# 问题1：简述接口测试定义

接口测试是测试系统间接口的一种测试，测试的对象主要是接口，主要是测试外部系统与所测试系统之间以及内部系统之间的交互点。

# 问题2：HTTP协议方法有哪些？

超文本传输​​协议（HTTP）是用于传输诸如HTML的超媒体文档的应用层协议

GET、POST、OPTIONS、HEAD、PUT 、DELETE、TRACE、CONNECT

# 问题3：post和get的区别？

参考答案:

1、GET使用URL或Cookie传参。而POST将数据放在BODY中。  
2、GET的URL会有长度上的限制，则POST的数据则可以非常大。  
3、POST比GET安全，因为数据在地址栏上不可见。  
4、一般get请求用来获取数据，post请求用来发送数据。

建议：  
1、get方式的安全性较Post方式要差些，包含机密信息的话，建议用Post数据提交方式；  
2、在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式；

# 问题4：Http协议有什么组成？

请求报文包含三部分：

* 请求方法 和 URI及使用的协议
* 请求头
* 请求正文

响应报文包含三部分：

* 协议和状态
* 响应头
* 响应正文

# 问题5：接口测试中依赖登录状态的接口如何测试？

参考答案：

依赖登录状态的接口的本质上是在每次发送请求时需要带上Session或者Cookie才能发送成功，在构建POST请求时添加必要的Session或者Cookie

# 问题6：在手工和自动化接口测试的过程中，上下游接口有数据依赖如何处理？

参考答案：

上下游接口的数据依赖就是准备测试数据。在工具中可以使用全局变量等方式将需要的数据进行传送。

假如一个事务需要顺序调用3个接口，A B C， C依赖于AB， 而AB有数据依赖，这时候就需要准备好A和B的数据。

数据一般有两种方式生成

* 动态方式：假如B依赖A创造的数据，那么每次执行B之前必须执行A去做数据创建
* 静态方式：独立统一的测试数据库, ABC需要的数据都可以从库里拿到

# 问题7：详细阐述接口测试和UI测试在测试活动中是如何协同测试的？

接口测试：验证数据传递，不依赖UI

UI测试：验证UI，通过操作驱动

# 问题8：什么是Http协议无状态协议?怎么解决Http协议无状态协议?

参考答案：

(1)、无状态协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息  
(2)、无状态协议解决办法： 通过1、Cookie 2、通过Session会话保存。

# 问题9：简述Http协议实现的原理机制？

参考答案

步骤如下：

1. 域名解析
2. 发起TCP三次握手
3. 发起HTTP请求
4. 服务器响应HTTP请求
5. 浏览器解析HTML代码，并请求HTML代码中的资源（JS、CSS、图片）
6. 浏览器对页面进行渲染，呈现给用户

