[1 接口测试 2](#_Toc204528397)

[1.1 接口的作用和组成 2](#_Toc204528398)

[1.1.1 接口的作用 3](#_Toc204528399)

[1.1.2 接口组成 3](#_Toc204528400)

[1.1.2.1 端点 3](#_Toc204528401)

[1.1.2.2 方法 3](#_Toc204528402)

[1.1.2.3 请求 4](#_Toc204528403)

[1.1.2.4 响应 4](#_Toc204528404)

[1.2 接口测试基本概念、目的、优势 4](#_Toc204528405)

[1.2.1 接口测试的目的 4](#_Toc204528406)

[1.2.2 接口测试的优势 5](#_Toc204528407)

[1.2.3 接口测试的验证 6](#_Toc204528408)

[1.3 接口测试的应用与接入阶段 6](#_Toc204528409)

[1.3.1 主要应用 6](#_Toc204528410)

[1.3.2 介入阶段 7](#_Toc204528411)

[1.4 接口测试的测试用例设计 7](#_Toc204528412)

[1.4.1 正向用例与负向用例 7](#_Toc204528413)

[1.4.2 接口协议与状态码测试 8](#_Toc204528414)

[1.4.3 参数组合与依赖测试 8](#_Toc204528415)

[1.4.4 数据驱动与异常流测试 8](#_Toc204528416)

[1.4.5 接口间状态传递与幂等性测试 9](#_Toc204528417)

[1.4.6 安全与性能专项测试 9](#_Toc204528418)

[1.4.7 Mock与第三方依赖测试 9](#_Toc204528419)

[1.5 接口测试的流程 9](#_Toc204528420)

[1.6 接口测试难点与重要概念 10](#_Toc204528421)

[1.6.1 接口流程原理 10](#_Toc204528422)

[1.6.2 RESTful API 10](#_Toc204528423)

[1.6.2.1 REST的基本概念 10](#_Toc204528424)

[1.6.2.2 RESTful API的设计原则 11](#_Toc204528425)

[1.6.2.3 RESTful API的优点 11](#_Toc204528426)

[1.6.3 认证与授权 12](#_Toc204528427)

[1.6.3.1 认证 12](#_Toc204528428)

[1.6.3.2 授权 14](#_Toc204528429)

[1.6.4 Mock 16](#_Toc204528430)

[1.6.4.1 MOCK 在接口测试中的应用 17](#_Toc204528431)

[1.6.4.2 MOCK 在接口测试中的关键作用 17](#_Toc204528432)

[1.6.4.3 MOCK 的实现方法 18](#_Toc204528433)

[1.6.5 幂等性 18](#_Toc204528434)

[1.6.5.1 常见的幂等性应用模式 18](#_Toc204528435)

[1.6.5.2 接口测试中的幂等性验证 19](#_Toc204528436)

[1.6.5.3 幂等性测试的重要性 20](#_Toc204528437)

[1.6.6 接口依赖 20](#_Toc204528438)

[1.6.6.1 接口依赖的类型 20](#_Toc204528439)

[1.6.6.2 接口依赖的挑战 20](#_Toc204528440)

[1.6.6.3 接口依赖的解决方案 21](#_Toc204528441)

[1.6.6.4 常见的依赖关系管理方法 21](#_Toc204528442)

[1.6.6.5 接口依赖的测试要点 22](#_Toc204528443)

[1.7 Charles抓包 22](#_Toc204528444)

[1.7.1 抓包的应用 22](#_Toc204528445)

[1.7.2 Charles基本介绍 22](#_Toc204528446)

[1.7.2.1 核心功能 22](#_Toc204528447)

[1.7.2.2 Charles抓包原理 23](#_Toc204528448)

[1.7.3 charles抓包详解 23](#_Toc204528449)

[1.7.4 抓包分析BUG 23](#_Toc204528450)

[1.7.5 常见定位BUG方案 23](#_Toc204528451)

[1.7.6 charles抓包实战 23](#_Toc204528452)

[1.8 POSTMAN 24](#_Toc204528453)

[1.8.1 postman基本操作 24](#_Toc204528454)

[1.8.2 postman环境配置 24](#_Toc204528455)

[1.8.3 postman参数关联 24](#_Toc204528456)

[1.8.4 postman的cookies管理 24](#_Toc204528457)

[1.8.5 postman自动断言 24](#_Toc204528458)

[1.8.6 postman脚本编写 24](#_Toc204528459)

[1.8.7 postman批量执行 24](#_Toc204528460)

[1.8.8 应用postman对上市公司的接口进行测试 24](#_Toc204528461)

# 接口测试

**接口**（Interface）指的是一种定义了**不同软件系统或组件之间交互规则的契约**。简单来说，接口定义了“谁”与“谁”如何通信，以及“如何”通信。

## 接口的作用和组成

接口指的是**API（Application Programming Interface）接口**，它是不同软件系统之间的通信桥梁。API接口通过定义一组标准化的方法，使得不同的系统或组件可以相互交换数据并完成操作，而不需要关心对方的内部实现。

API接口往往基于网络通信协议（如HTTP、WebSocket等）实现，因此它的核心特征是远程调用，即**调用者通过网络请求来访问接口，接口提供者根据请求处理相应的操作并返回结果**。

### 接口的作用

接口在软件系统中的作用，尤其是在接口测试语境中，具有以下核心作用：

1. **系统解耦​​**

通过标准化接口协议（如HTTP API）实现**系统间独立交互，屏蔽内部实现细节**，降低模块间耦合度。

1. ​**模块化与扩展性​​**

模块化设计将功能抽象为接口，其他模块**通过接口调用功能，确保系统扩展或修改时仅需维护接口规范，提升灵活性与兼容性**。

1. ​**标准化与一致性​​**

统一接口协议（如RESTful API）定义数据传输格式、请求方法等，确保跨系统数据交换的一致性。

1. ​**促进自动化测试​​**

接口的标准化特性便于编写自动化脚本，通过模拟请求快速验证功能逻辑、性能及异常处理，显著提升测试效率和覆盖率。

1. ​**提高系统互操作性​​**

基于标准化接口，不同技术栈或平台（如跨语言服务）可无缝协作，增强跨系统数据交换与功能调用的能力。

### 接口组成

**API接口的关键要素：**

* **端点（Endpoint）**：指API的访问URL地址。
* **方法（Method）**：指使用的HTTP动词，如GET、POST、PUT、DELETE等。
* **请求参数（Request Parameters）**：用于向API发送的数据，包括路径参数、查询参数、请求体等。
* **响应数据（Response Data）**：API返回的数据，通常是JSON或XML格式，包含业务逻辑处理后的结果。

#### 端点

API接口一般是一个由**域名+端点**构成的URL，其标识资源的位置。

例如：GET [https://api.weather.com/v1/city/{cityId}/forecast](https://api.weather.com/v1/city/%7bcityId%7d/forecast)中，https://api.weather.com是域名部分，/v1/city/{cityId}/forecast是API资源路径，也就是所谓的EndPoint、端点。

#### 方法

详见[HTTP](#_HTTP协议（HyperText_Transfer_Protocol）)、[Requsests](#_Requests)

