



# DevOpsDays

## Shanghai

— 2017.8.18-8.19 —

上海龙之梦酒店（长宁区延安西路1116号）

主办单位： 高效运维社区  
GreatOPS Community  Best Practice  
最佳实践





# 轻量化微服务测试实践

张桐 华为软件架构师

# 关于我

- 工作经历



- 个人译著

- 《Pact中文参考指南》译者
- 《AngularJS高级程序设计》译者
- 《精通LabVIEW程序设计》作者

- 目前从事

- 软件工程相关技术与实践
- CloudNative/微服务/DevOps相关实践的落地

# 目录

- ➔ **1** 微服务测试面临的挑战
- 2** 微服务测试策略
- 3** 轻量化微服务测试实践
- 4** 总结

# 微服务测试面临的挑战



- 微服务架构带来的新问题

- 微服务数量的大规模增长，导致微服务测试的总体成本越来越高
- 微服务之间依赖关系复杂，难以绘制出依赖关系图，导致不容易理解系统工作方式，新人上手成本高

# 微服务测试实践面临的挑战

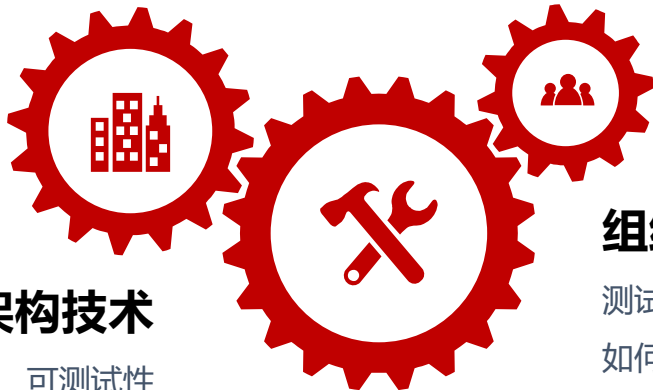