# 问题10：请列出常见的Http协议状态码

1XX 指示信息

2XX 请求发送成功

3XX 重定向

4XX 客户端发送的请求有语法错误

5XX 服务器错误

# 问题11：接口测试意义

通过测试保证服务端的正确性（分层测试）

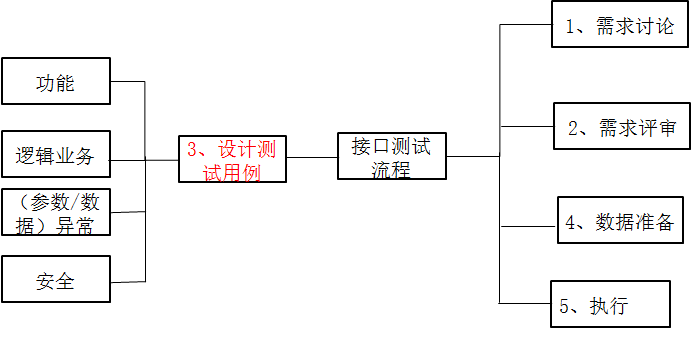
BUG更容易定位

自动化测试落地性价比更高，比UI更稳定

测试提前，降低研发成本，提高效率

更容易实现持续集成

# 问题12：接口测试流程



# 问题13：接口用例设计从哪些方面考虑？

功能方面

功能是否正常

功能是否按照接口文档实现

逻辑业务方面

是否依赖业务

异常处理方面

参数异常（关键字参数，参数为空，多、少参数，错误参数）

数据异常（关键字数据，数据为空，长度不一致，错误数据）

安全方面

cookie

header

唯一识别码

# 问题14：接口测试用例设计常用方法

边界值、等价类、决策表、场景法

# 问题15：常用的打包工具有哪些

ant 、Maven 、Gradle Jenkins

# 问题16：什么是持续性集成

**持续集成（Continuous Integration）这个术语源自 XP （极限编程）的一个最佳实践。**

**持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译、发布、自动化测试）来验证，从而尽快地发现集成错误。**

# 问题17：为什么要进行持续集成

1. **系统越来越庞大、复杂，集成、部署工作需要耗费大量时间**
2. **总是在无休止的修改bug，导致项目延期**
3. **总是在软件集成的后期发现重大问题**
4. **开发进度不可见**
5. **测试人员不能及早进入测试工作**
6. **减少重复的过程，节省时间，提高效率(重复性的活动包括代码编译，数据库集成，测试，评审，部署，信息反馈等)**

# 问题18：安装jenkins的命令

java –jar jenkins.war

# 问题19：简述Java接口自动化测试框架设计实现步骤

1.域名存放在属性文件中

2.常用get，post,方法进行封装，针对不同的参数，实现重载。

doGet(String url)

doGet(String url,String para)

doGet(String url,Map<String,Object> para)

doGetByCookie(String url,CookieStore cookie)

doPost(String url,String para)

doPost(String url,Map<String,Object> para) 针对Content-type是Form

doPost(String url,JSONObject para) 针对Content-type是JSON

doPostByCookie(String url,JSONObject para,CookieStore cookie)

3.断言的封装 Checker

4.数据驱动(把1000个商品遍历，做submit的操作；10个用户的登录，下单)

5.数据库的访问（JDBC）

6.监听器的使用

ApiListener extends TestListenerAdapter

重写 onFinish（）方法

7.发送邮件 JAVAmail

# 问题20：moco框架的使用

启动 moco的命令 moco-runner-0.12.0-standalone.jar的步骤

d:

cd d:/demo

java -jar moco-runner-0.12.0-standalone.jar http -p 9999 -c startup.json

实现GET接口 响应内容为text或者json

[

{

"description": *"第二个MOCO"*,

"request": {

"uri": *"/getdemo2"*,

"method": *"get"*

},

"response": {

"text": *"响应内容getdemo2"*

}

},

{

"description": *"第三个MOCO"*,

"request": {

"uri": *"/getdemo3"*,

"method": *"get"*,

"queries": {

"id": *"1"*,

"title": *"baidu"*

}

},

"response": {

"json": {

"message": *"success"*,

"code": *"200"*

}

}

}

]

实现POST接口

请求体类型为 *application/json"*

[

{

"description": *"moco login"*,

"request": {

"uri": *"/common/fgadmin/login2"*,

"method": *"post"*,

"headers": {

"content-type": *"application/json"*

},

"json": {

"phoneArea": *"86"*,

"phoneNumber": *"2000000000"*,

"password": *"netease123"*

}

},

"response": {

"json": {

"message": *"用户名错误"*,

"code": *"200"*

}

}

}

]

# 问题21：接口测试用例设计

参见《商城接口测试用例》

# 问题22：接口测试自动化-HttpClient GET POST 以及传递Cookie的实现

## GET方法-不带Cookie

**public** **static** String doGet(String url) **throws** Exception {

CloseableHttpClient client=HttpClients.*createDefault*();

//get请求参数在url中

HttpGet get=**new** HttpGet(url);

CloseableHttpResponse response =client.execute(get);

HttpEntity result\_entity=response.getEntity();

String result=EntityUtils.*toString*(result\_entity,"utf-8");

EntityUtils.*consume*(result\_entity);

response.close();

client.close();