请求方法定义了客户端和服务器之间**交互的方式**。最常用的请求方法有：

* **GET**：用于从服务器获取数据。
* **POST**：用于向服务器发送数据，通常用于创建或提交数据。
* **PUT**：用于更新已有的数据。
* **DELETE**：用于删除数据。
* **PATCH**：用于局部更新数据。

#### 请求

请求即发送特定HTTP方法对服务器进行请求时的数据包，一般包括请求体、请求头：

##### 请求头

请求头包含了关于请求的信息，如内容类型、授权信息、缓存控制等。常见的请求头字段有：

* **Content-Type**：指示请求体的数据格式，常见的有application/json、application/xml等。
* **Authorization**：用于传递认证信息，通常是Bearer Token或Basic Auth。
* **User-Agent**：指示发起请求的客户端类型（如浏览器、Postman等）。

##### 请求体

请求体是发送给服务器的数据，通常只在POST、PUT、PATCH等方法中使用。请求体的格式一般是JSON、XML或者表单数据等。需要配合设定的接口字段使用。

#### 响应

响应是服务器接收到请求后返回的消息，可以通过响应来验证请求、获取资源。

响应最重要的内容是状态码、响应体（、以及响应头）：

##### 状态码

HTTP状态码是API响应的重要组成部分，表示请求的处理结果。常见的状态码包括：

* 200 OK：请求成功并返回数据。
* 201 Created：请求成功，资源被创建。
* 400 Bad Request：请求参数错误或格式不合法。
* 404 Not Found：请求的资源不存在。
* 500 Internal Server Error：服务器内部错误。

一般来说，都要先检查响应状态码才能查看资源。

##### 响应体

响应体就是服务器返回的数据，通常是JSON或XML格式，包含请求结果或错误信息。一般服务器会根据特定的规则返回特定的资源、响应体。

## 接口测试基本概念、目的、优势

***接口测试****（API Testing）是指对软件系统中的各个模块通过API接口进行交互时，验证其是否按预期正常工作的测试活动*。API接口是不同系统或模块之间的通信桥梁，它规定了系统间数据交换和功能调用的标准。在接口测试中，测试人员不关注前端界面，而是直接通过API接口进行测试，模拟客户端向服务器发起请求，验证响应结果是否符合预期。

接口测试的**主要测试对象是系统的后端接口，通常不涉及UI**。

### 接口测试的目的

接口测试的核心目的是确保系统中的API接口按照预期的功能、性能和安全要求正常运行，并且能够处理正常、边界、错误等各种场景，具体有：

1. **验证功能正确性​​**

确保接口按设计处理请求并返回预期结果，包括参数有效性、响应数据（内容/格式/类型）的正确性。

1. ​**测试错误处理机制​​**

验证接口对非法输入（如参数缺失、格式错误）的容错能力，返回标准状态码（如400）及明确错误描述。

1. ​**评估性能与稳定性​​**

检测接口在正常/高并发场景下的响应时间、吞吐量及资源占用，确保满足性能阈值要求。

1. **​保障安全性​​**

防御SQL注入、XSS等攻击，强化身份验证（如Token校验）与权限控制，阻断未授权访问。

1. ​**确保数据完整性​​**

验证接口对特殊字符、空值、边界值的处理能力，保证跨系统数据传递的完整性和一致性。

1. ​**维护跨版本兼容性​​**

确保接口版本迭代时旧版本仍可用，避免升级导致现有系统异常。

### 接口测试的优势

接口测试相比其他类型的测试（如UI测试、单元测试等），具有一些独特的优势：

总结来其实就是：速度快、强自动化、独立。

1. **能够较早发现问题**

在系统开发的早期阶段就可以开始进行，尤其是在后端开发完成后，前端开发还未完成时。接口测试能够帮助开发团队尽早发现系统的功能性和稳定性问题，避免后期整合时出现大规模的缺陷。

* **优势**：减少了等待前端开发完成的时间，加快了测试周期。

1. **高效的自动化测试**

通常不依赖于图形用户界面（GUI），直接通过HTTP协议模拟请求和响应，因此非常适合自动化。接口测试可以通过编写脚本进行自动化测试，快速验证接口功能，尤其适合持续集成（CI/CD）环境中进行自动化验证。

* **优势**：可以显著提高测试效率，减少手动测试工作量，并且能快速响应代码的变化。

1. **测试范围广泛**

接口测试可以覆盖到系统的核心业务逻辑，特别是在微服务架构下，系统往往依赖多个接口进行交互。接口测试能够全面测试这些模块之间的通信是否正常，包括正常情况、异常情况、边界情况等多个场景。

* **优势**：能覆盖到更多的测试场景，尤其是在复杂的分布式系统和微服务架构中，接口测试非常重要。

1. **独立性强，不依赖于前端**

接口测试并不依赖于用户界面的实现，而是直接通过API接口进行验证。因此，测试人员可以独立于前端开发人员，尽早对后端API进行测试，而不需要等待前端接口的实现。

* **优势**：能够在前端开发之前或与前端并行进行测试，减少了前后端依赖的时间问题。

1. **提高系统的可靠性**

接口测试能够帮助验证接口的正确性、性能、稳定性和安全性，确保系统模块间的交互不会出现问题。这种验证对于系统的长期稳定性非常重要。

* **优势**：提高了系统的可靠性和稳定性，减少了生产环境中因接口问题导致的系统故障。

1. **能够模拟真实的用户行为**

接口测试模拟了真实用户与后端系统交互的过程，可以有效测试API接口在真实使用场景下的表现。通过模拟大量的请求和多种不同的场景，可以验证API接口在实际生产环境中的承载能力和稳定性。

* **优势**：测试更加贴近实际使用场景，能够发现潜在的性能瓶颈和稳定性问题。

1. **支持跨平台和跨系统的集成**

接口测试能够验证不同系统或服务之间的集成和互操作性，尤其是在微服务架构中，不同的微服务通过API进行通信。接口测试可以确保各个服务之间能够正确交互，保证系统的整体稳定性。

* **优势**：支持跨平台和跨系统的集成验证，对于多系统互通性验证非常重要。

### 接口测试的验证

接口测试的验证最主要就是判断是否符合需求文档、API设计文档的设计需求，包括：

1. **判断请求**:

是否正确, 系统默认的请求成功，会返回200[状态码](https://so.csdn.net/so/search?q=%E7%8A%B6%E6%80%81%E7%A0%81&spm=1001.2101.3001.7020), 假如请求错误返回400, 404, 500等状态码

1. **判断数据**:

返回数据的正确性与完整性，以及数据库的数据是否对应变更。

1. **判断安全性**:

