Moco使用介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订者 | 修订内容 | 时间 |
| 1.0 | 陈航 | 文档初始化 | 2016.06.20 |
| 1.0 | 陈航 | 添加Moco使用介绍 | 2016.06.20 |

目录

[1. Moco介绍 3](#_Toc454268734)

[2. Moco独立运行所需环境 3](#_Toc454268735)

[3. 如何运行Moco 4](#_Toc454268736)

[3.1. 启动http服务 4](#_Toc454268737)

[3.2. 启动https服务 4](#_Toc454268738)

[3.3. 效果测试 5](#_Toc454268739)

[4. Moco HTTP(s) API配置 6](#_Toc454268740)

[4.1. 如何在配置文件添加注释 7](#_Toc454268741)

[4.2. 约定请求Body 7](#_Toc454268742)

[4.3. 约定接口的uri 7](#_Toc454268743)

[4.4. 约定请求参数 8](#_Toc454268744)

[4.5. 约定请求方法 8](#_Toc454268745)

[4.6. 约定HTTP版本 8](#_Toc454268746)

[4.7. 约定请求头部 8](#_Toc454268747)

[4.8. 约定cookie 9](#_Toc454268748)

[4.9. 约定请求form 9](#_Toc454268749)

[4.10. 约定以指定xml作为请求body 9](#_Toc454268750)

[4.11. 用xpath对请求进行匹配 10](#_Toc454268751)

[4.12. 约定以指定json作为请求body 10](#_Toc454268752)

[4.13. 用正则表达式对请求进行匹配 11](#_Toc454268753)

[4.14. 匹配操作 11](#_Toc454268754)

[4.15. 设置Response content 12](#_Toc454268755)

[4.16. 设置Response 状态码 13](#_Toc454268756)

[4.17. 设置Response HTTP版本 13](#_Toc454268757)

[4.18. 设置Response 头部 13](#_Toc454268758)

[4.19. 设置重定向 13](#_Toc454268759)

[4.20. 设置cookie 14](#_Toc454268760)

[4.21. 挂载文件 14](#_Toc454268761)

[5. template的用法 14](#_Toc454268762)

# Moco介绍

在开发过程中，经常会使用到一些http网络接口，而这部分功能通常是由第三方开发团队或者是后端同事进行开发的，在我们开发时不能给我们提供服务，更有甚者，要集成的服务在开发时还不存在。这为我们的联调和测试造成了麻烦，常见的解决方案是搭建一个web server。

为什么要开发Moco这个框架？

具体到模拟服务上，处理的手法也是各种各样，因为服务以HTTP集成居多，无论是Web Service，还是REST，所以，一种典型的做法是，开发一个模拟服务，打成WAR包，部署到一个应用服务器上。而我们知道，一旦牵扯到应用服务器部署，就是非常耗时的，部署的时间量级通常是分钟级的。而且，模拟服务器通常不是一次性的工作，我们需要在开发过程中，反复调整，这就进一步增加了维护一个模拟服务器的成本。有的应用服务器是非常消耗资源的，要用专门的机器来部署它。更进一步，如果机器资源有限，团队就只能共享一台机器，这样，即便我为测试自己的部分做一个小的改动，很有可能因为得不到机器的使用权，而要等上几天时间

Moco就是针对这样一个特定的场景而生的。Moco是**一个简单搭建模拟服务器的程序库/工具**，这个基于 Java 开发的开源项目已经在 Github 上获得了不少的关注。该项目的简介是这样描述自己的：Moco 是一个简单搭建 stub 的框架，主要用于测试和集成。

开发团队只要根据自己的需要进行相应的配置，就会很方便得到一个模拟服务器。而且，由于 Moco 本身的灵活性，其用途已经不再局限于最初的集成测试，比如，Moco 可以用于移动开发，模拟尚未开发的服务；Moco 还可以用于前端开发，模拟一个完整的 Web 服务器等等。

Moco本身支持**API**和**独立运行**两种方式。通过使用API，开发人员可以在JUnit、JBehave等测试测试框架里使用Moco，极大程度地降低了集成点测试的复杂度

