# 前端测试

# 一、接口测试

## 1、接口定义及种类

接口就是内部模块对模块，外部系统对其他服务提供的一种可调用或者连接的能力的标准。仅仅是一个接口是不能进行传输的，我们还的对这个接口怎么进行传输进行一些设置和定义。

接口可以分为系统对外的接口和程序内部的接口。**系统对外的接口**：从别的网站或服务器上获取资源或信息时，通过一个写好的方法来获取数据，引用提供的接口就能使用他写好的方法，从而达到数据共享的目的。**程序内部的接口**：方法与方法之间，模块与模块之间的交互，程序内部抛出的接口。两个模块之间有交互时，就会抛出一个接口，供内部系统进行调用。

常见的接口分类http：get，post，delete，put。 get的提交方式是明文提交，把提交的参数跟在url后面发送给服务器，所以不安全，而且get提交的参数是有字符限制的且可以被当做书签保存，但是post的提交方式跟get完全不一样，post提交的参数是放在表单里的，所以不会存在字符限制，而且因为参数是放在表单里，不容易被看到，所以会比get更安全。

## 2、接口测试含义

不同端（前端，后端）的工作进度不一样，所以我们要针对最开始出来的接口，以及需要调用其他公司的一些接口进行接口测试及验证数据，从安全层面来说，只依赖前端进行限制已经完全不能满足系统的安全要求，需要后端同样进行控制，在这种情况下就需要从接口层面进行验证。接口测试对于测试来说其实是对接口协议的一种测试，这个协议指的是为了让这个接口实现某种需要的功能还设计的一种要求。

接口测试流程为：需求讨论，需求评审，场景设计，编写用例，准备数据，执行测试。

### 3、接口测试工具

　项目前后端调用主要是基于http协议的接口，所以测试接口时主要是通过工具或代码模拟http请求的发送与接收。工具有很多如：fiddler、postman、jmeter、soupUI、java+httpclient、robotframework+httplibrary等。

**Fiddler**：Fiddler是位于客户端和服务器端之间的代理，也是目前最常用的抓包工具之一 。它能够记录客户端和服务器之间的所有请求，可以针对特定的请求，分析请求数据、设置断点、调试web应用、修改请求的数据，甚至可以修改服务器返回的数据，功能非常强大，是web调试的利器。

**Postman**：Postman一款非常流行的API调试工具。 用来模拟HTTP请求，帮助后端人员进行单元测试的工具。它可以自定义请求URL、请求的类型(GET,POST等)，可以加入Head头信息以及HTTP body信息等，让我们简单直观的进行HTTP请求测试。

**jmeter**：JMeter是Apache组织开发的基于Java的压力测试工具。

**soupUI** ：soapui提供一个工具通过soap/http来检查，调用，实现web service和web service的功能/负载/符合性测试。

# 二、UI测试

用户界面测试（User interface testing），简称UI测试，来测试用户界面的功能模块的布局是否合理，整体风格是否一致和各个控件的放置位置是否符合客户使用习惯，更重要的是要符合操作便捷，导航简单易懂，界面中文字是否正确，命名是否统一，页面是否美观，文字、图片组合是否完美等等。

UI 测试（UI Test）只是对于前端的测试，是脱离真实后端环境的，仅仅只是将前端放在真实环境中运行，而后端和数据都应该使用 Mock 的。UI 测试的自动化程度还不高，大多数还依赖于手工测试。

# 三、端到端（E2E）测试

## 1、定义

端到端(e2e)测试/功能测试：是站在用户角度的测试，把我们的程序看成是一个黑盒子，用来确保整个应用会按照用户期望的那样运行。不关心代码内部的实现，只负责打开浏览器，把测试用例中设置的内容在页面上输入一遍，看是否能够得到想要的结果。

## 2、E2E测试框架

[**Selenium**](https://github.com/SeleniumHQ/selenium)**：**E2E测试鼻祖级的框架，有多种编程语言的版本。它是基于 webdriver 而不是 webkit 内核实现的，所以，Selenium 的浏览器兼容性相对于其他浏览器要好很多。

[**PhantomJS**](https://github.com/ariya/phantomjs)**：**开创性的 headless（无头）测试框架，何为 headless？即没有 UI 界面的浏览器。headless 最大好处在于可以像单元测试一样，在命令行中跑 e2e 测试。

[**nightmare**](https://github.com/segmentio/nightmare)**：**加强版的 PhantomJS。相对于 PhantomJS，无论是开发还是运行效率都得到了很大的提升。

[**nightwatch**](https://github.com/nightwatchjs/nightwatch)**：**使用 Node 调用 webdriver 实现。相对于 Selenium，开发和运行效率更高，迭代活跃。

[**cypress**](https://github.com/cypress-io/cypress)**：** 基于node.js环境的，安装简单、运行速度快，可以进行脚本实时调试。简化了设置测试、编写测试、运行测试和调试测试，支持端到端测试、集成测试和单元测试，支持测试在浏览器中运行的任意内容。

[**puppeteer**](https://github.com/puppeteer/puppeteer)**：**e2e 测试框架的集大成者，默认以 headless 模式运行，但是也可以通过配置变为 Chromium 运行。开发效率以及运行效率一流，最重要的是，它像 nightmare 一样，提供了测试代码生成插件——[puppeteer-recorder](https://github.com/checkly/puppeteer-recorder)。

综上所述，如果考虑浏览器兼容性，使用 [nightwatch](https://github.com/nightwatchjs/nightwatch)，反之，选择 [cypress](https://github.com/cypress-io/cypress) 或者 [puppeteer](https://github.com/puppeteer/puppeteer)，如果需要 headless 或者自动化生成代码的功能，那就使用 [puppeteer](https://github.com/puppeteer/puppeteer)。

