

# Table of Contents

接口测试课程	1.1
接口测试扩展	1.2
HTTPS	1.2.1
WebService	1.2.2
接口Mock测试	1.2.3
接口测试总结	1.2.4

佐智播客-黑马程序员

# 接口测试课程

## 课程目标

- 能够根据接口API文档编写接口测试用例
- 能够使用Postman工具进行接口测试，并能够对大量接口用例进行管理、对接口响应结果进行断言、处理多接口的依赖及生成测试报告
- 能够使用Python+Requests封装的接口测试框架，实现接口对象封装、测试用例编写、测试数据管理及生成测试报告

## 课程大纲

章节	知识点	
第1章 接口测试基础	1.接口及接口测试概念 2.HTTP协议 3.接口规范	4.接口测试流程 5.项目环境说明
第2章 Postman实现接口测试	1.Postman介绍和安装 2.Postman基本用法 3.Postman高级用法	4.Postman测试报告 5.项目实战
第3章 数据库操作	1. 数据库介绍 2. 数据库基本操作	3. 数据库事务操作
第4章 代码实现接口测试	1. Requests库 2. 集成UnitTest	3. 接口测试框架开发 4. 项目实战
第5章 持续集成	1. 持续集成介绍 2. Git与Git代码托管平台 3. Jenkins	4. 持续集成之Postman 5. 持续集成之代码
第6章 接口测试扩展	1. 接口Mock测试	2. 接口测试总结

# 接口测试扩展

## 目标

1. 能说出接口mock测试的应用场景，并掌握使用moco框架搭建mock服务
2. 熟练掌握接口测试相关面试题

佐智播客-黑马程序员

# HTTPS

## 目标

1. 了解密码学基础知识
2. 了解HTTPS的特点和通信流程
3. 掌握HTTP与HTTPS的区别

很多国内外的大型互联网公司开始大力推行HTTPS:

1. 从2017年开始, Chrome浏览器已把采用 HTTP 协议的网站标记为不安全网站;
2. 苹果要求2017年01月App Store中的所有应用都必须使用HTTPS协议;
3. 国内的微信小程序也要求必须使用HTTPS协议。

## 1. 密码学基础（了解）

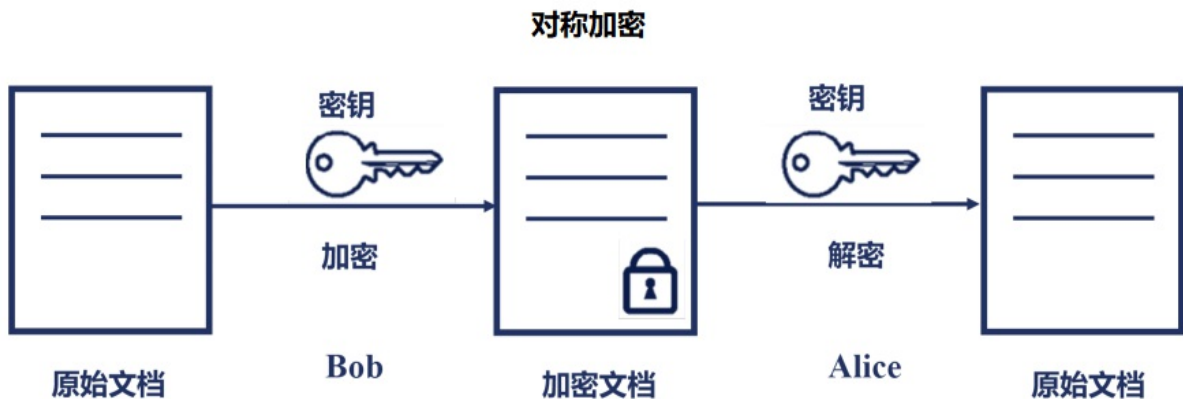
在正式学习HTTPS协议之前, 我们首先要了解一些密码学的相关知识。

- 明文: 明文指的是未被加密过的原始数据。
- 密文: 明文被某种加密算法加密之后, 会变成密文, 从而确保原始数据的安全。
- 密钥: 密钥是一种参数, 它是在明文转换为密文或将密文转换为明文的算法中输入的参数。