自动化测试  
怎样落地？



- 许多团队往往首先实现了架构调整（比如代码拆分），却并未形成工程实践与组织结构方面的有效适配

# 微服务测试实践面临的挑战



## 架构技术

可测试性  
架构解耦

## 组织结构

测试相关人员角色  
如何分工与协作

## 工程实践

持续集成流水线  
测试策略  
测试工具

微服务测试实践落地需要架构技术、工程实践与组织结构三方面的协同配合

# 如何应对这些挑战？



## 自动化

- 减少重复工作
- 提供快速反馈



## 轻量化

- 保持简单
- 选用成本低、易使用的工具



## 可视化

- 帮助从使用者角度去理解系统
- 降低沟通学习成本



## 去测试化

- 减少只负责手工测试的专职测试人员
- 让开发更关注自己的代码



# 目录

**1** 微服务测试面临的挑战

➔ **2** 微服务测试策略

**3** 轻量化微服务测试实践

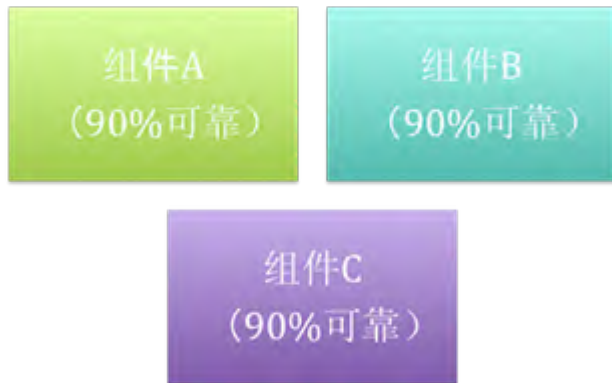
**4** 总结

# 微服务架构的系统化思考



- 基于微服务架构的软件构建过程，类似搭积木
