**IDEA+Gatling+Scala进行性能测试**

**搭建Intellij IDEA+Scala SDK+Scala插件开发环境**

参考

<https://gatling.io/docs/current/general/>

源码

<https://github.com/gatling/gatling>

<https://cloud.tencent.com/developer/news/2130>

[https://blog.csdn.net/u014612889/article/details/102699957](https://blog.csdn.net/u014612889/article/details/102699957" \t "_blank)

<https://testerhome.com/topics/20001>

<https://www.jianshu.com/p/90afbd06b69a>

**gradle配置环境变量**

#永久配置方法 sudo vi /etc/bash.bashrc source /etc/bash.bashrc

# 配置java环境变量 /usr/lib/jvm/java-11-openjdk-amd64/bin /opt/java/jdk-11.0.7

export JAVA\_HOME=/opt/java/jdk-11.0.7

#export JAVA\_HOME=/opt/java/java-1.8.0-openjdk-amd64

export CLASS\_PATH=.:$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

export JRE\_HOME=$JAVA\_HOME/jre

#gradle-6.2.2

export GRADLE\_HOME=/media/zgq/data/code/github.com/stress-test/gatling-api-test/gradle/gradle-6.2.2

export PATH=$GRADLE\_HOME/bin:$PATH

zgq@zgq-PC:/opt/java$ gradle -v

------------------------------------------------------------

Gradle 6.2.2

------------------------------------------------------------

Build time: 2020-03-04 08:49:31 UTC

Revision: 7d0bf6dcb46c143bcc3b7a0fa40a8e5ca28e5856

Kotlin: 1.3.61

Groovy: 2.5.8

Ant: Apache Ant(TM) version 1.10.7 compiled on September 1 2019

JVM: 11.0.7 (Eclipse OpenJ9 openj9-0.20.0)

OS: Linux 5.4.50-amd64-desktop amd64

**scala-环境变量配置**

下载scala开发工具包

[https://downloads.lightbend.com/scala/2.13.3/scala-2.13.3.deb](https://downloads.lightbend.com/scala/2.13.3/scala-2.13.3.deb" \t "_blank)

# 配置scala-sdk  
#export SCALA\_HOME=/media/zgq/data/code/github.com/scala/scala-2.13.3  
export SCALA\_HOME=/usr  
export PATH=${SCALA\_HOME}/bin:$PATH

环境变量配置 java1.8 &scala2.12版本兼容性

scala-sdk-2.12.7

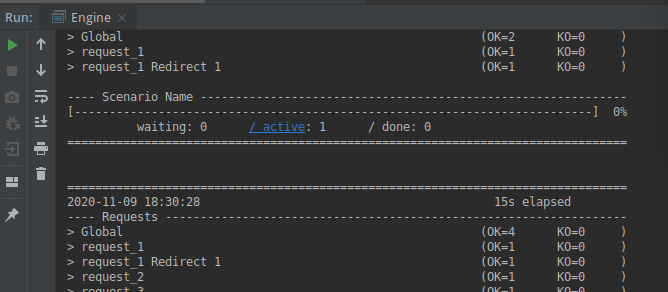
Gatling部署

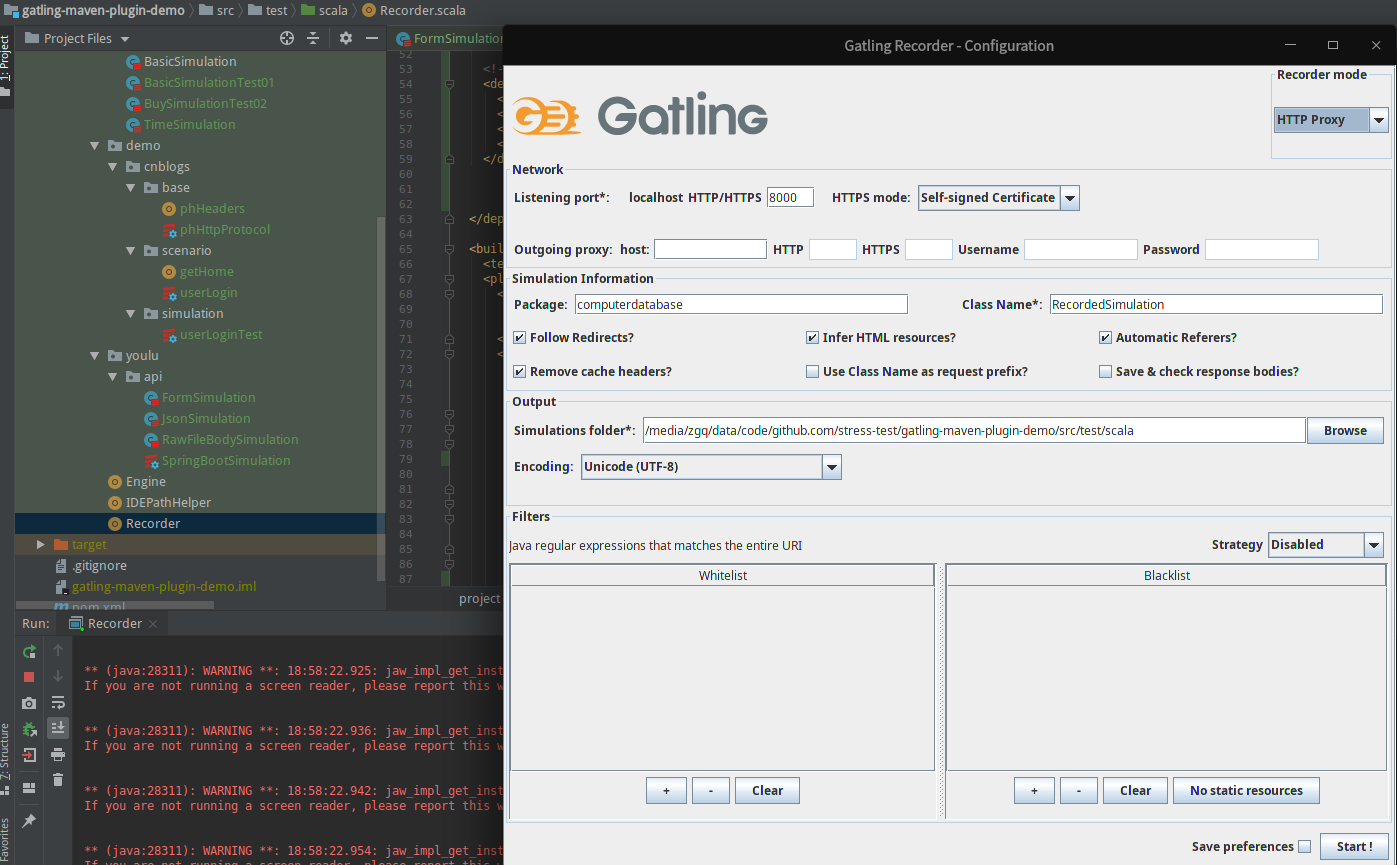
#gatling配置

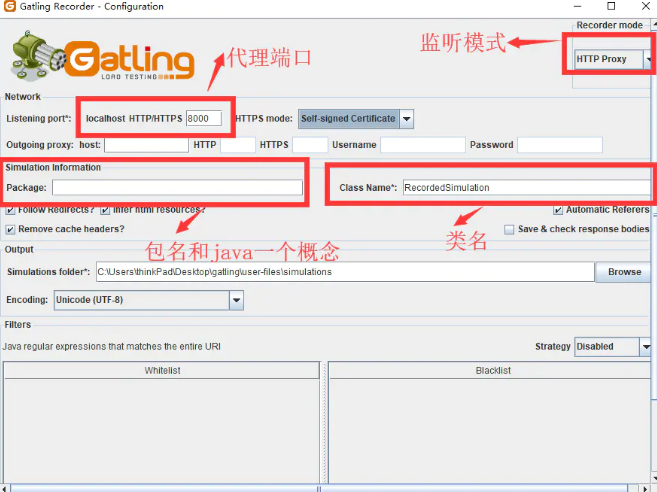
export GATLING\_HOME=/media/zgq/data/code/github.com/stress-test/gatling-charts-highcharts-bundle-3.3.1

export PATH=${GATLING\_HOME}/bin:$PATH

/media/zgq/data/code/github.com/stress-test/gatling-charts-highcharts-bundle-3.3.1/bin  
./bin/gatling.sh  
选0 连选择两次－－执行制定序号的用例







**scala的脚本demo**

package computerdatabase

import io.gatling.core.Predef.\_

import io.gatling.http.Predef.\_

import scala.concurrent.duration.\_

class lllll extends Simulation {

val httpProtocol = http

.baseUrl("http://13.19.21.0:58081") // Here is the root for all relative URLs

.acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8") // Here are the common headers

.doNotTrackHeader("1")

.acceptLanguageHeader("en-US,en;q=0.5")

.acceptEncodingHeader("gzip, deflate")

.userAgentHeader("Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/16.0")

val scn = scenario("Scenario Name").repeat(1){

// A scenario is a chain of requests and pauses

exec(http("request\_1") //自定义请求名称，用于报表展示

.get("/api/calculating-server/") //请求这个接口

) }

.pause(7) // Note that Gatling has recorded real time pauses 等待7秒

//设置线程数量

setUp(scn.inject(atOnceUsers(1)).protocols(httpProtocol))

}

package test.scala.cases

import io.gatling.core.Predef.\_

import io.gatling.http.Predef.\_

class BaiduSimulation extends Simulation {

// 设置请求的根路径

val httpConf = http.baseUrl("https://www.baidu.com")

/\*

运行10秒 during 默认单位秒,如果要用微秒 during(100 millisecond)

\*/

val scn = scenario("BaiduSimulation").during(1){

exec(http("baidu\_home").get("/"))

}

// 设置线程数

setUp(scn.inject(atOnceUsers(1000)).protocols(httpConf))