### 1.1 对称加密

对称加密又叫做私钥加密, 即信息的发送方和接收方使用同一个密钥去加密和解密数据。

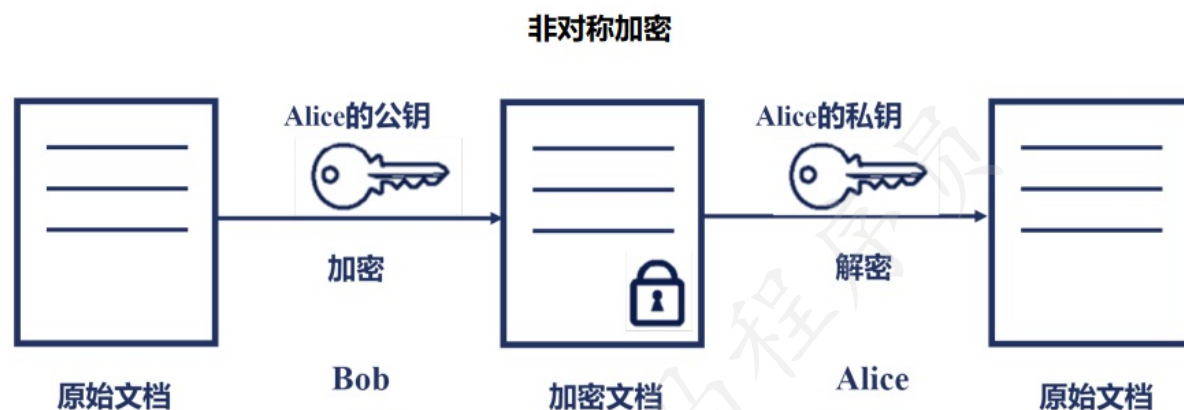
- 对称加密的特点是算法公开、加密和解密速度快, 适合于对大数据量进行加密
- 对称加密中用到的密钥叫做私钥, 私钥表示个人私有的密钥, 即该密钥不能被泄露。
- 其加密过程中的私钥与解密过程中用到的私钥是同一个密钥, 所以称之为“对称”加密。
- 由于对称加密的算法是公开的, 所以一旦私钥被泄露, 那么密文就很容易被破解, 所以对称加密的缺点是密钥安全管理困难
- 常见的对称加密算法: DES、3DES、TDEA、Blowfish、RC5、IDEA等



### 1.2 非对称加密

非对称加密也叫做公钥加密。非对称加密与对称加密相比，其安全性更好。

- 非对称加密算法需要两个密钥：公开密钥（简称公钥）和私有密钥（简称私钥），且二者成对出现。
- 私钥被自己保存，不能对外泄露。公钥指的是公共的密钥，任何人都可以获得该密钥。
- 用公钥或私钥中的任何一个进行加密，用另一个进行解密。
- 被公钥加密过的密文只能被私钥解密：明文 + 加密算法 + 公钥 => 密文，密文 + 解密算法 + 私钥 => 明文
- 被私钥加密过的密文只能被公钥解密：明文 + 加密算法 + 私钥 => 密文，密文 + 解密算法 + 公钥 => 明文
- 由于加密和解密使用了两个不同的密钥，所以叫作“非对称”加密
- 非对称加密的缺点是加密和解密花费时间长、速度慢
- 常见的非对称加密算法：RSA、Elgamal、Rabin、D-H、ECC等