接口一般不会随意暴露在网上被其他人任意调用,一般我们会对接口做出一些限制,比如请求次数、请求频率限制等等

## 接口测试的应用与接入阶段

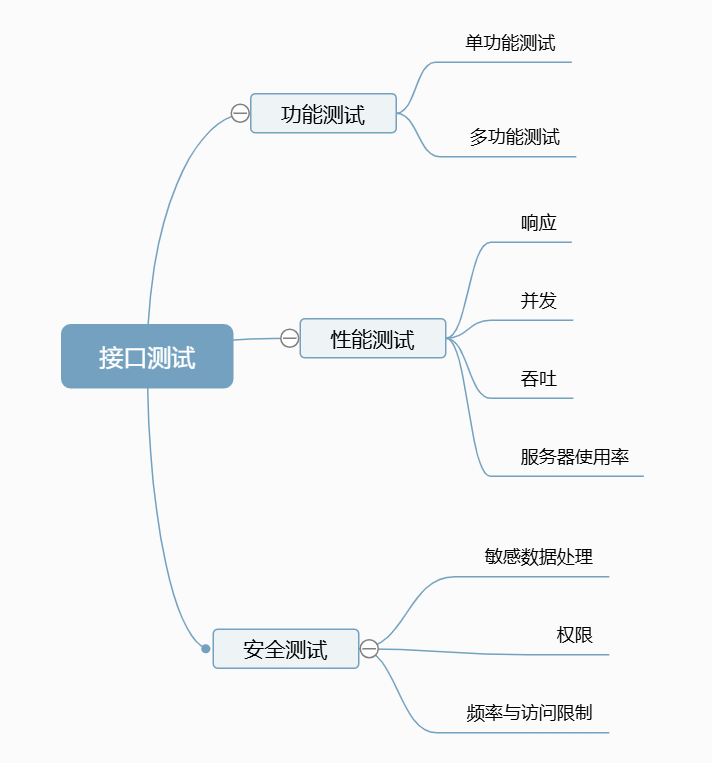
### 主要应用

接口测试可以实现如下功能：

1. ​**​功能验证​**​：确保接口参数、请求/响应格式、业务逻辑符合预期，例如校验数据传递、状态码和错误提示。
2. ​**​性能测试​**​：评估接口在高并发、大数据量下的稳定性（如响应时间、吞吐量）。
3. ​**​安全测试​**​：检测漏洞（如SQL注入、未授权访问）和权限控制（身份认证、数据加密）。
4. ​**​数据校验​**​：检查接口返回数据的完整性、一致性和格式（如JSON/XML结构）。
5. ​**​异常处理​**​：模拟异常场景（如超时、错误参数），验证系统的容错能力

常应用于：

1. ​**​微服务/分布式系统​**​
2. ​**​第三方服务对接​**​
3. ​**​前后端分离开发​**​
4. ​**​系统重构或升级​**​
5. ​**​多系统数据交互​**​



### 介入阶段

接口测试贯穿于软件开发的各个阶段，主要是因为从前期需求、接口设计就与接口测试有关联，然后到后期集成部署后，自动化的接口测试也是非常常见的应用，具体介入时机：

1. **需求与设计阶段**：接口设计审核。
2. **开发阶段**：接口功能验证。
3. **单元测试阶段**：接口独立测试。
4. **集成测试阶段**：验证模块间的接口协作。
5. **性能测试阶段**：压力测试和性能验证。
6. **回归测试阶段**：验证更新后接口的稳定性。
7. **生产环境监控阶段**：实时监控接口运行状态。

接口测试的及时介入能够有效提升软件的质量，确保系统的可靠性、稳定性和性能。

## 接口测试的测试用例设计

接口测试遵循**传统黑盒测试**的[核心方法](#_核心方法学)，但某种程度上其属于灰盒测试，其特殊性在于接口的​​协议交互性、数据驱动性、状态依赖性​​等特点，需要结合接口特性设计更贴合实际场景的测试用例。

整体来说，接口测试与传统测试的主要不同在于利用接口的特性，对接口传输的请求、传输、响应进行特殊的设计处理。

### 正向用例与负向用例

整体来说，用例设计包括正向用例和负向用例设计，​**​正向用例​**​验证接口在预期输入下的正确行为，而​**​负向用例​**​则确保接口能够优雅地处理异常场景。

* **正向用例**：测试接口在正常情况下的行为。
* **负向用例**：测试接口在异常输入、错误请求时的处理方式。

### 接口协议与状态码测试

**协议规范验证**：

* + 在测试HTTP接口时，需验证接口是否遵循RESTful规范，特别是请求方法（GET、POST、PUT、DELETE等）的正确性；
  + 验证Content-Type、Authorization等请求头是否按照**协议要求**传递。

**状态码覆盖**：

* + 针对不同的业务场景，设计用例覆盖常见的HTTP状态码（如200、201、400、401、403、404、500等），如：
    - 无效Token时，接口应返回401 Unauthorized；
    - 资源不存在时，返回404 Not Found；
    - 参数校验失败时，返回400 Bad Request。

**错误码映射**：

* + 验证接口返回的业务错误码是否与**文档**中的定义一致。比如，错误码1001应对应“用户不存在”。

### 参数组合与依赖测试

**必填参数校验**：

* + 测试接口在缺失必填参数时的响应（例如跳过 user\_id 参数时，接口是否正确报错）。

**参数互斥性**：

* + 对于互斥的参数（如某些参数只能同时存在或不能同时存在），需要设计用例验证接口能否正确处理这些情况。

**参数依赖链**：

* + 设计用例验证接口间的依赖关系。例如，某些接口需要先调用其他接口（如创建订单前需要用户登录），应覆盖这些依赖链的完整性。

**参数组合爆炸**：

* + 当接口支持多个参数时，可使用正交试验法或Pairwise工具（如AllPairs）来生成有效的参数组合，避免冗余的测试用例，同时保证全面覆盖。

### 数据驱动与异常流测试

**异常数据注入**：

* + 验证接口在面对异常数据时的表现，特别是：
    - 超长字符串、特殊字符（如SQL注入字符 ' OR 1=1--）、Emoji等输入，确保接口能正确处理或过滤这些数据；
    - 输入非法数据类型（如字符串代替数字，布尔值代替枚举类型）。

**数据格式校验**：

* + 验证接口响应的JSON/XML格式是否正确，确保字段类型、结构完整性（如 amount 字段是否为数值类型，日期字段是否符合 YYYY-MM-DD 格式）。

**空值/默认值处理**：

* + 检查接口对空值（null、空字符串、空数组等）的处理逻辑，确保接口能妥善处理缺失或不完整的数据。

### 接口间状态传递与幂等性测试

**状态一致性**：

* + 在执行涉及状态变更的接口时，验证接口操作后的状态是否符合预期。例如，支付接口调用后，订单状态是否从“待支付”变为“已支付”。

**幂等性测试**：

* + 对于支持重复提交的接口（如订单创建），多次发送相同请求应确保操作只执行一次，而不会产生副作用。

**接口依赖时序**：

* + 验证接口的时序要求。例如，某些接口需先调用 create 后再调用 update，需要确保接口按正确的顺序调用，否则应返回资源不存在等错误。

### 安全与性能专项测试

**安全性测试**：

* + 验证接口是否遵循安全协议（如HTTPS）；
  + 确保敏感信息（如密码、身份证号等）不会以明文形式传输；
  + 验证接口鉴权机制是否存在漏洞，如绕过Token验证直接访问接口；
  + 检查接口是否具备频率限制机制，防止恶意刷接口（例如，短时间内多次调用短信发送接口时是否触发限流）。