}

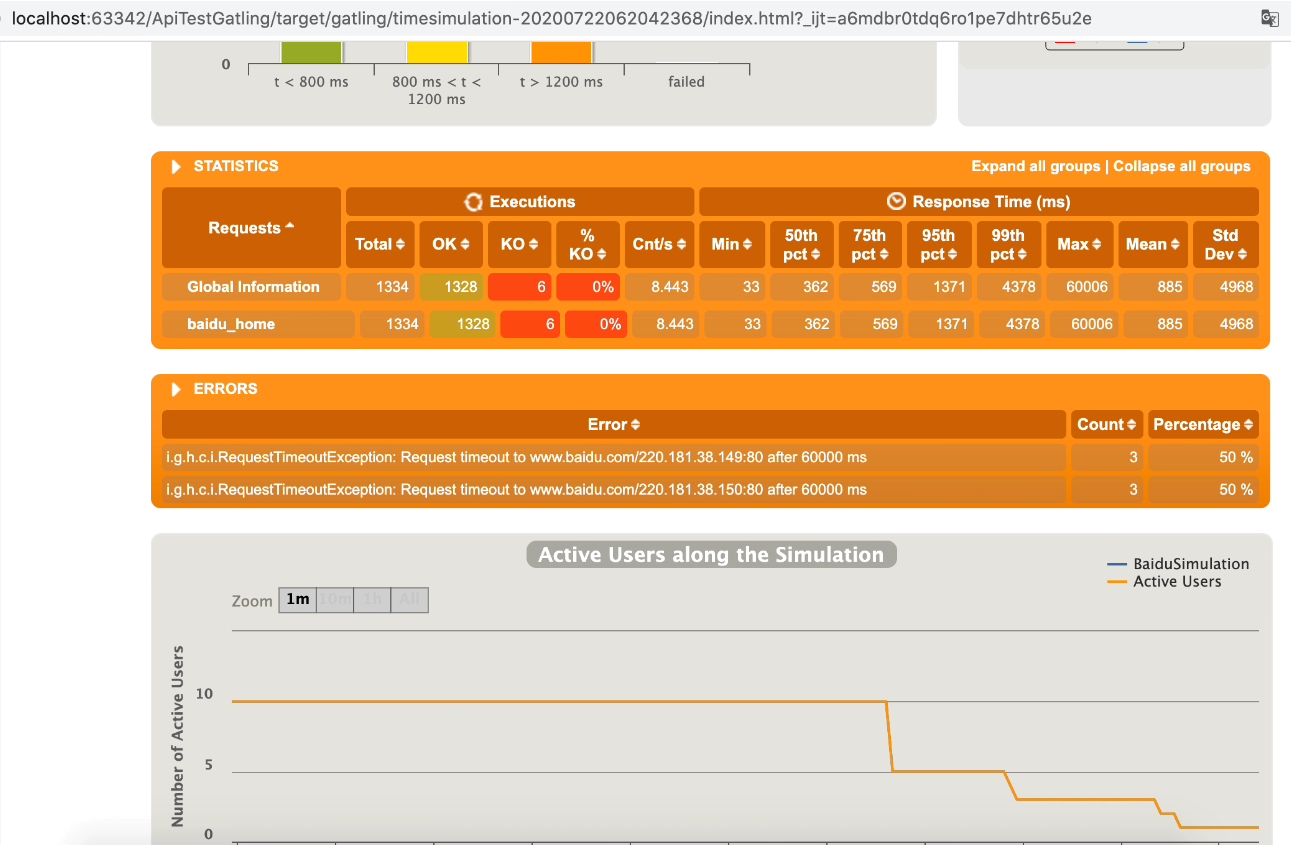
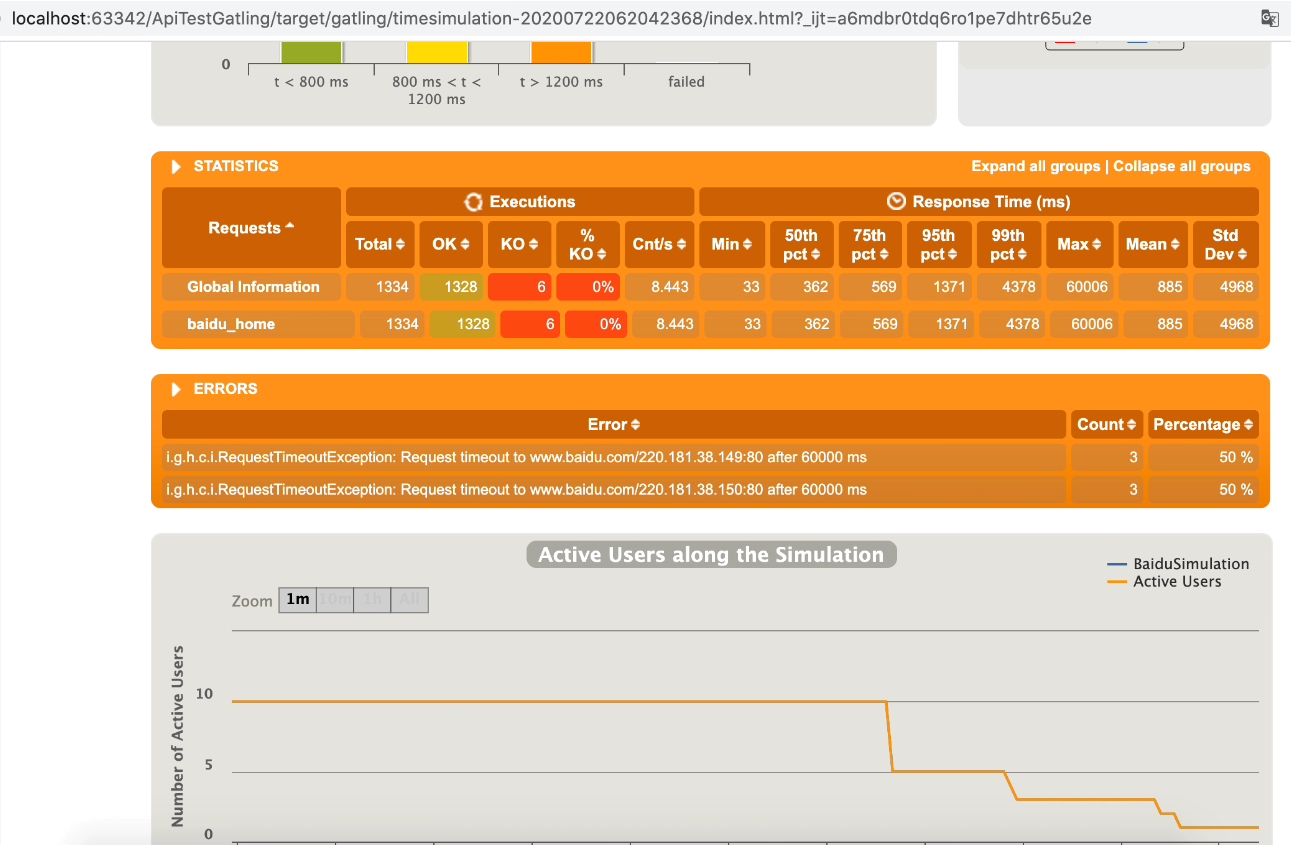
# **Gatling教程系列一实现POST请求测试(四)**

**gatling部署**

下载地址

[https://repo1.maven.org/maven2/io/gatling/highcharts/gatling-charts-highcharts-bundle/3.3.1/gatling-charts-highcharts-bundle-3.3.1-bundle.zip](https://repo1.maven.org/maven2/io/gatling/highcharts/gatling-charts-highcharts-bundle/3.3.1/gatling-charts-highcharts-bundle-3.3.1-bundle.zip" \t "_blank)

**生成测试报告**



**拓展学习**

[https://www.cnblogs.com/sanshengshui/p/9750478.html](https://www.cnblogs.com/sanshengshui/p/9750478.html" \t "_blank)

[http://gatling.io/docs/current/cheat-sheet/](http://gatling.io/docs/current/cheat-sheet/" \t "_blank)

[http://lihuazhang.github.io/2013/09/14/2013-09-14-gatling-shi-yong/](http://lihuazhang.github.io/2013/09/14/2013-09-14-gatling-shi-yong/" \t "_blank)

[http://gatling.io/docs/current/quickstart/](http://gatling.io/docs/current/quickstart/" \t "_blank)

**微服务dubbo扩展支持**

[https://github.com/youzan/gatling-dubbo](https://github.com/youzan/gatling-dubbo" \t "_blank)

[https://tech.youzan.com/gatling-dubbo/](https://tech.youzan.com/gatling-dubbo/" \t "_blank)

拓展参考：

[https://testerhome.com/topics/21438?locale=zh-CN](https://testerhome.com/topics/21438?locale=zh-CN" \t "_blank)

blazemeter 基本上就是压测行业的领头羊，多去他家的网站看看没毛病。

本文参考自：

[https://gatling.io/docs/2.3/general/simulation\_setup/](https://gatling.io/docs/2.3/general/simulation_setup/" \t "_blank)

[https://developer.rackspace.com/blog/Introduction-to-load-and-performance-testing-with-gatling-tool/](https://developer.rackspace.com/blog/Introduction-to-load-and-performance-testing-with-gatling-tool/" \t "_blank)

[https://stackshare.io/stackups/blazemeter-vs-gatling#description](https://stackshare.io/stackups/blazemeter-vs-gatling" \l "description" \t "_blank) 这个网站很有趣