- 模块开发 -> 集成 -> 服务 -> 集成 -> 系统
- 测试：先底层后上层，从局部到整体

# 如何保证系统的可靠性？

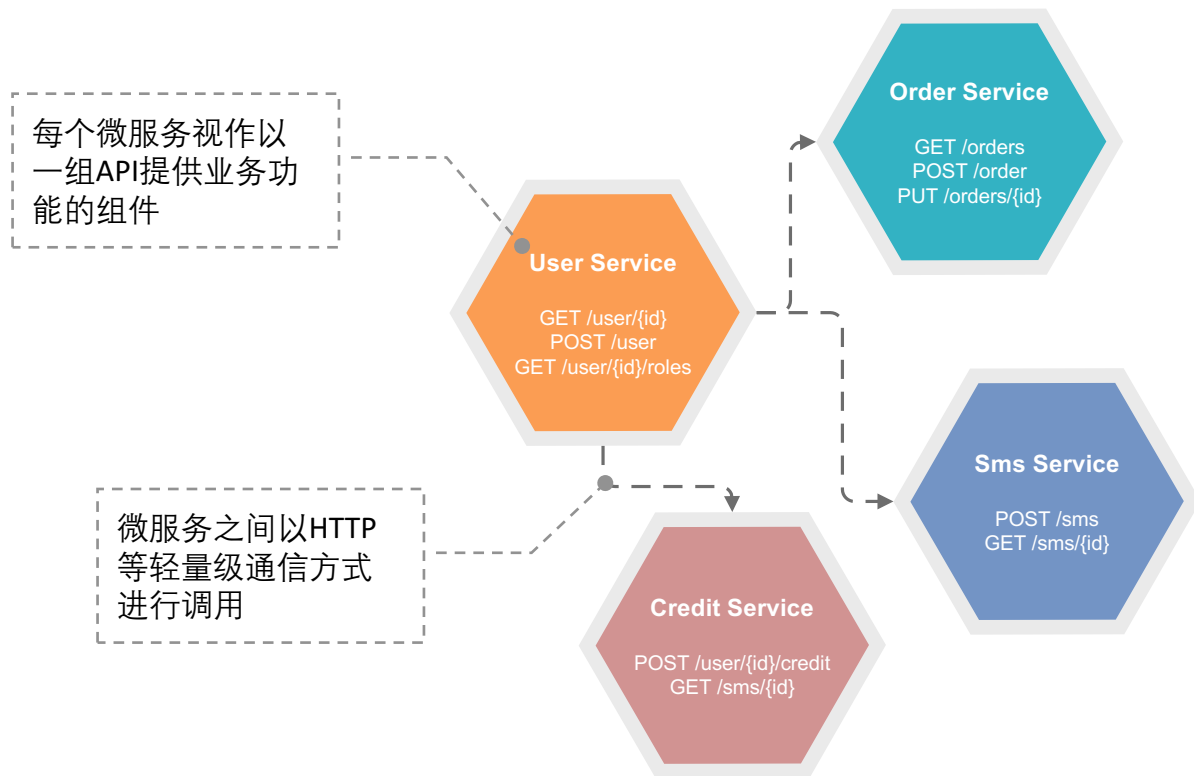


系统总体可靠性

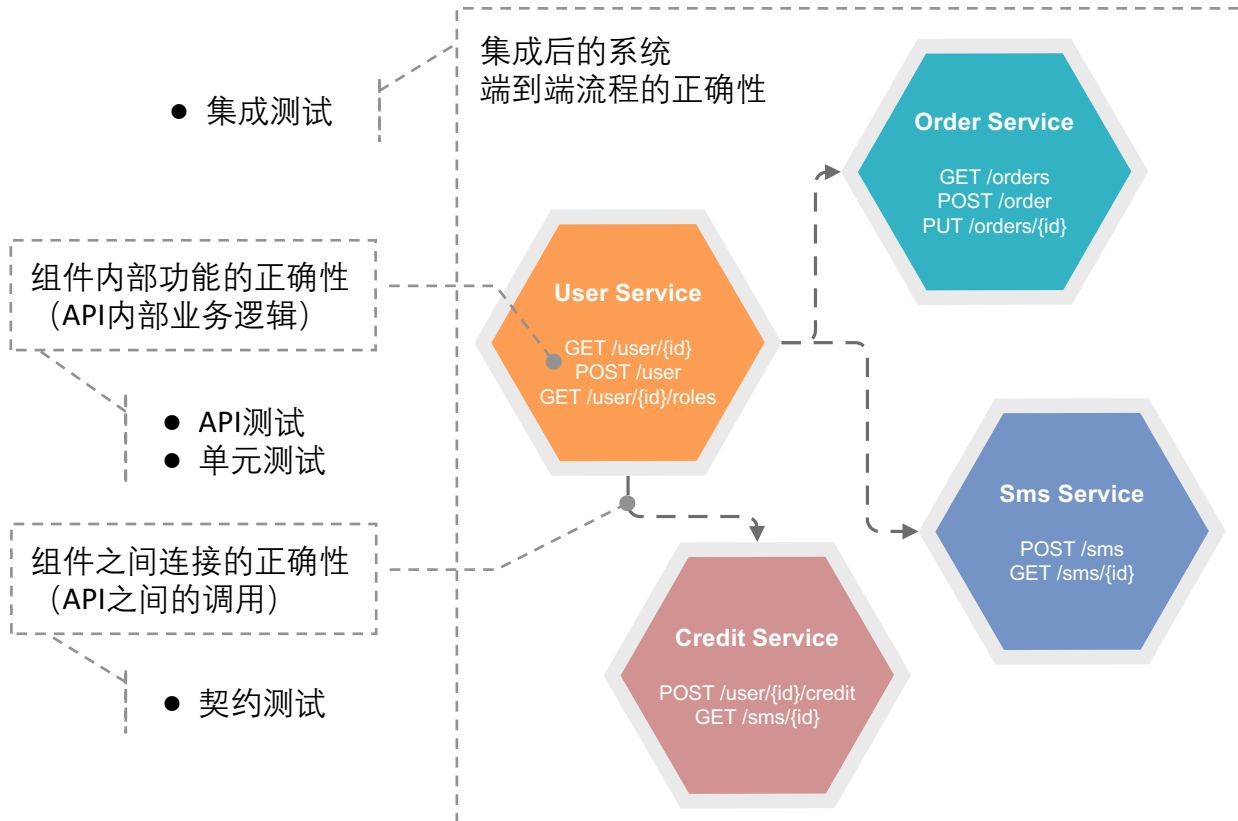
$$90\% \times 90\% \times 90\% = 72.9\%$$

- 从系统工程角度思考
- **线性系统**的可靠性等于各个组件的可靠性之乘积
- 复杂软件系统的可靠性可能更低，因为还有连接关系，是**非线性系统**
- 如何制定合理的测试策略？是否存在合适的模式？

# 仍然从微服务架构的特点出发.....



# 应该测试什么？



# 单元测试

- 范围：
  - 针对代码单元（通常是类）的测试
- 价值：
  - 最快速的反馈
  - 指导设计（掌握TDD的前提下）
  - 帮助重构、提升代码质量

# API测试

- 范围：
  - 针对业务单元/API的测试
  - 接口功能实现的完整度
  - 内部逻辑、异常处理等
- 价值：
  - 接口相对稳定，容易编写用例
  - 投入性价比高

