# http自动化桩的设计文档

# 背景：

之前在团队落地了接口自动化测试框架，极大的提高了测试效率以及代码覆盖率，所设计的测试用例对代码的覆盖率已经达到了70%，但是项目这边的通过标准是需要达到80%以上，于是我们对项目进行了精准的分析，除去可忽略的代码，发现对第三方依赖服务的测试点没有覆盖全面，主要是因为测试环境未接入第三方服务，开发只提供了第三方服务的线上测试环境，一开始我们提出桩的概念，可以理解为假想的服务，去替代第三方服务，模拟与被测服务的场景交互。但后面需要考虑校验异常场景，又实现了自动化桩，也就是将校验依赖服务异常测试用例转化成自动化脚本。

# 以往存在的痛点：

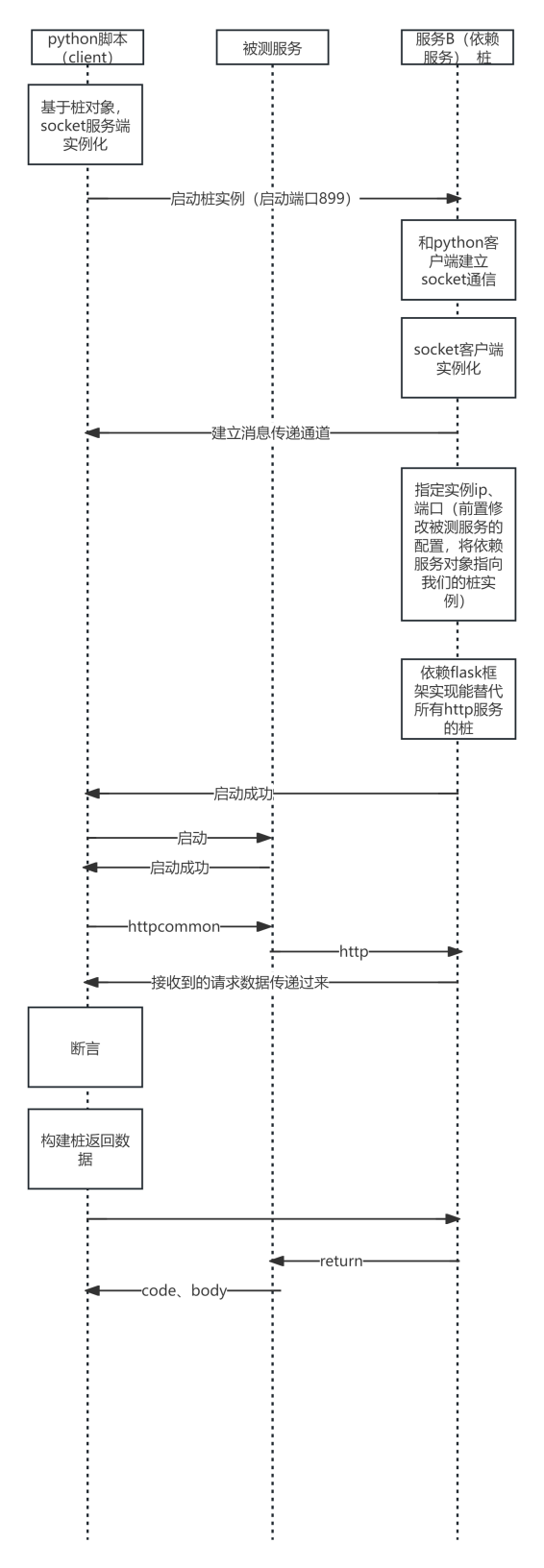
1. 三方服务异常的处理行为覆盖不到。开发一般会提供初级桩，只能保证通信过程，固定返回200的处理结果。但只能测试正常而且单一的交互场景，无法对满足交互异常情况的测试。
2. 被测服务请求三方服务的协议正确性难校验。如果三方服务可以接入测试，按之前只能靠抓包工具拦截真实服务，获取被测服务请求三方服务的接口，再手动对齐接口文档。但无法接入的话，仅靠初级桩是无法校验到协议正确性。
3. 异常场景无法落地成自动化。对于三方服务交互异常的场景如返回超时、返回错误的状态码、三方服务器突然宕机等。可以通过抓包工具去手工测试模拟接口拦截、数据的篡改、kill -9使服务下线。但这些异常场景的测试用例无法转换成自动化脚本。
4. 线上测试的成本过高。因为无法接入真实的三方服务测试，所以没法保障被测服务上线以后能和三方服务正常通信，还需要额外的线上调试成本，还可能影响线上的真实数据