**性能测试**：

* + 对单个接口进行并发压力测试，模拟大量并发请求，验证接口是否能稳定响应；
  + 测试接口响应时间是否符合服务水平协议（如SLA），例如95%的请求需在200ms内返回；
  + 验证系统在高并发下的数据库连接池、内存等资源是否有泄漏，避免造成性能瓶颈。

### Mock与第三方依赖测试

**模拟第三方异常**：

* + 使用Mock工具（如WireMock）模拟第三方接口的异常情况（如超时、错误码等），验证系统的容错能力。

**依赖隔离测试**：

* + 测试接口对第三方依赖的处理能力，确保接口在外部服务不可用时能够正常响应。例如，在支付接口依赖银行网关时，模拟银行接口的成功与失败，验证系统的业务逻辑是否正确。

## 接口测试的流程

接口测试流程与整体测试流程大致相同，但会在需求文档上加上接口说明文档的分析。

大致流程为：

1. **需求分析与测试计划**：理解接口的功能需求，确定测试目标和工具，选择合适的测试环境。
2. **测试用例设计**：

* **功能测试**：验证接口按预期工作。
* **异常流与边界值**：验证异常输入和边界情况。
* **安全性测试**：测试常见安全漏洞，如SQL注入等。
* **性能测试**：测试接口在高负载下的表现。
* **幂等性与状态传递**：验证接口的状态一致性和重复提交的处理。
* **Mock与依赖测试**：模拟第三方依赖，确保系统的容错能力。

1. **用例执行**：手动或自动化执行设计好的测试用例。
2. **结果验证与问题定位**：验证接口响应、定位问题并解决。
3. **性能与安全测试**：进行压力测试和安全性检查。
4. **回归测试与反馈**：修改后进行回归测试，提供反馈。
5. **测试报告与总结**：编写测试报告，分析测试结果，提出改进建议。

## 接口测试难点与重要概念

### 接口流程原理

前后端/系统之间通过接口进行交互的全过程机制，包括请求发送、处理逻辑、响应返回等各个环节的原理与流程。

### RESTful API

**RESTful API** 是基于 **REST**（Representational State Transfer，表现层状态转移）架构风格的一种**Web服务接口设计方式**，旨在通过网络提供和消费资源。RESTful API 是Web应用程序的核心，它遵循REST原则，并通过HTTP协议与客户端进行数据交互。它的设计注重简洁、可扩展性以及一致性，广泛应用于现代Web开发中。

**核心目标**是设计一个简单、轻量、可扩展的Web服务，使得客户端与服务器之间可以通过标准的HTTP协议进行资源的交互。它专注于资源的表现（如JSON或XML），并通过清晰、统一的接口让开发者能够轻松操作这些资源。

#### REST的基本概念

**REST**是由 **Roy Fielding** 在他的博士论文中提出的。其不是一种技术，而是一种架构风格，**强调分层系统的设计、无状态的通信和资源的表现形式**。

RESTful API遵循这些REST设计原则：

* **资源（Resources）**

在REST架构中，所有**可以被访问和操作的数据**都被视为“资源”。资源通过唯一的标识符（通常是URL）进行标识。每个资源都有自己的URI（统一资源标识符）。

* **HTTP方法**

RESTful API主要通过HTTP协议中的不同方法来操作资源。这些方法包括：

* **GET**：获取资源。例如，获取用户信息。
* **POST**：创建资源。例如，创建一个新的用户。
* **PUT**：更新资源。例如，更新一个已有用户的详细信息。
* **DELETE**：删除资源。例如，删除某个用户。
* **PATCH**：对资源进行部分更新。
* **无状态（Stateless）**

在REST架构中，每个请求都应该包含足够的信息，以使服务器能够理解该请求，不依赖于服务器之前的状态。这意味着每个请求都是独立的，**服务器不应该保存关于客户端状态的信息**。这确保了系统的可伸缩性和简洁性。

* **客户端-服务器架构（Client-Server）**

REST**分离了客户端和服务器的职责**。客户端负责用户界面的呈现和交互，服务器负责数据存储和资源管理。这种分离使得系统更加灵活，能够更容易地扩展和维护。

* **统一接口（Uniform Interface）**

RESTful API遵循统一的接口标准，确保不同的资源操作具有一致性。统一接口的设计使得客户端可以理解如何与不同资源交互。通过标准化的URI、HTTP方法以及标准的响应格式，RESTful API简化了客户端与服务器之间的交互。

#### RESTful API的设计原则

1. **每个资源都有唯一的URI（Uniform Resource Identifier）**：
   * 每个资源应通过URL唯一标识。例如，获取一个用户的接口可能是 GET /users/{id}，获取某个特定ID用户的接口为 GET /users/123。
2. **使用标准的HTTP动词**：
   * 使用HTTP的动词（GET、POST、PUT、DELETE等）来对应不同的操作：
     + **GET**：读取资源。
     + **POST**：创建资源。
     + **PUT**：更新资源。
     + **DELETE**：删除资源。
3. **返回适当的HTTP状态码**：
   * RESTful API应该根据请求的结果返回合适的HTTP状态码。例如：
     + **200 OK**：请求成功，通常是GET请求。
     + **201 Created**：资源创建成功，通常是POST请求。
     + **400 Bad Request**：请求格式错误。
     + **404 Not Found**：资源未找到。
     + **500 Internal Server Error**：服务器内部错误。
4. **无状态性**：
   * 每个请求必须是独立的，不依赖于之前的请求。所有信息应包含在请求中（如身份认证、请求参数等），服务器不应保存客户端的会话状态。
5. **使用标准的数据格式**：
   * RESTful API通常使用 **JSON** 或 **XML** 格式来传输数据。JSON是最常见的数据格式，因为它结构简洁、易于理解和处理。
6. **可缓存性**：
   * RESTful API响应应指明是否可缓存。如果一个请求的响应可以缓存，那么客户端可以保存响应，以便在下一次请求相同资源时使用缓存的数据，减少服务器负载。

#### RESTful API的优点

1. **简洁和可扩展**：
   * RESTful API设计简洁直观，操作易懂，遵循标准的HTTP协议和方法，具有较好的可扩展性，支持多种客户端（如Web、移动端等）。
2. **无状态性**：
   * 无状态设计意味着每个请求都是独立的，服务器无需保存状态，因此，系统可以更容易地扩展和负载均衡。
3. **灵活性**：
   * 客户端和服务器分离，客户端可以独立于服务器开发，并且可以根据需要更新和扩展接口。
4. **标准化**：
   * RESTful API遵循统一的接口规范，减少了学习成本，且各个系统之间的集成也更为方便。
5. **基于HTTP协议**：
   * RESTful API基于HTTP协议，可以使用现有的网络基础设施和Web服务工具，不需要额外的协议支持。

### 认证与授权

接口的**认证**（Authentication）和**授权**（Authorization）是确保接口安全性的重要机制。它们分别确保用户身份的合法性和用户是否有权限访问特定资源。前者关注“你是谁”，后者关注“你能干什么”。

接口测试中对认证和授权的验证至关重要，它保证了系统的安全性、稳定性和合法性。

#### 认证

认证（Authentication）是验证用户身份的过程，目的是确保用户是他们所声称的那个人。简而言之，认证是判断“你是谁”的过程。只有认证通过，才能继续执行后续的操作（如访问资源）。

