## Jenkins自动化编译配置指南

|  |  |  |
| --- | --- | --- |
| version | author | description |
| 1.0 | 张敬涛 | 配置+脚本附录 |

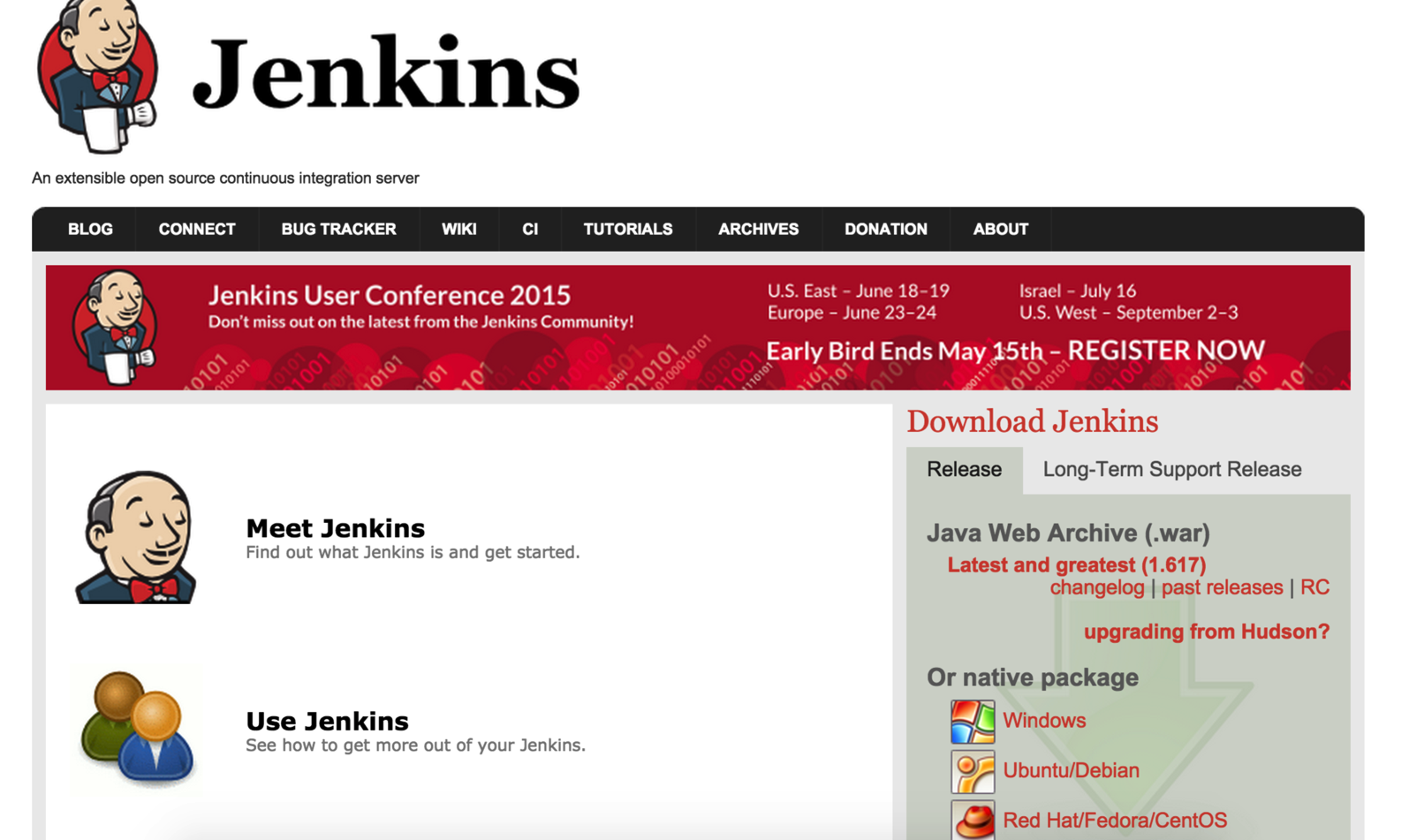
自动化编译是项目管理中重要的组成部分，能够有效解决软件编译版本混乱，测试的代码和 Git 代码管理服务器的代码不一致的问题以及由此引发的研发资源浪费的问题，把研发从日常的编译，安装工作中解放出来，同时也保证了测试测试的代码的可靠性和可追溯性。

Shell 编译脚本是自动化编译的重要组成部分之一，合理的保存符号文件和编译生成的安装文件能够保证一整套代码，符号文件和安装文件的匹配性，一旦发生异常问题，我们可以通过 Crash Log 通过符号文件对异常进行定位，虽然有时并不是100%的精确，但是至少可以提供一个方向。下面简单介绍下配置工作：