# 契约测试

- 范围：
  - 针对服务之间连接/调用的正确性
  - 服务提供者的功能是否能满足消费者需求
  
- 价值：
  - 以与集成测试相比更轻量化的方式快速验证服务间调用是否正确

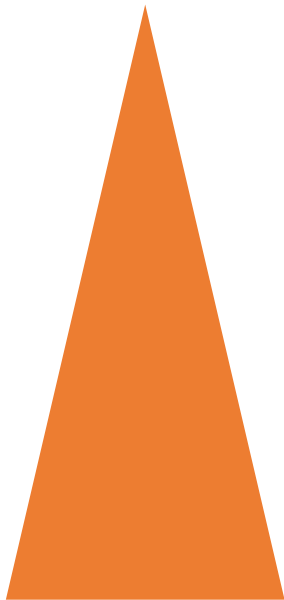


# 集成测试

- 范围：
  - 从用户角度验证功能的正确性
  - 端到端流程，加入用户场景和数据
  
- 价值：
  - 验证完整流程，业务价值含量最高

# 该如何组织这些测试？

## ——微服务测试策略中的模式与反模式



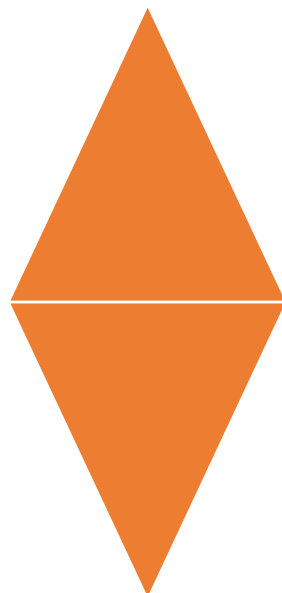
模式  
测试金字塔



反模式  
倒金字塔



反模式  
沙漏型

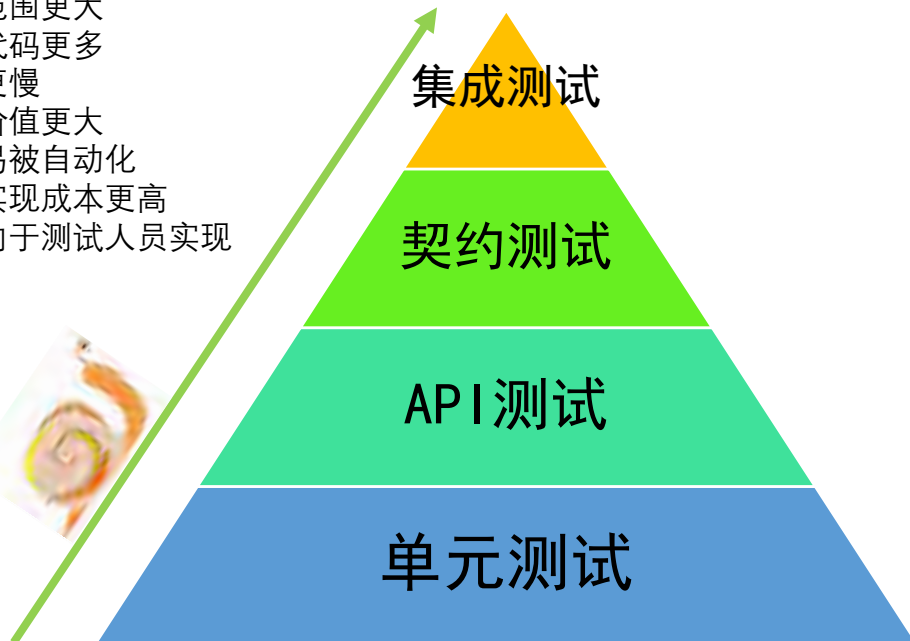


模式  
纺锤型

# 模式：测试金字塔

- 测试范围更大
- 覆盖代码更多
- 运行更慢
- 业务价值更大
- 更不易被自动化
- 单位实现成本更高
- 更倾向于测试人员实现

- 测试范围更小
- 覆盖代码更少
- 运行更快
- 业务价值更小