Moco可以提供以下服务：

* HTTP APIs
* Socket APIs
* REST API

Moco原理简介：Moco会根据一些配置，启动一个真正的HTTP服务（会监听本地的某个端口）。当发起请求满足一个条件时，它就给回复一个应答。Moco的底层没有依赖于像Servlet这样的重型框架，而是基于一个叫Netty网络应用框架直接编写的，这样一来，绕过了复杂的应用服务器，所以，它的速度是极快的

Moco已经在github上开源，可点击连接：<https://github.com/dreamhead/moco>

# Moco独立运行所需环境

Moco独立运行时所需准备的有：

* Java运行环境
* moco-runner-0.11.0-standalone.jar

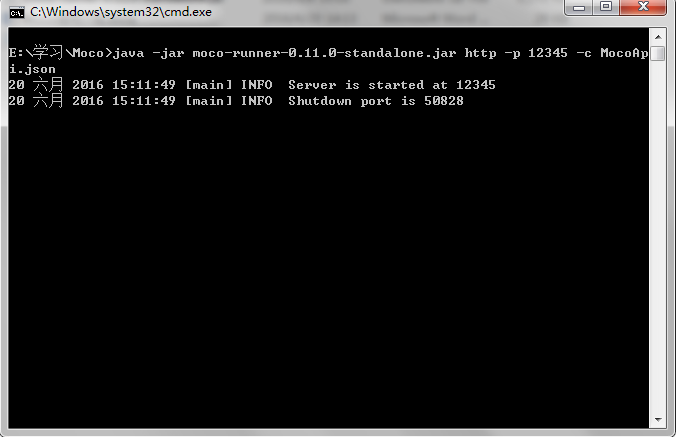
# 如何运行Moco

## 启动http服务

Moco的运行非常简单，只需要一行命令即可

如在命令行中运行：java -jar **<path-to-moco-runner>** http -p **<monitor-port>** -c **< configuration -file>**

* **<path-to-moco-runner>**：moco-runner-0.11.0-standalone.jar包的路径
* **<monitor-port>**：http服务监听的端口
* **<configuration -file>**：配置文件路径



这就在本地启动了一个http server，其中监听端口是12345，配置文件是MocoApi.json。只要在本机发起一个请求，如：<http://localhost:12345>，该请求就会被这个web server handle。

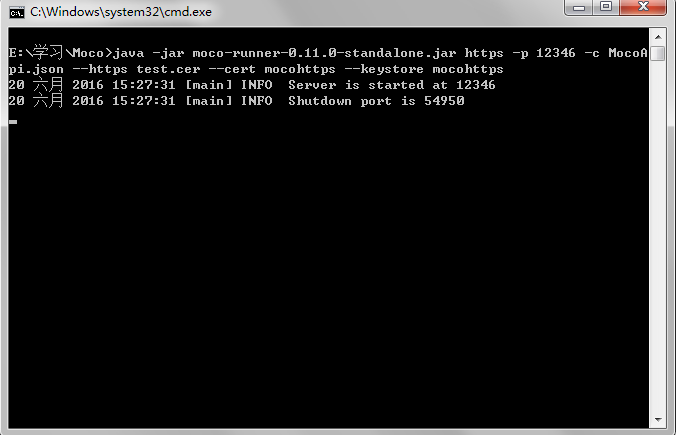
如果别的机子想访问这个服务，只要把localhost替换成本机IP即可。

## 启动https服务

启动https服务，需要先生成证书，并用如下命令启动服务：地方多发呆发地方的地方的地方的发呆发：

java -jar **<path-to-moco-runner>** https -p **<monitor-port>** -c **< configuration -file>** --https **<path-to-cert.jks >** --cert mocohttps --keystore mocohttps

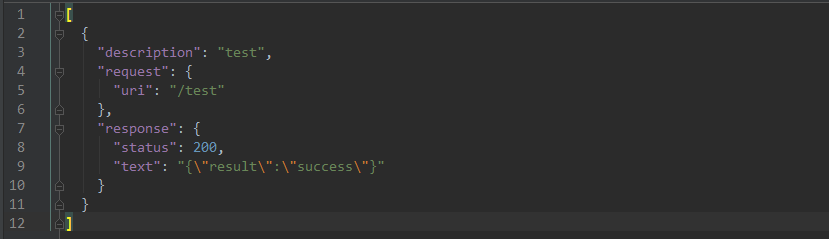
* **<path-to-moco-runner>**：moco-runner-0.11.0-standalone.jar包的路径
* **<monitor-port>**：http服务监听的端口
* **<configuration -file>**：配置文件路径
* **<path-to-cert.jks>**：证书路径