1. **准备工作**

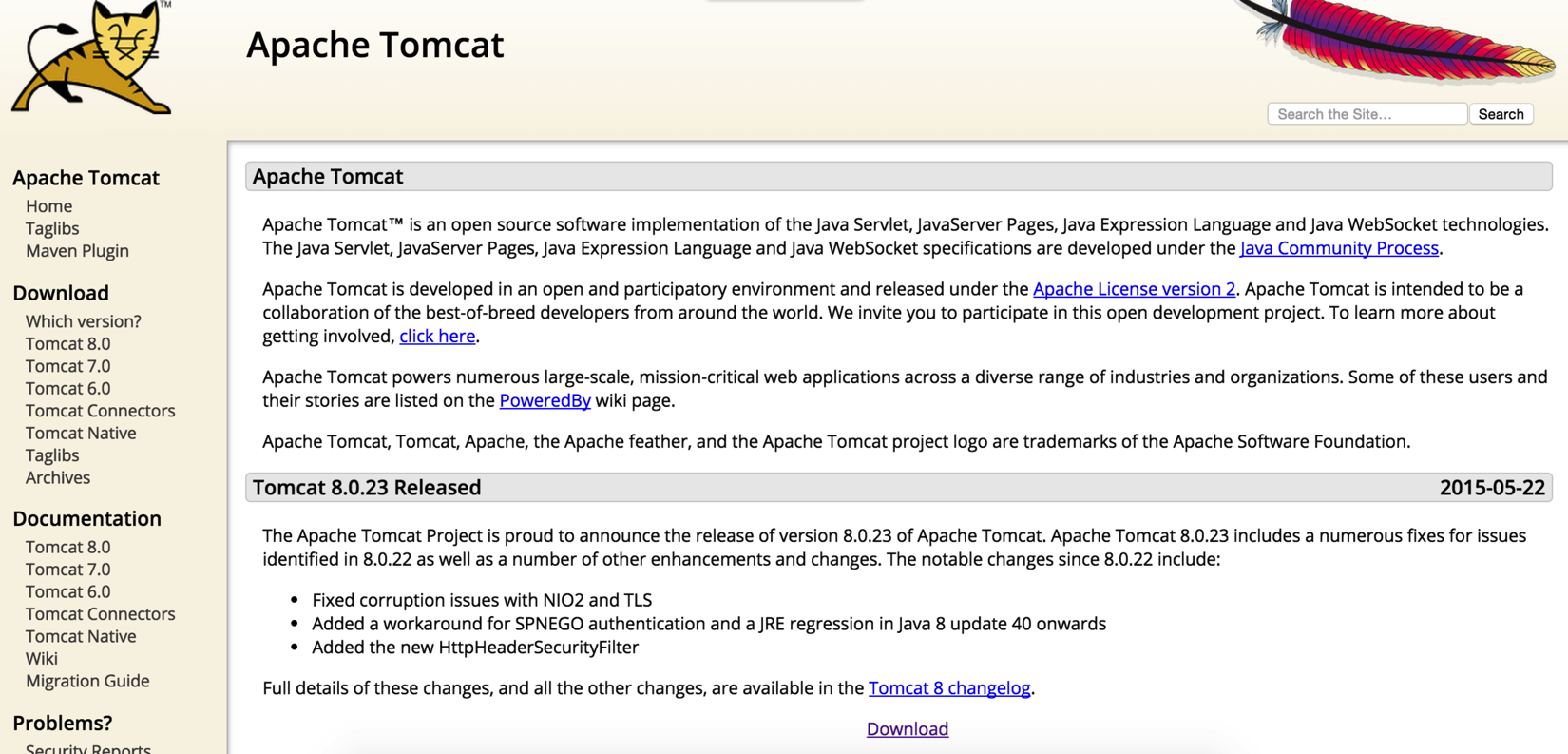
**下载：**

* **jenkins.war （http://jenkins-ci.org/）**

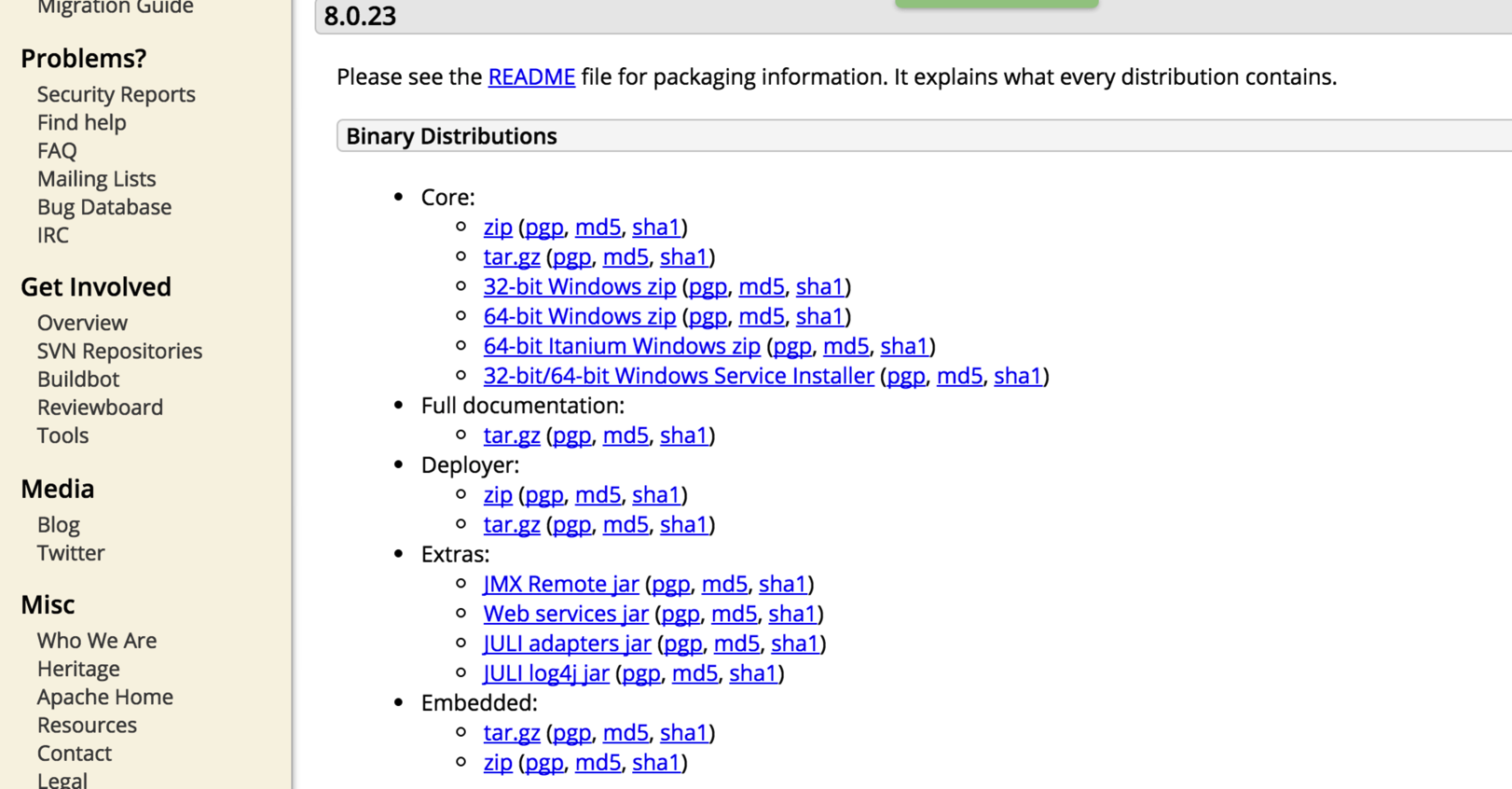
****

**建议下载 Java Web Archive(.war)文件。**

* **apache-tomcat （http://tomcat.apache.org/）**

****

**最新版本是8.0.23 Released**

****

**下载 Core 里面的zip或者 tar.gz。**

1. **部署jenkins**
2. **解压缩tomcat.**
3. **拷贝jenkins.war到apache-tomcat-8.0.23/webapps**
4. **进入apache-tomcat-8.0.23/bin目录运行命令：sh catalina.sh start 或者 ./catalina.sh start启动服务(针对Mac OS X和 Linux系统)。**