- 更容易被自动化
- 单位实现成本更低
- 更倾向于开发人员实现



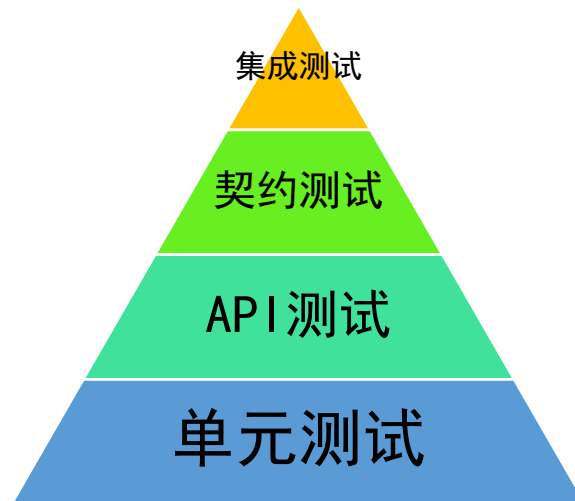
# 模式：测试金字塔

- 优点

- 容易快速定位问题
- 容易实现很高的自动化率
- 得到较高的自动化收益率

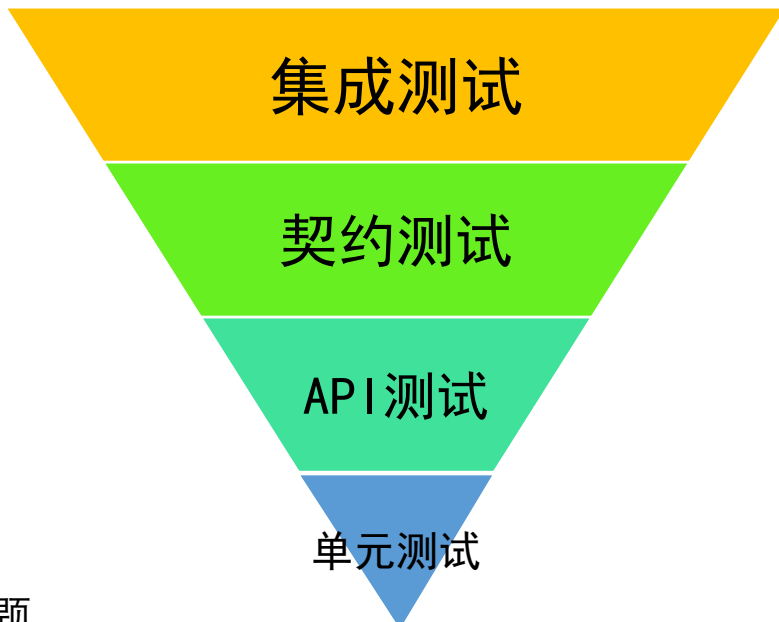
- 限制

- 对团队能力有较高要求
- 对于起步阶段的团队不容易落地



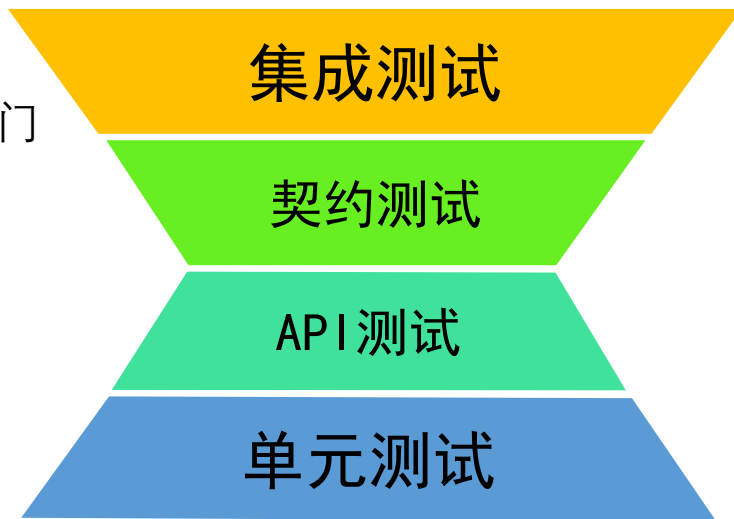
# 反模式：倒金字塔型（V型）

- 出现原因
  - 交付压力过大
  - 对自动化投入不足
- 缺点
  - 运行缓慢
  - 自动化率较低
  - 不容易快速定位问题



# 反模式：沙漏型（X型）

- 出现原因
  - 开发测试归属不同部门
  - 沟通协作存在隔阂
- 缺点
  - 自动化收益率较低
  - 存在重复用例



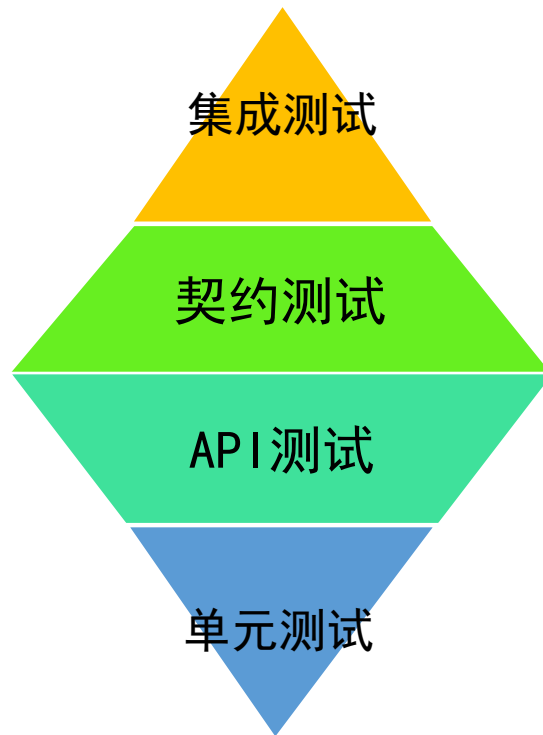
# 更适合于团队起步的模式：纺锤型

- 优点