**return** result;

}

## GET方法-带Cookie

**public** **static** String mapToString(Map<String, Object> para) {

StringBuilder sBuilder =**new** StringBuilder();

**int** size=para.size();

**for**(Entry<String, Object> entry:para.entrySet()) {

sBuilder.append(entry.getKey()+"="+entry.getValue());

size--;

**if** (size>=1) {

sBuilder.append("&");

}

}

**return** sBuilder.toString();

}

**public** **static** String doGet(String url,Map<String, Object> para,CookieStore cookie) **throws** Exception {

//创建一个带有Cookie信息的client对象

RequestConfig gConfig =

RequestConfig.*custom*().setCookieSpec(CookieSpecs.***STANDARD***).build();

CloseableHttpClient client=HttpClients.*custom*().

setDefaultRequestConfig(gConfig).setDefaultCookieStore(cookie).build();

//get请求参数在url中

System.***out***.println(ReadProperties.*getValue*("baseurl")+url+"?"+*mapToString*(para));

HttpGet get=**new** HttpGet(ReadProperties.*getValue*("baseurl")+url+"?"+*mapToString*(para));

CloseableHttpResponse response =client.execute(get);

HttpEntity result\_entity=response.getEntity();

String result=EntityUtils.*toString*(result\_entity,"utf-8");

EntityUtils.*consume*(result\_entity);

response.close();

client.close();

**return** result;

}

## POST方法1-请求体为 JSONObject

**public** **static** String doPost(String url, JSONObject body) **throws** Exception {

CloseableHttpClient client=HttpClients.*createDefault*();

HttpPost post=**new** HttpPost(ReadProperties.*getValue*("baseurl")+url);

post.setHeader("content-type","application/json");

HttpEntity request\_entity=**new** StringEntity(body.toString(),"UTF-8");

post.setEntity(request\_entity);

CloseableHttpResponse response =client.execute(post);

HttpEntity result\_entity=response.getEntity();

String result=EntityUtils.*toString*(result\_entity,"utf-8");

EntityUtils.*consume*(result\_entity);

response.close();

client.close();

**return** result;

}

## POST方法-请求体为 String-带Cookie

**public** **static** String doPost(String url, String body,CookieStore cookie) **throws** Exception {

RequestConfig gConfig =

RequestConfig.*custom*().setCookieSpec(CookieSpecs.***STANDARD***).build();

CloseableHttpClient client=HttpClients.*custom*().

setDefaultRequestConfig(gConfig).setDefaultCookieStore(cookie).build();

HttpPost post=**new** HttpPost(url);

post.setHeader("content-type","application/json");

HttpEntity request\_entity=**new** StringEntity(body);

post.setEntity(request\_entity);

CloseableHttpResponse response =client.execute(post);

HttpEntity result\_entity=response.getEntity();

String result=EntityUtils.*toString*(result\_entity,"utf-8");

EntityUtils.*consume*(result\_entity);

response.close();

client.close();

**return** result;

}

## 返回Cookied的方法

**public** **class** Common {

**public** **static** CookieStore getCookie(String u\_name,String password) **throws** Exception, IOException {

JSONObject user=**new** JSONObject();

user.element("phoneArea", "86");

user.put("phoneNumber", u\_name);

user.element("password", password);

//创建带自定义Cookie的Client

RequestConfig gConfig =

RequestConfig.*custom*().setCookieSpec(CookieSpecs.***STANDARD***).build();

CookieStore cookie=**new** BasicCookieStore();

CloseableHttpClient client=HttpClients.*custom*().

setDefaultRequestConfig(gConfig).setDefaultCookieStore(cookie).build();

String loginurl="http://study-perf.qa.netease.com/common/fgadmin/login";

HttpPost post =**new** HttpPost(loginurl);

//设置请求体

HttpEntity userEntity=**new** StringEntity(user.toString());

post.setHeader("Content-Type","application/json");

post.setEntity(userEntity);

CloseableHttpResponse response=client.execute(post);

HttpEntity responseEntity=response.getEntity();