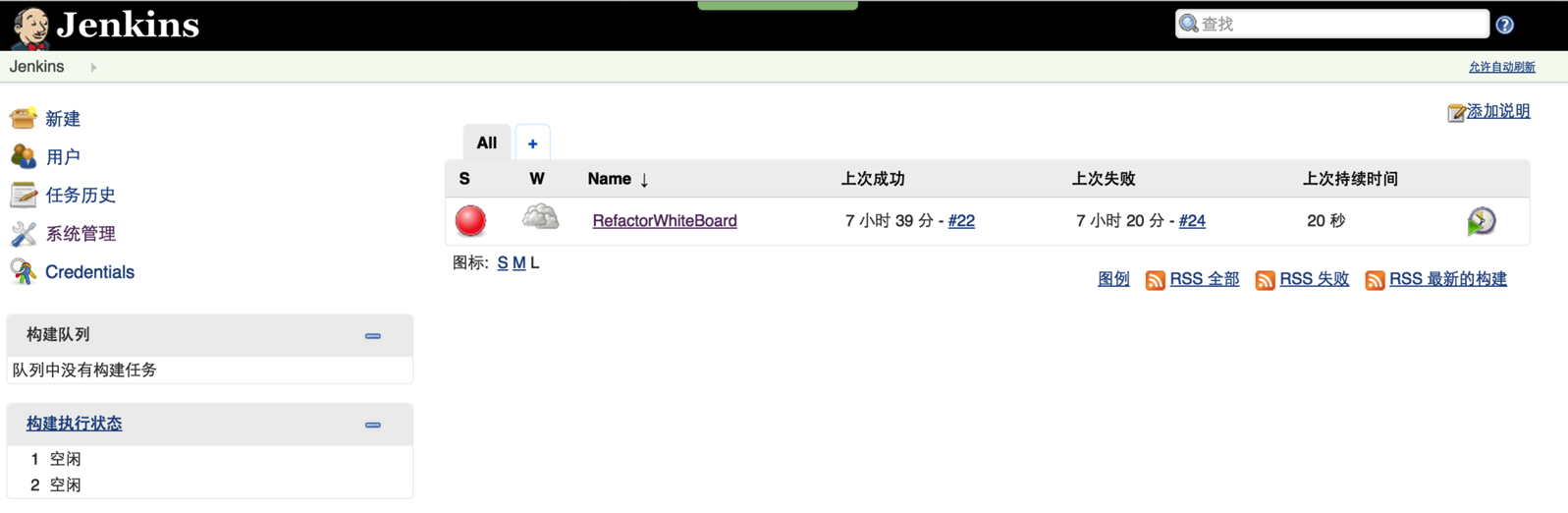
**eg:**

cd Documents/Tools/apache-tomcat-8.0.23/bin

sh catalina.sh start

**停止jenkins命令：sh catalina.sh stop 或者 ./catalina.sh stop**

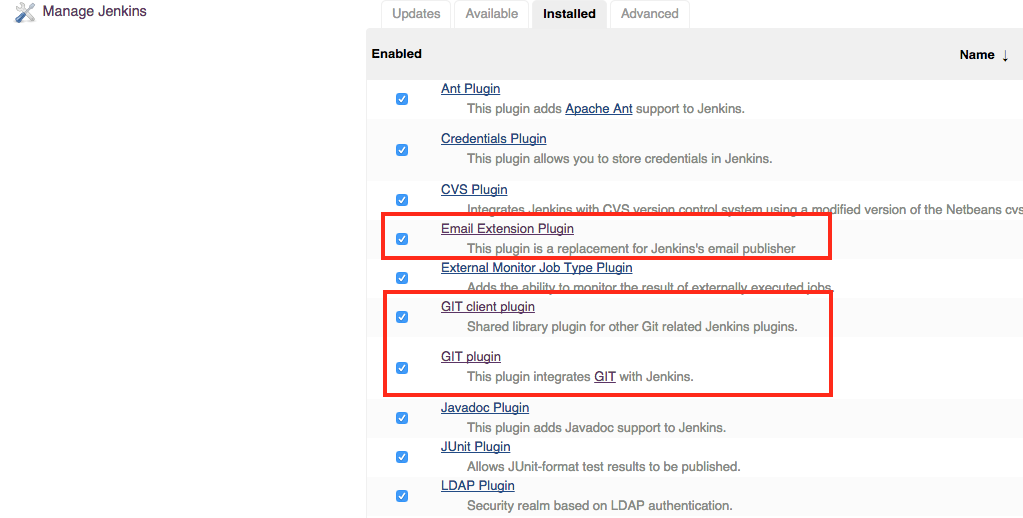
1. **配置jenkins**
2. **打开jenkins主页：**[**http://localhost:8080/jenkins/**](http://localhost:8080/jenkins/)



