QCon 全球软件开发大会
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

BEIJING 2017

大规模分布式系统架构下调测能力构建之道

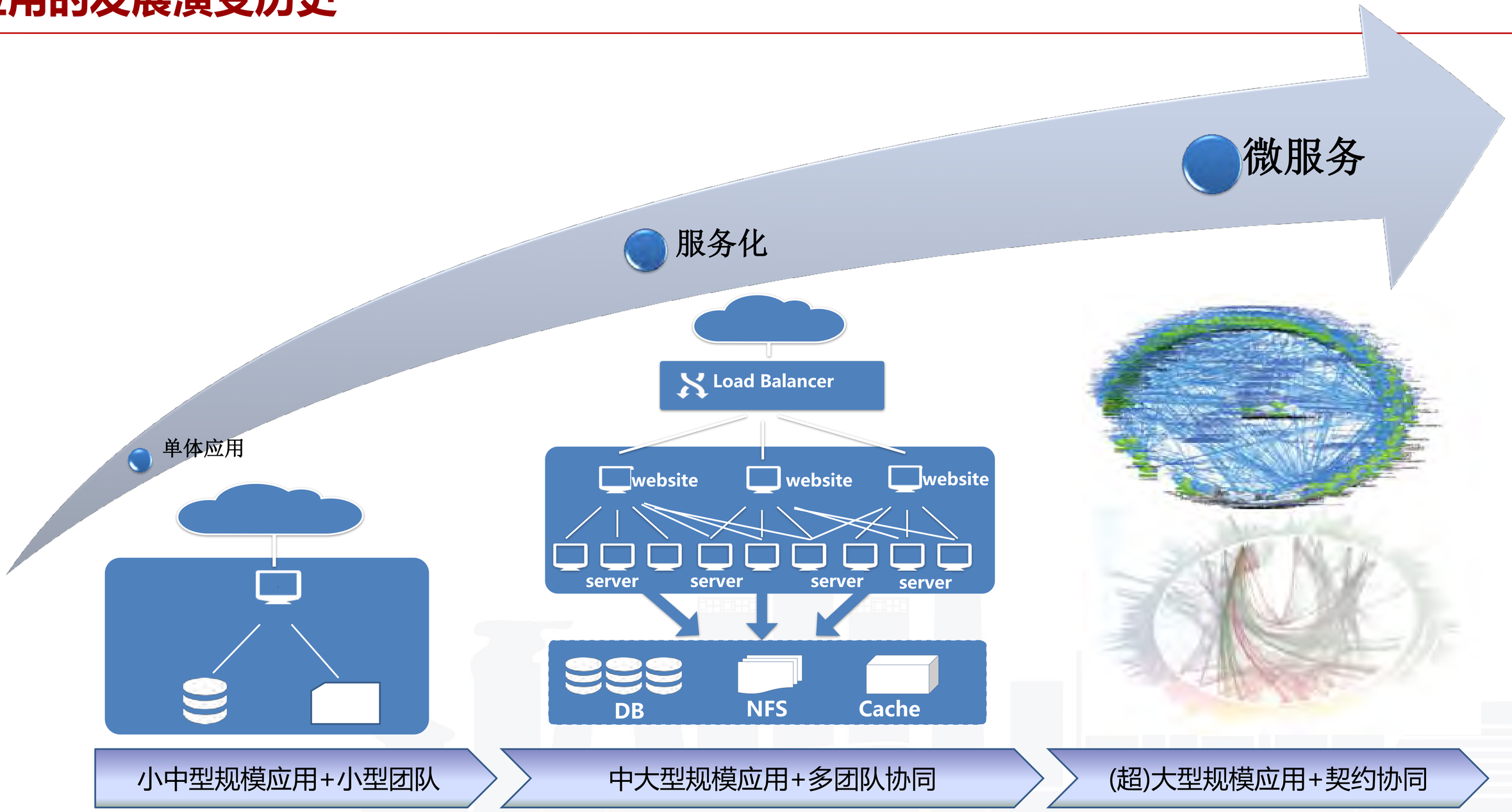
天弘基金 李鑫

主办方 Geekbang. 极客邦科技 InfoQ

大纲

- 分布式环境下开发的调测效率问题
- 应对之道
  - 远程应用服务
  - 契约测试
  - 分布式消息服务
  - 分布式缓存
  - 分布式服务的“租户”隔离策略
- 总体调测框架实践
- 分布式环境下调测方法论