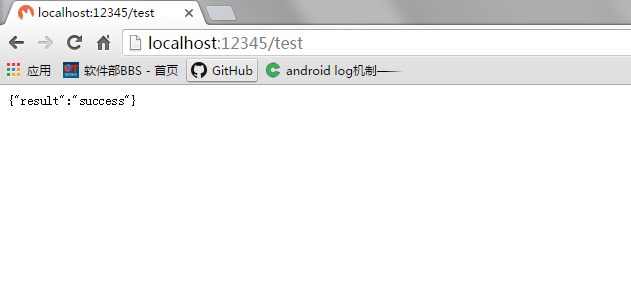
这就在本地启动了一个http server，其中监听端口是12346，配置文件是MocoApi.json，证书文件是test.cer

## 效果测试

1. 设置配置文件：



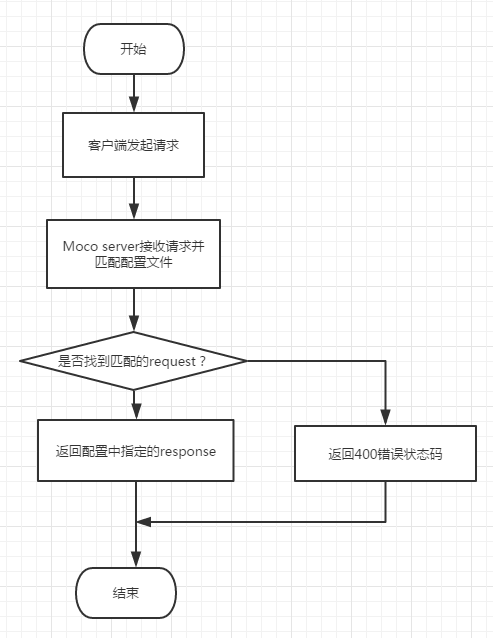
1. 启动服务：java -jar moco-runner-0.11.0-standalone.jar http -p 12345 -c MocoApi.json
2. 浏览器发起请求：<http://localhost:12345/test>
3. 返回结果如下：



# Moco HTTP(s) API配置

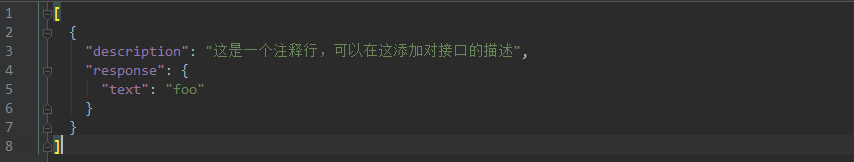
启动服务之后，必然会根据需求stub出各种各样接口反馈，我们会把这个配置放在一个json文件中，启动Moco的时候，需要指定使用的配置文件路径，这样配置就可以生效了。Moco服务可以检测到配置文件的变更，假如你修改了配置文件，不需要重新启动Moco，服务照样可以生效。更详细的配置介绍请查看：<https://github.com/dreamhead/moco/blob/master/moco-doc/apis.md>

配置文件的工作原理大致如下：

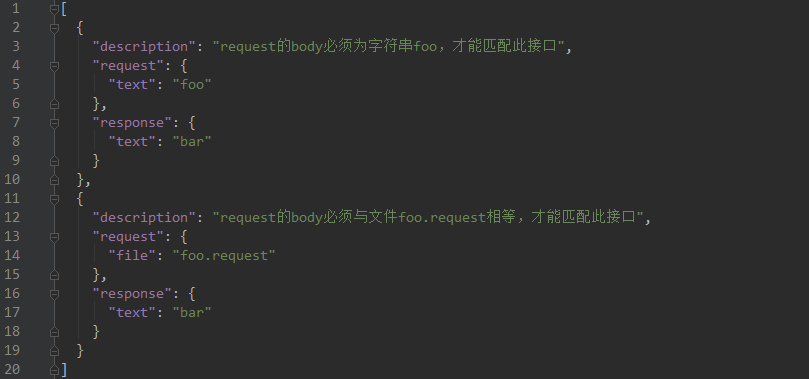


## 如何在配置文件添加注释

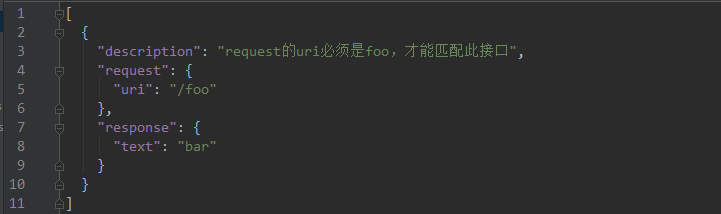
json不支持注释，想要添加注释的话，可以在description字段中加入描述