1. **进入主配置界面：点击“系统管理”**



1. **安装需要的插件：选择“管理插件”**

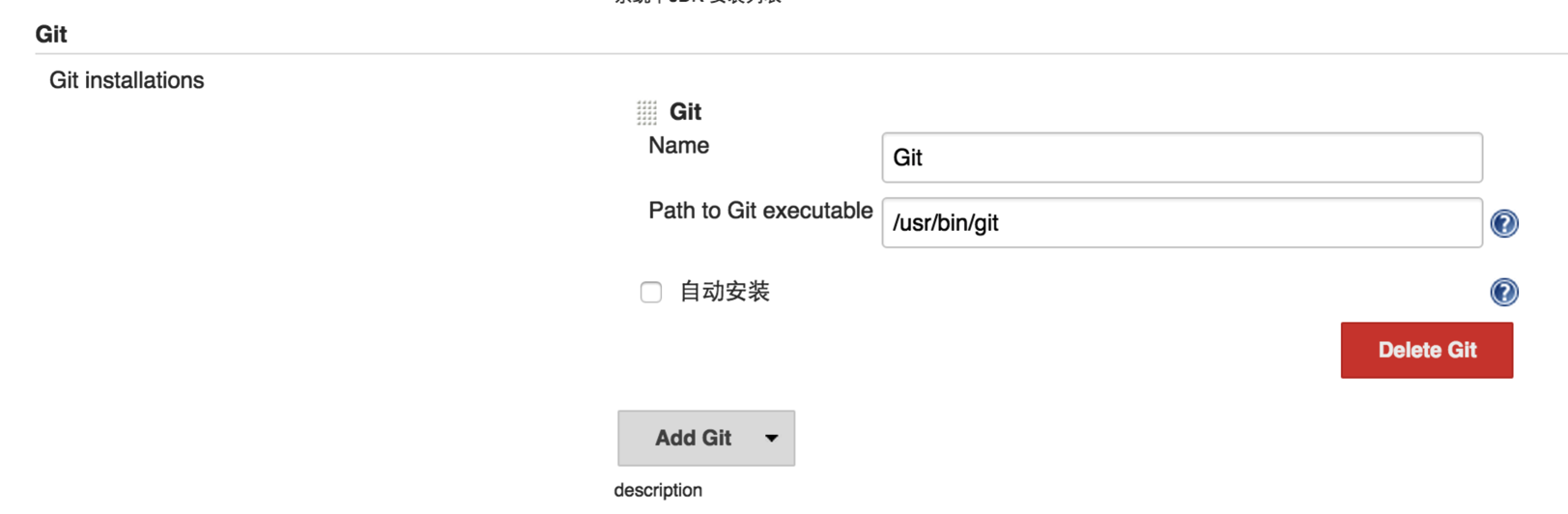
****

**GIT client plugin ，GIT plugin，Email Ext Plugin和Multiple SCMs为必装插件。**

1. **系统设置：**



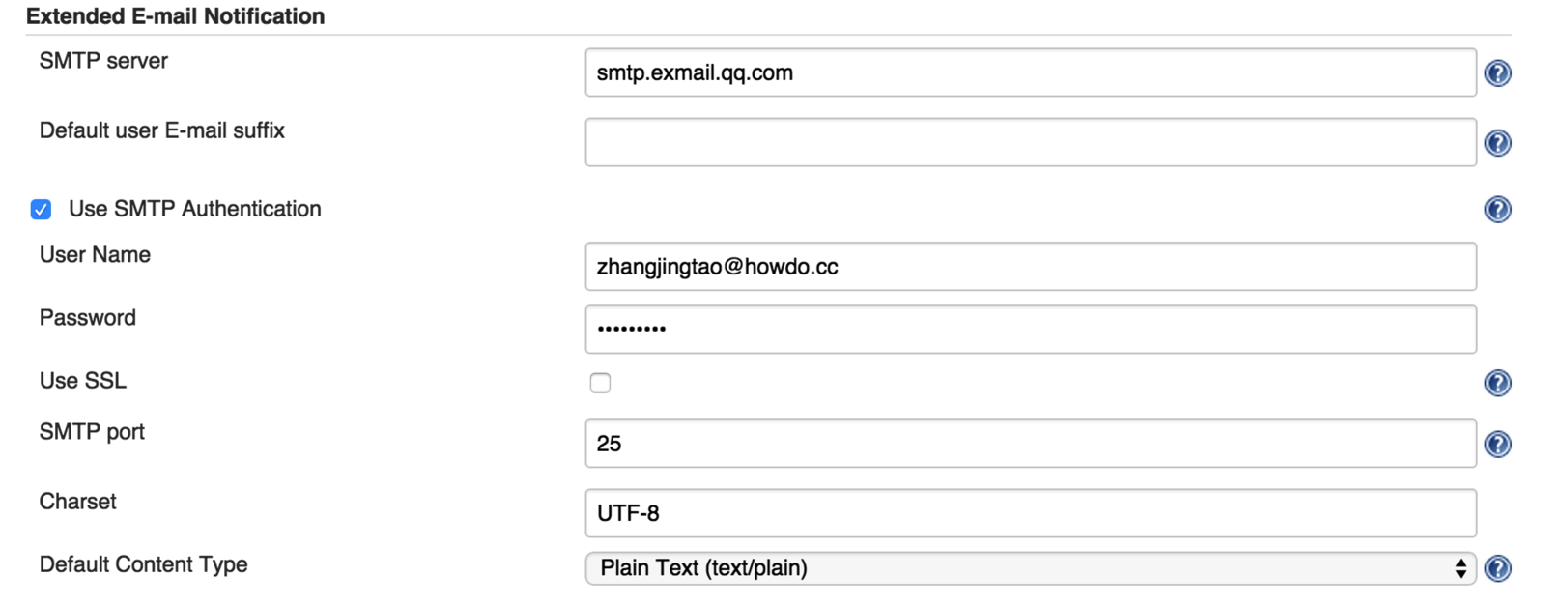
**配置 Git 程序**



**设置服务及管理员信息：**

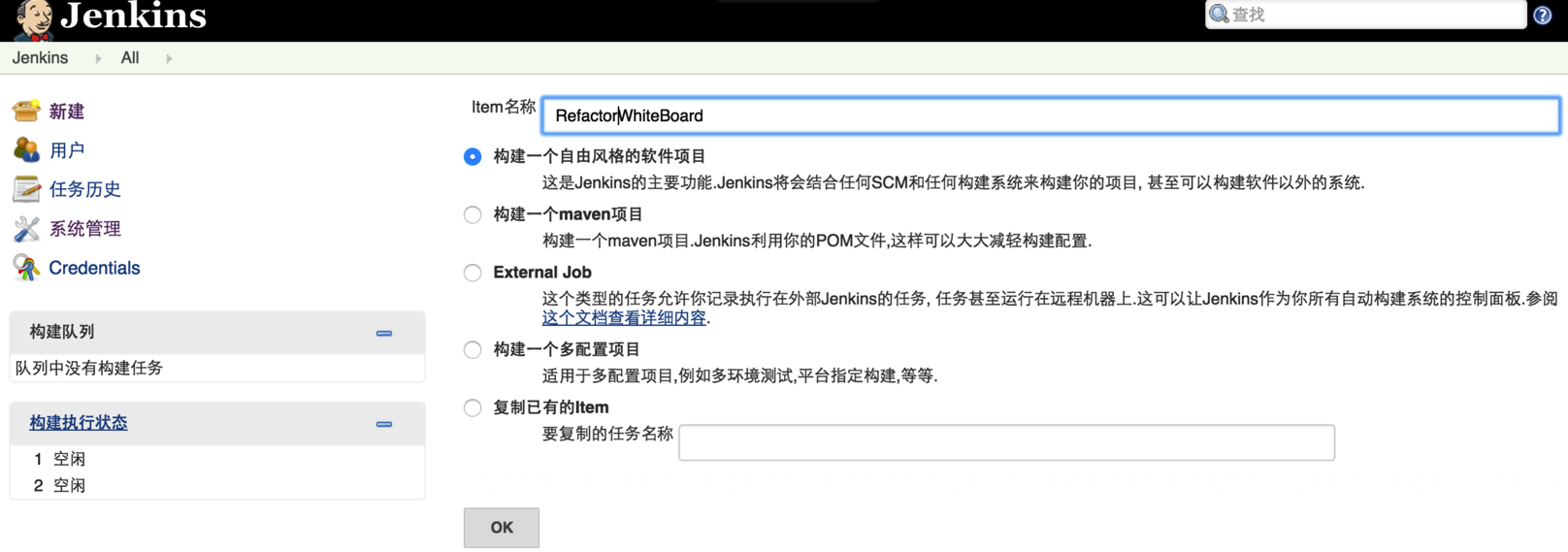


**设置 Email 通知服务**

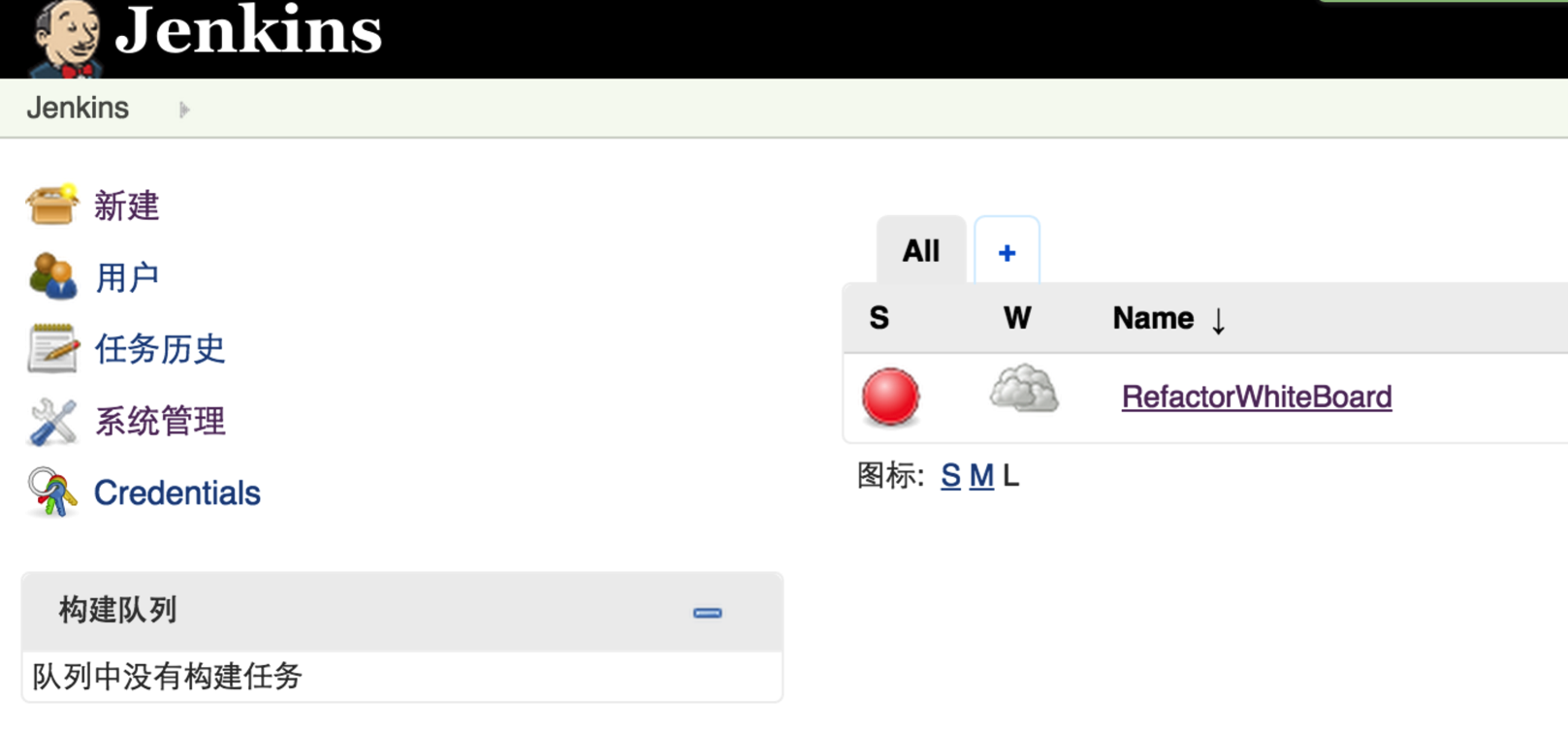


**需要点击高级配置 SMTP 服务认证。**

**新建工程构建:**

****

**添加完成：**



1. **配置构建: 点击刚刚创建的构建名称(这里是 RefactorWhiteBoard)**



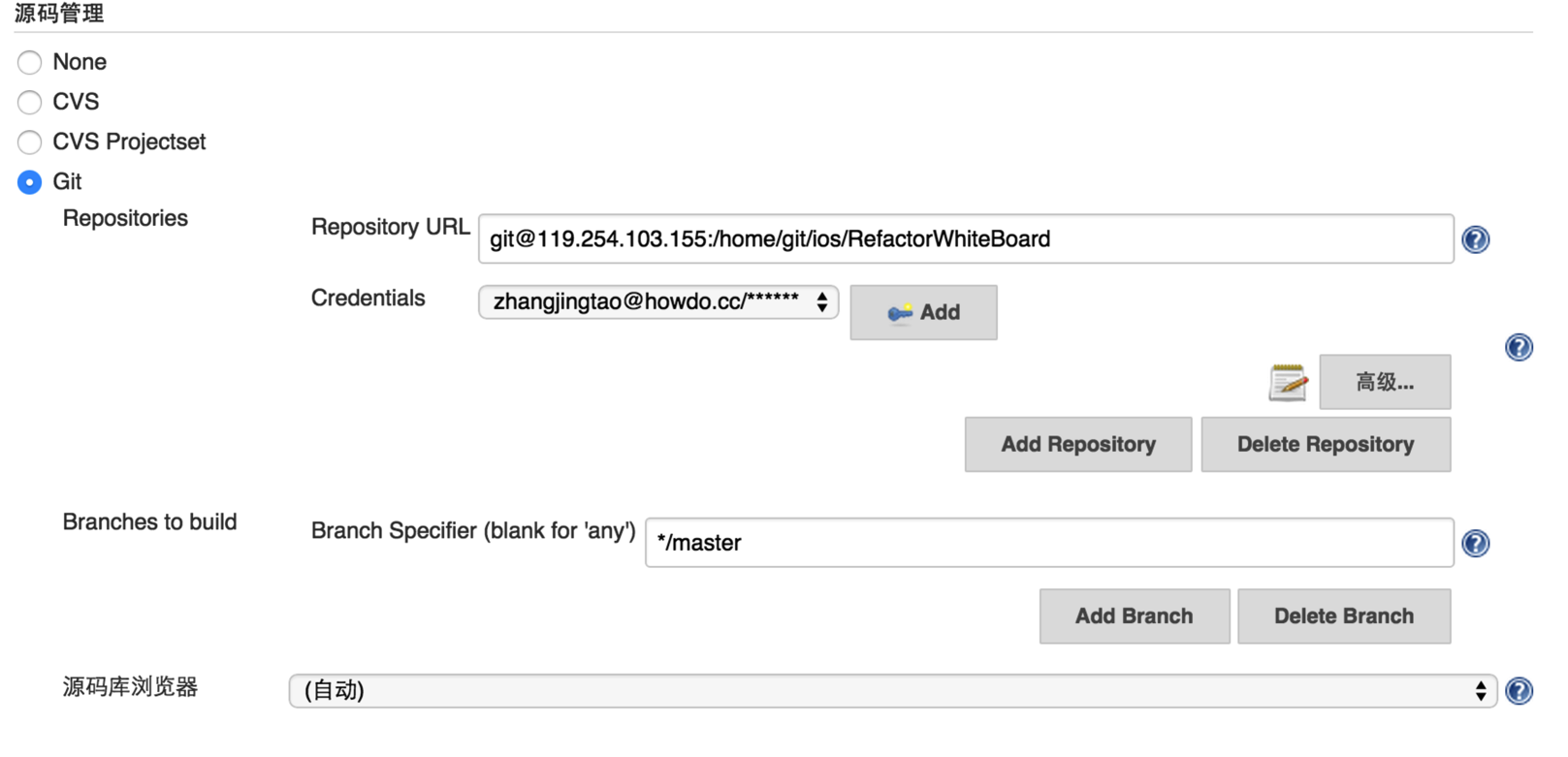
**点击左侧的配置：**

****

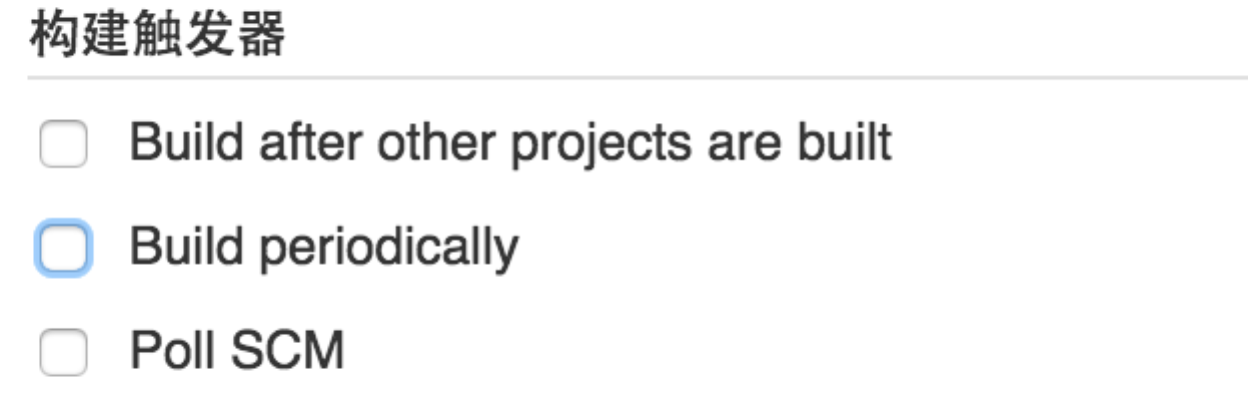