String result=EntityUtils.*toString*(responseEntity,"UTF-8");

EntityUtils.*consume*(responseEntity);

response.close();

client.close();

**return** cookie;

}

}

## 从属性文件读取信息

**public** **class** ReadProperties {

**public** **static** String getValue(String key) **throws** IOException {

FileInputStream input=**new** FileInputStream("conf/conf.properties");

Properties properties=**new** Properties();

properties.load(input);

input.close();

**return** properties.getProperty(key);

}

}

# 问题23：接口测试自动化-HttpClien 如果请求参数如包含非英文字符，需要encode转码

例如：

http://127.0.0.1:8080/supermarket/analysis/lookupprice?goodsCode={"pId":"123456"}

URLEncoder.*encode("{\"pId\":\"123456\"}", "UTF-8");*

# 问题24：接口测试自动化-HtmlUnit

## GET方法

**public** **static** String doGet(String url) **throws** Exception {

WebClient client = **new** WebClient();

WebRequest get = **new** WebRequest(**new** URL(url), HttpMethod.***GET***);

Page page = client.getPage(get);

WebResponse response = page.getWebResponse();

String result = response.getContentAsString();

client.close();

**return** result;

}

## GET方法带-Cookie

**public** **static** String doGet(String url,Set<Cookie> cookie) **throws** Exception {

WebClient client = **new** WebClient();

Iterator<Cookie> iterator=cookie.iterator();

**while**(iterator.hasNext()) {

client.getCookieManager().addCookie(iterator.next());

}

WebRequest get = **new** WebRequest(**new** URL(url), HttpMethod.***GET***);

Page page = client.getPage(get);

WebResponse response = page.getWebResponse();

String result = response.getContentAsString();

client.close();

**return** result;

}

## POST 方法

**public** **static** String doPost(String url, String body) **throws** Exception {

WebClient client = **new** WebClient();

WebRequest post = **new** WebRequest(**new** URL(url), HttpMethod.***POST***);

client.addRequestHeader("content-type", "application/json");

post.setRequestBody(body);

Page page = client.getPage(post);

WebResponse response = page.getWebResponse();

String result = response.getContentAsString();

client.close();

**return** result;

}

# 问题25：前端组成

负责：浏览器端的展现，页面包括了图片，文本框，样式。

Web前端有三部分组成，分别是HTML、CSS和JavaScript

# 问题26：Fiddler

Fiddler是一个HTTP协议调试代理工具，它能够记录并检查所有的本机与网络之间的http通讯，设置断点，查出所有的“进出”Fiddler的数据。其运行机制是在本机127.0.0.1监听8888端口的HTTP代理

# 问题27：JSON

JSON（JavaScript Object Notation ）轻量级的数据交换语言，以文字为基础，且易于阅读。

对象（object）:一个对象以“{”开始，并以“}”结束。每个对象包含一系列排序的名称/值对，每个名称/值对之间使用“,”区分。

名称/值（collection）：名称和值之间使用“:”隔开，一般的形式是：{key1:value,key2:value2}

JSON的四个特征：

数据在键值对中

数据由逗号分隔

方括号保存数组

花括号保存对象

# 问题28：Spring Boot注解

@SpringBootApplication 代表是Spring Boot 的启动类。

@RestController:1,表示一个Controller。2，表示当前这个Controller 下的所有的方法都会以json 格式的数据返回

@RequestMapping：将请求URL映射到整个类上 或某个特定的方法上,

# 问题29：Spring Boot实现接口

## GET方法

@RequestMapping("hello")

public String test1() {

return "hello ,test李焕贞";

}

## 返回带有Cookie信息的Get接口开发

@RequestMapping(value="/getCookie",method=RequestMethod.GET)

public String getCookies(HttpServletResponse response) {

Cookie cookie=new Cookie("login","true");

response.addCookie(cookie);

return "获得cookie成功";

}

## 要求携带Cookie访问的Get接口开发

@RequestMapping(value="/getwithCookie",method=RequestMethod.GET)

public String getWithCookies(HttpServletRequest request) {

Cookie [] cookies=request.getCookies();

if(Objects.isNull(cookies)) {

return "错误，必须携带Cookie";

