# 一、自测平台功能点：

## 1、测试环境自动部署：

* 代码同步
* 数据同步
* 配置同步
* 启动检查

## 2、冒烟测试

* 启动summary，检查是否启动成功；
* 500qps打压10分钟；
* 检查日志中标红标签是否闭合（MARKRED\_WRONG）；
* 检查日志中是否有空摘要情况；
* 检查压力日志中xml解析是否报错；

## 3、性能对比：

* 同步与disk数据同环的日志并生成词表；
* 启动query压力，模拟机器负载；
* 启动summary，检查是否启动成功，打压；
* 统计性能结果

## 4、diff对比：

* 准备词表，dump数量支持可配， type类型支持可配（一般来说需要去除type6请求）
* 启动summary，检查是否启动成功；
* 5qps打压，避免请求乱序；
* 自动提取日志结果；

## 5、对比title/summary的一致性波动率：

* 随机词表；
* 启动summary，检查是否启动成功；
* 5qps打压，避免请求乱序；
* 逐条对比title及summary，不一致的结果需要记录下来（入库或文件），同时统计不一致率

## 6、代码覆盖率

* Check diff\_cover工具（可以一次性先check好，记录地址，方便后续cp脚本）
* Check summary代码，同步 diff\_cover工具下的脚本；
* 使用gcov编译；
* 启动summary，检查是否启动成功
* 打压
* 生成并输出报告，报告放在nginx目录下，直接返回链接地址查看；

## 7、静态代码检查

* 使用cppcheck进行对test和base代码进行检查；
* 返回新增error和warning

## 8、任务和机器的分配、调度问题

# 二、概要设计

## 1、django框架：

主要是和mysql的交互，不涉及和其他处理脚本的交互

* 前端效果展示
* 用户填写信息的入库

## 2、任务和机器的分配、调度（testcache\_task\_dispatcher.py）：

该脚本常驻，一直循环执行，不会退出

* 读库，找到待处理的任务；
* 读库，找到空闲的机器；
* 将任务分配到该机器上执行，修改任务和机器的状态，入库；

## 3、触发任务执行（controller.py）

该脚本常驻，一直循环执行，不会退出

* 查看当前机器上是否有正在运行的任务，如果有，每5秒循环检测一下，直到任务执行完成；
* 查看当前机器上是否有等待运行的任务，如果有，调起执行任务的脚本处理任务；
* 查看当前机器上是否有待取消的任务，如果没有，循环检测；如果有，发信号终止任务（这里包括一些业务上的处理，如杀掉相关进程等），并将status置为canceled

## 4、执行任务

# 三、实现细节问题：

## 1、form表单提交

form表单提交数据的几种方式：<https://www.jianshu.com/p/8c947e7fee00>

Form 表单提交知识的总结：<https://juejin.im/post/5bc6aa8be51d450e9943804d>

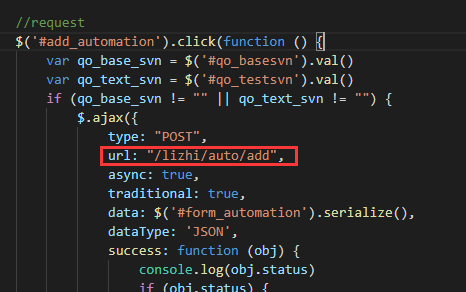
前提：要提交的内容在一个form里，

1、提交按钮设置为：type=submit，action=参数发送到的url接口



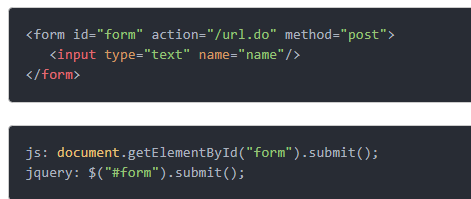
这种默认的提交方式，一般会进行页面的跳转，跳转到action中的url地址，cache自测平台用的是这种方式

