1. 产品介绍
2. 基本组成

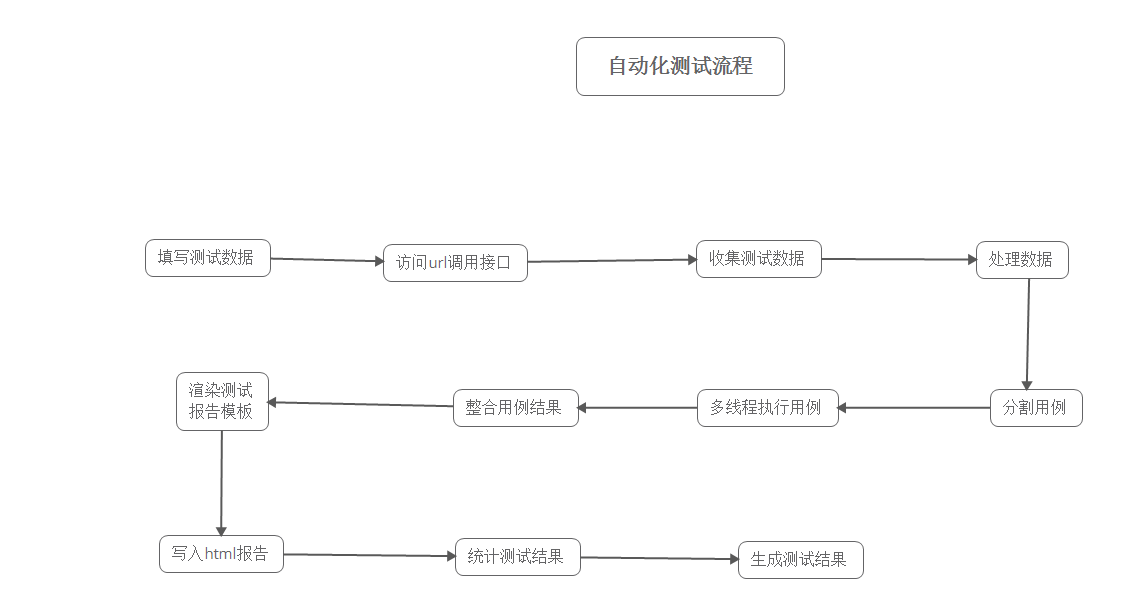
该产品分为前端后台部分，采用了web的形式进行操作，用户可以通过浏览器来完成相应的操作，如执行用例等。

1. 编程语言及框架

前端语言包含: html, css, javascript，框架为bootstrap。

后端语言为Python 主体框架为flask。

1. 实现原理



1. 用户将接口测试用例存放在某个固定的目录(该目录可配置)，通过调用GET请求的方式来达到执行用例的目的，一般该操作被简化为在浏览器访问URL地址，URL地址里包含有需要执行的excel文件名。上述过程是通过flask的接口路由(route)来实现的。

2. 当接口获取到文件名之后，并会在后台读取excel里的每一条用例，目前有用例配置开关，通过excel里的enable参数控制用例是否执行。获取到每一条用例的信息如url地址，用例名称，params，请求方式，断言方式然后存放到case\_list列表里。

3. 由于excel里储存的是文本，所以需要对json数据进行解析。

4. 之后便是使用Python下的requests库， 访问接口并拿到接口的状态码进行对比，如果与预期状态码相符，则对接口的返回值与预期值进行断言，断言方式也是在excel中配置。如成功则说明用例通过，失败则说明用例执行失败。

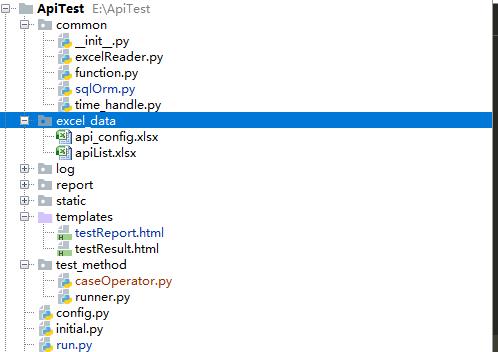
5. 将结果整合，并通过jinja2引擎渲染html模板，将用例的失败、成功的数据都渲染到html里，并生产html报告文件方便以后查看。

6. 接口返回本次用例运行结果，期间有计算用例运行时间以及统计运行个数，失败个数以及成功个数等。

7. 当用例较多的时候，程序会自动拆分用例，线程数也可配置。不过Python本身是有全局锁(GIL)，多线程运用不到CPU的多核优势。但是效率上也明显比单一的循环快。