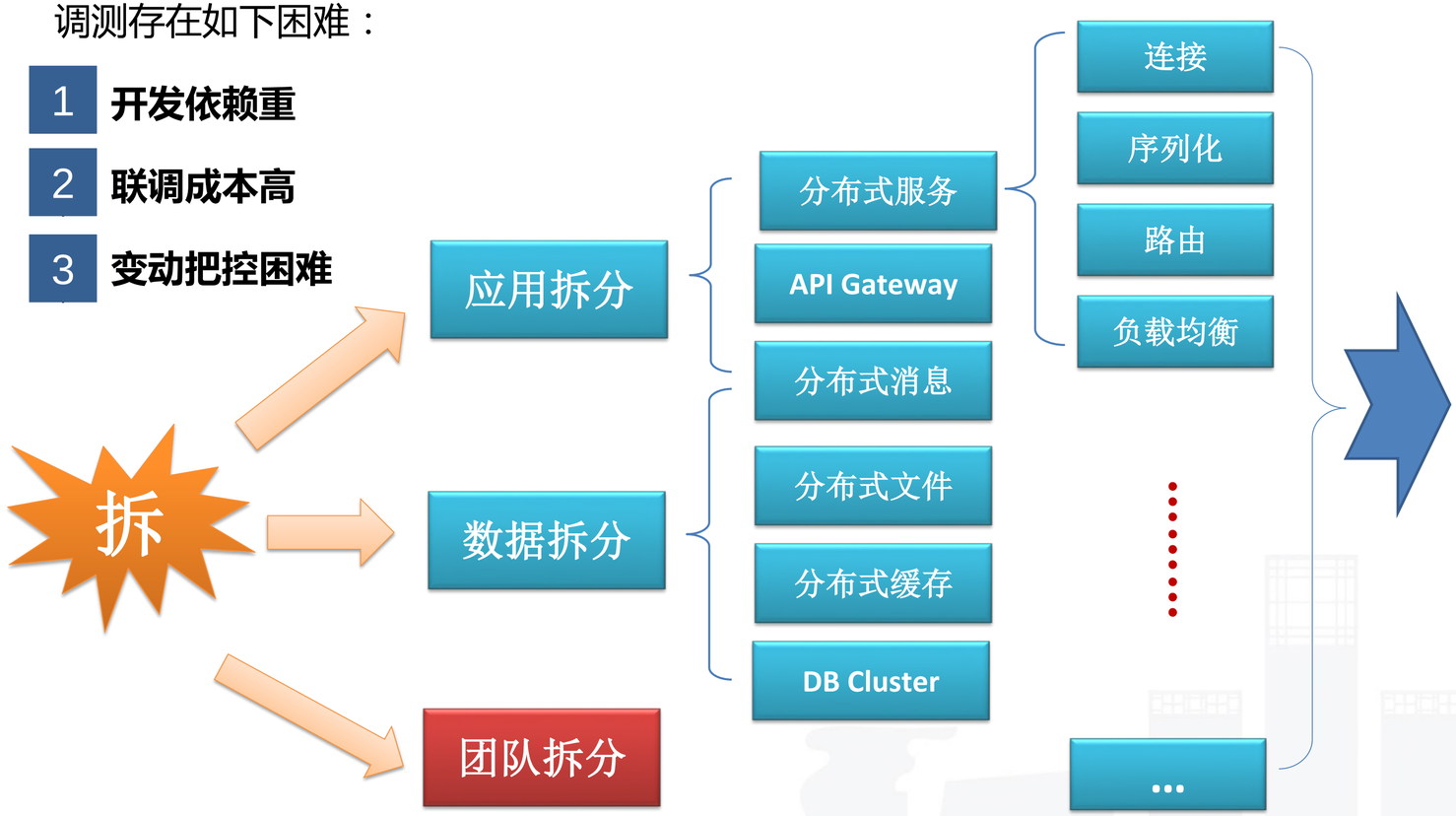
应用的发展演变历史


微服务
服务化
单体应用
Load Balancer
website
website
website
server
server
server
server
DB
NFS
Cache
小中型规模应用+小型团队
中大型规模应用+多团队协同
(超)大型规模应用+契约协同

服务化后的问题

多团队协同的分布式环境下，不仅环境“重”，而且外部的服务是别的团队开发的，你很难得到一个稳定、快速的外部服务提供环境，在此背景下进行功能/接口调测存在如下困难：

1. **开发依赖重**
2. **联调成本高**
3. **变动把控困难**



解决之道：通过技术手段，降低系统对外部的依赖，而“MOCK”，是最有效的手段。

	问题示例
1.	我依赖一个远程服务，但在负责它的团队把它上线之前，我什么也做不了
2.	我负责的功能依赖一堆的远程服务，为了本地调测，我必须从头到尾梳理代码，再写一堆的mock语句把他们全mock掉了。每当业务逻辑变化了，代码中要增减相应的mock语句；每当依赖服务上线后，要把测试用例中对应的mock语句去掉。对测试用例的修改工作贯穿于整个开发工作之中。
3.	我和某同事通过服务框架进行调用联调，结果另一同事也把同名服务挂了上来，不幸的是，他的服务版本和我们的不一样，结果，一系列的灵异状况发生了...
4.	依赖的远程服务逻辑发生变化了，但负责它的团队没有通知到我，结果上线后直接导致生产事故。
5.	我和其他团队共用一套分布式缓存服务，为了防止数据覆盖，我只能和别人轮流排队上线调试，我等了一天，结果只能用一个小时。
6.	我写了个消息Consumer，布到线上准备测测，结果，别人的消息先到了，一下把我的预置数据全搞乱了...
7.	我的开发机性能还行，我想自己装一套依赖环境提高开发调测效率，结果消息服务、缓存服务、服务框架装完一启动，80%的系统资源就没有了，想想还有一堆的应用服务还没装，直接泪崩...
8.	办公区断网，所有线上依赖环境都无法访问了，得，今天啥也干不了了，回家洗洗睡吧....
…	.....