Gatling是一个开源的性能测试框架，它是基于Scala, Akka and Netty开发的。

这三个家伙是什么，你可以不用关心。

Gatling 支持的是http协议，所以可以用来压测任何http-server。但是其他的协议，你就要额外想办法了。

不同于Jmeter使用xml文件控制压测场景，Gatling通过Scala脚本进行业务编排。

一般情况下可以通过录制获取Scala测试脚本，当然也可以完全手写Scala脚本。

下面我们揭开Scala脚本的神秘面纱。

这些问题好难啊O2O

我在想利用docker能不能实现完美的分布式Gatling集群呢？

怎么去利用zookeeper成为注册中心呢？

我怎么可以监听每个agent的性能数据呢？

多个agent之前的数据怎么汇总呢？

怎么和infludb进行集成呢？

怎么和Jenkins工具集成呢？

我的其他性能测试系列文章

Jmeter系列

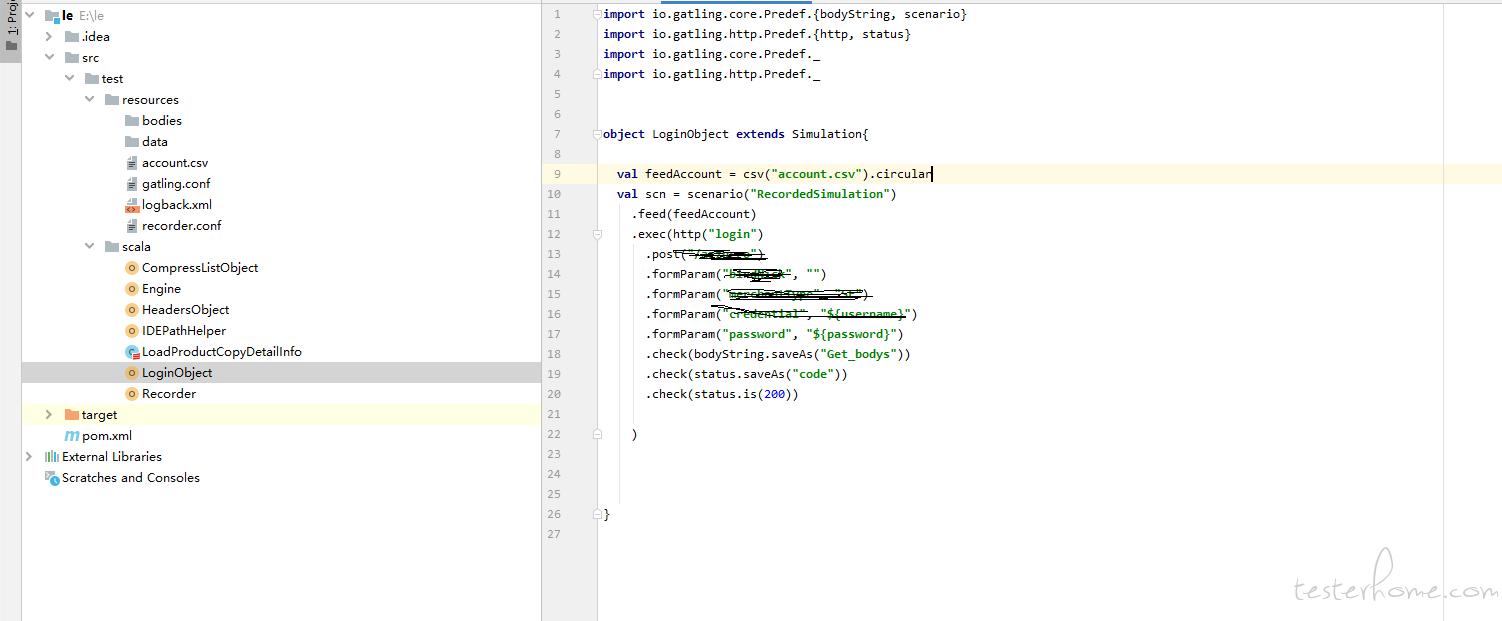
JMeter性能测试工具简介 [https://blog.csdn.net/lineuman/article/details/95235990](https://blog.csdn.net/lineuman/article/details/95235990" \t "_blank)

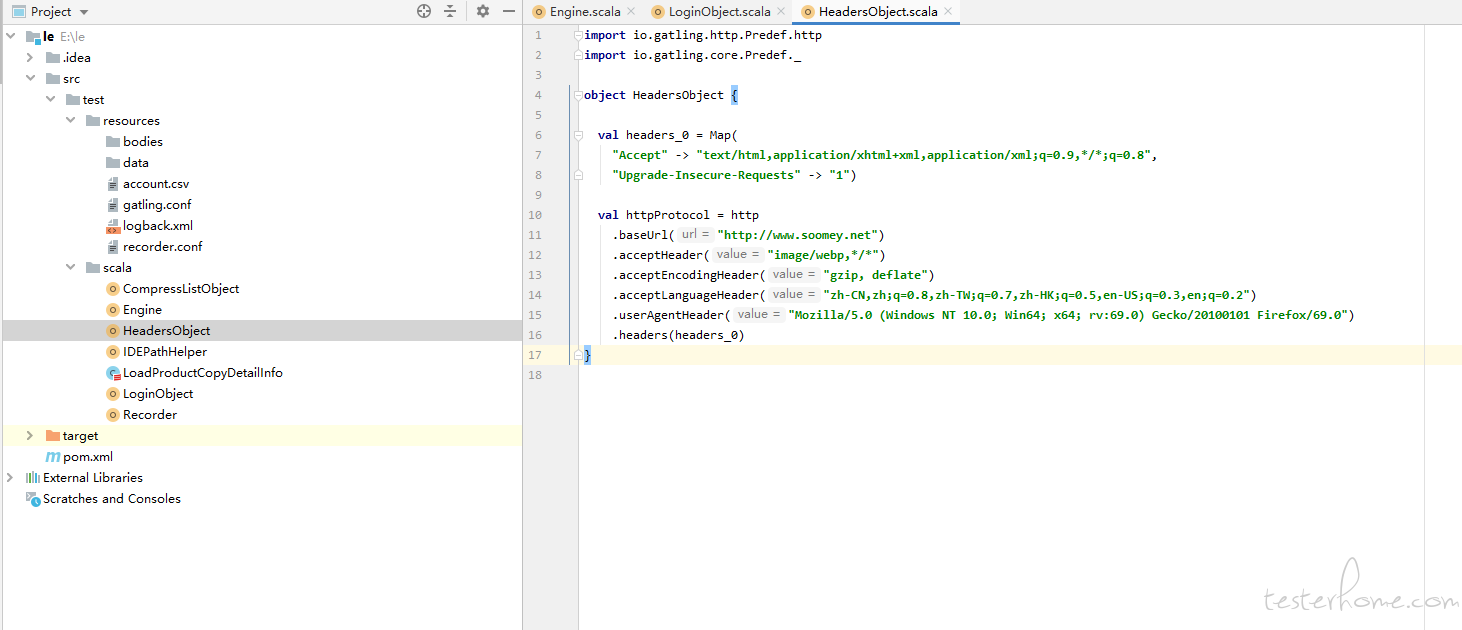
Jmeter获取性能指标 [https://blog.csdn.net/lineuman/article/details/94413092](https://blog.csdn.net/lineuman/article/details/94413092" \t "_blank)

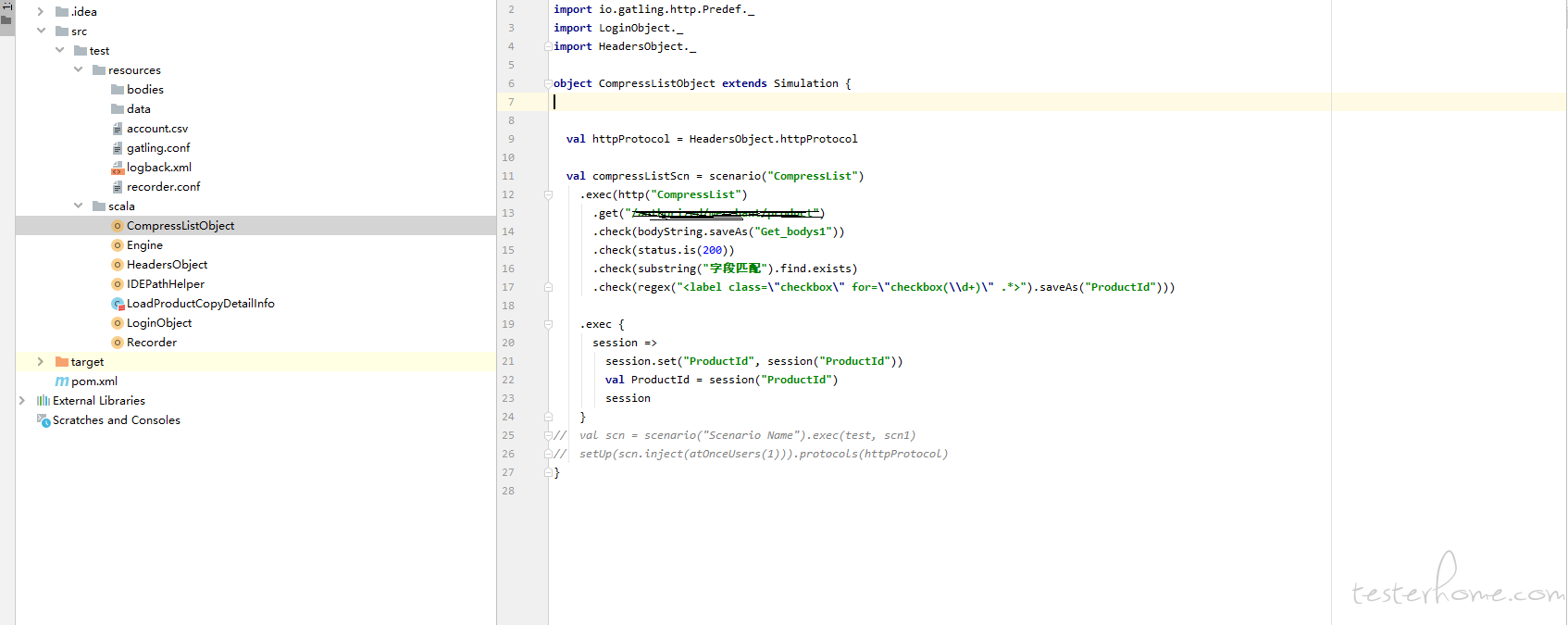
Gatling系列

Gatling性能测试工具使用简介 [https://blog.csdn.net/lineuman/article/details/94759198](https://blog.csdn.net/lineuman/article/details/94759198" \t "_blank)

