

通用Mock平台实践



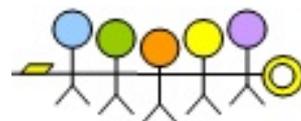
第一部分：Mock平台的使用背景

- Mock平台的由来
- Mock平台的价值



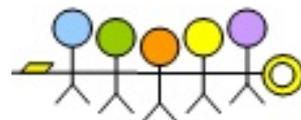
MOCK平台的由来

- 在联调环境不可用的时候，我们需要一个系统来模拟外部银行，商户与支付宝系统的互交。
- **Mock**是一个每人都能想到的东西，但是。。
- 我们需要怎样的一个模拟系统？
 - 接口模拟小程序？
 - 无存储的模拟服务器？
 - 面向用户的模拟平台？



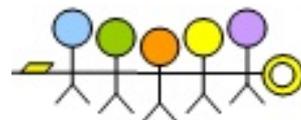
MOCK平台的由来

- 支付宝外部接口现状：
 - 支付宝有**450**余商户接口。
 - 支付宝有**300**余活跃着的银行渠道。
 - 不同的前置网关，不同的入口配置。
- 测试中可能碰到了问题：
 - 需要能够快速地在交付过程中开发**mock**接口。
 - 接口能适应 升级包/技术改造 带来的改变。
 - 新测试人员在不是很了解接口规范的情况下也能完成测试。



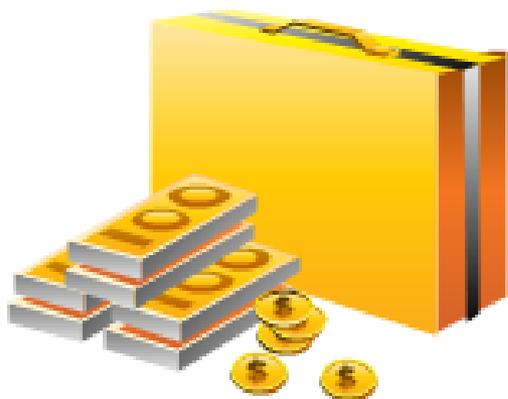
MOCK平台的由来

- 支付宝接口模拟关键字：
 - 轻量的开发。
 - 变化的适应。
 - 适当的封装。
- 以上要求决定了我们的mock需要是一个：
 - **Server, Client**兼备的。
 - 配置型的。
 - 模板化的。
 - 可共享的。



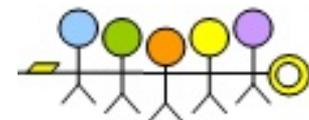
MOCK平台的由来

- 这样的一个平台会带来不小的开发量，那么，它到底会给我们带来什么呢？



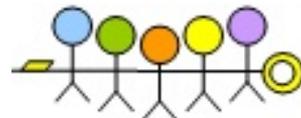
MOCK的价值

- 原始价值：
 - 模拟系统，解决不可测问题。
- 后面发现的附带价值：
 - 更容易地制造异常，帮助总结异常模型。
 - 接口与测试数据的保存与交流。
 - 完成流程的切断的测试。（跳跃数据初始化过程）
 - 帮助完成报文内容的查看和测试。



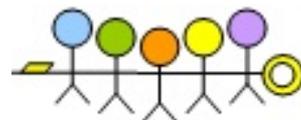
第二部分：Mock系统实践

- Mock的需求者
- 基于需求的思考
- （例）API平台的配置
- （例）银行Mock的设计
- 与其他测试技术的结合



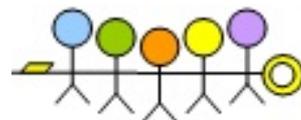
MOCK的需求者

- 商户：协议类似，互交方式和接口数据结构不同。
- 银行：上百渠道，协议，互交，内容均不同。
- 测试者：测试不同业务，**mock**需要能够适配。
- 项目管理者：不同项目，需要不同版本的接口。
- 测试主管：有的测试者需要测试正常，有的测试者需要测试异常，不能有影响。



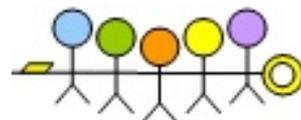
基于需求的思考

- 实现mock需要考虑的问题：
 - 接口覆盖-三个维度
 - 系统互交方式
 - 通信协议
 - 传输报文的内容
 - 不同的测试者，需要不同的mock数据。
 - 协议码（基于报文的判断系统）
 - 监听型日志
 - 权限控制