# 三、单元测试

## 1、定义

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

## 2、前端单元测试框架



**图1 主流前端单元测试框架比较**

**Mocha：**Mocha 是生态最好，使用最广泛的单测框架，但是他需要较多的配置来实现它的高扩展性。

**Ava：**Ava 是更轻量高效简单的单测框架，但是自身不够稳定，并发运行文件多的时候CPU负载会过高。

**Jasmine：**Jasmine 是单测框架的“元老”，开箱即用，但是异步测试支持较弱。

**Jest：** Jest 基于 Jasmine, 做了大量修改并添加了很多特性，同样开箱即用，异步测试支持良好。配置少，API简单。

**Karma：**Karma 能在真实的浏览器中测试，强大适配器，可配置其他单测框架，一般会配合 Mocha 或 Jasmine 等一起使用。

**Vue Test Utils:** Vue Test Utils 是 Vue.js 官方的单元测试实用工具库。 (https://vue-test-utils.vuejs.org/zh/)

# 四、集成测试

## 1、定义

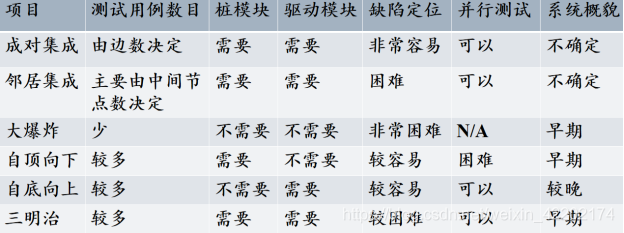
集成测试，也叫组装测试或联合测试。在单元测试的基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，进行集成测试。

集成测试通常被应用在：耦合度较高的函数/组件、经过二次封装的函数/组件、多个函数/组件组合而成的函数/组件等。

集成测试的目的在于，测试经过单元测试后的各个模块组合在一起是否能正常工作。会对组合之后的代码整体暴露在外接口进行测试，查看组合后的代码工作是否符合预期。达到以下目的：1）**保证UI：**确保浏览器渲染效果正常，实现快照对比；2）**克服兼容：**在集成测试端，快速进行多种；浏览器的测试，发现兼容问题；3）**易于回归：**易于流程控制，用户交互模拟，方便测试用例编写；

无明确主流框架推荐。

## 2、集成测试方法



**图2 集成测试方法比较**

**（1）成对集成：**

基本思想：将每个集成测试用例限定在一对调用单元上，每个集成测试用例都是最小的集成单元，仅涉及一对调用的接口

测试用例设计：两个典型的模块成对集成；

**（2）邻居集成：**

概念：邻居是指某个指定模块及其所有直接调用该模块的上层模块以及所有被该模块直接调用的下层模块；

基本思想：将每个集成测试用例限定在某个节点的邻居上，针对某个模块的集成测试用例应同时包含该模块及其邻居；

**（3）基于独立路径的测试：**

基本思想：将函数调用图看做程序图，每个从根节点到叶子节点的调用形成了路径，每条独立路径即可构成一个集成测试用例；

**（4）大爆炸集成：**

1.基本思想：将所有经过单元测试的模块一次性组装到被测系统中进行测试，完全不考虑模块之间的依赖性和可能的风险；

2.举例：将所有7个模块放在一起进行测试，即仅需一个测试用例，达到用例规模的最小化；

**（5）自顶向下集成：**

1.基本思想： 从主控模块(主程序，即根节点)开始，按照系统程序结构，沿着控制层次从上而下，逐渐将各模块组装起来

**（6）自底向上的集成：**

1. 基本思想：从底层模块(即叶子节点)开始，按照调用图的结构，从下而上，逐层将各模块组装起来

2. 该集成测试方法无须开发桩模块，但需对未经集成测试的模块开发驱动模块；

**（7）三明治集成：**

1.基本思想：将自顶向下和自底向上集成方法结合起来的集成策略。在调用图上按照一定的策略，分别自顶向下和自底向上展开集成，并在子树上进行大爆炸集成；

2.优势：结合了自顶向下和自底向上的集成的优势；

# 五、代码规范

## 1、Eslint

ESLint 是一个开源的 JavaScript 代码检查工具。代码检查是一种静态的分析，常用于寻找有问题的模式或者代码，并且不依赖于具体的编码风格。对大多数编程语言来说都会有代码检查，一般来说编译程序会内置检查工具。

JavaScript 是一个动态的弱类型语言，在开发中比较容易出错。因为没有编译程序，为了寻找 JavaScript 代码错误通常需要在执行过程中不断调试。像 ESLint 这样的可以让程序员在编码的过程中发现问题而不是在执行的过程中。

ESLint 的初衷是为了让程序员可以创建自己的检测规则。ESLint 的所有规则都被设计成可插入的。ESLint 的默认规则与其他的插件并没有什么区别，规则本身和测试可以依赖于同样的模式。为了便于人们使用，ESLint 内置了一些规则，当然，你可以在使用过程中自定义规则。ESLint 使用 Node.js 编写，这样既可以有一个快速的运行环境的同时也便于安装。

## 2、JavaScript Standard

定义了JavaScript编写时需要遵守的规范，如：文件及变量命名规范、代码书写规范等。

## 3、Prettier

Prettier 是一个代码格式化工具。

## 4、JSHint

JSHint 是一个 JavaScript 的代码质量检查工具，主要用来检查代码质量以及找出一些潜在的代码缺陷。