分布式服务框架mock能力构建

●将mock能力内置到分布式服务框架中

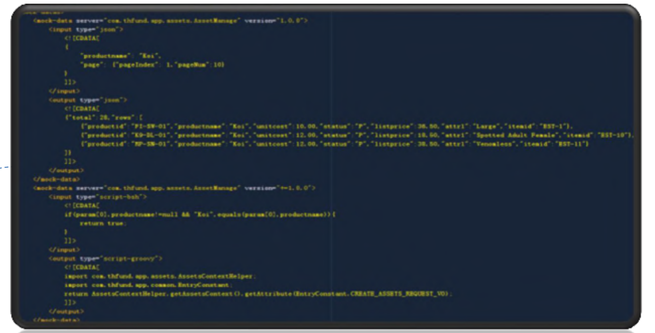
开发mock能力过滤器，在服务调用链路上对服务调用进行挡截。

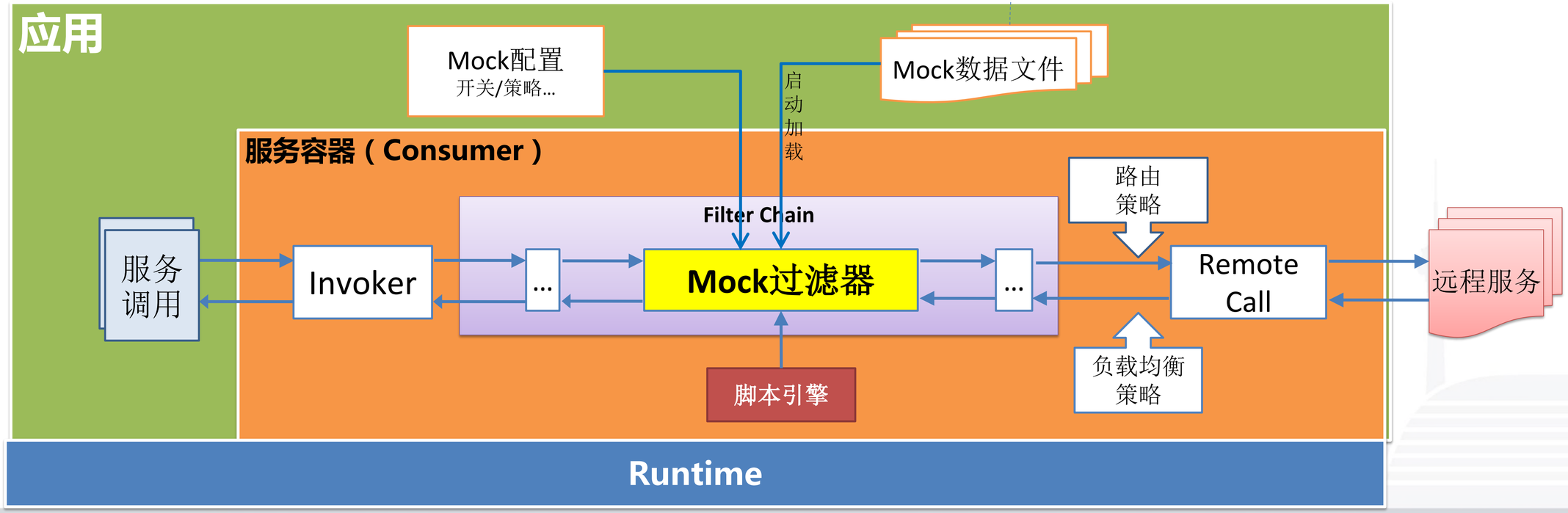
● “开关机制”控制mock能力启停

mock能力启用时，服务容器初始化期间将加载mock数据文件到内存中，每个服务请求将和mock数据的入参定义进行比对，满足（静态匹配 或动态匹配）规范的请求，直接用mock数据的出参定义作为服务调用的结果。

●应用无感知

无需修改应用代码，单元测试代码中也无需添加任何的mock语句。





Mock数据规范

- **格式规范：**
  1.静态匹配
  2.动态匹配
  3.多版本支持（服务版本）
- **管理规范：**
  1.服务谁提供，就由谁负责相应Mock数据的制作。
  2.mock数据可被使用者二次修改或替换。
  3.mock数据独立工程管理。

```
<mock-data server="com.thfund.app.assets.AssetManage" version="1.0.0">
    <input type="json">
        <![CDATA[
        {
            "productname": "Koi",
            "page": {"pageIndex": 1,"pageNum":10}
        }
        ]]>
    </input>
    <output type="json">
        <![CDATA[
        {"total":28,"rows":[
            {"productid":"FI-SW-01","productname":"Koi","unitcost":10.00,"status":"P","listprice":36.50,"attr1":"Large","itemid":"EST-1"},
            {"productid":"K9-DL-01","productname":"Koi","unitcost":12.00,"status":"P","listprice":18.50,"attr1":"Spotted Adult Female","itemid":"EST-10"},
            {"productid":"RP-SN-01","productname":"Koi","unitcost":12.00,"status":"P","listprice":38.50,"attr1":"Venomless","itemid":"EST-11"}
        ]}
        ]]>
    </output>
</mock-data>
```

静态匹配（json）

```
<mock-data server="com.thfund.app.assets.AssetManage" version="+=1.0.0">
    <input type="script-bsh">
        <![CDATA[
        if(param[0].productname!=null && "Koi".equals(param[0].productname)){
            return true;
        }
        ]]>
    </input>
    <output type="script-groovy">
        <![CDATA[
        import com.thfund.app.assets.AssetsContextHelper;
        import com.thfund.app.common.EntryConstant;
        return AssetsContextHelper.getAssetsContext().getAttribute(EntryConstant.CREATE_ASSETS_REQUEST_VO);
        ]]>
    </output>
</mock-data>
```

动态匹配（脚本）

在线抓取Mock数据

手工编写应用服务的mock数据往往工作量巨大，尤其是对一些数据驱动的业务，比如电信营业厅的套餐开户、基金业务中的基金购买等，**手工制作这些mock数据费时费力**。通过对线上（一般是测试环境）实际服务调用的真实数据的抓取来制作mock数据能非常有效的降低Mock数据制作的工作量，同时还能提升Mock数据的质量。

