是一样的。Assert.assertEuqel(String ex,String real) 的形式。学习和使用过程中，需要大家对 JUnit 的断言特别的熟悉，事实 Assert 的使用也是特别的简单方便。

Fit 功能

目前 S-Tea 提供了 fit 的功能，fitnesse 是一个通过编写 wiki 风格的 case 来运行程序的脚本的服务器，目前我们本地的服务器上有一个我搭建的 fitnesse 的服务器，我们可以通过 wiki 或者表格的方式来编写 Case，我们使用 Robot 类中的关键字就能够编写，fit 也相当于提供了一个关键字驱动，不过它能够更方便的编写 fit 的处理模式。我们能够定制更多的表格规范，目前提供了两种定制的表格解析方式。有兴趣的可以参考 fitnesse 的资料，并且通过参考源码来了解更多的内容。

RobotFramework 关键字

S-tea 提供了一套 Robot Framework 的关键字，目前的关键字并不是很完善。关键字的部分已经被单独的作为一个项目维护，目前提供的是 jbehave，robot，fitnesse 的方式，它整合关键字的部分来实现更方便的自动化测试，不过现阶段是处于开发阶段中，关键字就是方法名，我们通过读源码就能够知道关键字的意思。有兴趣的可以通过源码了解更多。

Page-Object 模式

我们一直听这个模式，但是从来没有注意和使用过，这个模式的意思就是我们把我们需要测试的页面直接包装到一个类里面，当作资源，然后我们的 case 去调用这个资源类里面的功能和方法直接去使用。这样我们提高了复用性，也就节约了开发时间和维护成本。所以我推荐大家的所有的 case 编写都通过 page-object 的模式来编写，s-tea 提供了一种简约的调用 page-object 的模式。下面我们举例说明，假如我们有一个页面是百度首页，我们提供的方法是搜索功能，这样我们定义一个 Baidu 的 class，然后我们就可以去编写代码了。先参考 Baidu 类的代码:

```
import org.sky.auto.page.source.CurrentPage;

public class Baidu extends CurrentPage{

    public void search(String text){
        sElement("百度首页-搜索框").sendKeys(text);
        sElement("百度首页-搜索按钮").click();
    }
}
```

我们的 case 类的代码如下：

```
@Test
public void baiduSearchTest(){
    PageModel.load(Baidu.class).search("北京");
    Window.selectNewWindow();
    AutoBase.currentpage().assertTitle("北京-百度搜索");
}
```

这样我们就能够进行是否打开了搜索页面的验证，这样在我们的 case 中没有出现任何的 selement 等元素操作的代码，这样不太可能出现大量的代码维护工具，可以降低我们的开发成本和提高复用性，想使用的人可以多思考一下这种方式带来的好处，并且提倡大家多多使用。

结束语

欢迎大家使用 s-tea，s-tea 提供的功能已经很强大，但是在我们的使用过程中可能会出现各种各样的问题，并且 s-tea 一直处于维护状态中。S-tea 可以解决很多问题，但是它却有很多问题解决不掉。我们可以借助更多的方式和手段来实现自动化测试，并且 webUI 自动化测试的成本高，效率低的特点是种硬伤，所以我们对于自动化测试必须要有独到的见解来实现更高价值的测试，我推荐大家使用测试驱动开发的方式来编写脚本，我们通过编写一个肯定会失败的测试用例来验证程序和功能的正确性。

更多的 s-tea 的功能如果想了解，需要我们去通过 javadoc 和源码去了解，去深入。
celeskyking@163.com 是我的邮箱，如果有新的需求和 bug，欢迎邮件通知。