}

for(Cookie cookie:cookies) {

if(cookie.getName().equals("login") &&

cookie.getValue().equals("true")) {

return "访问成功";

}

}

return "错误，必须携带Cookie";

}

## 要求携带参数的Get请求的开发

/\*

\* url?para1=value&para2=value2

\*/

@RequestMapping(value="/getByWithPara1",method=RequestMethod.GET)

public Map<String,Integer> getBypara1(@RequestParam Integer goodId) {

Map<String,Integer> goods=new HashMap<String,Integer> ();

goods.put("笔记本", 5000);

goods.put("洗衣液", 78);

return goods;

}

@RequestMapping(value="/getByWithPara2",method=RequestMethod.GET)

public String getName(@RequestParam String name) {

return "你输入的是"+name;

}

/\*

\* url/get/withpara/1/2

\*/

@RequestMapping(value="/getByWithPara2/{start}/{end}",method=RequestMethod.GET)

public Map<String,Integer> getBypara2(@PathVariable Integer start,

@PathVariable Integer end) {

Map<String,Integer> goods=new HashMap<String,Integer> ();

goods.put("笔记本", 8000);

goods.put("洗衣液", 78);

return goods;

}

## POST方法-Content-Type","application/x-www-form-urlencoded;

@RestController

@RequestMapping("/post")

public class MyPostMethod {

public static Cookie cookie;

@RequestMapping(value = "/loginmap", method = RequestMethod.POST)

public String doPostLogin(HttpServletResponse response,

@RequestParam(value = "u\_Name", required = true) String userName,

@RequestParam(value = "u\_Pwd", required = true) String userPwd) {

if (userName.equals("admin") && userPwd.equals("admin")) {

cookie = new Cookie("login", "true");

response.addCookie(cookie);

return "登录成功";

}

return "登录失败";

}

public static Cookie cookie;

@RequestMapping(value = "/loginmap", method = RequestMethod.POST)

public String doPostLogin(HttpServletResponse response,

@RequestParam(value = "u\_Name", required = true) String userName,

@RequestParam(value = "u\_Pwd", required = true) String userPwd) {

if (userName.equals("admin") && userPwd.equals("admin")) {

cookie = new Cookie("login", "true");

response.addCookie(cookie);

return "登录成功";

}

return "登录失败";

}

# 问题30 安装request命令

**在线安装：pip install requests**

**升级安装：pip install -U requests**

**检查是否安装成功：pip show requests**

**导入 Requests 模块：import requests**

Requests 允许使用params关键字参数，以一个字符串字典来提供参数化

# 问题31：request使用

## GET方法

**res=requests.get(url)**

或者 res=requests.get(url,params={"goodsId":1})

**print(res.json())**

## POST方法

user={"username":"lihuanzhen","password":"123456"}  
headers={"content-type":"application/x-www-form-urlencoded"}  
resp=requests.post(url,data=user,headers=headers)  
resp\_json=resp.json()

user={"phoneArea":"86","phoneNumber":"20000000000","password":"netease123"}  
header={"content-type":"application/json"}  
res1=requests.post(login\_url,data=json.dumps(user),headers=header)  
print(res1.json())

获取Cookie:r.cookies

使用Cookie:requests.get(url, cookies=cookies)

user={"phoneArea":"86","phoneNumber":"20000000000","password":"netease123"}  
header={"content-type":"application/json"}  
res1=requests.post(login\_url,data=json.dumps(user),headers=header)  
print(res1.json())

res2=requests.get(address\_url,cookies=res1.cookies)

# 问题32：Pyhon接口自动化框架的实施步骤

**1、Common类存储域名，存储获得cookie方法**

**2、封装SendHttp类发送Http请求**

**3、使用unittest管理测试用例**

**4、使用HTMLTestRunner生成测试报告**

**5、使用xlrd包操作excel文件，实现数据驱动**

**6、使用smtplib将测试报告发送邮件**