基于需求的思考

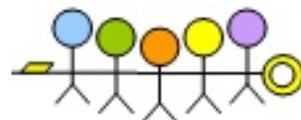
- 如何在项目用，配置及应用**mock**
 - 初期审核及文档
 - 开发，测试，**PM**角色协调
 - 做到关注**output**
- **实践mock情况简介：**
 - 对于**Apitest**平台(**client**型)的配置方法。
 - 对于银行**mock**的设计。 (**anymock2.0**)



API平台的配置

- **Apitest平台：Client型的接口模拟：**
 - 全面的参数化。
 - 合理的（接口配置）权限控制。

业务参数			备注
服务费	service_fee	<input type="text" value="10"/>	<input type="text"/>
邮费	transport_fee	<input type="text" value="5"/>	<input type="text"/>
外部交易号	out_trade_no	<input type="text" value="794044146422508"/> <input type="button" value="刷新"/>	<input type="text"/>
调整价格	adjust_fee	<input type="text" value="-1"/>	<input type="text"/>
操作id	op_id	<input type="text"/>	<input type="text"/>
交易号	trade_no	<input type="text" value="2012060712596474"/>	<input type="text"/>



API平台的配置

- 接口的拼装。
 - 不同的编码方式。
 - 业务与协议参数。
 - 各种签名的生成。
 - 随机数id的生成。

跳转URL查看

生成的跳转URL：

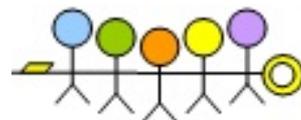
```
http://aliapi.stable.alipay.net/gateway.do?trade_no=2012060712596474&transport_fee=5&sig  
notify_url=http%3A%2F%2Fapi.test.alipay.net%2Fatinterface%2Freceive_notify.htm&adjust_fee  
out_trade_no=7940441464225088&sign=15d195c228cf825f23571bedec125060&service_fee=  
service=trade.etrade.modifyCODFee&partner=2088001159940003
```

用于签名的字符串(签名方式[MD5] 编码方式[GBK])：

```
adjust_fee=-1&notify_url=http://api.test.alipay.net/atinterface/receive_notify.htm&out_trade_n  
partner=2088001159940003&service=trade.etrade.modifyCODFee&service_fee=10&trade_no=  
transport_fee=5
```

签名结果：

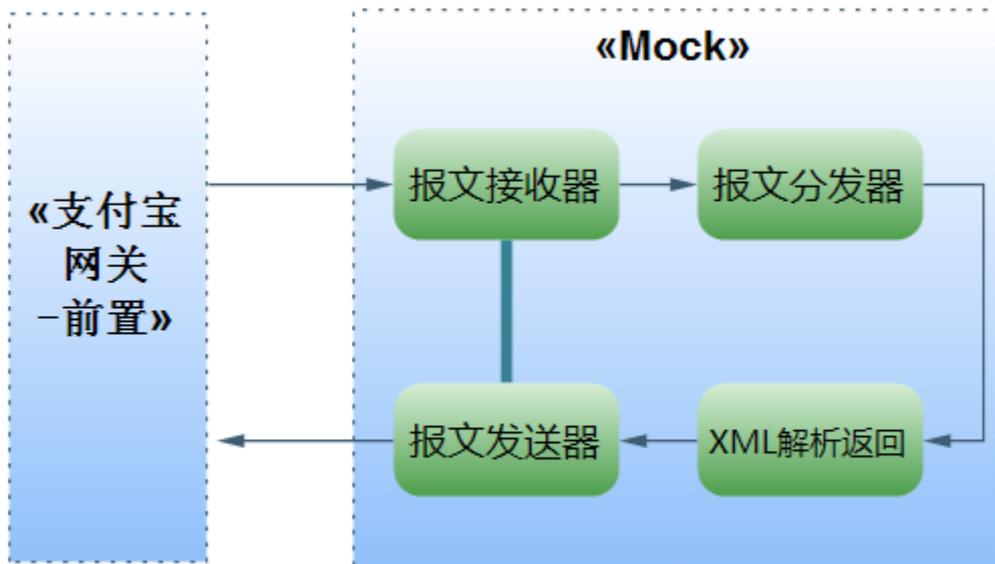
```
15d195c228cf825f23571bedec125060
```