**可以添加描述。**

**点击高级（右下角）可以配置自定义工作空间**

**设置 Git:**

****

**构建触发器：**

****

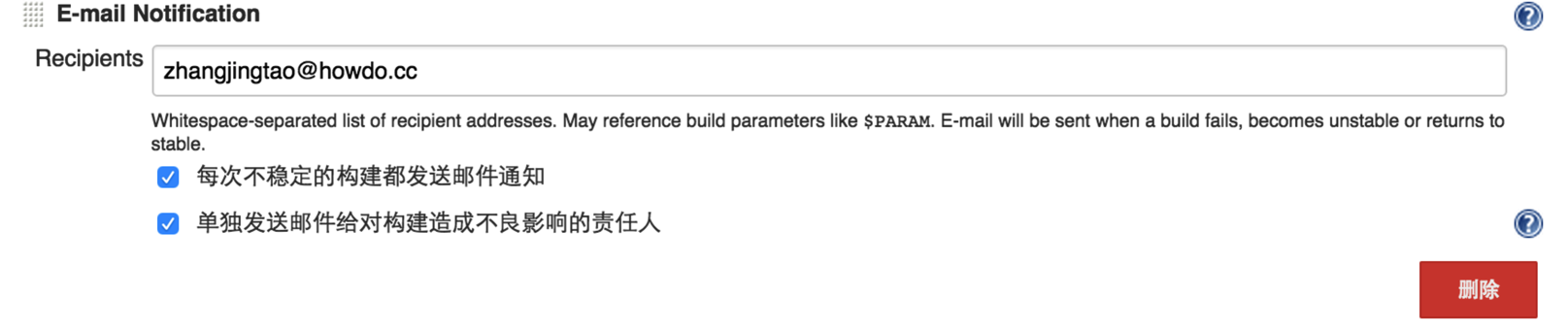
**根据需要进行配置。**

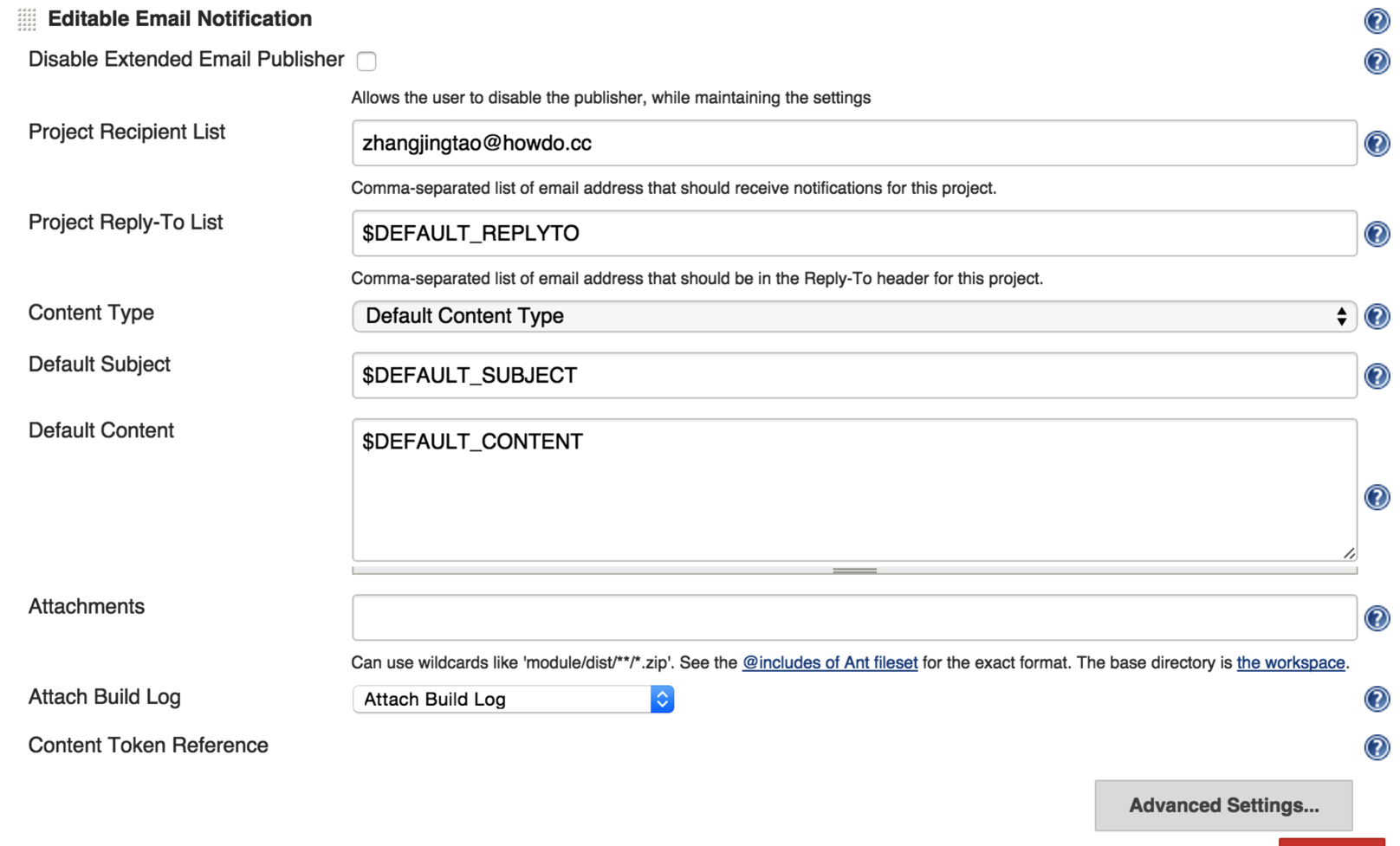
**设置构建时需要执行的脚本：**

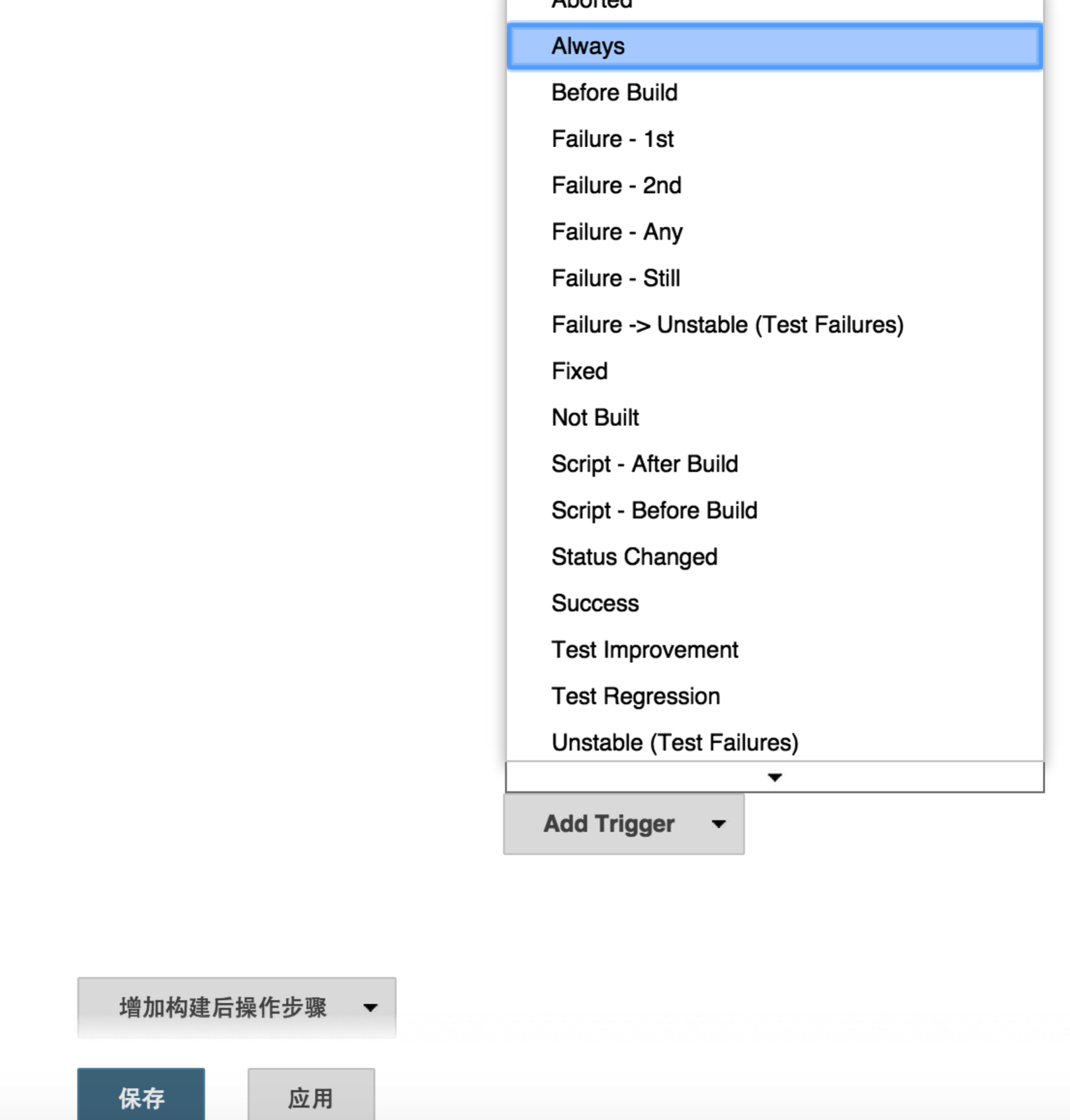
****

**根据不同的构建项目进行Shell 脚本的编写，这里的 Build shell脚本参见附录1 :build\_xcode.sh。**

**设置邮件通知：**

****

****



**必须对邮件增加触发条件，这里增加是Always.**

**保存配置修改。**

1. **开始构建**

****

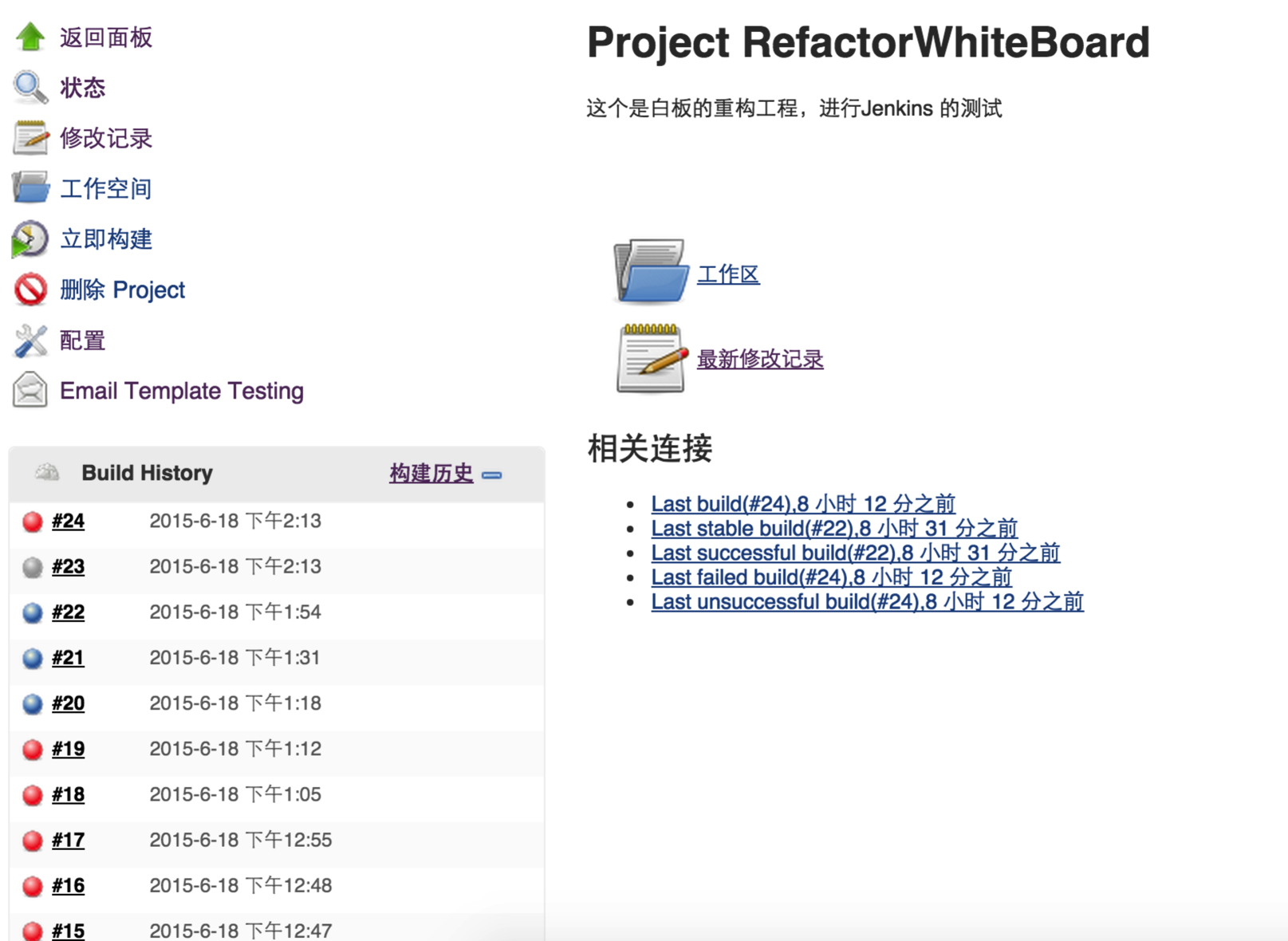
**选择立即构建**

**或者从主界面：**

****

**最右面的图片按钮。**

1. **获取结果**

****

**蓝色代表成功，红色失败，点击进去可以看失败的log，灰色代表取消。**

1. **版本自动更新：**

**参见附录2：version.sh.**

**附录1：**

**build\_xcode.sh**

#!/bin/sh