下面是常见的认证方法：

##### 基本验证

基本验证是最简单的一种认证方式，客户端通过在请求的头部携带用户名和密码进行身份验证。用户名和密码通常使用 Base64 编码，并在 Authorization 头部传输给服务器。

* **原理**
* 客户端向服务器发送请求时，在请求头中包含一个 Authorization 字段，格式为：Authorization: Basic base64(username:password)。
* 服务器接收到请求后，解码 Authorization 字段并校验用户名和密码是否正确。如果验证通过，允许继续处理请求，否则返回 401 Unauthorized 错误。
* **接口测试中的注意点**
* **有效性验证**：确保 Authorization 头部传递的用户名和密码正确时，返回成功响应（如 200 OK 或 201 Created）。
* **无效凭证验证**：当用户名或密码错误时，服务器应返回 401 Unauthorized 状态码，且不能泄露过多错误信息（如明确提示“用户名错误”或“密码错误”）。
* **安全性测试**：由于 Basic Authentication 的凭证是明文传输的，必须确保请求使用 HTTPS 来加密传输。
* **边界测试**：测试不同长度的用户名、密码，确保服务器不会因为过长的输入而崩溃或泄露信息

##### Token令牌认证

Token 令牌认证是一种使用 Token 进行认证的方式。**客户端在登录时提供用户名和密码，服务器验证通过后生成一个 Token，客户端在后续的请求中携带这个 Token 进行认证**。

* 就像进入公司门禁需要刷卡，Token相当于门禁卡：无需反复输入密码，刷卡（携带Token）即可证明身份。
* **Token详解**

**Token​**​是一个短期的、加密的字符串，由服务端生成并颁发给客户端，用于​**​代替用户的身份信息​**​。

* **常见Token类型**
* **JWT（JSON Web Token）​​**

由三部分组成（Header.Payload.Signature），用点分隔。

* + ​​Header​​：算法和类型（如HS256、JWT）。
  + ​​Payload​​：携带用户信息（如用户ID、角色）和过期时间。
  + ​​Signature​​：对前两部分的签名，防篡改。
* ​​**​​OAuth 2.0 Access Token​​**
* **​​**用途​​：第三方应用通过授权服务器（如微信、Google）获取的访问令牌。
* ​​特点​​：通常有时间限制，需定期刷新**。**
* **​​自定义Token​​**
* **​**服务端用加密算法（如UUID或随机字符串）生成，存储于数据库或缓存中。
* ​​验证流程​​：客户端传递Token，服务端查询是否有效。
* **Token生命周期**
* **​​生成​​**：用户登录成功后，服务端生成Token并返回给客户端。
* ​​**传递**​​：客户端将Token放入请求头（如 Authorization: Bearer <Token>）。
* ​​**验证​​**：服务端解密Token，检查签名、有效期和权限。
* ​​**过期/刷新​​**：

​​短期Token​​：过期后需重新登录（如JWT的exp字段）。

​​长期Token​​：通过Refresh Token重新获取Access Token。

* **原理**
* 客户端发送用户名和密码给认证服务器，认证服务器验证通过后生成一个 Token（如 JWT）。
* 客户端在随后的请求中通过 HTTP 请求头（通常为 Authorization: Bearer <token>）携带该 Token。
* 服务器接收到请求后，通过验证 Token 的有效性来判断是否授权。如果 Token 有效，允许访问接口，否则返回 401 Unauthorized。
* **接口测试中的注意点**
* **Token 生成测试**：确保服务器能够根据正确的用户名和密码生成有效的 Token，并将其返回给客户端。
* **Token 验证测试**：测试 Token 是否正确传递至 Authorization 头部。验证服务器是否能正确验证 Token，并对无效或过期 Token 返回 401 Unauthorized 错误。
* **Token 过期验证**：Token 通常有有效期，测试接口是否能正确处理过期的 Token，返回 401 Unauthorized 或相应的错误信息。
* **Token 安全性**：验证 Token 是否有足够的安全性，特别是对 JWT 进行签名和加密，防止 Token 被篡改。

##### API Key密钥认证

API Key 认证是一种通过唯一的密钥来识别和验证客户端的方式。每个用户或客户端应用会被分配一个唯一的 API 密钥，客户端在请求时将该密钥传递给服务器。

* **原理**
* 客户端将 API 密钥作为请求的一部分（通常是 URL 参数或请求头部）发送给服务器。
* 服务器根据传递的 API 密钥验证客户端身份，允许或拒绝访问。
* **接口测试中的注意点**
* **有效密钥验证**：确保服务器能够根据有效的 API 密钥提供访问权限，返回成功的响应。
* **无效密钥验证**：测试无效或缺失的 API 密钥时，服务器应返回 401 Unauthorized 或 403 Forbidden 错误。
* **密钥泄露防护**：测试 API 密钥的传递方式，确保 API 密钥不会在日志、URL 中暴露，必须使用 HTTPS 来加密传输。
* **权限验证**：测试是否能够对不同的 API 密钥分配不同的权限，确保权限控制的粒度。

##### Session-Based认证

Session-Based 认证是一种基于会话的认证机制，客户端登录后，服务器会创建一个会话（Session），并返回一个**会话 ID**。客户端在每次请求时通过 Cookie 或请求头将会话 ID 发送给服务器。

* **原理**
* 用户登录成功后，服务器生成一个会话 ID，并通过 Cookie 或响应头返回给客户端。
* 客户端在随后的请求中携带会话 ID，服务器通过查找该会话 ID 是否有效来判断用户是否已认证。
* **接口测试中的注意点**
* **会话创建和验证**：测试服务器是否能够成功创建会话，并且会话 ID 是否正确返回给客户端。
* **会话超时测试**：测试会话在一段时间后是否会过期，并确保服务器返回 401 Unauthorized 或相关错误。
* **会话管理**：测试不同的会话 ID，确保每个用户的会话是独立的，且会话不会被篡改。
* **安全性测试**：验证会话 ID 是否通过加密传输（如 HTTPS），防止会话劫持或伪造。
* **跨站请求伪造（CSRF）防护**：确保会话认证不会受到 CSRF 攻击，测试是否启用了 CSRF 防护机制。

#### 授权

授权（Authorization）是指在用户身份验证之后，控制和限制用户或应用程序访问某些资源或操作的权限。授权机制确保只有被授权的用户能够执行特定的操作或访问特定的数据。

下面是常见的授权方法：

##### 基于角色的访问控制（RBAC）

基于角色的访问控制（Role-Based Access Control，RBAC）是最常见的一种授权方式。用户被分配到特定的角色，系统根据角色的权限来限制用户访问资源。每个角色对应一组权限，用户只能访问其角色允许的资源和执行授权的操作。