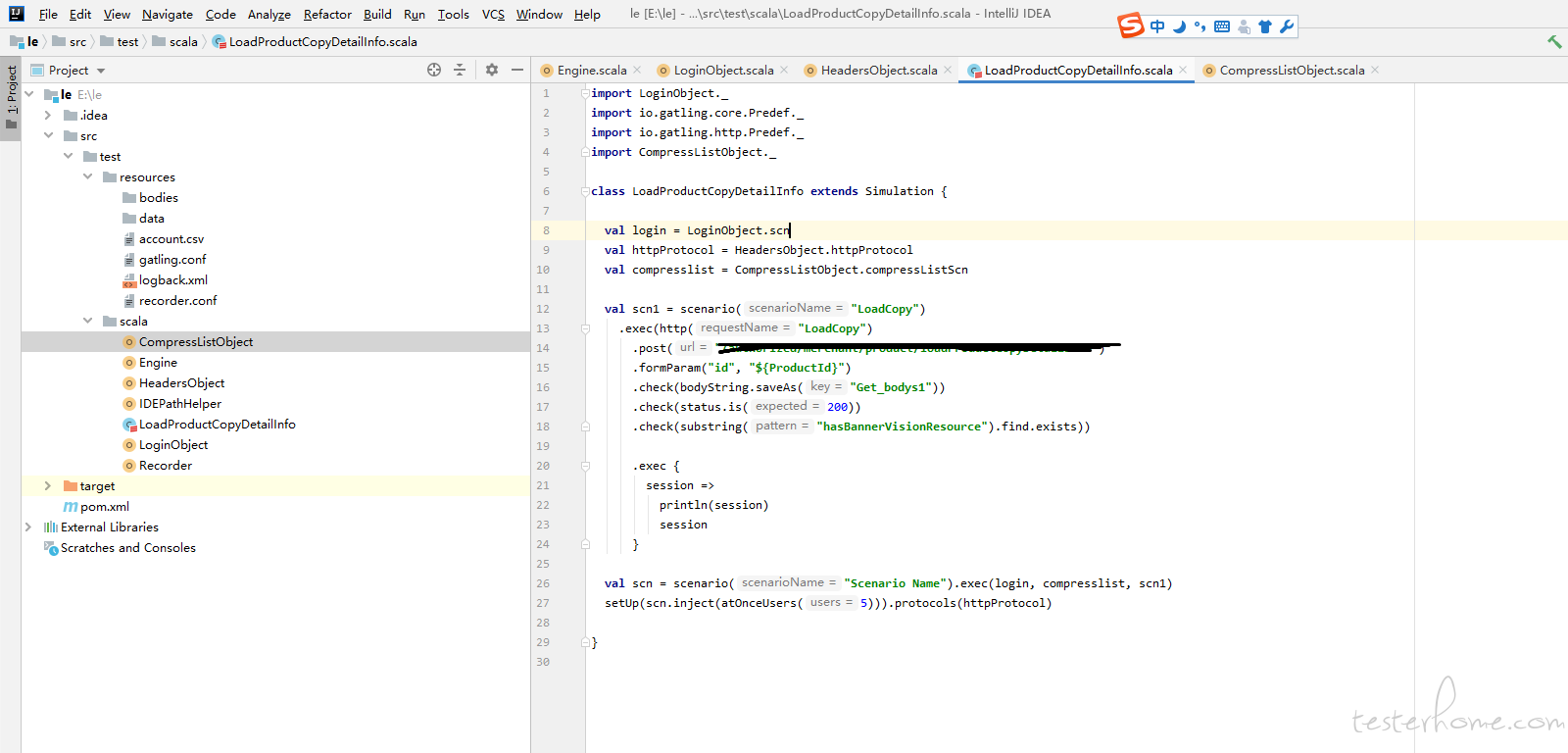
gatling相比jmeter能更好的支持高并发，我的本地环境是由gatling+scala+intellij组成，当时搭建环境也费了不少时间。阿弥陀佛总算是搭建成功了。gatling脚本主要由三部分组成，网上最多的也是这部分的资料，就不过多讲解了。  
最开始一直卡在cookie那一块，结果都不是问题。gatling会自动应用cookie，也就是说主要你登录了之后，其他后续的脚本执行的时候会自动带上cookie，不用人为再去获取cookie塞进去。当时不知道，直到看到官方文档中的一句话才知道原来是这样，这个坑耽误了很多时间。  
还有一个就是分层的思想，和ui自动化的po模式类似。把一些公用的方法单独抽出来写成object class，再由其他脚本调用

[](https://testerhome.com/uploads/photo/2020/85eb806c-34e6-4d4c-ad22-da68e5e8dcd9.png!large)

[](https://testerhome.com/uploads/photo/2020/6ac872aa-792f-4fd9-84ba-807982db317f.png!large)

[](https://testerhome.com/uploads/photo/2020/426be551-f3bd-4baa-87a3-06efed58c61c.png!large)

像我图片上显示的，我就是把headers请求头的内容和登录的内容单独抽出来，其中还有通过正则表达式取到id给其他脚本用的脚本也单独抽出来写成了object class，然后要执行的测试脚本中直接引用这些公用的模块，执行用例，省了很多重复编写的代码，易于理解。  
如下图片就是调用一个接口并且引用那个接口返回的id，入参。

[](https://testerhome.com/uploads/photo/2020/9307f551-7e6f-42d1-9827-acd7f35acb71.png!large)

折腾了那么多天，总算能实现接口间的调用和不同用户的并发了，下一步接近用maven运行所有用例的问题，有问题再记录

### **1. Gatling 介绍**

Gatling是一款基于Scala 开发的高性能服务器性能测试工具，它主要用于对服务器进行负载等测试，并分析和测量服务器的各种性能指标。Gatling主要用于测量基于HTTP的服务器，比如Web应用程序，RESTful服务等。

### **2. Gatling 优势**

1. 使用Akka Actors和Async IO，并发测试性能更高
2. 基于一套开源的Gatling DSL API，使用Scala代码，脚本易维护
3. 支持录制并生成测试脚本，从而可以方便的生成测试脚本
4. 支持使用IntelliJ，便于开发
5. 支持实时生成Html动态轻量报表，从而使报表更易阅读和进行数据分析
6. 支持Jenkins，以便于进行持续集成

### **3. Gatling 教程目录(持续更新)**

[Gatling 教程一：快速开始 Gatling 的使用](https://xiuxiuing.gitee.io/blog/2018/09/18/gatlinguse/" \t "/home/zgq/Documents\\x/_blank)：[https://testerhome.com/articles/16529](https://testerhome.com/articles/16529" \t "/home/zgq/Documents\\x/_blank)  
[Gatling 教程二：配置 IntelliJ 开发环境](https://xiuxiuing.gitee.io/blog/2018/09/18/gatlingintellij/" \t "/home/zgq/Documents\\x/_blank)：[https://testerhome.com/articles/16530](https://testerhome.com/articles/16530" \t "/home/zgq/Documents\\x/_blank)  
[Gatling 教程三：使用脚本录制功能](https://xiuxiuing.gitee.io/blog/2018/09/18/gatlingrecorder/" \t "/home/zgq/Documents\\x/_blank)：[https://testerhome.com/articles/16531](https://testerhome.com/articles/16531" \t "/home/zgq/Documents\\x/_blank)  
[Gatling 教程四：测试脚本编写及 API 介绍](https://xiuxiuing.gitee.io/blog/2018/09/18/gatlingcode/" \t "/home/zgq/Documents\\x/_blank)：[https://testerhome.com/articles/16532](https://testerhome.com/articles/16532" \t "/home/zgq/Documents\\x/_blank)

有什么问题，欢迎在评论区交流，如果你觉得还不错，欢迎[打赏支持]  
可以加入 [Gatling交流会](https://testerhome.com/gatling" \t "/home/zgq/Documents\\x/_blank), 一起交流，共同进步

# [Gatling教程](https://testerhome.com/columns/Gatling" \o "Gatling教程) Gatling 教程一：快速开始 Gatling 的使用

### **1. 介绍**

Gatling是一款基于Scala 开发的高性能服务器性能测试工具，它主要用于对服务器进行负载等测试，并分析和测量服务器的各种性能指标。目前仅支持http协议，可以用来测试web应用程序和RESTful服务。

除此之外它拥有以下特点：

* 支持Akka Actors 和 Async IO，从而能达到很高的性能
* 支持实时生成Html动态轻量报表，从而使报表更易阅读和进行数据分析
* 支持DSL脚本，从而使测试脚本更易开发与维护
* 支持录制并生成测试脚本，从而可以方便的生成测试脚本
* 支持导入HAR（Http Archive）并生成测试脚本
* 支持Maven，Eclipse，IntelliJ等，以便于开发
* 支持Jenkins，以便于进行持续集成
* 支持插件，从而可以扩展其功能，比如可以扩展对其他协议的支持
* 开源免费

### **2. 使用**

下载解压即可使用：[http://gatling.io/#/download](http://gatling.io/" \l "/download" \t "/home/zgq/Documents\\x/_blank)  
解压后目录结构：

├── bin  
├── conf  
├── gatling-tests  
├── lib  
├── results  
├── target  
└── user-files

### **3. 文件目录介绍**

* bin目录下有2个脚本，gatling和recorder， gatling用来运行测试， recorder用来启动录制脚本的UI的（不推荐使用）。
* conf目录是关于Gatling自身的一些配置。
* lib目录是Gatling自身依赖的库文件。
* results目录用来存放测试报告的。
* user-files目录是用来存放测试脚本的。
* target: 你启动运行组件后，gatling会为你编译好所有的.scala脚本，而编译后的class文件就会在这里;

当运行gating脚本的时候，其会扫描user-files目录下的所有文件，列出其中所有的Simulation(一个测试类，里面可以包含任意多个测试场景)。选择其中一个Simulation，然后填写Simulation ID和运行描述，这个都是为报告描述服务的。

### **4. Demo 运行**

在Gatling的下载包里默认包含了一个demo示例，  
在项目目录下执行shell命令：

/work/gatling-2.3.1 $ ./bin/gatling.sh

Gatling 会遍历user-files，列出所有的Simulation

GATLING\_HOME is set to /work/gatling-2.3.1  
Choose a simulation number:  
 [0] computerdatabase.BasicSimulation  
 [1] computerdatabase.advanced.AdvancedSimulationStep01  
 [2] computerdatabase.advanced.AdvancedSimulationStep02  
 [3] computerdatabase.advanced.AdvancedSimulationStep03  
 [4] computerdatabase.advanced.AdvancedSimulationStep04  
 [5] computerdatabase.advanced.AdvancedSimulationStep05

这里我们在终端中输入 0，代表选择computerdatabase.BasicSimulation执行，  
之后输入两次 回车，就可以开始执行了，执行完成会在results目录下生成网页报告。

Reports generated in 0s.  
Please open the following file: /work/gatling-2.3.1/results/basicsimulation-1535978730687/index.html