# 自动化桩落地的价值：

1. 可对桩的返回进行修改，涵盖状态码、返回体内容以及返回时间的调整，以此覆盖所有交互异常场景。
2. 能够对被测服务请求三方服务的协议正确性展开校验，并提供针对请求依赖服务协议的断言方法。
3. 可将测试用例转换为自动化脚本，达成自动化测试的目的。

项目名：httpStubFramework

# 交互图：



桩的交互流程：

step1：在python客户端实现socket服务端，提供通信方式

step2：在桩模块下，实现socket客户端，和python客户端的socket服务建立通信，明确channel并实例化。

step3：根据桩启动配置所描述的端口在桩模块基于flask框架启动http的桩，满足任意接口和任意请求方式。

step4：启动被测服务，满足依赖关系。

step5：测试用例开始执行，使用桩模块提供的异步http请求方法调用被测服务的接口。

step6：被测服务在入参协议校验完成后开始请求依赖服务获取业务的处理结果，被测服务调用依赖服务的接口（实际上就是请求桩）。

step7：桩服务会通过channel发送消息，将从被测服务接到的请求打包起来发送到python客户端的socket服务。

step8：python客户端会通过拉取消息的方式接收消息。

step9：python客户端需要定义桩的结果返回，定义完成通过socket提供的send进行消息的回调。

step10：桩接收到回调消息后包装成回调消息直接进行被测服务的消息回复。

step11：被测服务通过接口返回返回业务的处理结果。

step12：python客户端能正常完成用例的流转过程。

step13：python客户端执行完全量用例后下线桩服务。

# 自动化桩的模块

**httpCommon**

1. 提供http的异步通信方法，可以将http的发送和接收实现异步的请求方式。
2. 提供的方法HttpCommon().http\_requests() http\_requests的形参继承requests.requests()的
3. 提供获取http请求结果的方法

**httpStubOperator**

定义了桩的操作方法，类名StubOperator

**httpServerStub**

桩实例

# 自动化桩使用的协议：

桩的启动配置stub\_instantiation.py

1. from httpStubFramework.httpStubOperator import StubOperator # 在该模块实现桩对象的实例化，需要导入模块下的类
2. # 每一个桩都需要先定义一套配置
3. stub\_config = {
4. 'http\_app\_port': 8770, # 定义了http桩实例的端口，ip默认本机
5. 'socket\_client\_port': 8771, # 定义了桩通道通信过程socket的客户端端口
6. 'socket\_server\_port': 8772 # 定义了桩通道通信过程socket的服务端端口
7. }
8. # 如果要定义一个桩，可以根据桩的业务定义实例化的名称
9. fileAppStub = StubOperator(桩的配置信息)

桩启动方法（自动化框架执行入口main）

1. from stub\_instantiation import fileAppStub
2. fileAppStub.start\_stub()

桩下线方法（自动化框架执行入口main）

1. from stub\_instantiation import fileAppStub
2. fileAppStub.shutdown\_stub()

桩http异步方法应用（用例层在调用被测服务http接口时的接口调用方法）

1. from httpCommon import HttpCommon
2. hc = HttpCommon() # 使用前需要进行实例化
3. hc.http\_requests() # 发送http接口请求，实参传递满足requests.requests的形参
4. # 等http接口业务处理完成后
5. code = hc.status\_code # 获取接口返回的状态码
6. text = hc.res\_text # 获取接口返回的返回体文本

从桩接收消息（用例层）

1. from stub\_instantiation import fileAppStub
2. data = fileAppStub.receive() # 接收桩受收到的请求协议消息

接收消息模板：

{

"body": dict(re\_data), # 收到的请求体

"headers": dict(re\_headers), # 收到的请求头

"cookies": dict(re\_cookies), # 收到的cookie

"path": path, # 收到的路由地址

"method": re\_method, # 收到的请求方式

"params": dict # params参数，以字典形式返回

}

定义桩的结果返回（用例层）

1. from stub\_instantiation import fileAppStub
2. msg = {
3. 'body': {}, # 返回体在这一侧定义，目前只支持json返回体的返回
4. 'code': 200 # 定义了桩的返回状态码
5. }
6. fileAppStub.send(msg) # 将定义好的返回体返回给桩