## 2. HTTPS介绍

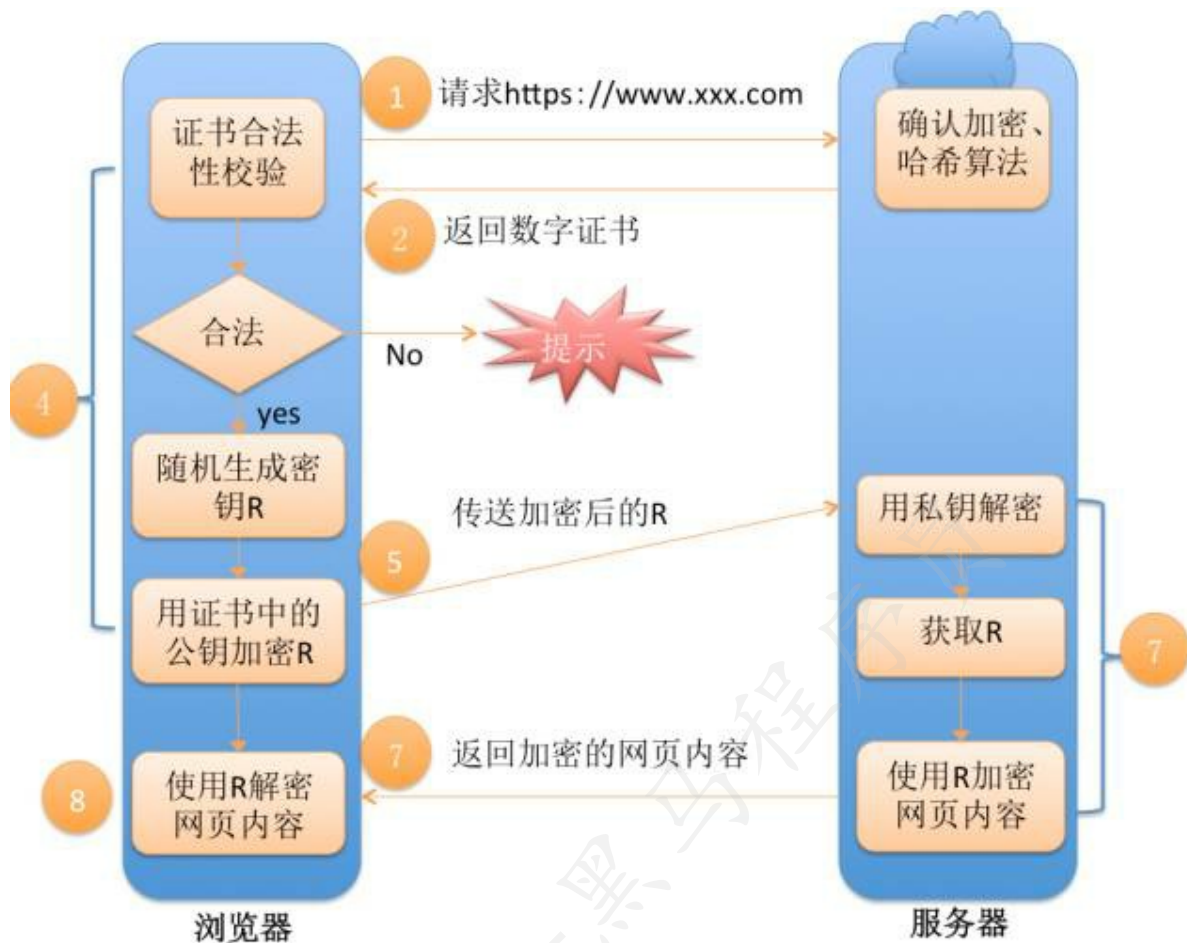
- HTTPS: (Hyper Text Transfer Protocol over SecureSocket Layer)，是以安全为目标的 HTTP 通道，在 HTTP 的基础上通过传输加密和身份认证保证了传输过程的安全性。
- HTTPS 在 HTTP 的基础下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。
- 广泛用于网络上安全敏感数据的通讯，例如交易支付等方面。

### 2.1 HTTPS特点

- 数据保密性：保证数据内容在传输的过程中不会被第三方查看
- 数据完整性：防止传输的内容被中间人冒充或者篡改
- 身份校验安全性：通过证书认证客户端访问的是自己的服务器，保证数据到达用户期望的目的地

### 2.2 HTTPS通信过程

1. 浏览器向服务器的443端口发起请求，请求携带了浏览器支持的加密算法和哈希算法
2. 服务器收到请求，选择浏览器支持的加密算法和哈希算法
3. 服务器将数字证书返回给浏览器，这里的数字证书可以是向某个可靠机构申请的，也可以是自制的
4. 浏览器进入数字证书认证环节，这一部分是浏览器内置的TLS完成的。认证通过后获取到公钥
5. 浏览器生成一个随机数R，并使用网站公钥对R进行加密，然后将加密的R传送给服务器
6. 服务器用自己的私钥解密得到R
7. 服务器以R为密钥使用了对称加密算法加密网页内容并传输给浏览器
8. 浏览器以R为密钥使用之前约定好的解密算法获取网页内容