2、ajax提交form表单



这种方式适用于不跳转新页面，而是在当前页面局部刷新，静军的平台一般是这种方式。

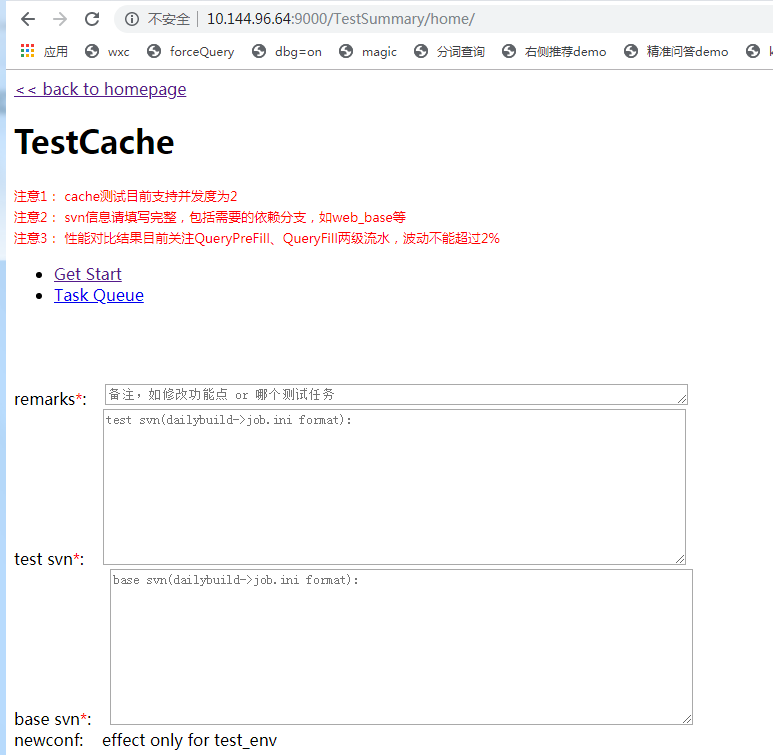
3、js提交form表单



js事件触发表单提交，通过button、链接等触发事件，js调用submit()方法提交表单数据，jquery通过submit()方法

## 2、样式不生效问题

在10.144.96.64上部署django2.1.1版本，创建project和app，然后将自测平台中的cache相关部分拷贝过来。启动后，发现页面没有样式，如下：



查看页面源码，发现相应的css和js确实访问不到。

注意：css放在head中，js放在body中

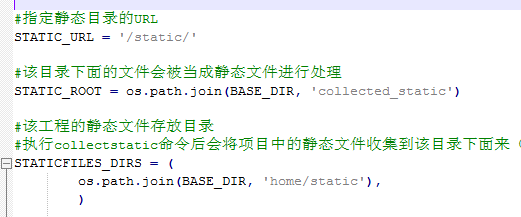


经查，发现是static（静态文件目录）设置的路径不对，有两种解决方式：

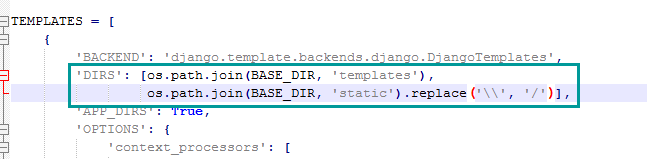
1、将js和css下载下来，放在一个nginx上，href中填写这个nginx上可访问的地址；

2、通过在django中设置static静态资源的路径来访问（自测平台和静军的平台都用这种方式）

在settings.py中添加如下内容：



发现样式还是不生效，还需要在settings.py的TEMPLATES中增加如下内容：



还是不生效，未解决，，，

## 3、forms表单

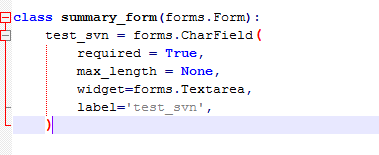
Forms表单的使用：<http://www.liujiangblog.com/course/django/152>