#

# build\_xcode.sh

#

# How to build Project in batch way:

# 1. build\_xcode.sh debug

# 2. build\_xcode.sh release

if [ $# -eq 0 ]; then

echo "Usage: build\_xcode.sh [debug|release]"

exit 1

fi

DEPLOYMENT\_TARGET=8.0

SCRIPT\_DIR=$(pwd -P)

Project\_ROOT=${SCRIPT\_DIR}/..

BUILD\_TYPE=$1

DEVICE="iphonesimulator$SDK\_VERSION"

if [ "$BUILD\_TYPE" == "debug" ]; then

BUILD\_TYPE="Debug"

DEVICE="iphonesimulator$SDK\_VERSION"

elif [ "$BUILD\_TYPE" == "release" ]; then

BUILD\_TYPE="Release"

DEVICE="iphoneos$SDK\_VERSION"

else

echo "input debug or release to make the build."

exit 1

fi

# The pathes of projects

WHITEBOARD\_PROJ="${SCRIPT\_DIR}/RefactorWhiteBoard.xcodeproj"

ALL\_PROJECTS="$WHITEBOARD\_PROJ"

#python ./version.py $Project\_ROOT/.

echo "DEVICE = $DEVICE"

echo "-= Begin Clean =-"

for PROJ in $ALL\_PROJECTS

do

echo "--= Clean $PROJ =--"

xcodebuild -project $PROJ clean > /dev/null

done

echo "-= Clean Complete =-"

echo

echo

echo "-= Begin Build =-"

for PROJ in $ALL\_PROJECTS

do

echo "--= Build $PROJ $BUILD\_TYPE =--"

xcodebuild -project $PROJ -configuration $BUILD\_TYPE -sdk $DEVICE IPHONEOS\_DEPLOYMENT\_TARGET=${DEPLOYMENT\_TARGET} build

if [ $? -ne 0 ]; then

echo "Failed to build project: $PROJ..."

exit 1

fi

done