## 2.3 HTTP与HTTPS有什么区别？

HTTP和HTTPS的区别主要如下：

- HTTP是超文本传输协议，信息是明文传输，在传输敏感数据时不安全；HTTPS在HTTP的基础上加入了SSL/TLS协议，数据是加密的更安全
- HTTPS协议需要到CA申请证书，一般免费证书较少，因而需要一定费用
- 使用的默认端口不一样，HTTP是80，HTTPS是443
- HTTP速度更快，HTTPS需要进行加密和解密的处理，消耗的时间比HTTP要多

# WebService

## 目标

1. 了解WebService的概念和工作原理
2. 知道如何对WebService接口测试

## 1. WebService介绍

- WebService就是一种跨编程语言和跨操作系统平台的远程调用技术，提供一个供外部调用的服务。
- 例如：气象局把自己的系统服务以WebService服务的形式暴露出来，让第三方网站和程序可以调用这些服务功能。

### 1.1 WebService工作原理

WebService包含的核心技术有：HTTP、XML、SOAP、WSDL

- WebService采用HTTP协议实现客户端和服务端之间数据的传输
- WebService采用XML来封装数据（请求数据和响应数据都采用XML封装），XML主要的优点在于跨平台
- WebService通过HTTP协议发送请求时，会增加了一些特定的HTTP消息头，用来说明HTTP消息的内容格式，这些特定的HTTP消息头和XML内容格式就是SOAP协议规定的
- WebService服务器端通过一个WSDL文件来说明可以对外提供的服务

SOAP（Simple Object Access Protocol）：简单对象访问协议是交换数据的一种协议规范，是一种轻量的、简单的、基于XML的协议，它被设计成在WEB上交换结构化的和固化的信息。目前SOAP的版本主要有SOAP1.1和SOAP1.2。

WSDL（Web Services Description Language）：网络服务描述语言是一种使用XML编写的文档，该文档用来描述某个WebService。WSDL就像是一个说明书，用于描述WebService及其方法、参数和返回值。

### 1.2 HTTP接口与WebService接口的区别

- HTTP接口
  - 使用HTTP协议，请求数据一般是表单数据或JSON形式，返回数据一般是JSON数据
  - 常用的请求方式是GET和POST
- WebService接口
  - 使用SOAP协议，通过HTTP传输，请求数据和响应数据都是xml格式
  - 请求方式是POST
  - WebService能处理较复杂的数据类型

### 1.3 WebService接口测试工具

- Postman
- JMeter
- SoapUI：一款专业的web service测试软件，包括开源版和专业版

- Python + Requests

## 2. WebService接口测试

### 2.1 案例

调用天气预报Web服务，获取天气信息。

- WebService地址: <http://www.webxml.com.cn/WebServices/WeatherWebService.asmx>
- WSDL地址: <http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl>

### 2.2 getSupportCity

getSupportCity接口是用来查询本天气预报Web Services支持的国内外城市或地区信息

- 输入参数: byProvinceName = 指定的洲或国内的省份，若为ALL或空则表示返回全部城市；
- 返回数据: 一个一维字符串数组 String()，结构为: 城市名称(城市代码)。

以下是 SOAP 1.2 请求和响应示例。所显示的占位符需替换为实际值。

```
POST /WebServices/WeatherWebService.asmx HTTP/1.1
Host: www.webxml.com.cn
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <getSupportCity xmlns="http://WebXml.com.cn/">
      <byProvinceName>{string}</byProvinceName>
    </getSupportCity>
  </soap12:Body>
</soap12:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <getSupportCityResponse xmlns="http://WebXml.com.cn/">
      <getSupportCityResult>
        <string>{string}</string>
        <string>{string}</string>
      </getSupportCityResult>
    </getSupportCityResponse>
  </soap12:Body>
</soap12:Envelope>
```

### 2.3 Postman测试WebService接口

操作步骤:

1. 新建请求 getSupportCity

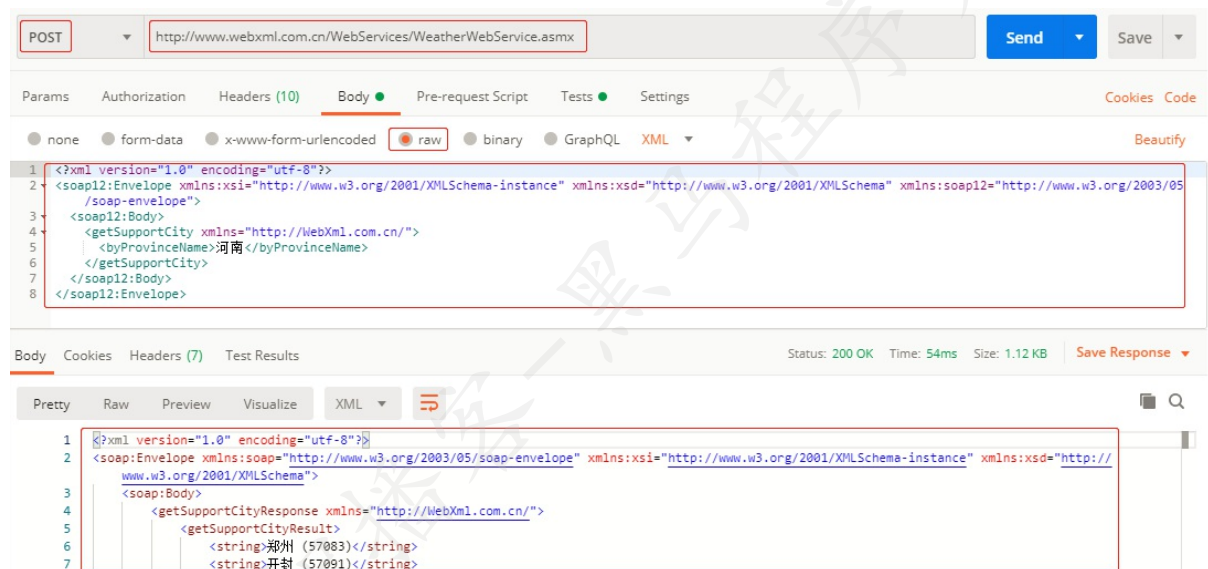


2. 设置请求方式为POST
3. 输入请求URL: `http://www.webxml.com.cn/WebServices/WeatherWebService.asmx`
4. 设置请求头: `Content-Type: application/soap+xml; charset=utf-8`
5. 设置请求体数据:

```
<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <getSupportCity xmlns="http://WebXml.com.cn/">
      <byProvinceName>河南</byProvinceName>
    </getSupportCity>
  </soap12:Body>
</soap12:Envelope>
```

6. 发送请求, 查看响应结果

示例截图:



# 接口Mock测试

## 目标

1. 知道接口Mock测试的作用
2. 知道接口Mock测试的实现方式
3. 掌握如何使用moco框架搭建mock服务

## 1. 什么是接口Mock测试？

应用场景思考？

1. 在前后端分离的项目中，假如后端代码还未开发完，前端代码需要调用后端接口进行调试，该怎么办？
2. 本公司的电商平台需要对接第三方支付接口，如何测试支付失败的场景？

### 1.1 概念

Mock：模拟的、仿制的、虚假的

Mock测试：在测试过程中，对于某些不容易构造或者不容易获取的对象，可以用一个虚拟的对象来代替的测试方法。

接口Mock测试：在接口测试过程中，对于某些不容易构造或者不容易获取的接口，可以用一个模拟接口来代替。

### 1.2 作用

- 可以用来解除测试对象对外部服务的依赖，使得测试用例可以独立运行
- 替换外部服务调用或一些速度较慢的操作，提升测试用例的运行速度
- 模拟异常逻辑，异常逻辑往往很难触发，通过Mock可以人为的控制触发异常逻辑
- 团队可以并行工作