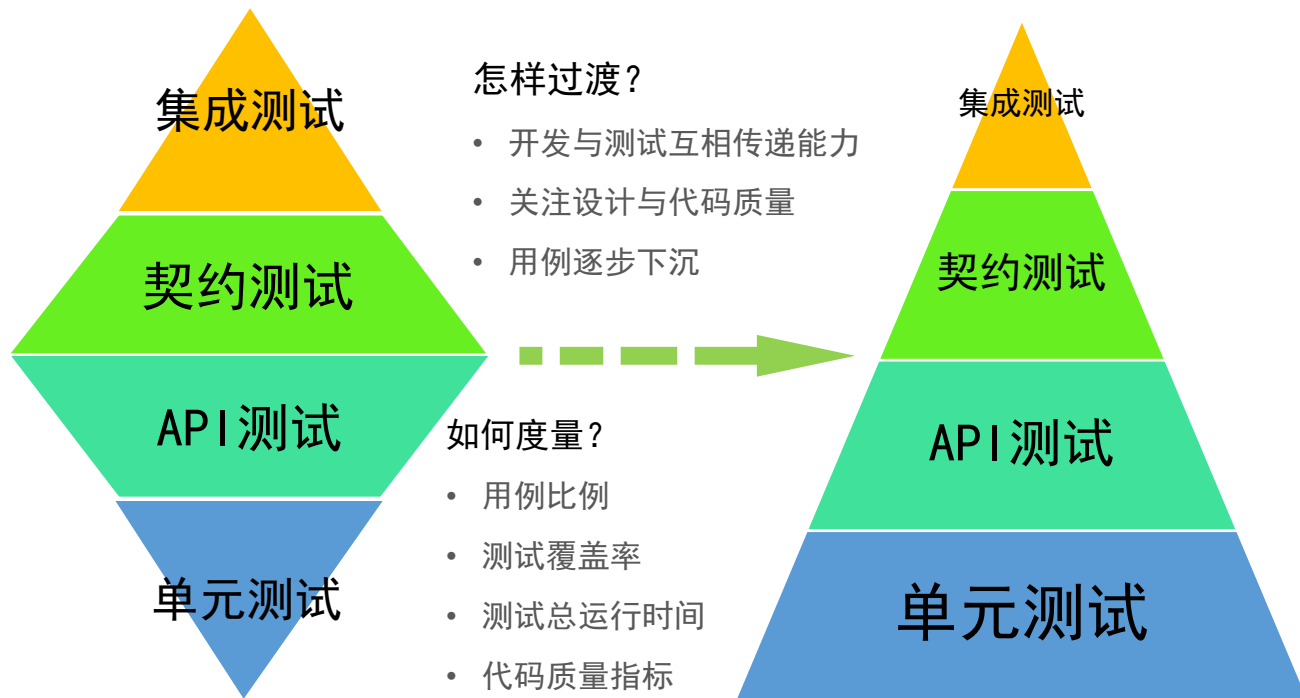
- 投入成本相对较小
- 相对较高的自动化率
- 相对较高的自动化收益率
- 增强开发测试之间的协作

- 适用于

- 起步阶段的团队



# 从纺锤型向金字塔型过渡





# 目录

**1** 微服务测试面临的挑战

**2** 微服务测试策略

**➔ 3** 轻量化微服务测试实践

**4** 总结

# 工程实践——单元测试

- 单元测试的目标并非100%代码覆盖率
- 对细粒度业务单元测试，而非绝对代码单元
- Do's :
  - 从学习如何设计代码开始，从精选一部分核心代码开始
  - Mock外部服务/数据库，或使用内存数据库
- Don'ts :
  - 不要使用测试生成工具完全代替手写
  - 不要调用真实的外部服务或数据库

# 工程实践——单元测试

- 工具：
  - 单元测试框架：JUnit, TestNG
  - Mock工具：Mockito, PowerMock, Easymock
  - 内存数据库：HyperSQL DB, H2
  - Testing with BDD：Spock Framework