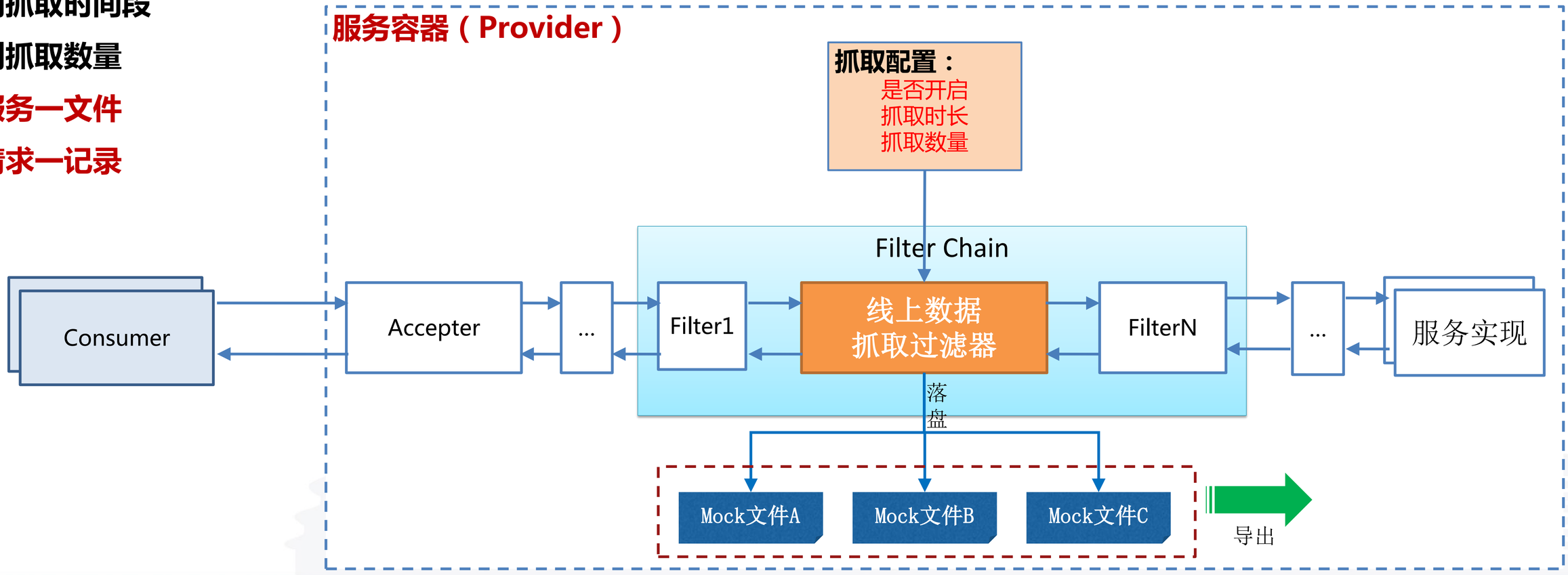
1.定制现网数据抓取过滤器

2.定制抓取时间段

3.定制抓取数量

4.一服务一文件

5.一请求一记录



应用服务直连调测

- **无需注册中心**

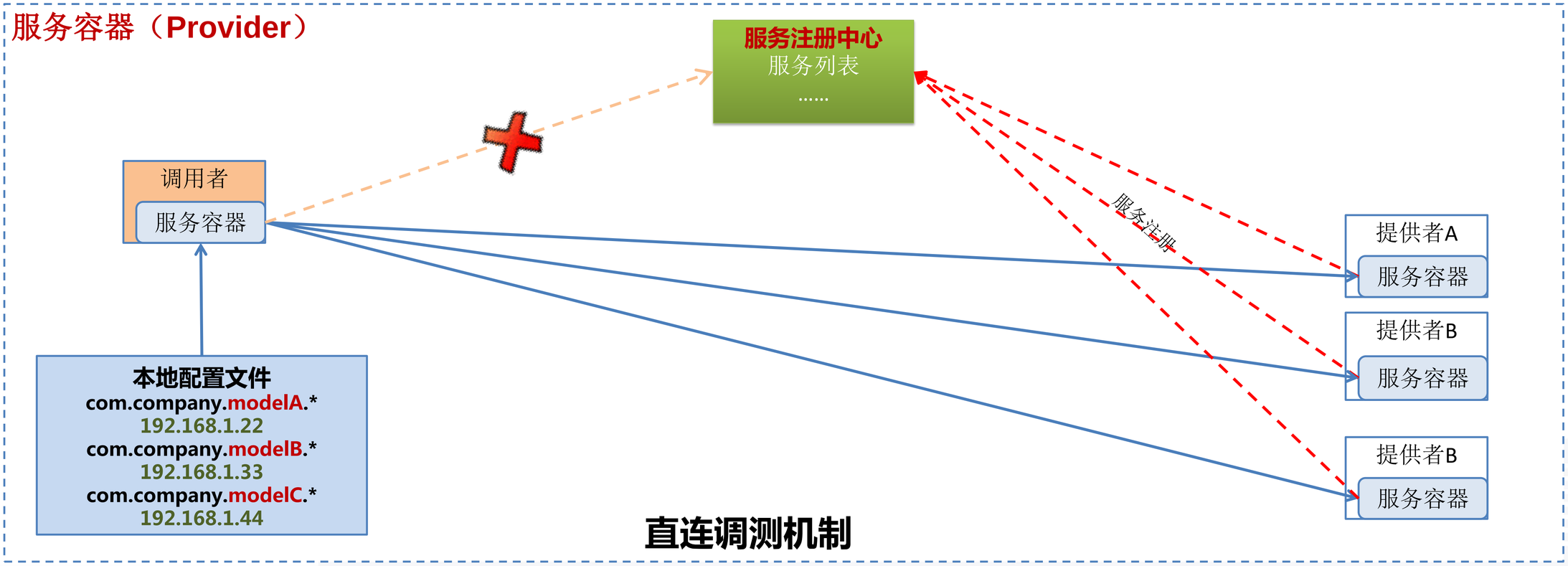
不需要服务消费者注册，直接通过本地配置文件指定的IP地址来绕过“服务路由”及“负载均衡”机制。