* **原理**
* 系统中定义多个角色（例如：管理员、普通用户、访客等）。
* 每个角色拥有特定的访问权限。
* 用户根据其角色被授予或限制对某些资源（API、数据、操作等）的访问权限。
* **注意点**
* **权限校验**：确保 API 接口根据用户的角色进行权限校验，只有具备足够权限的用户才能执行相应操作。
  + **有效角色测试**：测试具有正确角色的用户是否能够访问其权限范围内的接口，并且操作成功。
  + **无效角色测试**：测试没有权限的用户（例如普通用户尝试执行管理员操作）是否被拒绝访问，接口应返回 403 Forbidden。
* **边界测试**：测试角色权限的边界情况，验证权限的粒度是否足够细致（如管理员和普通用户在相同接口上的权限差异）。
* **授权数据验证**：测试在 RBAC 控制下，用户仅能访问被授权的资源，防止越权访问。
* **角色修改测试**：测试用户角色变化后，权限是否随之正确调整（例如，将管理员降级为普通用户，确保其不能执行管理员操作）。

##### 基于属性的访问控制（ABAC）

基于属性的访问控制（Attribute-Based Access Control，ABAC）是一种灵活的授权控制模型，它依据多个属性（如用户属性、资源属性、环境条件等）来进行决策。不同于 RBAC 中基于角色的权限管理，ABAC 允许更精细的权限控制。

* **原理**
* 系统定义用户属性、资源属性以及环境条件等多个维度。
* 系统根据这些属性以及定义的访问策略来判断是否授予访问权限。
* **注意点**
* **属性校验**：确保在 API 请求时，用户和资源的属性是否正确被验证。例如，检查请求中的用户角色、请求时间等是否符合策略规则。
  + **合法属性测试**：确保 API 可以根据用户属性、资源属性等参数正确授权。
  + **非法属性测试**：验证当请求不符合属性要求时，系统应返回 403 Forbidden 或相关错误消息。
* **策略验证**：测试是否可以在接口请求中验证与用户属性相关的策略，并确保策略逻辑在实际应用中得到正确实现。
* **细粒度权限测试**：ABAC 可以提供非常细致的权限控制，测试时要检查属性组合的灵活性，确保系统能处理不同维度的权限判定。

##### 基于策略的访问控制（PBAC）

基于策略的访问控制（Policy-Based Access Control，PBAC）是一种更加灵活的授权控制方法，它允许通过策略来定义和管理访问权限，策略由多个规则组成。PBAC 是 ABAC 的一个扩展，强调通过策略引擎来管理访问权限。

* **原理**
* 用户、资源和环境条件被映射到不同的策略规则中。
* 系统根据策略的定义来判断用户是否有权限访问资源。
* 策略的定义通常使用标准语言（如 XACML）来表示。
* **注意点**
* **策略验证**：测试策略引擎是否能够正确执行访问策略。例如，针对用户角色、时间、设备等属性设置的访问策略是否能够正确控制接口访问。
  + **有效策略测试**：根据策略规则，验证请求是否符合授权条件。
  + **无效策略测试**：验证请求不符合策略规则时，系统是否正确返回 403 Forbidden。
* **策略更新测试**：测试策略更新后的影响，确保新策略能即时生效并对接口访问产生预期效果。
* **多维度策略测试**：确保不同维度的策略组合（如用户角色 + 访问时间 + 设备类型等）可以正确控制 API 的访问。

##### 其它方法

随着现代技术的发展，认证和授权越来越多地通过一种“**一体化**”的方式来处理，其中 **Token**（如 OAuth 2.0）和 **API Key** 成为常见的解决方案。这些方法不仅实现了认证，还能同时完成授权控制。

* **OAuth 2.0（Token）**

OAuth 2.0 通过同一个 Token 同时完成认证（用户身份）和授权（访问权限），令牌内包含身份信息和权限范围。

* **认证**：用户通过授权服务器验证身份，返回访问令牌（Token）。
* **授权**：Token 包含权限信息，决定用户可以访问哪些资源或执行哪些操作。
* **API Key**

API Key 既是身份验证凭证，也是授权凭证，服务器根据 API Key 验证身份并控制访问。

* **认证**：客户端使用 API Key 进行身份验证。
* **授权**：API Key 控制对特定资源的访问权限。

### Mock

**MOCK** 是软件测试中的一种技术，用于模拟系统中的某个组件或模块的行为。通过创建模拟对象，测试人员能够控制依赖项的行为，集中测试某个具体的单元或模块的功能，而无需依赖于外部系统或复杂的依赖关系。

与实际系统交互时，模拟对象根据预设的行为提供返回值，而不需要执行实际的操作。MOCK 的核心目的就是让测试更加可控、可靠，避免不必要的依赖。

**例如：**测试一个商城系统，可能实际购买需要调用第三方支付接口，但测试时肯定不可能每次都真实的去支付，此时就可以使用Mock来实现模拟支付。

#### MOCK 在接口测试中的应用

接口测试通常涉及多个外部依赖项（例如数据库、第三方服务、网络请求等）。直接测试这些外部依赖可能带来以下问题：

* **外部服务不可用**：外部 API 或服务可能不稳定，导致测试失败。
* **慢速响应**：与外部系统的通信可能导致测试变得缓慢。
* **无法控制的依赖**：外部依赖的状态难以控制，可能影响测试结果的准确性。

使用 **MOCK** 技术可以帮助解决这些问题，模拟外部依赖，以便集中验证接口本身的行为和逻辑。MOCK 在接口测试中的应用场景如：

* **模拟外部服务**

外部服务（如第三方支付接口、消息队列、邮件服务等）可能不适合在测试环境中直接调用。通过 **MOCK**，可以模拟这些服务的行为，让测试集中在接口自身的逻辑上。

* **例子**：在测试支付接口时，可以通过 **MOCK** 模拟支付网关的响应，而不实际发起支付请求。
* **模拟数据库**

有时测试需要依赖数据库，但在单元测试或集成测试时直接操作数据库可能非常缓慢或不便。通过 **MOCK**，可以模拟数据库的行为，控制数据的返回和交互。

* **例子**：测试一个需要查询用户信息的接口时，可以模拟数据库查询的返回结果，而不依赖于实际的数据库。
* **模拟网络请求**

接口可能会依赖其他外部 API 服务或网络资源。在测试过程中，我们通常无法依赖这些外部服务的可用性或响应时间，使用 **MOCK** 可以模拟这些网络请求的返回，确保测试环境的稳定性。

* **例子**：测试一个与天气 API 交互的接口时，可以通过 **MOCK** 模拟天气 API 的响应，确保测试过程中接口能够正确处理不同的返回数据。

#### MOCK 在接口测试中的关键作用

* **隔离测试**

MOCK 可以隔离被测试的接口及其依赖关系，从而保证测试的准确性。通过模拟外部依赖，可以确保测试不受外部环境变化的影响。

* **举例**：在测试一个依赖于外部 API 的接口时，使用 MOCK 模拟外部 API 的响应，避免了测试过程中因 API 服务不可用而导致的失败。
* **控制测试环境**

通过 MOCK，可以模拟各种异常情况和边界条件，从而验证接口在不同情况下的表现。这些条件可能很难在实际环境中复现，例如网络超时、服务不可用等。

* **举例**：模拟第三方服务的错误返回（如 500 错误、服务超时等），确保接口能够正确处理这些异常情况。
* **提高测试效率**

使用 MOCK 可以避免对外部系统的依赖，减少不必要的等待时间，从而提高测试的执行效率。例如，模拟数据库查询时，测试可以直接使用预设的返回数据，而不需要查询真实数据库。