# [Gatling教程](https://testerhome.com/columns/Gatling" \o "Gatling教程) Gatling 教程二：配置 IntelliJ 开发环境

### **1. 安装IntelliJ**

进入官网下载安装 IntelliJ IDEA：[https://www.jetbrains.com/idea/](https://www.jetbrains.com/idea/" \t "/home/zgq/Documents\\x/_blank)  
IntelliJ IDEA 分为两个版本，分别为：

* 旗舰版（Ultimate）
* 社区版（Community）

旗舰版是收费的，社区版则是免费的。至于两者的区别嘛，就是旗舰版比社区版的功能更为齐全！如果你想用 IntelliJ IDEA 进行大型项目开发的的话，啥也别说了，下载旗舰版就没错了；如果你只是想体验一下开发的乐趣，下载社区版就足够了。  
旗舰版破解教程：[传送门](https://xiuxiuing.gitee.io/blog/2018/10/17/intellijidea/" \t "/home/zgq/Documents\\x/_blank)

### **2. 配置Maven环境**

[Maven安装教程](https://www.yiibai.com/maven/maven_environment_setup.html" \t "/home/zgq/Documents\\x/_blank)  
也可以使用IntelliJ自带的Maven  
一般不配置镜像的话，下载lib包的速度很慢，找到Maven目录中的settings.xml，  
在<mirrors></mirrors>中添加如下内容，保存：

<mirror>  
 <id>alimaven</id>  
 <name>aliyun maven</name>  
 <url>http:*//maven.aliyun.com/nexus/content/groups/public/</url>*  
 <mirrorOf>central</mirrorOf>   
</mirror>

### **3. 安装Scala**

打开 IDEA ，点击【IntelliJ IDEA】 -> 【Preferences】 -> 【Plugins】，搜索 “Scala”，搜索到插件然后点击底部的 【Install JetBrains plugin...】安装重启即可。

### **4. 新建Demo工程**

在 IntelliJ中选择 New Project -> Maven -> Create form archetype -> Add Archetype，在弹出框中输入一下内容：

GroupId: io.gatling.highcharts  
ArtifactId: gatling-highcharts-maven-archetype  
Version: 2.3.1

点击查看最新版本：[最新版本](https://search.maven.org/search?q=g:io.gatling.highcharts AND a:gatling-highcharts-maven-archetype&core=gav" \t "/home/zgq/Documents\\x/_blank)  
之后输入你项目的GroupId(包名)和ArtifactId(项目名)来完成项目创建，项目创建完成后，Maven会自动配置项目结构。  
项目目录说明：

* bodies：用来存放请求的body数据
* data：存放需要输入的数据
* scala：存放Simulation脚本
* Engine：右键运行跟运行 bin\gatling.bat 效果一致
* Recorder：右键运行跟运行 bin\recorder.bat 效果一致，录制的脚本存放在scala目录下
* target：存放运行后的报告

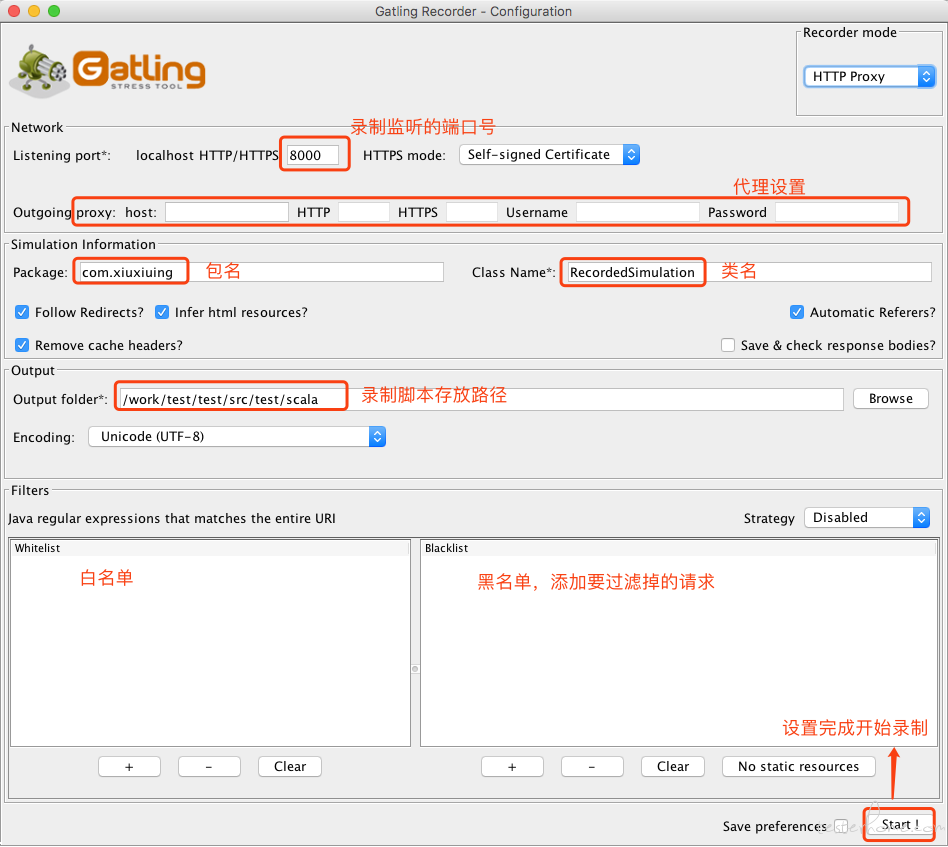
至此就可以使用IntelliJ愉快的开发啦。

# [Gatling教程](https://testerhome.com/columns/Gatling" \o "Gatling教程) Gatling 教程三：使用脚本录制功能

Gatling 提供了可视化的接口录制功能，我们无须写任何代码，通过录制功能可以自动生成性能测试所需要的脚本。

上篇文件如何配置IntelliJ的开发环境，本文将谈谈如何使用录制功能。

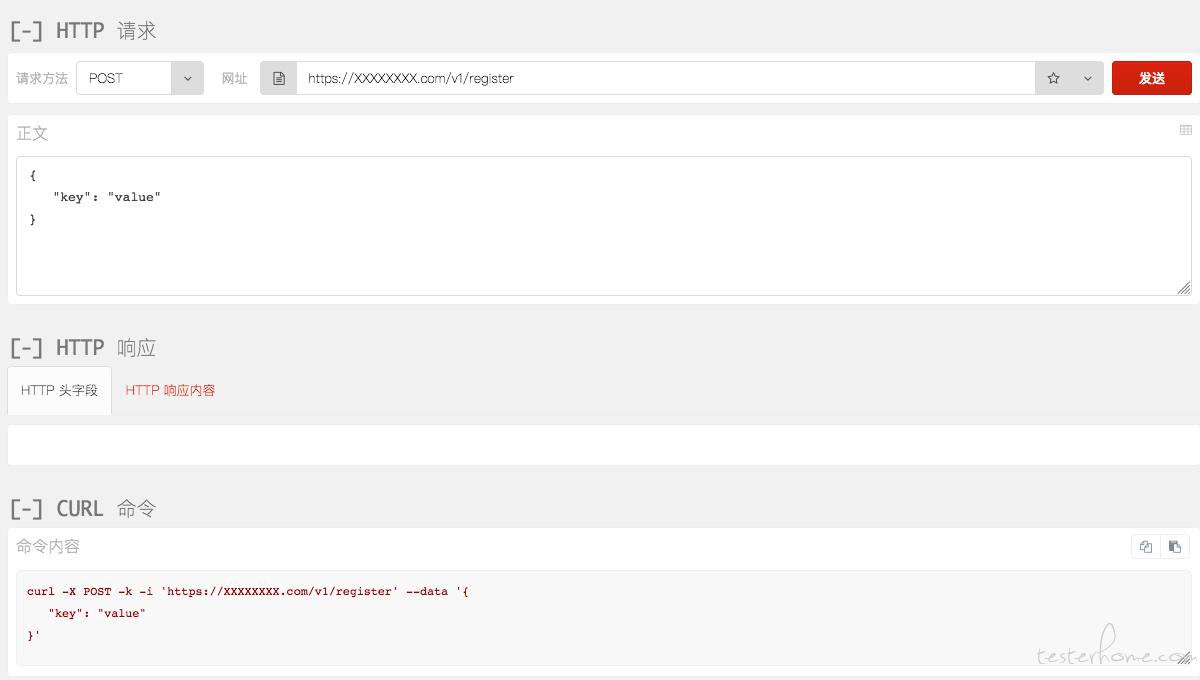
1. 在IntelliJ中选择Recorder文件右键运行，即可打开Gatling录制的界面，如果不是使用IntelliJ来开发，运行/bin/recorder.sh 命令也是可以启动录制界面的。

[](https://testerhome.com/uploads/photo/2018/0dd54b25-8f52-4d02-b9b7-fe36b2a21663.png!large)

1. 设置完成后，点击start，开始录制。
2. 在Firefox设置中，设置网络代理

[](https://testerhome.com/uploads/photo/2018/2b3b7e75-6660-4e30-bdef-6bed0624d959.png!large)