### 1.3 实现方式

接口mock实现的核心思想是搭建一个Mock Server，通过该服务提供mock接口。常见的实现方式有：

- 使用第三方mock平台
- 自己开发mock服务
- 使用mock框架搭建mock服务

## 2. Python + Flask开发mock服务（了解）

### 2.1 Flask介绍

Flask是一个用Python编写的轻量级Web应用程序框架。使用该框架可以非常方便的开发Web项目。

## 2.2 环境搭建

安装Flask需要先安装好Python开发环境，建议使用2.6或更高版本

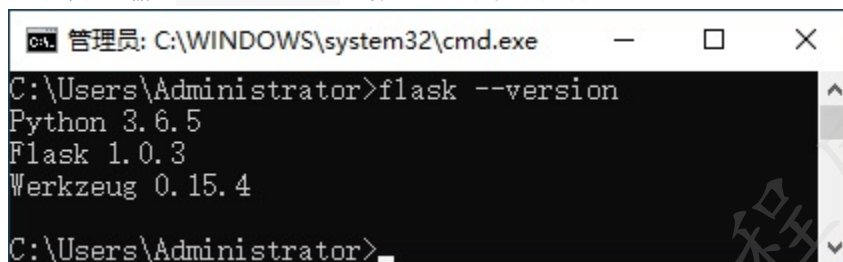
### 安装Flask

使用pip工具安装，注意需要联网

```
pip install Flask
```

### 验证

在命令行里输入 `flask --version` 查看Flask是否安装成功



```
管理员: C:\WINDOWS\system32\cmd.exe
C:\Users\Administrator>flask --version
Python 3.6.5
Flask 1.0.3
Werkzeug 0.15.4
C:\Users\Administrator>
```

## 2.3 示例演示

### 编写Flask程序

打开编辑器，创建apimock.py文件

```
from flask import Flask, jsonify

# 创建一个应用对象
app = Flask(__name__)

# 定义视图函数，设置路由规则
@app.route("/index")
def index():
    return "hello mock"

@app.route("/login", methods=["POST"])
def login():
    data = {
        "code": 10000,
        "uid": 1001,
        "token": "xxx"
    }
    return jsonify(data)

if __name__ == '__main__':
    # 启动WEB服务器
    app.run()
```

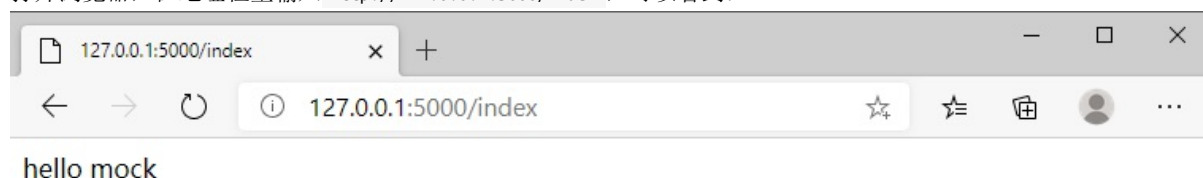
### 启动运行

可以使用下面两种方式来运行程序：

- 手动运行： `python apimock.py`
- pycharm运行：像正常运行普通python程序一样即可

## 访问

打开浏览器，在地址栏里输入 `http://127.0.0.1:5000/index`，可以看到：



## 3. Moco框架

### 3.1 Moco简介

- Moco是一个简单搭建模拟服务器的框架（工具），可以模拟http、https、socket等协议
- 基于Java开发的开源项目，Github地址：<https://github.com/dreamhead/moco>
- 原理：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）。当发起的请求满足某个条件时，就会返回指定的响应数据

### 3.2 环境搭建

Moco运行时所需环境包括：

- Java运行环境
  - 安装JDK，并配置环境变量
- moco-runner-1.1.0-standalone.jar
  - 下载地址：<https://repo1.maven.org/maven2/com/github/dreamhead/moco-runner/1.1.0/moco-runner-1.1.0-standalone.jar>

### 3.3 如何运行Moco

#### 1>创建配置文件

创建配置文件test.json，并输入如下内容：

```
[
  {
    "description": "首页",
    "request": {
      "uri": "/index"
    },
    "response": {
      "text": "hello world"
    }
  }
]
```

## 2>启动http服务

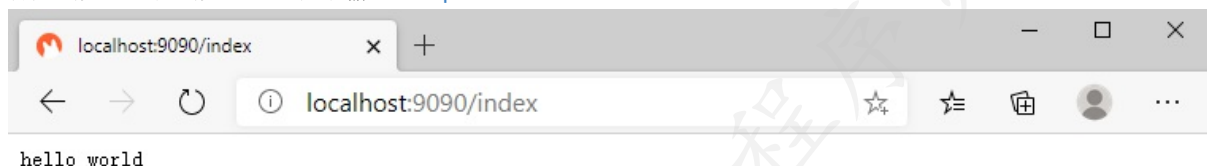
启动命令:

```
java -jar <path-to-moco-runner> http -p <monitor-port> -c <configuration-file>
示例:
java -jar moco-runner-1.1.0-standalone.jar http -p 9090 -c test.json
```