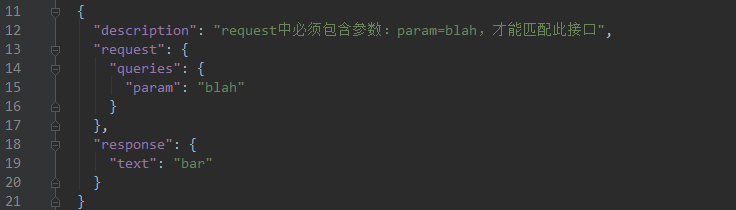
## 约定请求Body



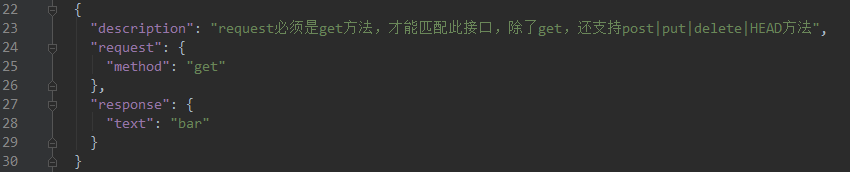
## 约定接口的uri



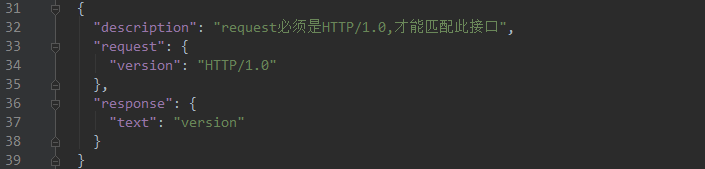
## 约定请求参数



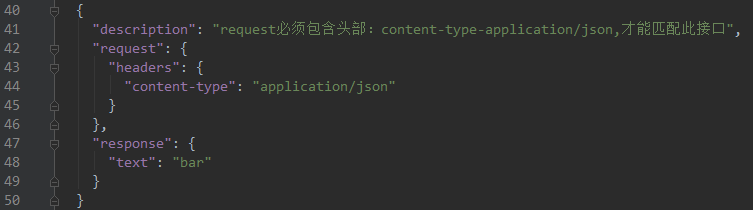
## 约定请求方法



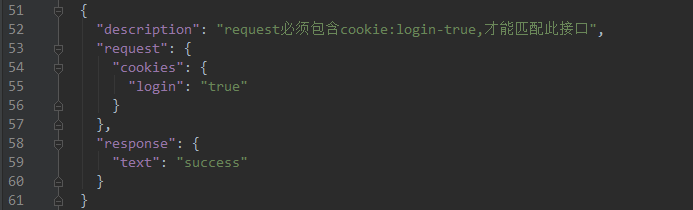
## 约定HTTP版本



## 约定请求头部

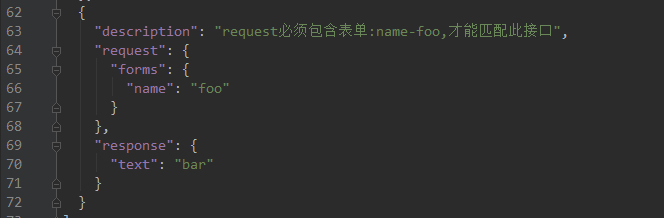


## 约定cookie

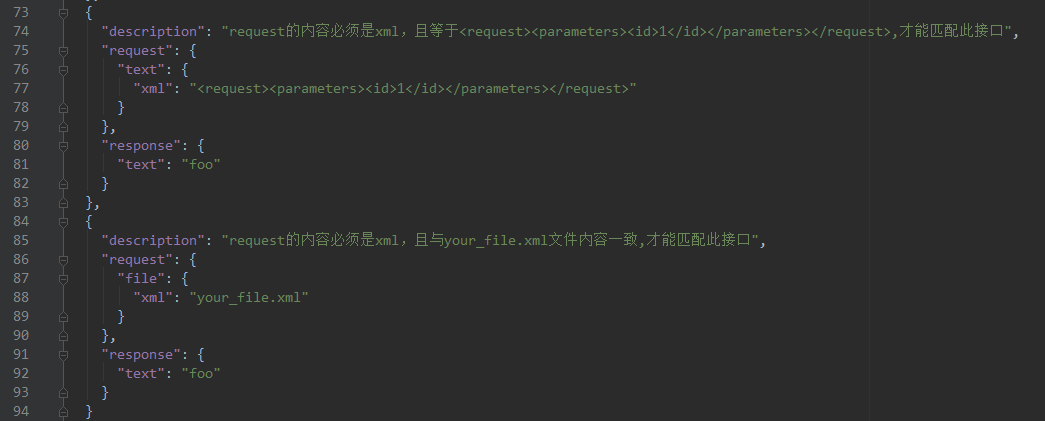


## 约定请求form

表单可以添加多项，多项的时候，必须全部匹配，接口才算匹配成功



## 约定以指定xml作为请求body



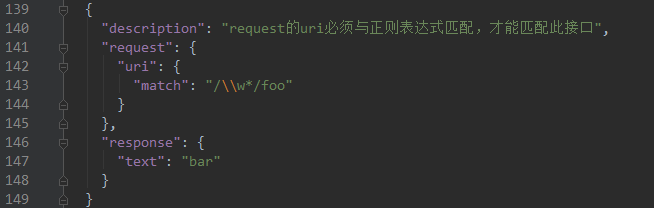
