测试桩与mock

- 讲师：pansir

简介

基本概念

mock测试指的就是测试过程中，对于某些不容易获取的对象，创建一个虚拟对象来方便测试。讲人话：做个假对象或者响应来实现某种特定场景的测试。

mock这个概念最开始是针对单元测试而言

扩展概念(了解)

double：置换，所有模拟测试对象的统称，也称替身

stub：测试桩，实现特定的方法被调用，返回一个模拟值。最开始指的是单元测试中，使用stub取代数据源，固定返回一个合理的模拟数据

spy：负责汇报情况，持续追踪什么方法被调用了，以及调用过程中传递了哪些参数

mock：先设定期望，然后执行测试，最后验证结果与预期是否一致

fake：具备完整功能实现和行为的对象，比如：不访问生产环境的数据库，而是使用内存中的数据库来生成一个数据持久化对象

实际使用中，我们不需要纠结这些定义，因为经常是混着用，甚至可以互换

服务虚拟化：将测试套件从它的环境依赖中解耦出来的技术，广义上来说，包含：模拟(mock) 和 打桩(stub)

产生原因

对于开发而言，无论前端还是后端：

1. 底层数据复杂，依赖严重，按照真实场景去测，耗时长，成本高
2. 依赖的底层单元还没开发完成，或者现有资源难以满足当前模块测试需求。如果直接把假数据写死到代码中（开发入侵），数据写死不便于修改
3. 一些特定的异常场景，较难复现。比如：需要批量构造大量不同规则的数据，手写肯定是不可能。
4. 前后端分离的场景，由于前后端不能准确描述接口长什么样，导致前端开发困难，bug率高。通常需要后端提供一个mock桩，也可由前端自行创建

对于测试而言：

1. 某些第三方依赖模块缺失，导致测试场景不完整，覆盖率低，甚至漏测
2. 最终数据结果并不落地到存储组件中，无法确保最终结果，而仅通过日志判断，有一定风险
3. 测试某些复杂页面展示，而数据来源是db，构造数据低效且易产生脏数据

应用场景

什么时候需要mock呢？并不是所有测试都需要使用mock，简单的场景，没必要使用，可能耗时更多。异常场景，复杂场景，影响工作效率的场景建议使用

- 单元测试/接口测试中测试对象依赖其他对象，这些对象构造复杂，耗时，根本无法构造，甚至未交付
- 测试页面或图表复杂的展示功能，数据来自多个表，需要频繁测试，频繁修改
- 只测试目标服务的功能，不关心依赖对象的逻辑正确性和稳定性

mock方法

mock有很多方法：

1. 通过第三方网站，模拟http响应数据
2. 通过工具模拟mock数据
3. 使用编程语言，编写测试桩

第三方网站

登录网站：https://www.mocky.io/，输入json数据，点击：generate my http resoponse，获取url：http://www.mocky.io/v2/5e0c98af2f000074002831be 使用该url即可获得指定数据

- 优点：构造简单，使用简单，应付一下简单测试场景很方便
- 缺点：数据死板，不灵活；调用方需要修改对应的接口，不便维护；涉及到业务逻辑判断，则无法满足

工具

市面上存在的mock工具有很多，列出几个供参考：

- YApi（`http://yapi.demo.qunar.com/`）：去哪儿网研发，基于mock.js，可自定义规则
- RAP（`http://rap2.taobao.org`）：阿里妈妈研发，基于mock.js，界面交互好，生成规则可视化编辑，推荐
- Nei（`https://nei.netease.com/`）：网易研发，基于mock.js，可自定义规则

优点：工具化，模块化，易学易用，方便管理

缺点：只有http接口；涉及业务逻辑判断仍无法满足

编程

使用编程语言完成测试桩开发，包含但不限于python

测试桩大致分为4类：

1. mock测试桩：提供接口，模拟mock数据返回，满足业务场景测试
    - http测试桩：提供http接口给被测服务使用，模拟第三方数据返回
    - socket测试桩：提供socket接口给被测服务使用，模拟第三方数据返回
    - ice测试桩：提供ice接口给被测服务使用，模拟第三方数据返回
2. 数据接收测试桩：

    - http回调接收桩：提供http接口，接收被测服务的请求，通常不需要返回，但回调结果需要转存，可用作日常测试与自动化测试
    - 消费转存测试桩：消费中间件（RabbtiMQ/Kafka）数据，转存到存储组件，可用作日常测试与自动化测试
3. 定时任务测试桩：定时触发某些操作，例如：每天凌晨/每月底凌晨调接口，验证数据记录分表功能
4. 特殊功能测试桩：提供特殊的服务功能，例如：将接口数据转发到不同的中间件；数据的解析与反序列化