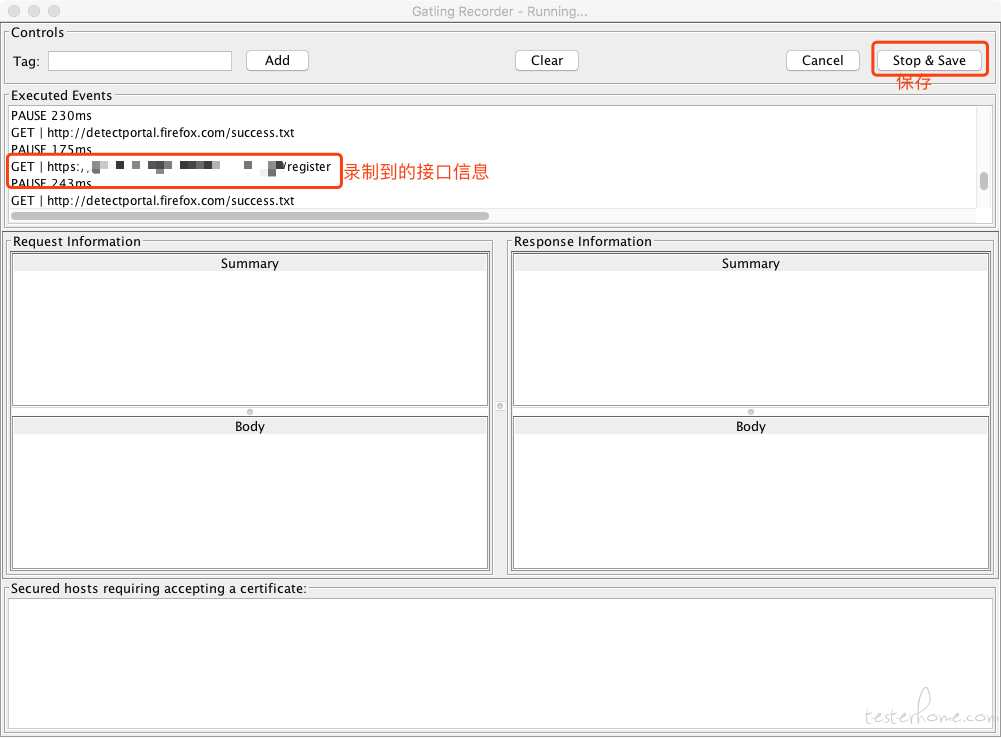
1. 使用类似PostMan的工具，模拟发送http请求

[](https://testerhome.com/uploads/photo/2018/98a1b091-2443-4fb3-9ef9-c5b7b00fdfc6.png!large)

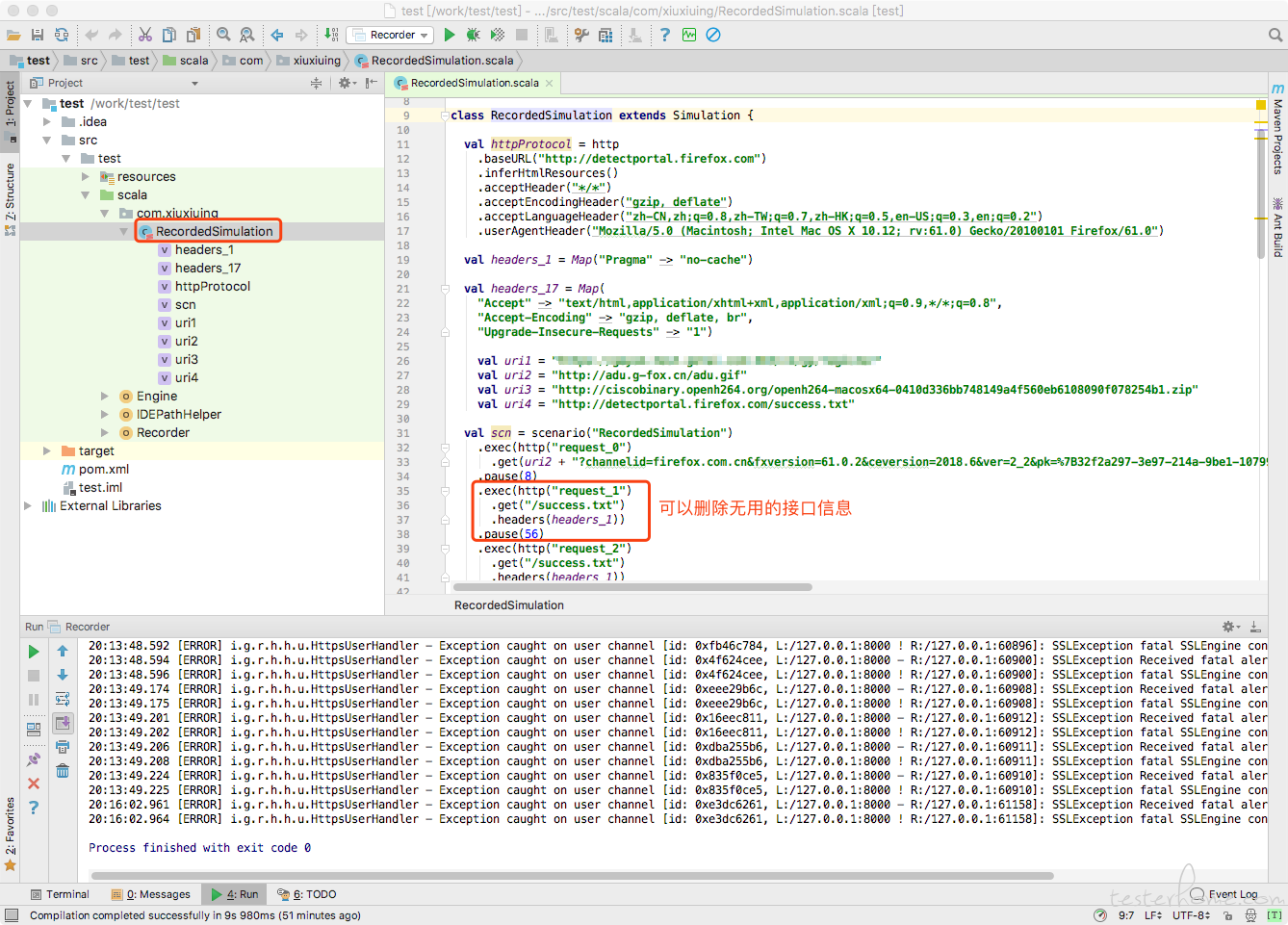
1. 如果使用的Https协议的话，需要添加例外

[](https://testerhome.com/uploads/photo/2018/8c4430aa-847c-437f-89a8-11dadce900b5.png!large)

1. Gatling客户端则会监听到接口请求，接口录制完成后，点击Stop & Save

[](https://testerhome.com/uploads/photo/2018/08c1bb10-d207-43da-818e-5080b9209665.png!large)

1. 在IntelliJ中可以看到录制完后自动生成的脚本

[](https://testerhome.com/uploads/photo/2018/13feb8b1-7f2a-4121-b7e1-6b43229254ec.png!large)

由于录制过程中会有我们不需要的请求，可以在脚本中删除。

点击Engine脚本右键即可运行我们录制的脚本了。

com.xiuxiuing.RecordedSimulation is the only simulation, executing it.  
Select simulation id (**default** is 'recordedsimulation'). Accepted characters are a-z, A-Z, 0-9, - and \_  
  
Select run description (optional)  
  
Simulation com.xiuxiuing.RecordedSimulation started...

# [Gatling教程](https://testerhome.com/columns/Gatling" \o "Gatling教程) Gatling 教程四：测试脚本编写及 API 介绍

Gatling基于Scala开发的压测工具，我们可以通过录制自动生成脚本，也可以自己编写脚本，大家不用担心，首先脚本很简单常用的没几个，另外gatling封装的也很好我们不需要去专门学习Scala语法，当然如果会的话会更好。

#### **脚本示例**

**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.http.Predef.\_**  
**import** **scala.concurrent.duration.\_**  
**class** BaiduSimulation **extends** Simulation {  
 *//设置请求的根路径*  
 val httpConf = http.baseURL("https://www.baidu.com")  
 */\*  
 运行100秒 during 默认单位秒,如果要用微秒 during(100 millisecond)  
 \*/*  
 val scn = scenario("BaiduSimulation").during(100){  
 exec(http("baidu\_home").get("/"))  
 }  
 *//设置线程数*  
*// setUp(scn.inject(rampUsers(500) over(10 seconds)).protocols(httpConf))*  
 setUp(scn.inject(atOnceUsers(10)).protocols(httpConf))  
}

#### **脚本编写**

Gatling脚本的编写主要包含下面三个步骤

* http head配置
* Scenario 执行细节
* setUp 组装

我们以百度为例，进行第一个GET请求测试脚本的编写，类必须继承 Simulation

1. 配置下head，只是简单的请求下百度首页，所以只定义下请求的base url，采用默认的http配置即可 //设置请求的根路径 val httpConf = http.baseURL("https://www.baidu.com")

声明Scenario，指定我们的请求动作

val scn = scenario("BaiduSimulation").  
exec(http("baidu\_home").get("/"))

scenario里的参数：scenario name  
exec()里的参数就是我们的执行动作，http("本次请求的名称").get("本次http get请求的地址")

设置并发数并组装

*//设置线程数*  
setUp(scn.inject(atOnceUsers(10)).protocols(httpConf))

atOnceUsers：立马启动的用户数，可以理解为并发数

这样我们一个简单的脚本就完成了，可以运行看下效果。

### **高级教程**

##### **Injection -- 注入**

注入方法用来定义虚拟用户的操作

setUp(  
 scn.inject(  
 nothingFor(4 seconds), *// 1*  
 atOnceUsers(10), *// 2*  
 rampUsers(10) over(5 seconds), *// 3*  
 constantUsersPerSec(20) during(15 seconds), *// 4*  
 constantUsersPerSec(20) during(15 seconds) randomized, *// 5*  
 rampUsersPerSec(10) to 20 during(10 minutes), *// 6*  
 rampUsersPerSec(10) to 20 during(10 minutes) randomized, *// 7*  
 splitUsers(1000) into(rampUsers(10) over(10 seconds)) separatedBy(10 seconds), *// 8*  
 splitUsers(1000) into(rampUsers(10) over(10 seconds)) separatedBy atOnceUsers(30), *// 9*  
 heavisideUsers(1000) over(20 seconds) *// 10*  
 ).protocols(httpConf)  
)