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

## 3>接口访问

打开浏览器, 在浏览器地址栏中输入: <http://localhost:9090/index>



## 3.4 Moco常用配置参数

1. 定义请求方式, 通过 `method` 参数定义

```
[
  {
    "description": "首页",
    "request": {
      "uri": "/index",
      "method": "post"
    },
    "response": {
      "text": "hello world"
    }
  }
]
```

2. 定义请求参数, 通过 `queries` 参数定义

```
[
  {
    "description": "首页",
    "request": {
      "uri": "/index",
      "method": "get",
      "queries": {
        "area": "010",
        "kw": "hello"
      }
    },
    "response": {
      "text": "hello world"
    }
  }
]
```

```
]
```

3. 定义请求头，通过 `headers` 参数定义

```
[
  {
    "description": "登录",
    "request": {
      "uri": "/login",
      "method": "post",
      "headers": {
        "area": "010"
      }
    },
    "response": {
      "text": "hello world"
    }
  }
]
```

4. 定义表单请求体，通过 `forms` 参数定义

```
[
  {
    "description": "登录",
    "request": {
      "uri": "/login",
      "method": "post",
      "forms": {
        "username": "tom",
        "password": "123456"
      }
    },
    "response": {
      "text": "login success"
    }
  }
]
```

5. 定义JSON请求体，通过 `json` 参数定义

```
[
  {
    "description": "登录",
    "request": {
      "uri": "/login",
      "method": "post",
      "headers": {
        "Content-Type": "application/json"
      },
      "json": {
        "username": "tom",
        "password": "123456"
      }
    },
    "response": {
      "text": "hello world66666"
    }
  }
]
```

6. 定义HTTP响应状态码，通过 `status` 参数定义

```
[
  {
    "description": "首页",
    "request": {
      "uri": "/index2"
    },
    "response": {
      "status": 500,
      "text": "error"
    }
  }
]
```

#### 7. 定义JSON响应数据，通过 json 参数定义

```
[
  {
    "description": "登录",
    "request": {
      "uri": "/login"
    },
    "response": {
      "headers": {
        "Content-Type": "application/json;charset=UTF-8"
      },
      "json": {
        "code": "10000",
        "msg": "操作成功",
        "data": {
          "uid": 2,
          "token": "xxx"
        }
      }
    }
  }
]
```

### 3.5 Moco引入配置文件

moco支持在配置文件中引入其他配置文件，这样可以分服务/模块定义配置文件，便于对配置文件的管理。

实现步骤：

1. 分服务/模块定义配置文件，如分别定义 index.json 和 login.json 文件

```
[
  {
    "description": "首页",
    "request": {
      "uri": "/index"
    },
    "response": {
      "text": "hello world"
    }
  }
]
```

```
[
  {
    "description": "登录",
    "request": {
```

```
    "uri": "/login"
  },
  "response": {
    "text": "success"
  }
}
]
```

2. 定义启动配置文件，如 `config.json` 并引入其他配置文件

```
[
  {"include": "index.json"},
  {"include": "login.json"}
]
```

3. 启动服务

```
java -jar moco-runner-1.1.0-standalone.jar http -p 9090 -g config.json
```

注意：通过 `-g config.json` 指定配置文件



# 接口测试总结

## 目标

1. 掌握接口测试的常见面试题

## 1. 常见面试题

1. 你们公司是如何做接口测试的？
2. 基于代码的接口测试框架是如何封装的？
3. 接口之间有依赖时怎么处理？
4. 如何判断接口测试的结果（成功或失败）？
5. 常见的接口请求方式和区别？
6. GET和POST请求的区别？
7. 常见的响应状态码有哪些？
8. 发送HTTP请求时，传递参数的途径有哪些？
9. 持续集成如何做的？自动化测试多久构建一次？
10. 使用工具和代码实现接口测试的区别？

## 2. 场景分析

发表文章

标题 静夜思

内容

作者：李白  
床前明月光，疑是地上霜。  
举头望明月，低头思故乡。

P 24字

封面 ☐ 单图 ☐ 三图 ☒ 无图 ☐ 自动

频道: 产品

发表 存入草稿

问题：在发布文章模块中，输入正确的标题、内容、频道等信息后，点击“发表”按钮后没有任何反应。请描述如何分析定位存在的问题？

佐智播客-黑马程序员