服务提供者不能采用token验证模式

- **基于包名过滤服务**

团队往往开发某类业务服务，这类服务一般都具有相同的包名，因此，可以通过配置包名和服务IP的映射关系，

让服务框架自动将一批服务和特定的IP关联到一起。



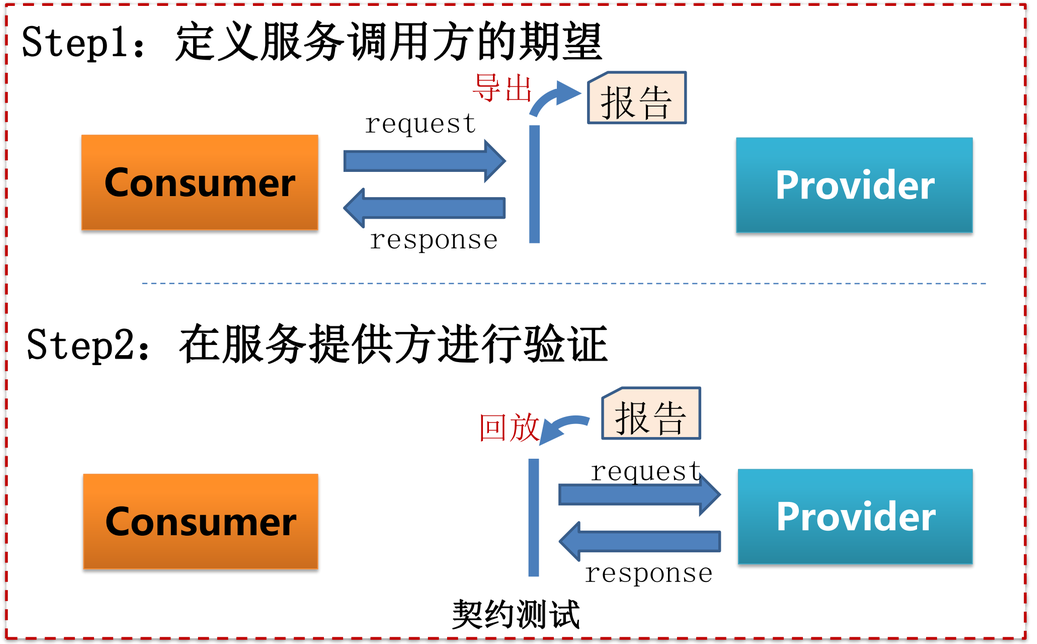
应用服务契约测试

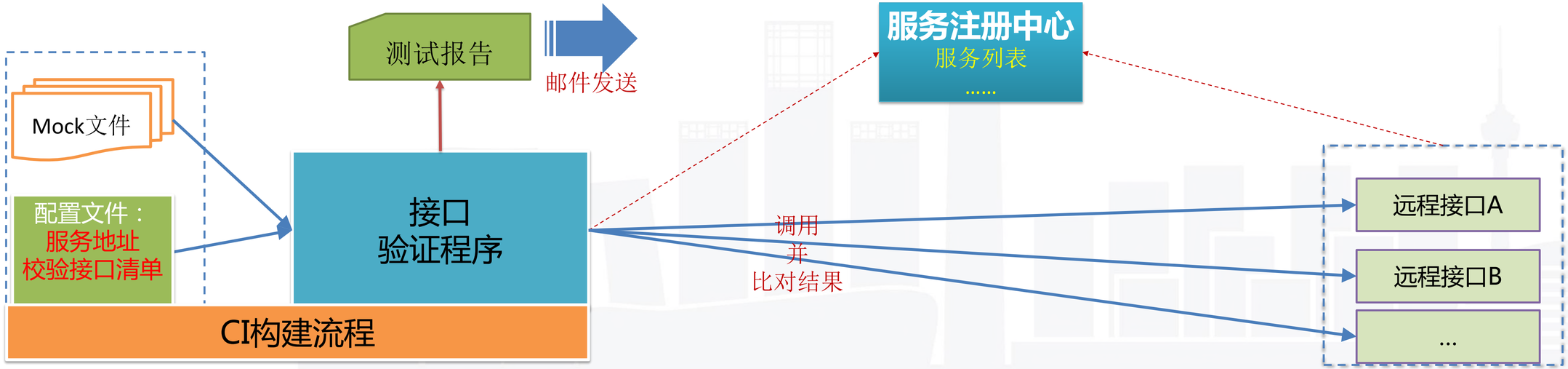
通过mock手段可以解决外部不可控因素对本地调测的影响，如果真实的外部服务改变了，我们的mock数据也要随之改变，但

问题是： **我们如何及时感知到服务接口/逻辑发生变化了？**

解决方法：契约测试

通过将契约测试集成到CI流程中，在构建的过程中完成接口的联调测试，和*接口变动的验证测试*，从而实现对接口变动的及时感知。



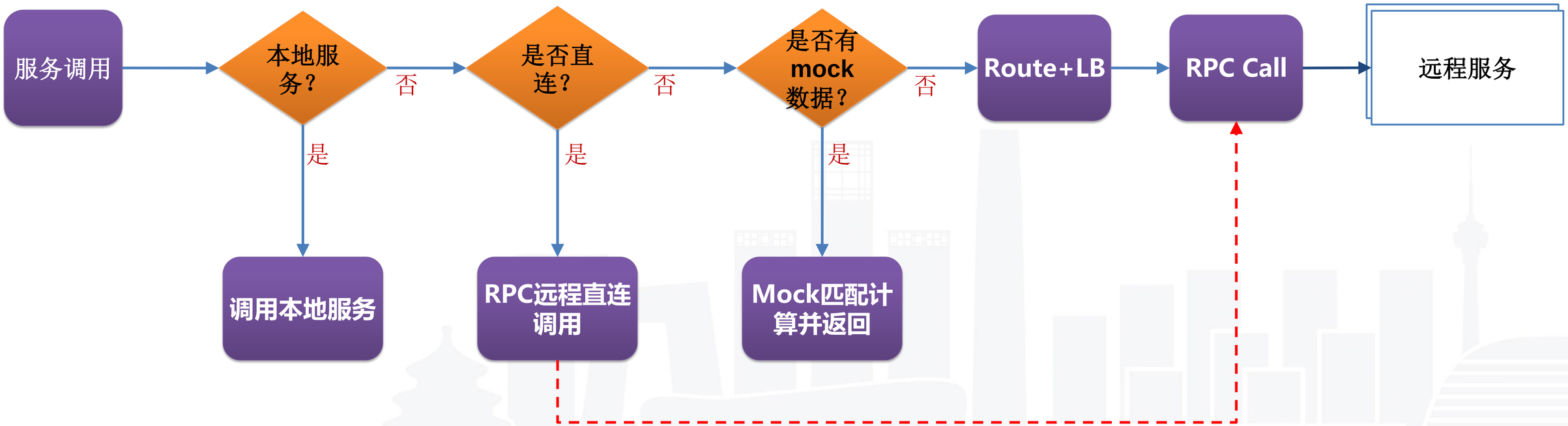


应用服务综合mock能力

在实际应用场景中，应用所依赖的服务往往很分散：

1. 一部分依赖服务从本机的Runtime环境即可获取，
2. 一部分需要从协同团队的远程开发机上临时获取（联调模式），
3. 一部分服务可能还未完成开发并发布，需要通过mock机制进行模拟。