各字段介绍：<https://www.jianshu.com/p/c4f1c25feb1c>

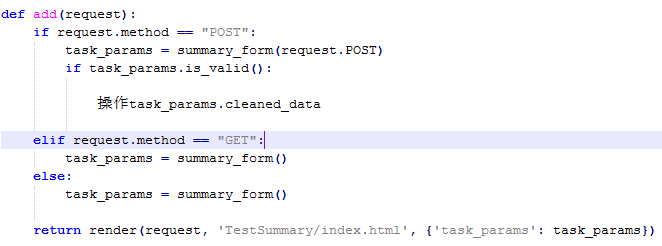
1）在当前app内新建一个forms.py文件；

2）导入forms模块，所有的表单类继承自forms.Form类

3）一个输入框对应一个字段，如下是test\_svn的定义：



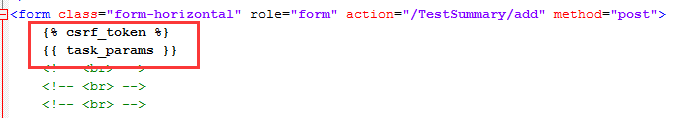
4）在视图函数中的使用，大体框架如下：



前提：在template中定义的method是POST

* 如果是POST请求，则使用summary\_form来解析该post请求，结果放入task\_params对象中；
* 使用is\_valid检查task\_params是否合法，这里主要是根据summary\_form中对各个输入字段的定义，检查输入是否合法；
* 如果合法，django会将所有的表单数据转存到cleaned\_data的属性中，其是一个字典，然后根据业务逻辑对其进行处理；
* 如果不合法，会直接跳到return语句，用解出来的参数渲染index.html页面，即返回一个包含先前数据的表单给前端页面，方便用户修改；
* 如果是非POST请求，返回空的表单，让用户可以填入数据；

5）在模板中的使用如下：

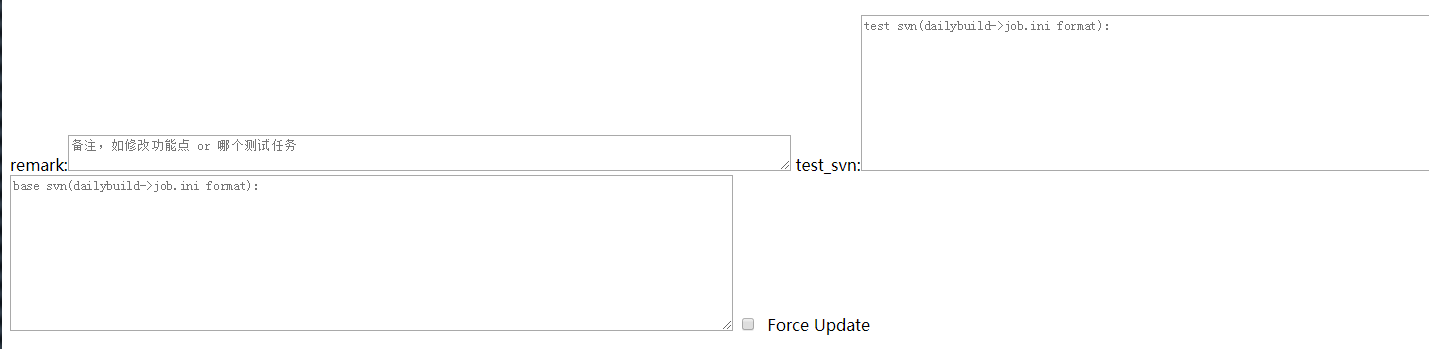


在form标签中，加入上面两项，如果不加csrf\_token，页面跳转会报错

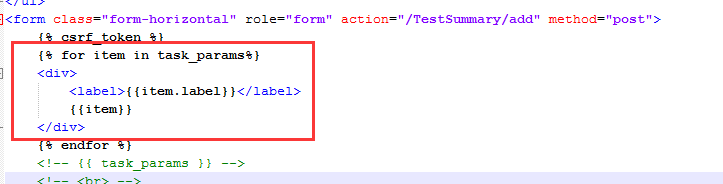
如上表示使用整个task\_params，如果只用其中某几项，可以采用task\_params.test\_svn