银行Mock的设计

- 银行mock，覆盖较为全面的接口mock平台：
 - 接口覆盖范围广，适配程度高。
 - 扩展性与可插拔特性。
 - 使用者分层：高级的-Mock模板配置人员。



```
*Template
1 <root xmlns:schemaLocation="http://www.schema.anymock.aliy
2 <flow>
3   <filter id="com.alipay.anymock.web.home.SampleContro
4     <cd:CDKey>identifier</cd:CDKey>
5     <cd:CDValue>/httpmodell/gate.htm</cd:CDValue>
6   <operation id="com.alipay.anymock.biz.shared.compo
7     <cd:CDKey>codeRule</cd:CDKey>
8     <cd:CDValue>[method]</cd:CDValue>
9     <cd:CDKey>state</cd:CDKey>
10    <cd:CDValue>2</cd:CDValue>
11  </operation>
12  <operation id="com.alipay.anymock.biz.shared.compo
13    <cd:CDKey>template</cd:CDKey>
14    <cd:CDValue><![CDATA[&lt;!--separator:code--&gt;
15    <cd:CDKey>template_error</cd:CDKey>
16    <cd:CDValue><![CDATA[&lt;?xml version="1.0&
17    <cd:CDKey>defaultRule</cd:CDKey>
18    <cd:CDValue>>null</cd:CDValue>
19  </operation>
20 </filter>
21 </flow>
22 </root>
```

银行Mock的设计

— 普通的接口使用人员：基于模板的视图

Http-键值对接口模板组件 [\[打开端口\]](#) [\[删除模板\]](#) [\[返回上级模板\]](#)

anymock的日志见主机 10.253.3.4 的 anymock-components.log 和 anymock-error.log 两个文件

HTTP 服务器组件

URL地址[identifier]

协议码解析组件

协议码[codeRule] 消息类型[state]

键值对解析组件

错误规则[template_error] 默认规则[defaultRule]

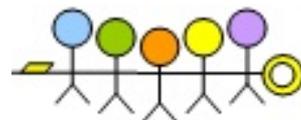
规则[template] [请点击这里](#) [查看](#)

— 个性化日志（监听型日志）。



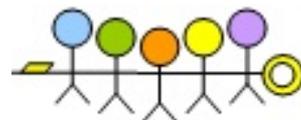
银行Mock的实践

- **mock**组件，帮助**mock**完成拓展的功能：
 - 报文分发器。-基于关键字，基于特殊规则。
 - 各类报文处理**util**。-基于不同格式报文。
 - 报文验证器。-**server**型
 - 自定义组件的加入。-提供签名验签等入口。
- **mock**的其他条件：
 - 业务测试时，签名验签功能的无效化。
 - **Jar**包类型加密，加签的报文，**Mock**掉**jar**包。



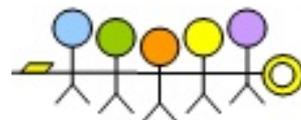
与其他测试技术的结合

- 对于各类测试的支持
 - 自动化测试
 - 提供接口设置。
 - 提供URL或者脚本触发。
 - 性能测试
 - 几乎没有并发控制。
 - 模板都是在缓存中的。
 - 组件模型的可拆卸。



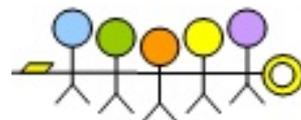
第三部分：Mock平台的挑战与展望

- 面临的挑战
- 平台的未来



我们面临的挑战

- 接口管理方面：
 - 接口公有与私有。
 - （共用环境中）服务资源的冲突。
- 接口覆盖方面：
 - 签名/验签/加密过程的 易用性与覆盖率共存。
 - 老接口与内部接口的覆盖。（无明确接口文档）
 - 长连接/**session**的功能应用。



平台的延伸和未来

● Mock的思索：

- 哪些东西是真正需要被mock掉的。
- 我们如何获得接口规范和需要相关数据。
- **Mock**数据的共享化和私有化的管理。

● Mock新功能：

- **proxy**功能，寻找着它更好的实现和的使用场景。
- 和测试脚本相结合的**mock**，服务**mock**正在制作中。