我们需要综合利用前面所介绍的各种调测机制来保障日常开发中对应用服务的正常调测：



分布式消息Mock

- **MQ Mock**

通过本地JVM自带BlockingQueue队列模拟分布式队列，提供消息的发布、暂存、订阅、消费处理能力

- **代理门面**

将MQ的调用用代理模式做一层封装。

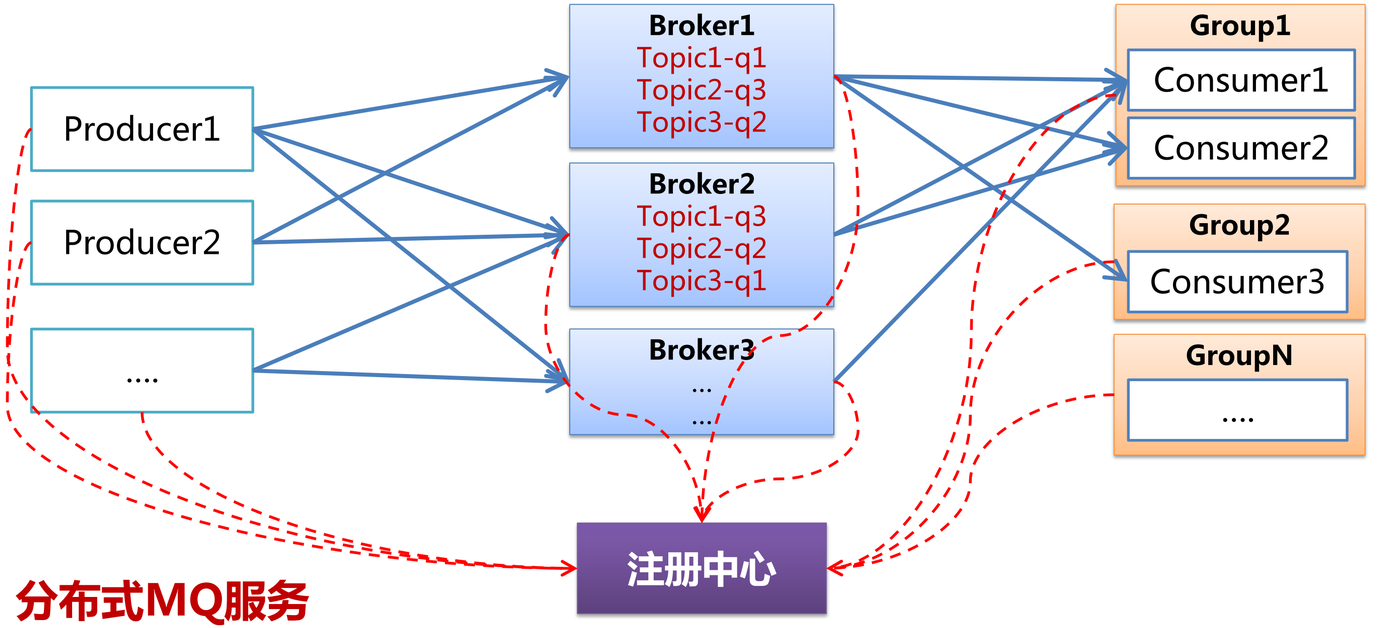
- **开关机制**

支持多级开关切换

- **消息预取**

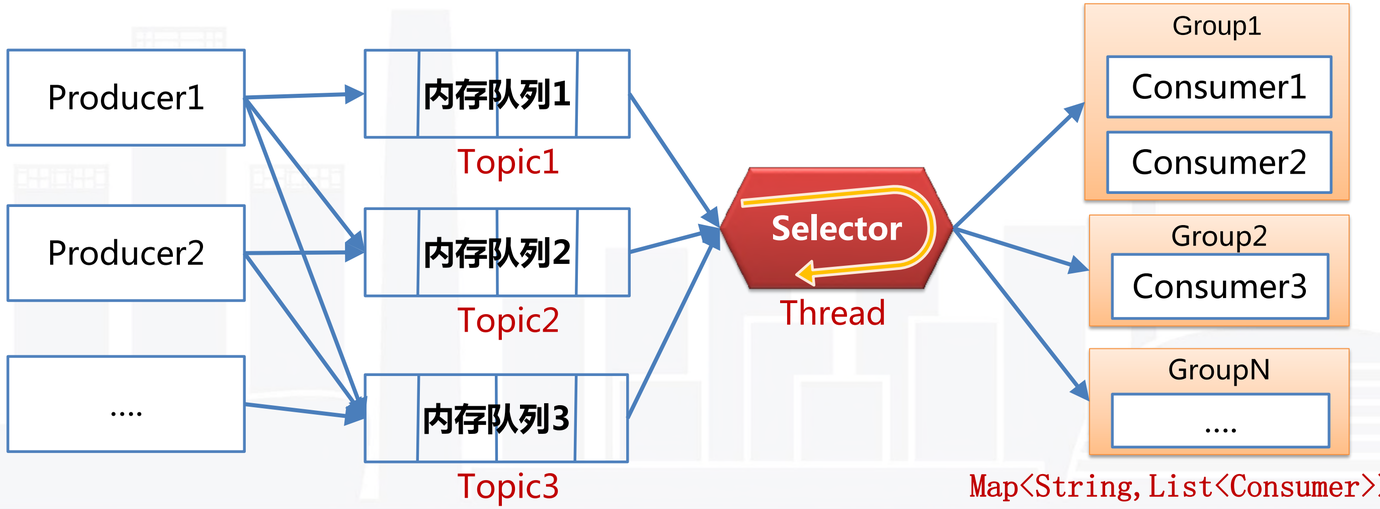
通过消息预判定，解决找不到消费者导致消息被抛弃的问题。

优势：能用极低的资源损耗实现单机环境下对分布式MQ服务的仿真模拟



分布式MQ服务

单机MQ Mock服务



分布式缓存Mock

- **Memcached**

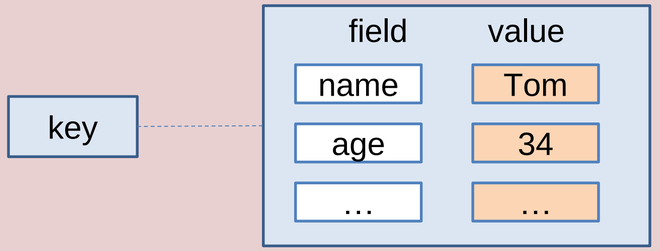
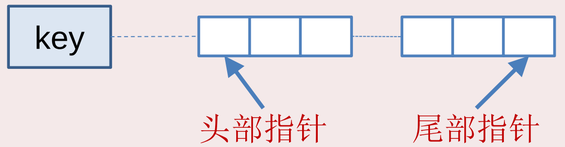
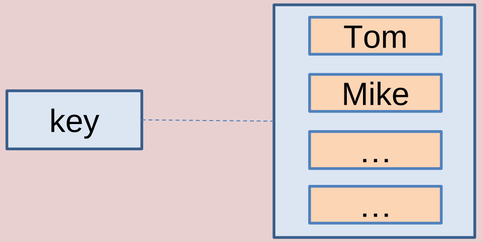
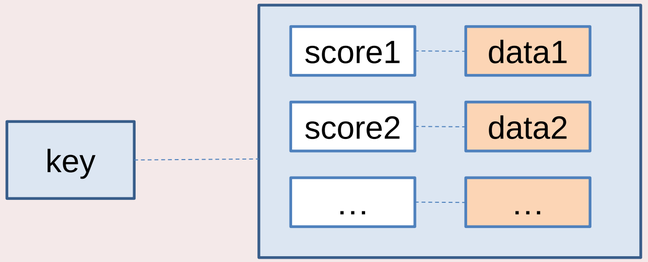
  采用普通的ConcurrentHashMap类进行模拟即可。