* **举例**：测试与数据库交互的接口时，可以模拟数据库的返回结果，而不需要执行实际的查询操作，节省测试时间。
* **验证交互行为**

MOCK 不仅模拟外部依赖的返回，还能验证接口与这些依赖之间的交互。通过 MOCK 可以检查接口是否正确地调用了模拟对象，调用次数、参数等是否符合预期。

* **举例**：在测试一个发送邮件的接口时，可以通过 MOCK 验证接口是否正确调用了邮件服务的发送函数，并传递了正确的参数。

#### MOCK 的实现方法

在接口测试中，常见的 MOCK 实现方法包括：

* **手动 MOCK**

手动模拟外部接口的响应和行为，适用于简单的测试场景。这通常通过创建一个简单的模拟对象，并手动定义其返回的行为。

* **举例**：手动编写一个 Mock 对象，该对象会返回模拟的用户数据，替代真实的数据库查询。
* **使用 MOCK 框架**

在实际开发中，我们通常使用现成的 **MOCK 框架** 来帮助生成和验证 MOCK 对象。常用的 MOCK 框架包括：

* **Mockito**（Java）
* **unittest.mock**（Python）
* **Jest**（JavaScript）
* **nock**（JavaScript，用于模拟 HTTP 请求）

### 幂等性

**幂等性（Idempotency）** 是指在多次执行相同操作时，无论执行多少次，系统的状态和结果应该是相同的，不会产生副作用。在接口测试中，幂等性确保了相同的请求不会因为重复提交而引发错误或改变系统的状态，尤其在处理网络请求、并发操作、事务等方面，幂等性尤为重要。

幂等性意味着：

* 对同一个接口请求进行一次或多次相同的操作，系统返回的结果是相同的，且系统状态不发生额外变化。
* 即使客户端多次发送相同的请求，服务器的响应和数据状态保持一致，没有重复操作的副作用。

#### 常见的幂等性应用模式

* **使用幂等性Key**

客户端在每次请求时传递一个唯一的标识符（如idempotency key），用于标识请求是否已被处理过。

* **应用场景**：支付接口、订单创建等。
* **实现原理**：服务器根据传递的 idempotency key 判断请求是否已经处理过。如果是重复请求，返回之前的响应而不是重复执行操作。
* **操作幂等化**

对于操作本身的设计，确保即使请求重复也不会产生副作用。

* **应用场景**：更新或删除操作。
* **实现原理**：对于重复的更新请求，只会更新一次数据，对于删除请求，系统应处理“已删除”状态，而不报错。
* **事务处理与锁机制**

在复杂操作中使用事务和锁机制，确保多次操作不会导致状态不一致。

* **应用场景**：支付、库存扣减等。
* **实现原理**：使用分布式事务或锁，确保多个相同请求不会重复执行。

#### 接口测试中的幂等性验证

在接口测试中，验证幂等性主要是确保接口的行为符合预期，避免重复请求产生副作用。

**一般来说，进行测试时幂等标识需要查阅接口文档自行带上，而真是使用前端逻辑会自动处理。**

测试人员通常会验证以下几个方面：

* **重复请求验证**
* **目标**：验证接口是否能正确处理重复的请求，不会产生副作用。
* **方法**：向接口发送相同的请求多次，确保返回结果一致，且不会创建重复的数据或执行多次操作。
* **测试示例**：
  + **POST 请求**：对相同的 POST 请求进行重复发送，验证是否创建了多个相同资源。
  + **PUT 请求**：对相同的 PUT 请求重复发送，检查是否返回相同的响应，且数据保持一致。
  + **DELETE 请求**：重复发送相同的 DELETE 请求，检查是否不会导致重复删除。
* **幂等性 Key 测试**
* **目标**：确保通过幂等性 Key 控制请求重复性。
* **方法**：在请求中包含幂等性 Key，发送多次相同请求，确保服务器根据 Key 识别请求是否已处理过，并返回相同的响应。
* **测试示例**：在支付接口中使用相同的 idempotency key 发送多个支付请求，检查是否产生重复扣款。
* **状态一致性验证**
* **目标**：确保接口在处理重复请求时，系统的状态保持一致。
* **方法**：通过多次请求，检查系统资源是否保持一致，且没有不必要的变化。
* **测试示例**：
  + 在更新接口上多次提交相同的更新请求，检查是否产生多次更新。
  + 在删除操作上多次调用删除接口，确保返回正确的响应（如资源已删除）而不是错误信息。
* **网络异常与重试验证**
* **目标**：验证在网络错误或超时情况下，系统是否能够正确处理重试请求而不产生副作用。
* **方法**：模拟网络中断或超时后重新发送相同请求，确保系统能识别并正确处理重复请求。
* **测试示例**：模拟支付请求因网络问题失败，重试时检查支付是否成功且不重复扣款。

#### 幂等性测试的重要性

幂等性测试在接口测试中有以下几大关键意义：

1. **保证数据一致性**：防止因重复请求导致的数据不一致或重复创建。
2. **确保用户体验**：避免用户在重复提交请求时遭遇错误或重复操作，如支付重复扣款、订单重复创建等。
3. **系统稳定性**：避免因网络问题或请求超时等原因，导致系统状态异常或重复操作，从而确保系统的健壮性和稳定性。
4. **增强安全性**：幂等性测试能够确保接口在处理重复请求时不会导致权限问题、资源泄露或安全漏洞。

### 接口依赖

**接口依赖** 是指在一个系统的多个接口之间存在调用关系或数据依赖的情况。在接口测试中，某些接口的测试可能需要依赖其他接口的返回数据或执行结果。这种依赖关系是测试中的一个重要因素，因为它影响着测试的设计、执行顺序、数据准备、以及最终测试结果的有效性。

#### 接口依赖的类型

* **数据依赖**

数据依赖是最常见的接口依赖类型。在这种情况下，某个接口的测试**需要依赖于另一个接口提供的数据或响应结果**。例如，在电子商务平台中，支付接口可能需要依赖于订单接口返回的订单详情来执行支付操作。

* **例子：**支付接口（POST /pay）依赖于订单创建接口（POST /order）生成的订单号和订单信息，支付接口才能成功执行支付。
* **执行依赖**

执行依赖指的是**某个接口的调用需要其他接口先执行成功**。即，接口的测试或调用必须按照某个顺序进行。在这种情况下，一个接口的调用只有在另一个接口的调用成功之后才能继续进行。

* **例子：**创建用户接口（POST /users）成功后，才能测试用户信息更新接口（PUT /users/{id}）。
* **状态依赖**

状态依赖是指在某些接口测试中，依赖于某个接口的状态或结果。例如，一个系统的某个资源可能需要先在另一个接口中进行初始化，才能在后续的接口中正确操作。

* **例子：**在进行某项操作（如发布内容）之前，需要调用 POST /init 接口对资源进行初始化。

#### 接口依赖的挑战

* **依赖性带来的顺序问题**

如果接口之间有依赖关系，测试的顺序必须严格遵守。若前一个接口调用失败，后续的接口测试可能会受到影响。这要求测试人员在设计测试用例时要确保接口调用顺序和数据依赖关系得以遵循。