1. nothingFor(duration)：设置一段停止的时间
2. atOnceUsers(nbUsers)：立即注入一定数量的虚拟用户 setUp(scn.inject(atOnceUsers(50)).protocols(httpConf))
3. rampUsers(nbUsers) over(duration)：在指定时间内，设置一定数量逐步注入的虚拟用户 setUp(scn.inject(rampUsers(50) over(30 seconds)).protocols(httpConf))
4. constantUsersPerSec(rate) during(duration)：定义一个在每秒钟恒定的并发用户数，持续指定的时间 setUp(scn.inject(constantUsersPerSec(30) during(15 seconds)).protocols(httpConf))
5. constantUsersPerSec(rate) during(duration) randomized：定义一个在每秒钟围绕指定并发数随机增减的并发，持续指定时间 setUp(scn.inject(constantUsersPerSec(30) during(15 seconds) randomized).protocols(httpConf))
6. rampUsersPerSec(rate1) to (rate2) during(duration)：定义一个并发数区间，运行指定时间，并发增长的周期是一个规律的值 setUp(scn.inject(rampUsersPerSec(30) to (50) during(15 seconds)).protocols(httpConf))
7. rampUsersPerSec(rate1) to(rate2) during(duration) randomized：定义一个并发数区间，运行指定时间，并发增长的周期是一个随机的值 setUp(scn.inject(rampUsersPerSec(30) to (50) during(15 seconds) randomized).protocols(httpConf))
8. heavisideUsers(nbUsers) over(duration)：定义一个持续的并发，围绕和海维赛德函数平滑逼近的增长量，持续指定时间（译者解释下海维赛德函数，H(x)当x>0时返回1，x<0时返回0，x=0时返回0.5。实际操作时，并发数是一个成平滑抛物线形的曲线） setUp(scn.inject(heavisideUsers(50) over(15 seconds)).protocols(httpConf))
9. splitUsers(nbUsers) into(injectionStep) separatedBy(duration)：定义一个周期，执行injectionStep里面的注入，将nbUsers的请求平均分配 setUp(scn.inject(splitUsers(50) into(rampUsers(10) over(10 seconds)) separatedBy(10 seconds)).protocols(httpConf))
10. splitUsers(nbUsers) into(injectionStep1) separatedBy(injectionStep2)：使用injectionStep2的注入作为周期，分隔injectionStep1的注入，直到用户数达到nbUsers setUp(scn.inject(splitUsers(100) into(rampUsers(10) over(10 seconds)) separatedBy atOnceUsers(30)).protocols(httpConf))

##### **循环**

val scn = scenario("BaiduSimulation").  
 exec(http("baidu\_home").get("/"))

上面的测试代码运行时只能跑一次，为了测试效果，我们需要让它持续运行一定次数或者一段时间，可以使用下面两个方式：

* repeat repeat(times，counterName) times:循环次数 counterName:计数器名称，可选参数，可以用来当当前循环下标值使用，从0开始 val scn = scenario("BaiduSimulation").repeat(100){ exec(http("baidu\_home").get("/")) }
* during during(duration, counterName, exitASAP) duration:时长，默认单位秒，可以加单位milliseconds，表示毫秒 counterName:计数器名称，可选。很少使用 exitASAP：默认为true,简单的可以认为当这个为false的时候循环直接跳出,可在 循环中进行控制是否继续 /\* 运行100秒 during 默认单位秒,如果要用微秒 during(100 millisecond) \*/ val scn = scenario("BaiduSimulation").during(100){ exec(http("baidu\_home").get("/")) }

##### **POST请求**

post参数提交方式：

JSON方式

**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.core.scenario.Simulation**  
**import** **io.gatling.http.Predef.\_**  
**class** JsonSimulation **extends** Simulation {  
val httpConf = http.baseURL("http://127.0.0.1:7001/tst")  
*//注意这里,设置提交内容type*  
val headers\_json = Map("Content-Type" -> "application/json")  
val scn = scenario("json scenario")  
.exec(http("test\_json") *//http 请求name*  
.post("/order/get") *//post url*  
.headers(headers\_json) *//设置body数据格式*  
*//将json参数用StringBody包起,并作为参数传递给function body()*  
.body(RawFileBody("request.txt"))  
setUp(scn.inject(atOnceUsers(10))).protocols(httpConf)  
}

Form方式

**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.http.Predef.\_**  
**class** FormSimulation **extends** Simulation {  
val httpConf = http  
.baseURL("http://computer-database.gatling.io")  
*//注意这里,设置提交内容type*  
val contentType = Map("Content-Type" -> "application/x-www-form-urlencoded")  
*//声明scenario*  
val scn = scenario("form Scenario")  
.exec(http("form\_test") *//http 请求name*  
.post("/computers") *//post地址, 真正发起的地址会拼上上面的baseUrl http://computer-database.gatling.io/computers*  
.headers(contentType)  
.formParam("name", "Beautiful Computer") *//form 表单的property name = name, value=Beautiful Computer*  
.formParam("introduced", "2012-05-30")  
.formParam("discontinued", "")  
.formParam("company", "37"))  
setUp(scn.inject(atOnceUsers(1)).protocols(httpConf))  
}

RawFileBody

**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.core.scenario.Simulation**  
**import** **io.gatling.http.Predef.\_**  
**class** JsonSimulation **extends** Simulation {  
val httpConf = http.baseURL("http://127.0.0.1:7001/tst")  
*//注意这里,设置提交内容type*  
val headers\_json = Map("Content-Type" -> "application/json")  
val scn = scenario("json scenario")  
.exec(http("test\_json") *//http 请求name*  
.post("/order/get") *//post url*  
.headers(headers\_json) *//设置body数据格式*  
*//将json参数用StringBody包起,并作为参数传递给function body()*  
.body(RawFileBody("request.txt"))  
setUp(scn.inject(atOnceUsers(10))).protocols(httpConf)  
}

txt的文件内容为JSON数据，存放目录/resources/bodies下

##### **Feed 动态参数**

Gatling对参数的处理称为Feeder[供料器]，支持主要有：

* 数组 val feeder = Array( Map("foo" -> "foo1", "bar" -> "bar1"), Map("foo" -> "foo2", "bar" -> "bar2"), Map("foo" -> "foo3", "bar" -> "bar3"))
* CSV文件 val csvFeeder = csv("foo.csv")//文件路径在 %Gatling\_Home%/user-files/data/
* JSON文件 val jsonFileFeeder = jsonFile("foo.json") //json的形式： [ { "id":19434, "foo":1 }, { "id":19435, "foo":2 } ]
* JDBC数据 jdbcFeeder("databaseUrl", "username", "password", "SELECT \* FROM users")

Redis

可参看官方文档http:*//gatling.io/docs/2.1.7/session/feeder.html#feeder*

使用示例：

**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.core.scenario.Simulation**  
**import** **io.gatling.http.Predef.\_**  
**import** **scala.concurrent.duration.\_**  
*/\*\*  
\* region请求接口测试  
\*/*  
**class** DynamicTest **extends** Simulation {  
val httpConf = http.baseURL("http://127.0.0.1:7001/test")  
*//地区 feeder*  
val regionFeeder = csv("region.csv").random  
*//数组形式*  
val mapTypeFeeder = Array(  
 Map("type" -> ""),  
 Map("type" -> "id\_to\_name"),  
 Map("type" -> "name\_to\_id")).random  
*//设置请求地址*  
val regionRequest =  
 exec(http("region\_map").get("/region/map/get"))  
 *//加载mapType feeder*  
 .feed(mapTypeFeeder)  
 *//执行请求, feeder里key=type, 在下面可以直接使用${type}*  
 .exec(http("province\_map").get("/region/provinces?mType=${type}"))  
 *//加载地区 feeder*  
 .feed(regionFeeder)  
 *//region.csv里title含有provinceId和cityId,所以请求中直接引用${cityId}/${provinceId}*  
 .exec(http("county\_map").get("/region/countties/map?mType=${type}&cityId=${cityId}&provinceId=${provinceId}"))  
*//声明scenario name=dynamic\_test*  
val scn = scenario("dynamic\_test")  
 .exec(during(180){ regionRequest  
 })  
*//在2秒内平滑启动150个线程(具体多少秒启动多少线程大家自己评估哈,我这里瞎写的)*  
setUp(scn.inject(rampUsers(150) over (2 seconds)).protocols(httpConf))  
}

注意：通过下面的代码只会第一次调用生成一个随机数，后面调用不变

exec(http("Random id browse")  
 .get("/articles/" + scala.util.Random.nextInt(100))  
 .check(status.is(200))

Gatling的官方文档解释是，由于DSL会预编译，在整个执行过程中是静态的。因此Random在运行过程中就已经静态化了，不会再执行。应改为Feeder实现，Feeder是gatling用于实现注入动态参数或变量的，改用Feeder实现:

val randomIdFeeder =   
 Iterator.continually(Map("id" ->   
 (scala.util.Random.nextInt(100))))  
  
feed(randomIdFeeder)  
 .exec(http("Random id browse")  
 .get("/articles/${id}"))  
 .check(status.is(200))

feed()在每次执行时都会从Iterator[Map[String, T]]对象中取出一个值，这样才能实现动态参数的需求。

# [性能测试工具](https://testerhome.com/topics/node3) Gatling 自动化压测实践

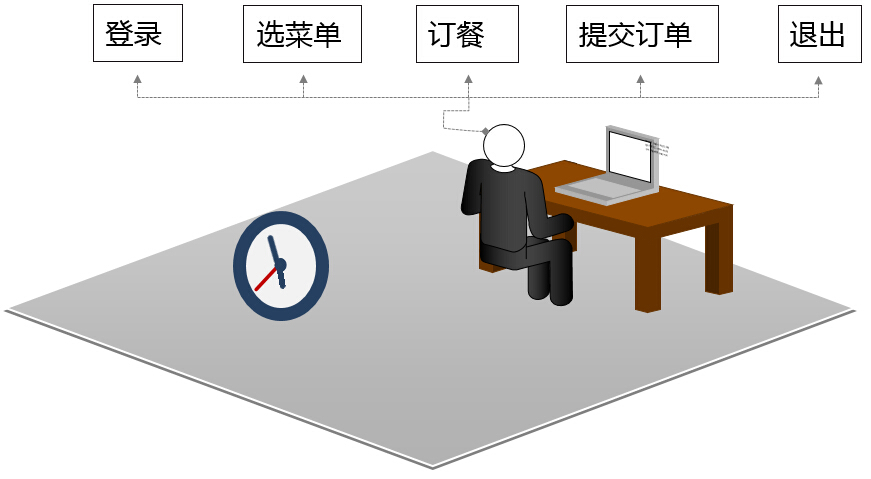
Gatling是一款基于Scala 开发的高性能服务器性能测试工具，它主要用于对服务器进行负载等测试；想使用Gatling进行压测的原因之一是想体验一下Scala编程的感觉，玩一下；第二，工作上也确实有这样的需求；

压测工作简单来说就是利用压测应用，来测试一下服务器的响应性能参数；然后把这些工作全部自动化，集成到jenkins中来运行。

整个工作的子任务分解可以由下图来表示:

[](https://testerhome.com/photo/2015/05d30e1ecdaddbbabe57f3e1c24f790d.jpg)

压测使用的是一个常见的web应用，该web应用的具体使用的业务场景如下：

[](https://testerhome.com/photo/2015/252359c50b94ed79977bca1d09f171dc.jpg)

针对该应用的压测Scala源代码如下：  
文件名：performance.scala

**package** performance  
  
**import** **scala.concurrent.duration.\_**  
  
**import** **io.gatling.core.Predef.\_**  
**import** **io.gatling.http.Predef.\_**  
**import** **io.gatling.jdbc.Predef.\_**  
  
**class** Performance **extends** Simulation {  
  
 *//用户名、餐馆ID 存储文件*  
 val user = csv("/root/.jenkins/workspace/testGatling/src/test/scala/data/user.csv").random  
 val res = csv("/root/.jenkins/workspace/testGatling/src/test/scala/data/restaurant.csv").random  
 val ip = csv("/root/.jenkins/workspace/testGatling/src/test/scala/data/ip.csv").random  
  
 val httpProtocol = http  
 .baseURL("http://${ip}:8180")  
  
 val headers\_0 = Map("Accept" -> "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8")  
  
 val scn = scenario("Emenu")  
 *//打开订餐首页*  
 .exec(http("index")  
 .get("/E\_Menu/userlogin/login.jsp")  
 .headers(headers\_0))  
  
 *//登录*  
 .pause(1 second,20 second)  
 .feed(user)  
 .feed(res)  
 .exec(http("login")  
 .post("/E\_Menu/userlogin/login")  
 .headers(headers\_0)  
 .formParam("username", "${username}")  
 .formParam("password", "${password}")  
 )  
  
 *//选择餐馆*  
 .pause(1 second,20 second)  
 .exec(http("menu")  
 .get("/E\_Menu/menu.action?res\_num=${rest\_id}")  
 .headers(headers\_0)  
 )  
 *//点菜*  
 .pause(1 second,20 second)  
 .exec(addCookie(Cookie("username", "${username}")))  
 .exec(addCookie(Cookie("res\_num", "${rest\_id}")))  
 .exec(addCookie(Cookie("food\_num0", "105")))  
 .exec(addCookie(Cookie("food\_num1", "104")))  
 .exec(addCookie(Cookie("food\_num2", "104")))  
 .exec(addCookie(Cookie("food\_num3", "106")))  
 .exec(addCookie(Cookie("total", "52")))  
 .exec(http("list") *//点完菜，开始订*  
 .get("/E\_Menu/list.action")  
 .headers(headers\_0)  
 )  
 *//下单*  
 .pause(1 second,20 second)  
 .exec(http("order")  
 .get("/E\_Menu/order.action?people=5&time=2025-08-31")  
 .headers(headers\_0)  
 )  
  
 *// //回首页*  
 .pause(1 second,20 second)  
 .exec(http("restaurant")  
 .get("/E\_Menu/restaurant.action")  
 .headers(headers\_0)  
 )  
  
 *//用户信息*  
 .pause(1 second,20 second)  
 .exec(http("userinfo")  
 .get("/E\_Menu/userinfo?username=${username}")  
 .headers(headers\_0)  
 )  
  
 *//我的订单*  
 .pause(1 second,20 second)  
 .exec(http("userorder")  
 .get("/E\_Menu/userorder.action?username=${username}")  
 .headers(headers\_0)  
 )  
  
 *//退出*  
 .pause(1 second,20 second)  
 .exec(http("exit")  
 .get("/E\_Menu/userlogin/login.jsp")  
 .headers(headers\_0))  
  
 setUp(scn.inject(atOnceUsers(100))).protocols(httpProtocol)  
}

写完scala代码，并且保证它在本地可以调试通过，下一步就是需要将代码集成进jenkins；  
先确保jenkins Gatling Plugin在jenkins上被安装；  
之后，写入用来执行的shell调控代码：

#################### 环境准备 ####################  
pwd  
export IP="xx.xx.xx.xx"  
export user="root"  
export pwd="xxxxxxxxxx"  
export tomcat\_path="/root/apache-tomcat-7.0.56/bin"  
export killtomcat="/root/killtomcat.sh"

#################### 定义函数 ####################  
kill\_tomcat7(){  
expect -c "  
spawn ssh $user@$IP sh $killtomcat;  
expect {  
yes/no { send yes\r;exp\_continue }  
\*password: { send $pwd\r }  
};  
expect eof;  
"  
}

start\_tomcat(){  
expect -c "  
spawn ssh $user@$IP sh $tomcat\_path/$1;  
expect {  
yes/no { send yes\r;exp\_continue }  
\*password: { send $pwd\r }  
};  
expect eof;  
"  
}  
#################### tomcat压测 ####################

### **停止tomcat**

kill\_tomcat7

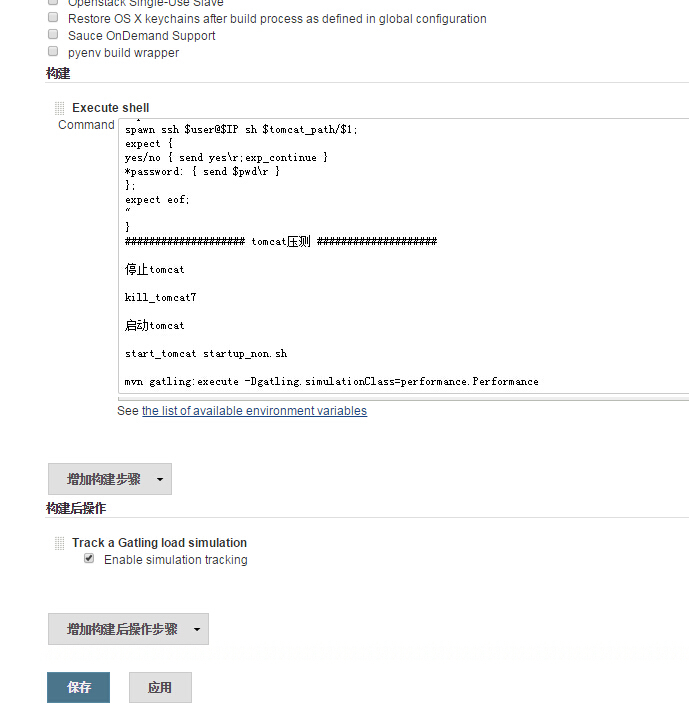
### **启动tomcat**

start\_tomcat startup\_non.sh

mvn gatling:execute -Dgatling.simulationClass=performance.Performance

#################### 完毕 ####################

参考截图如下：

[](https://testerhome.com/photo/2015/6d9ad213fa42ca5e5f5a2c0753eb7a95.jpg)

在完成调试改bug工作之后，可以尝试运行一下，得到压测结果 Gatling report ，然后可以进行相关的数据分析，这里不再赘述。  
Jenkins 集成 Gatling Report参考截图如下：

[](https://testerhome.com/photo/2015/aeb94af0e2b7268eb2623cc82ed3b8bd.jpg)

[https://www.yuque.com/docs/share/e71c1056-9b0c-4c77-862a-7f0af225abdb?#%20%E3%80%8AGatling%E6%BA%90%E7%A0%81%E7%BC%96%E8%AF%91%E3%80%8B](https://www.yuque.com/docs/share/e71c1056-9b0c-4c77-862a-7f0af225abdb?" \l " %E3%80%8AGatling%E6%BA%90%E7%A0%81%E7%BC%96%E8%AF%91%E3%80%8B)

# **Gatling+Jenkins流水线实现性能测试自动化**

****背景：****Gatling作为一款新兴的性能测试工具，具有高性能高稳定性，方便的代码维护管理等特点。

****环境：****gatling-charts-highcharts-bundle-3.31 + jenkins

****步骤：****

一、安装

1.安装Gatling

官网（<https://gatling.io/open-source/start-testing/>）下载免费安装版，解压即可。前提是，操作系统已安装Java8及以上版本。

2.使用

这里不是本文重点，所以可以参考其他好文。例如，<https://blog.csdn.net/qq_37023538/article/details/54290035>

3.jenkins上Gatling插件安装

这个可以找度娘问问

4.流水线配置

1）在Jenkins上创建“流水线”

2）配置脚本如下：

param1="0" *// Gatling场景测试输入的参数1*

param2="0" *// Gatling场景测试输入的参数2*

taskname="Gatling-Workflow" *// 流水线名称*

node{

stage('test'){

try{

sshagent(["testpri"]){

sh """

cd /tools/gatling/bin;

sh exec\_gatling.sh $param1 $param2 // exec\_gatling.sh中调用gatling.sh运行Gatling

"""

}

}catch(exc){

echo "failed"

}finally{

stage('collect'){

sshagent(["testpri"]){

def project\_path=sh returnStdout: true, script: ". /tools/sendgatlingreport.sh ${taskname}" *// sendgatlingreport.sh将Gatling报告copy到jenkins指定目录，这样才能展示Gatling报告*

project\_path=project\_path.trim()

publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: true, reportDir: "result/${project\_path}", reportFiles: 'index.html', reportName: 'HTML Report', reportTitles: ''])

}

}

}

}

}

3）exec\_gatling.sh。*注意：该脚本放在gatling.sh同级目录下，否则无法调用gatling.sh。*

#!/bin/bash

set -x

param1=$1

param2=$2

echo "$param1">tmp.s

echo "$param2">tmp.s

if true; then

./gatling.sh

fi < ./tmp.s

4）sendgatlingreport.sh

#!/bin/bash

taskname=$1

path=/tools/gatling/results/

cd $path

cp -r $path /root/.jenkins/workspace/$taskname/ *# jenkins的job目录*

dir=`ls -l | awk '/^d/ {print $Nf}'` *# 找到job目录下的文件夹。这里默认只一个*

echo $dir

问题：

这里流水线配置时，gatling和jenkins放在同一执行机环境中，因此在copy gatling的执行结果报告时，没有scp的这些操作。如果gatling与jenkins不在同一执行机环境，则需要ssh到gatling所在机器上，然后执行scp操作。

sh """

ssh -o StrictHostKevChecking=no -l root <gatling所在地址>

scp -rq root@<jenkins所在地址>:/目的目录 root@<gatling所在地址>/<gatling结果报告目录>