8. 生成的html结果里， 有查看详情按钮，点击后会调用查看报告的接口，达到查看更多信息的目的。

1. 项目结构



1. common目录

1.1 excelReader.py

存放excel的相关操作， 如打开excel文件，释放excel文件，读取excel文件等。

1.2 function.py

存放获取函数名，封装了json里的dumps和loads方法。

1.3 sqlOrm.py

存放建立orm的相关类和orm的相关操作。

1.4 time\_handle.py

存放对时间、日期做对应处理的类和函数。

1. excel\_data

1.1 api\_config.xlsx

存放url的启用/禁止状态。

1.2 apiList.xlsx

存放接口测试用例。

1. log

3.1 ApiTest.log

存放日志。

1. report

4.1 xxxxxx.html

存放以时间命名的html报告。

1. static

flask自带目录，主要存放js脚本和css样式，这里我用来放bootstrap框架。

1. templates

6.1 testResult.html

存放测试结果模板，用数据渲染。

6.1 testReport.html

存放测试报告模板，用数据渲染。

1. test\_method

7.1 caseOperator.py

存放处理case报告，结果等的相关功能函数，如分割case，多线程功能等。

7.2 runner.py

存放最核心的request请求断言等功能函数。

1. config.py

存放配置文件，包括excel目录，工作目录，报告目录等等。

1. initial.py

服务初始化，防止嵌套引用。

1. run.py

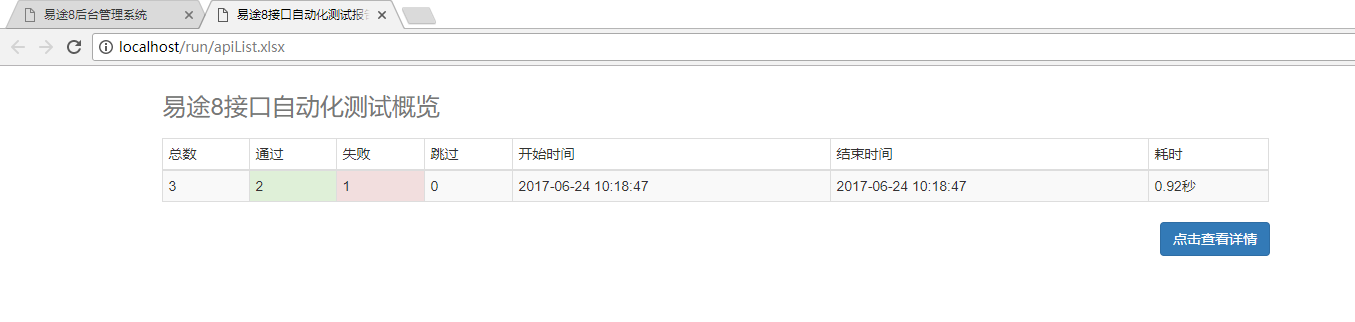
运行服务，通过命令python run.py即可启动，注册了3个接口。

部分代码展示：

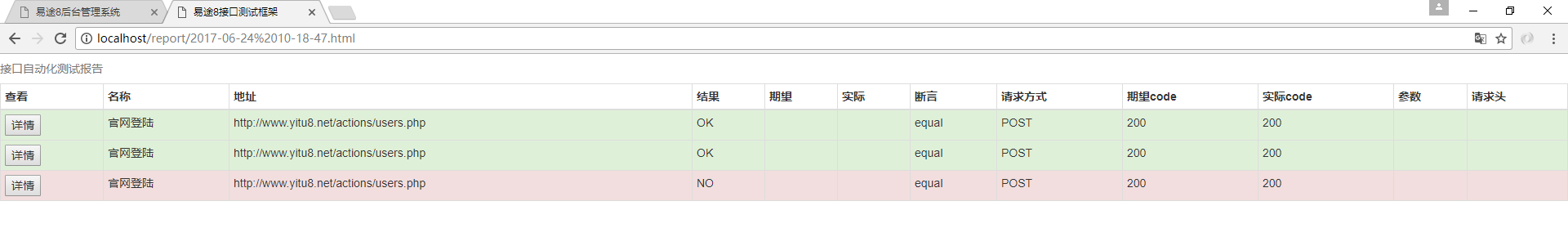


接口部分

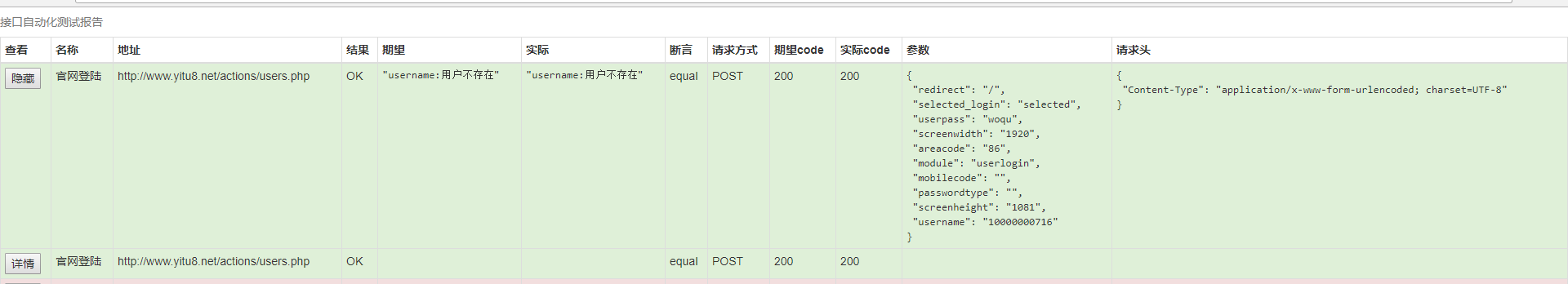
1. 效果展示
2. 测试结果



1. 报告



1. 报告详情页



1. 缺陷

既然用了web框架，其他的细节可以考虑美化一些，但是个人的前端技术比较不足，所以页面能省就省了。有几个小的值得改善的地方，一是用例执行的时候需要等待，不太友好。二是暂时只支持单个excel文件。三是断言方式不算完善，有待改善。四是数据库的相关处理还未用到。五是报告等还不太完美。六是文件目录结构设置有些不合理。