* **测试环境的复杂性**

多个接口之间的依赖可能导致测试环境的设置非常复杂。每次运行测试时，测试环境可能需要准备特定的上下文或数据，这使得测试的自动化和持续集成变得更加困难。

* **数据一致性**

当接口之间存在依赖时，数据一致性变得至关重要。一个接口的返回数据可能是另一个接口测试的输入。如果第一个接口的返回数据不准确或不符合预期，可能会导致后续接口测试失败，从而影响整体的测试结果。

* **测试难度增加**

当接口之间有复杂依赖关系时，手动测试可能会变得非常繁琐和易出错。测试人员需要跟踪和管理所有依赖关系，确保每个接口的返回数据都能正确传递给后续接口。这要求测试框架或测试工具能够高效地处理这些依赖。

#### 接口依赖的解决方案

* **接口依赖模拟（Mock）**

在接口测试中，Mock 技术广泛应用于解决接口依赖问题。通过模拟被依赖接口的返回值，测试人员可以独立地测试接口，而不必等待其他接口的实际执行。

* 例子：如果某个支付接口依赖于订单接口的返回数据，可以通过 Mock 技术模拟订单接口的返回数据，从而在不依赖真实订单接口的情况下测试支付接口。
* **依赖接口的隔离**

通过将依赖接口进行隔离，可以独立测试各个接口，减少不同接口之间的紧密耦合。可以考虑使用一些自动化框架，通过设置合适的前置条件，先调用依赖接口来设置数据，再调用目标接口进行测试。

* 例子：在用户登录接口测试前，可以先调用用户注册接口（POST /register）进行注册，确保有可用的用户数据进行登录测试。
* **依赖关系管理**

在复杂的接口依赖环境中，可以通过依赖关系管理工具来帮助测试人员管理接口的调用顺序和数据传递。许多现代的自动化测试框架和工具（如 Postman、Jenkins、TestNG 等）提供了依赖关系的管理功能。

* 例子：在自动化测试中，通过设置依赖链条，确保先执行用户注册接口，再执行登录接口，最后执行账户余额查询接口。
* **数据准备与清理**

接口之间的依赖关系往往需要测试人员在每次测试前准备数据，并在测试结束后清理数据。自动化脚本可以在每次测试前动态准备好所需的测试数据，并在测试后清理掉这些数据，保证测试环境的干净。

* 例子：自动化脚本在执行支付接口测试前，自动创建订单数据，并在测试完成后清理相关订单数据。

#### 常见的依赖关系管理方法

* **API 链接测试**

这种方法是通过**串联多个 API** 请求来构建依赖链。

* **例：**一个支付接口可能会依赖于订单接口、库存接口等。通过手动或自动化的方式先调用订单接口，获取订单信息，再传递给支付接口。
* **实现方式：**在测试脚本中设置接口的调用顺序，通过 API 返回的数据传递给下一个接口进行测试。
* **数据传递与共享**

在接口测试中，可以通过数据传递机制，在前后端接口之间共享测试数据。通过存储测试数据的上下文信息，使得后续的接口可以使用这些数据。

* **实现方式：**在脚本中使用变量或存储库来保存一个接口返回的数据（如订单号、支付状态等），然后将其作为输入传递给后续的接口。
* **接口的合成与依赖模拟**

在一些较复杂的测试场景中，可以将多个依赖接口合成一个接口进行模拟，减少测试过程中对真实服务的依赖，从而提高测试效率。

* **实现方式：**通过模拟（Mock）或桩（Stub）来模拟依赖接口的行为，避免在测试过程中依赖真实服务。

#### 接口依赖的测试要点

1. **验证依赖关系：**测试时需要验证接口依赖关系是否正确，如依赖接口返回的状态码、数据格式等。
2. **边界条件测试：**确保在接口依赖数据的边界情况下，接口能够正常处理并返回预期结果。
3. **失败场景测试：**模拟依赖接口失败的场景（如接口未响应、数据格式错误等），验证主接口是否能够正确处理错误。
4. **并发测试：**测试接口在并发访问下，依赖关系是否会导致数据错乱或资源竞争等问题。

## Charles抓包

### 抓包的应用

### Charles基本介绍

Charles（又称“青花瓷”）是一款功能强大的跨平台网络分析工具，主要用于HTTP/HTTPS协议的数据抓取与调试，更专注于应用层协议（如 HTTP/HTTPS）的抓包与调试

#### 核心功能

* **协议支持​​**
* 支持HTTP/HTTPS协议的全量抓取，包括请求头、响应体、Cookie等细节。
* 通过SSL代理实现HTTPS流量的解密，采用“中间人”（MITM）技术，需安装信任证书以解析加密内容。
* ​​**调试与模拟​​**
* ​​**断点调试​​**：允许在请求发送前或响应返回前拦截并动态修改数据，便于测试异常场景。
* ​​**带宽控制**​​：模拟弱网环境（如2G/3G），测试应用在低带宽、高延迟下的表现。
* ​​**请求重放**​​：重复发送历史请求，用于接口调试或性能压测。
* ​​**高级映射​​**
* ​​**远程映射**​​：将请求重定向到指定服务器或本地文件，便于开发环境与生产环境隔离。
* ​​**本地修改**​​：直接替换服务器返回内容，无需修改代码即可测试不同响应数据

#### Charles抓包原理

整体来说，Charles通过中间人（Man-in-the-Middle, MITM）技术插入客户端与服务器之间，将原本直接通信的双方握手过程拆解为​**​双重代理交互​**​，从而实现抓包。

* **原始通信：**

**​​客户端↔ 服务器**

* **启用Charles后：**

**​​ 客户端 ↔ Charles ↔ 服务器**

下面是详细介绍

* **代理架构​​**
* Charles 作为中间代理服务器，拦截并转发客户端与服务器之间的所有流量。客户端需配置代理指向 Charles（默认端口 8888），实现请求/响应的双向拦截与记录。
* ​​**HTTPS 解密流程​​**
* ​​**证书伪造​​：**Charles 生成**自签名根证书**并诱导客户端信任，拦截 HTTPS 请求时动态伪造与目标域名匹配的服务器证书返回给客户端。
* ​​**双重握手​​：**客户端与 Charles 完成 SSL 握手（使用伪造证书），Charles 再与真实服务器完成标准 SSL 握手，形成两个独立加密通道。
* ​​**密钥截获​​：**客户端生成的对称加密密钥（如 Pre-master）被 Charles 用伪造证书私钥解密，后续通过该密钥解密所有 HTTPS 通信内容。
* ​​**数据透明化​​**
* Charles 利用截获的对称密钥，对客户端和服务器的双向加密流量进行实时解密与加密转发，实现明文抓包，同时保持两端正常通信

### charles抓包详解

### 抓包分析BUG

### 常见定位BUG方案

### charles抓包实战

## POSTMAN

### postman基本操作

### postman环境配置

### postman参数关联

### postman的cookies管理

### postman自动断言

### postman脚本编写

### postman批量执行

### 应用postman对上市公司的接口进行测试