# 工程实践——API测试

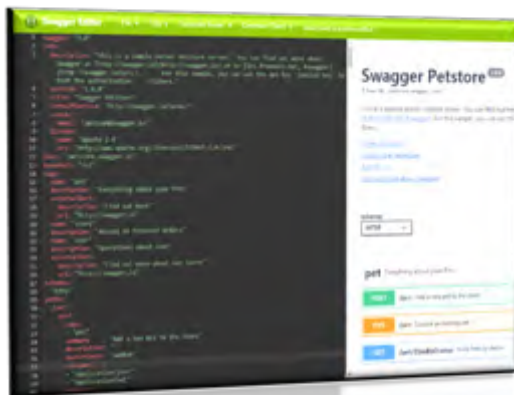
- API测试完成的最佳方式：开发与测试协作完成
- Do's :
  - 从对接口的定义和管理开始
  - Mock外部服务
  - 可以使用真实数据库
- Don'ts :
  - 不要调用真实的外部服务

# 工程实践——API测试

- Swagger Editor
- 接口定义与设计工具



- Swagger UI
- 可视化、可交互的接口信息展示工具



# 工程实践——API测试

- 开源微服务开发框架：ServiceComb



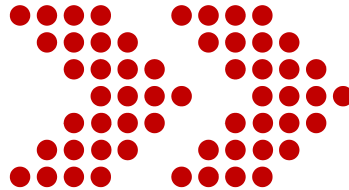
```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableEurekaClient
@EnableZuulProxy
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

REST API Code  
by SpringMVC  
or JAX-RS



swagger-generator-jaxrs

swagger-generator-springmvc

基于代码自动生成  
Swagger接口文件

```
swagger:
  info:
    title: "Swagger"
    version: "1.0"
  paths:
    /pets:
      get:
        description: "Get all pets from the store"
        responses:
          200:
            description: "successful operation"
            schema:
              type: array
              items:
                $ref: "#/definitions/pet"
      post:
        description: "Add a new pet to the store"
        responses:
          201:
            description: "successful operation"
            schema:
              $ref: "#/definitions/pet"
        requestBody:
          description: "Pet object that needs to be added to the store"
          required: true
          schema:
            $ref: "#/definitions/pet"
    /pets/{petId}:
      get:
        description: "Get pet by ID"
        responses:
          200:
            description: "successful operation"
            schema:
              $ref: "#/definitions/pet"
      delete:
        description: "Delete pet by ID"
        responses:
          204:
            description: "successful operation"
            schema:
              type: null
    /pets/{petId}/status:
      patch:
        description: "Update pet status"
        responses:
          200:
            description: "successful operation"
            schema:
              $ref: "#/definitions/pet"
        requestBody:
          description: "Update status"
          required: true
          schema:
            type: string
```

Swagger File

# 工程实践——API测试

- 轻量易用的桩服务生成工具：moco
  - 支持http/https/socket通信
  - 与JUnit/Gradle/Maven集成



- 丰富的Java/REST APIs Design


1

依据接口信息编辑json文件