- **redis**

情况比较复杂，需要考虑不同的数据格式。

- **String**
- **Hash**
- **List**
- **Set**
- **Sorted Set**

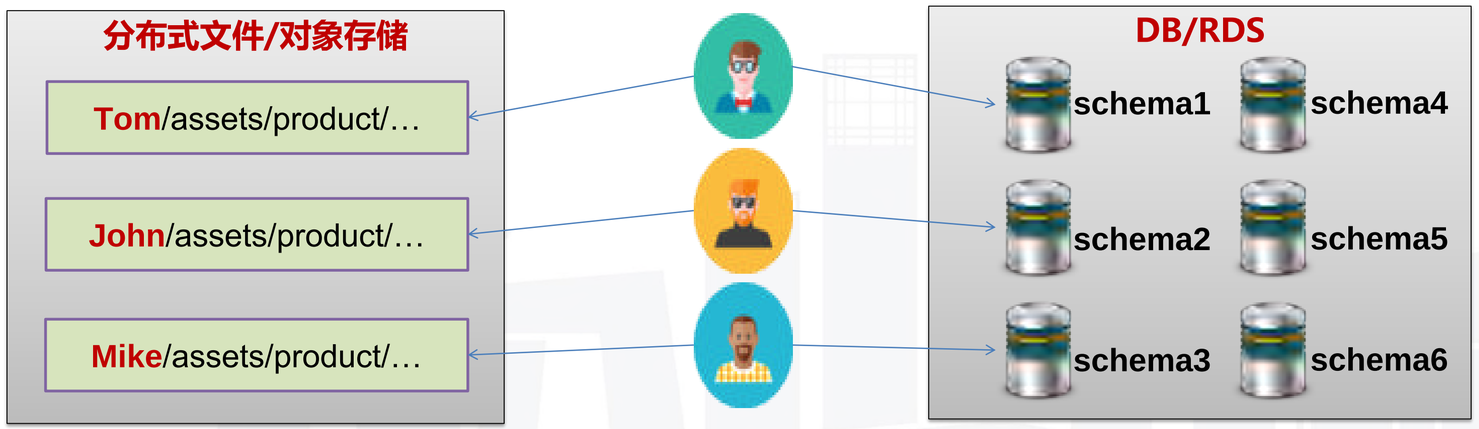
问题：数据时效性问题如何解决？

数据格式	模型	对应Java模型
Hash		Map<String,Map>
List		Map<String,BlockingQueue>
Set		Map<String,HashSet>
Sorted Set		Map<String,TreeMap>

其它分布式服务的调测支持

逻辑上支持“租户”隔离模式的分布式服务一般对多团队并行开发/调测的支持较好，分布式环境下，一般不会成为工程效率提升的瓶颈。针对这类服务，不用急于构建其Mock服务，可以根据其特性，寻找成本较低的实现途径。

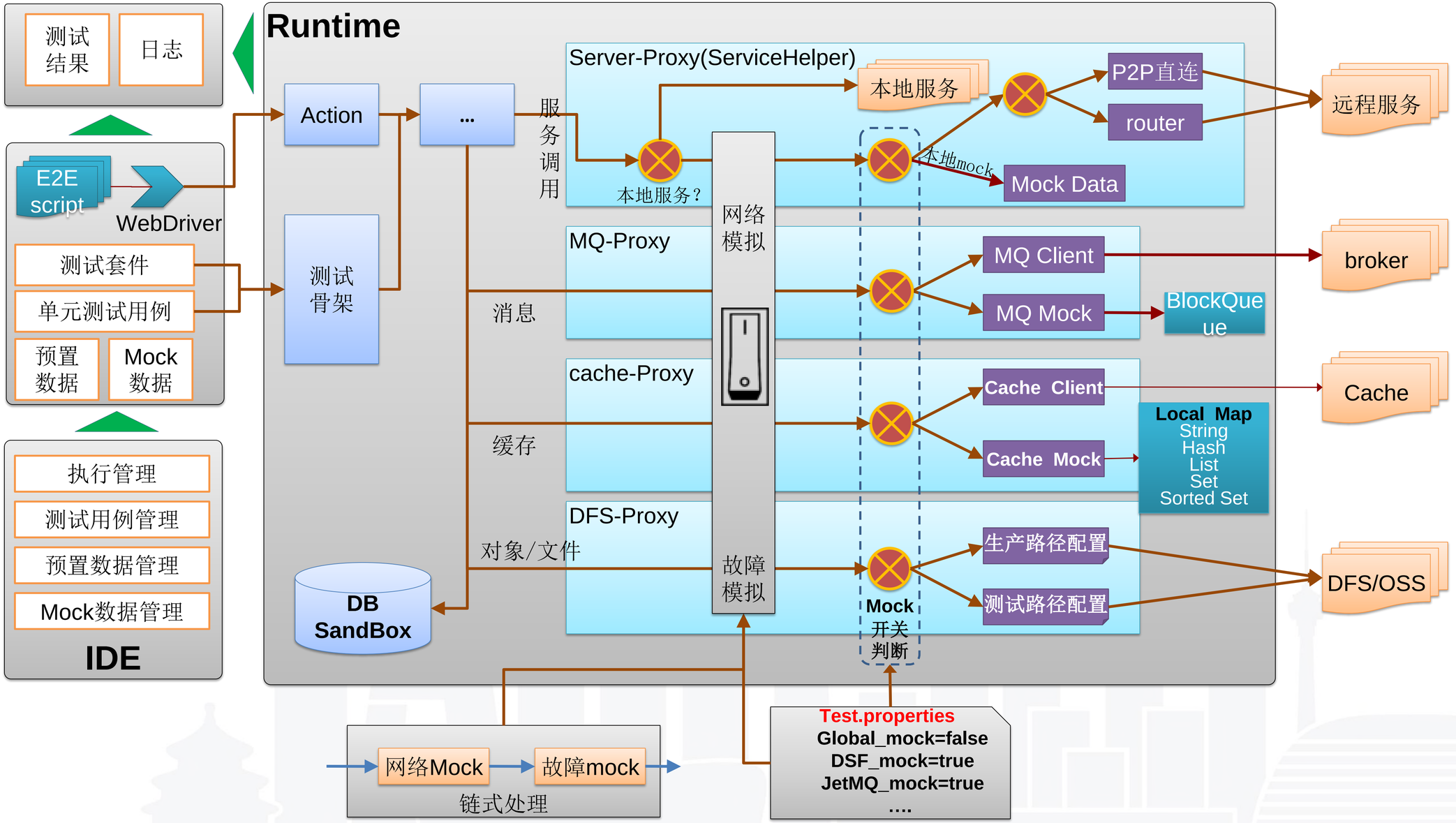
- **对象存储**
  1.分布式环境下，为不同开发团队/人员配置**独立的文件路径（Direction、Bucket）**可以有效进行资源隔离。
  2.单机环境下，可以参考MQ，采用本地File I/O实现的存储机制来mock 。
- **DataBase SandBox**
  1.为每个开发人员或测试人员提供一个单独（schema/catalog）的测试数据库。
  2.预置数据独立管理，并通过测试框架提供预置数据**前置导入**及**后置清除**能力。