## 用xpath对请求进行匹配



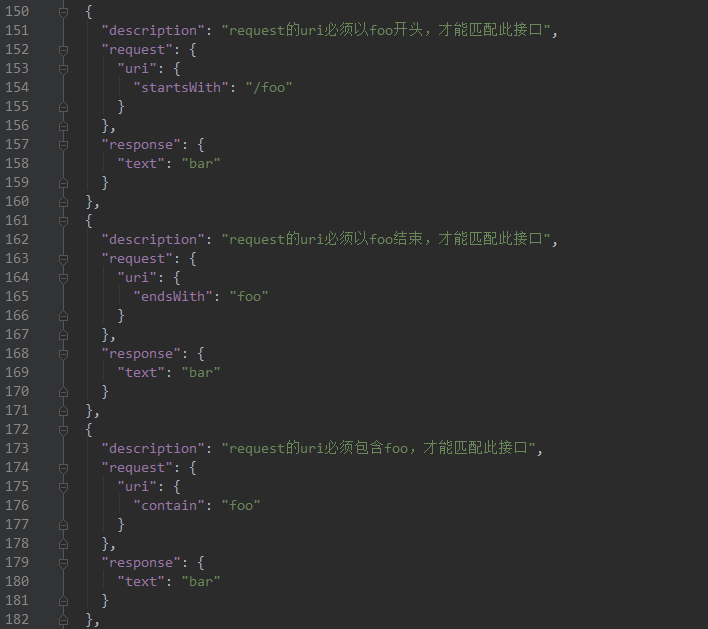
## 约定以指定json作为请求body



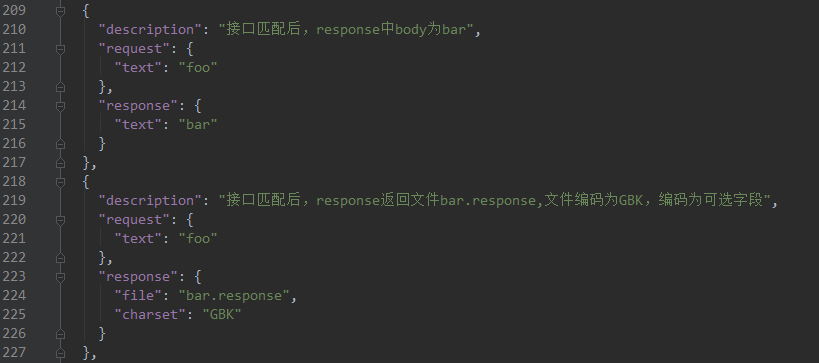
## 用正则表达式对请求进行匹配



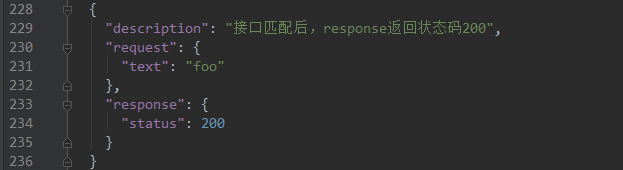
## 匹配操作

## 设置Response content



## 设置Response 状态码



## 设置Response HTTP版本



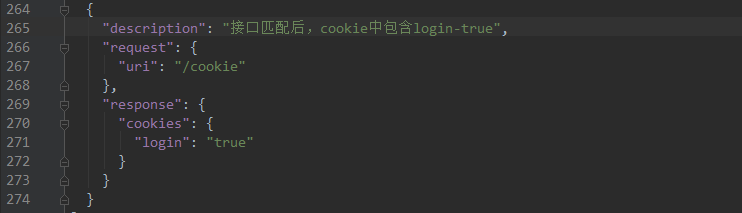
## 设置Response 头部



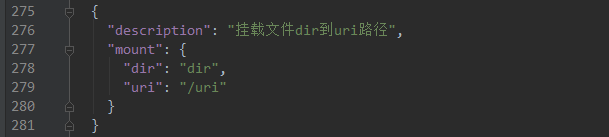
## 设置重定向



## 设置cookie



## 挂载文件

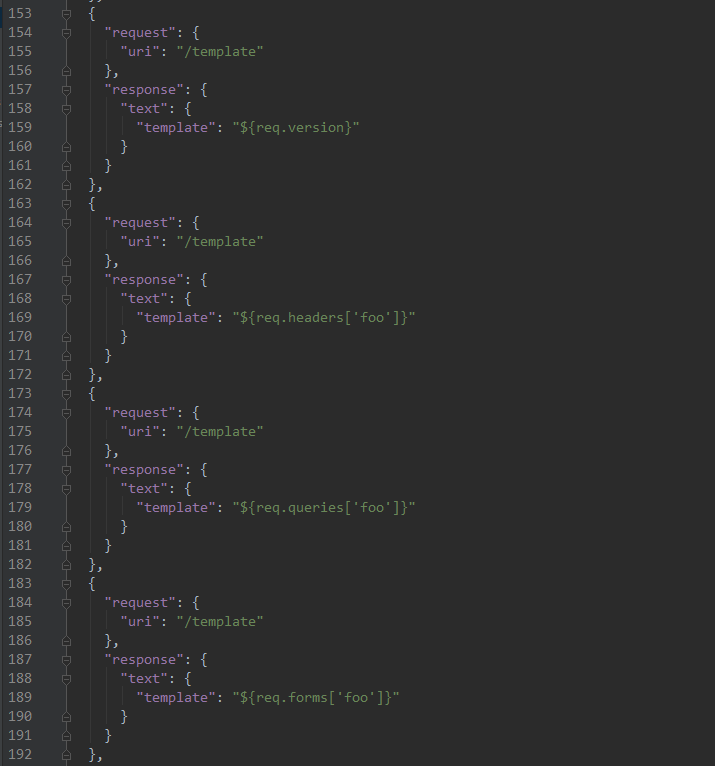


# template的用法

Moco内置了一些变量，在response中可以使用这些变量，让反馈更加智能，以下列举了常用的变量。

* req.version
* req.version
* req.method
* req.content
* req.headers
* req.queries
* req.forms
* req.cookies

使用举例如下：



# Moco在单元测试中使用

Moco除了可以单独运行外，还可以在单元测试中运行，测试过程中，Moco会启动一个web server来处理我们的请求。



运行在单元测试中的moco server也可以选择加载json配置文件



通过stub后台，便可对http请求进行测试了

# Moco的不足

Moco的使用很简单，配置也很方便，目前更是提供了http、rest、socket服务。但是也仅仅是能stub出接口，模拟出简单的场景。如果接收到请求后需要做一些处理，如需查询数据库、进行运算、或者一些复杂的操作，就无能为力了。所以是否选用Moco，就取决于开发者是否只是需要一个简单的模拟服务器。