```
[{
  "response": {
    "text": "Hello Shanghai"
  }
}]
```

2

```
java -jar moco-runner-<version>-standalone.jar http -p 12306 -c foo.json
```

Got it! http://localhost:12306  Hello Shanghai

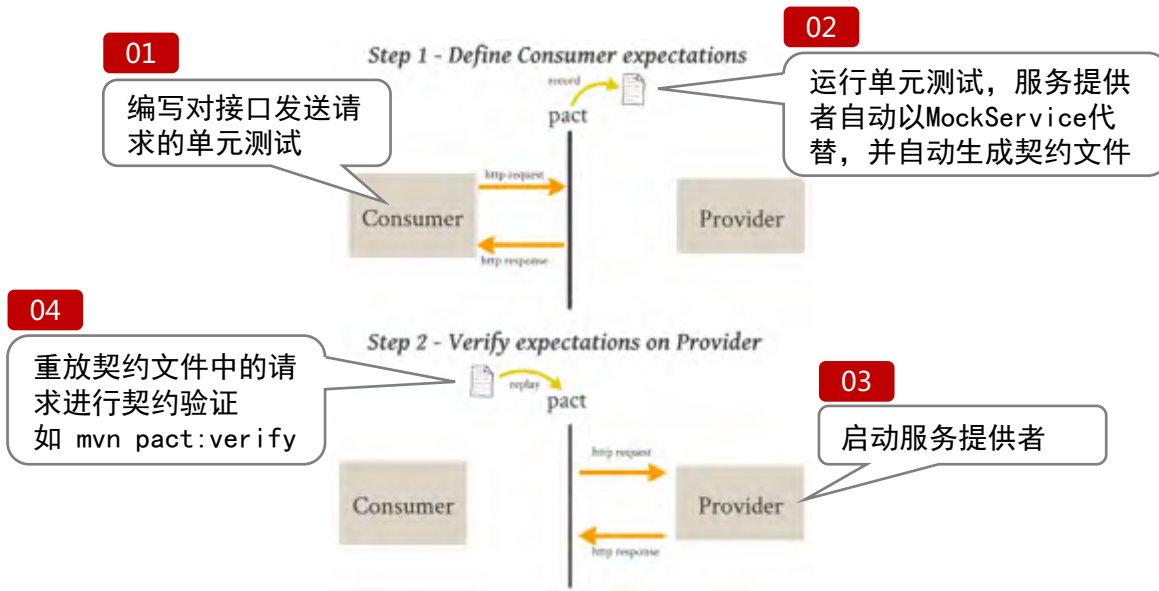
# 工程实践——契约测试

- 契约测试的最重要价值：降低服务集成的难度
- Do's :
  - 从消费者驱动，生成契约开始
  - Mock外部服务
  - 可以使用真实数据库
- Don'ts :
  - 不要调用真实的外部服务



# 工程实践——契约测试

- 消费者驱动的契约测试工具：Pact



# 工程实践——契约测试

## 轻量化

将【集成测试】简化为  
【单元测试+接口测试】

01

03

## 测试解耦

服务消费者与提供者解耦，甚至可以在没有提供者实现的情况下开展  
消费者端测试

02

04

## 一致性

通过测试保证契约与实现的一致性

## 测试前移

可以被开发阶段运行，并作为CI的一部分，便于尽早发现问题



# 工程实践——契约测试

- 推荐资料：《Pact中文参考指南》
- <https://www.pact.net.cn>

- Pact是怎样工作的
- 各种语言下的实现及使用指南
- 最佳实践
- FAQ及其它技术细节

# 工程实践——集成测试

- 集成测试的意义：从用户角度验证完整流程
- Do's :
  - 测试环境部署自动化（基础设施即代码）
  - 脚本编写自动化（使用录制工具等）
  - 使用真实的数据库和数据
- Don'ts :
  - 尽量少做集成测试（尽量测试前移）

# 工程实践——集成测试

- 环境部署自动化工具
  - Docker
  - Docker-Compose
  - Shell Scripts
- 集成测试/验收测试自动化工具
  - Selenium
  - Cucumber

# 组织实践——角色划分与如何协作

## 开发工程师

实现功能

编写代码与单元测试

共同参与编写API测试与契约测试

## 自动化测试工程师

主导制定测试策略

选择或开发合适的自动化工具

帮助其他团队成员更容易做测试



## 测试工程师

对具体业务测试的实现

编写集成测试，执行手工测试

共同参与编写API测试与契约测试

## 需求分析师

分析与定义需求

进行验收测试

# 目录

**1** 微服务测试面临的挑战

**2** 微服务测试策略

**3** 轻量化微服务测试实践

**➔ 4** 总结





# 总结

## SUMMARY

### 01 挑战

微服务测试的挑战来源于微服务架构的特点,应对这些挑战仍要从对微服务架构的系统化思考入手

### 02 策略

根据团队实际情况选择合适的测试策略,逐渐过渡到理想策略

### 03 实践

工程实践

- Do's and Don'ts
- 选择合适的轻量化工具

组织实践

- 角色划分与如何协作





高效运维社区  
GreatOPS Community



会议

培训

咨询

- 8月18日 DevOpsDays 上海
- 全年 DevOps China 巡回沙龙
- 11月17日 DevOps金融上海

- EXIN DevOps Master 认证培训
- DevOps 企业内训
- DevOps 公开课
- 互联网运维培训
- 企业DevOps 实践咨询
- 企业运维咨询



商务经理：刘静女士  
电话 / 微信：13021082989  
邮箱：liujing@greatops.com



# Thanks

荣誉出品

高效运维社区

国际最佳实践管理联盟



想第一时间看到  
高效运维社区公众号  
的好文章吗？

请打开高效运维社区公众号，点击右上角小人，如右侧所示设置就好