echo "-= Build Complete =-"

echo

echo

exit 0

**附录2：**

**version.sh**

if [ "${CONFIGURATION}" = "Release" ]; then

VERSIONNUM=$(/usr/libexec/PlistBuddy -c "Print CFBundleVersion" "${PROJECT\_DIR}/${INFOPLIST\_FILE}")

NEWSUBVERSION=`echo $VERSIONNUM | awk -F "." '{print $3}'`

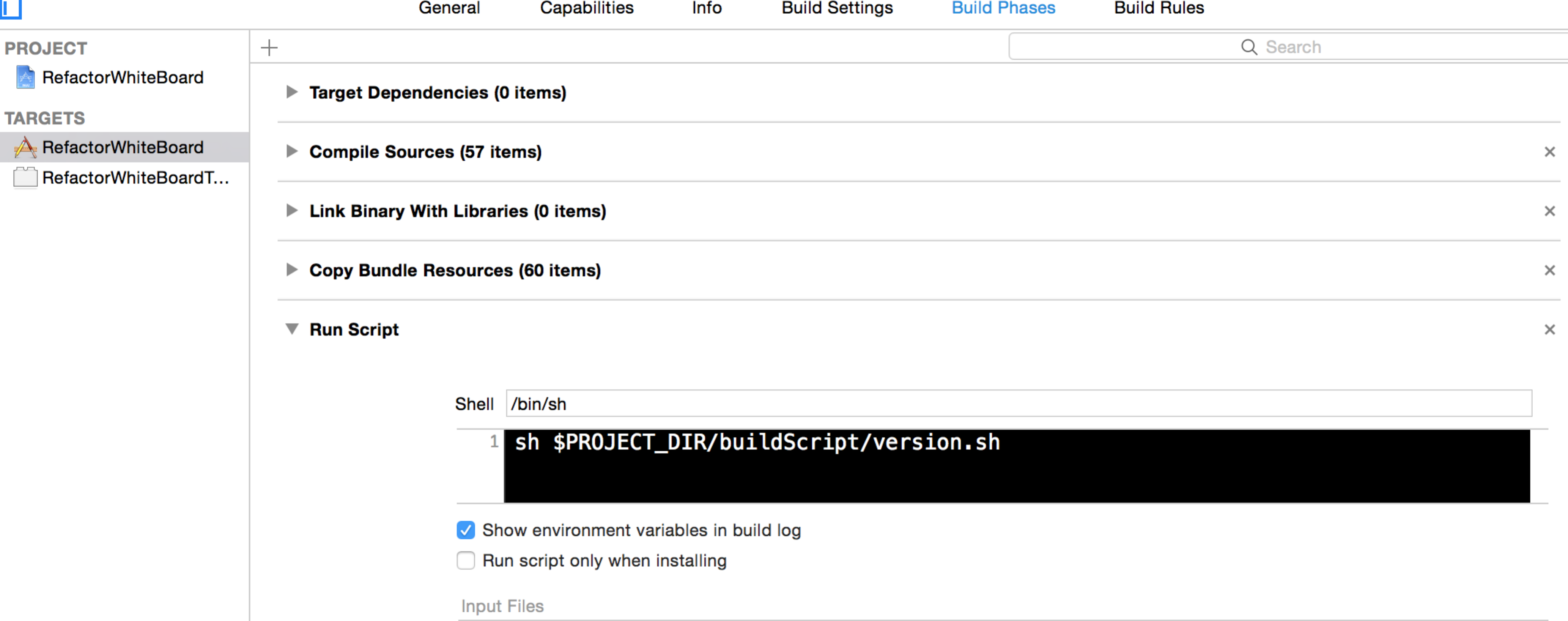
NEWSUBVERSION=$(($NEWSUBVERSION + 1))

NEWVERSIONSTRING=`echo $VERSIONNUM | awk -F "." '{print $1 "." $2 ".'$NEWSUBVERSION'"}'`

/usr/libexec/PlistBuddy -c "Set :CFBundleVersion $NEWVERSIONSTRING" "${PROJECT\_DIR}/${INFOPLIST\_FILE}"

fi

**version.sh 进行每次 Build 后的版本自增，需要在xcode 中进行脚本执行：**

****