资源的“逻辑”隔离机制

完整的调测框架

- SDK
- 测试骨架
- 多级开关控制
- 网络及故障模拟



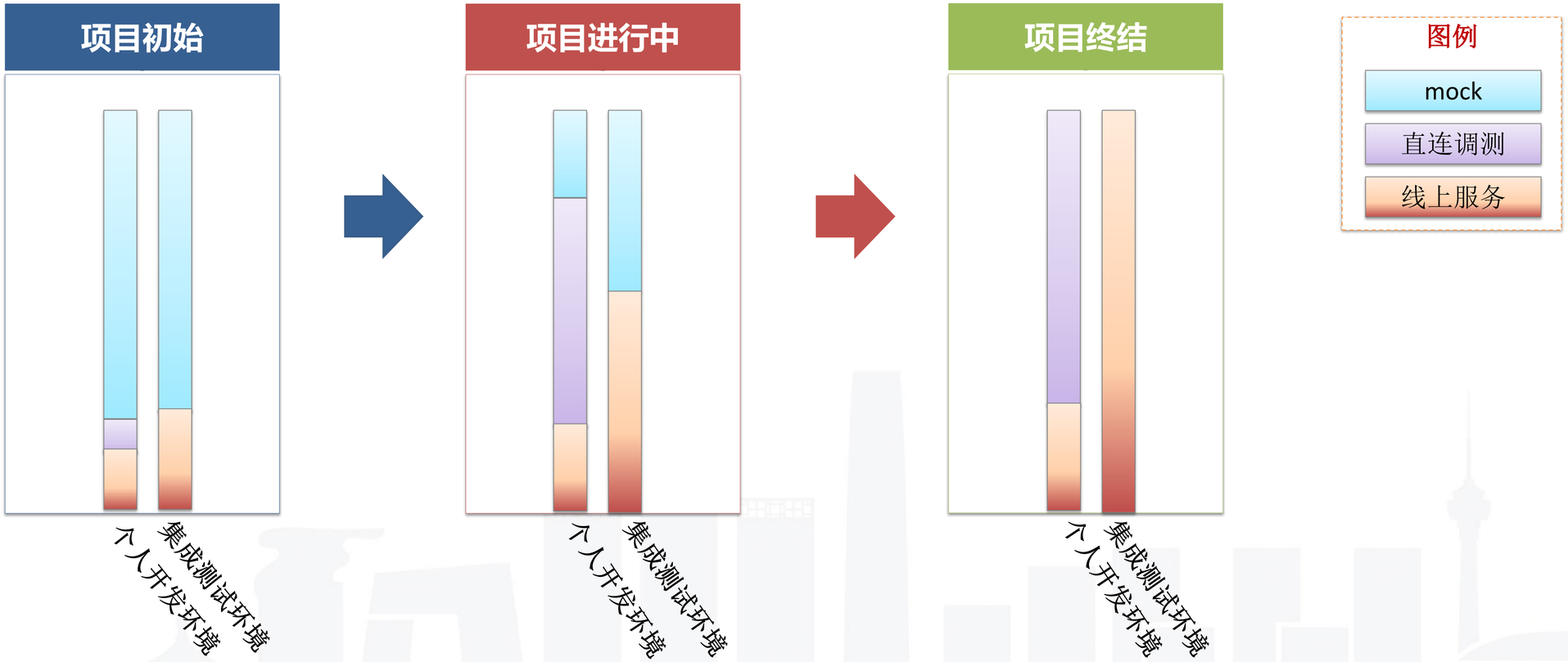
调测方法论

- **个人开发环境**

  Mock+直连调测+线上服务

- **集成测试环境**

  Mock+线上服务



项目各个阶段灵活组合使用各调测手段

总结

- 介绍了分布式环境下开发调测所遇到的各种效率问题
- 分布式环境下工程效率提升的应对之道
  - 远程应用服务采用服务mock+直连调测
  - 通过契约测试保障mock数据及协议的可靠性
  - 分布式消息服务的mock实践
  - 分布式缓存的mock实践
  - 分布式存储（File+DB）的“逻辑”隔离策略
- 调测能力的总体框架及实践展示
- 分布式环境下调测方法论

Q/A

联系方式：

李鑫

天弘基金　　高级架构师

E-mail:lixin.storm@foxmail.com

QQ:25893288

欢迎交流：

专注于大规模分布式应用及治理、中间件云化及服务化(PaaS)、APM监控、基础开发平台、移动应用平台、企业架构等技术领域。

个人公众号

天弘基金 TIANHONG ASSET MANAGEMENT

稳健理财 值得信赖

天弘爱理财

关注QCon微信公众号，
获得更多干货！

Thanks!

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

主办方 Geekbang. 极客邦科技 InfoQ