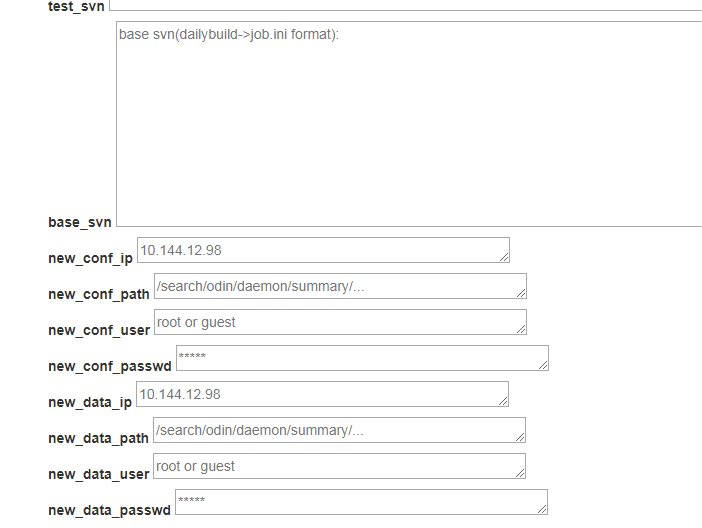
的方式。

改进：直接使用task\_params，django默认的使用table来部署元素的，会导致布局错乱



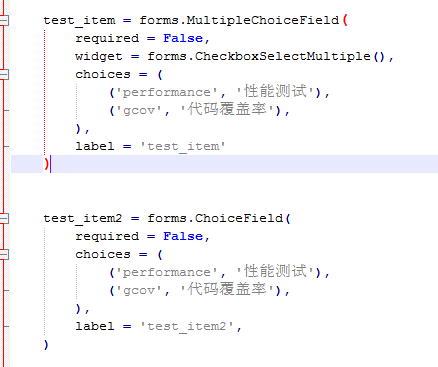
可以改为以下方式，每一项放在一个div中，这样就保证每一行是一项：

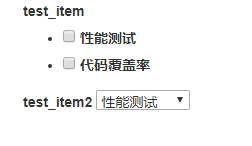




6）checkbox及select的使用

在forms.py中定义如下字段：test\_item是复选框，test\_item2是select





## 4、dispatcher问题

复用自测平台的dispatcher脚本来实现任务和机器分配及调度，其原理为：

1、读库，检查是否有待处理的任务；

2、读库，检查是否有空闲节点，具体是先查询status=1或2的runningIP，即正在使用的机器节点，然后遍历所有机器节点，看是否在该列表中，以确定是否有空闲节点；

3、将待处理任务分配给空闲节点，在库中修改status；

发现程序不能正常获得空闲节点，表现为：

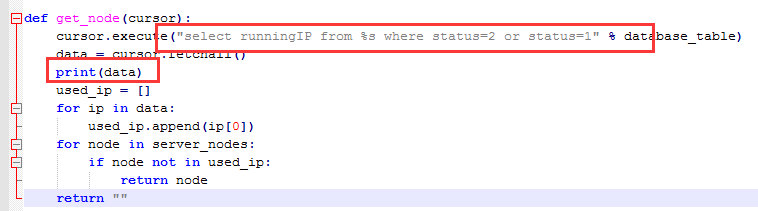
1）库里有5个任务，status=0（等待）

2）运行脚本，第一个任务的status=1（ready）

3）修改第一个任务的status=3，即该任务不再占用该机器

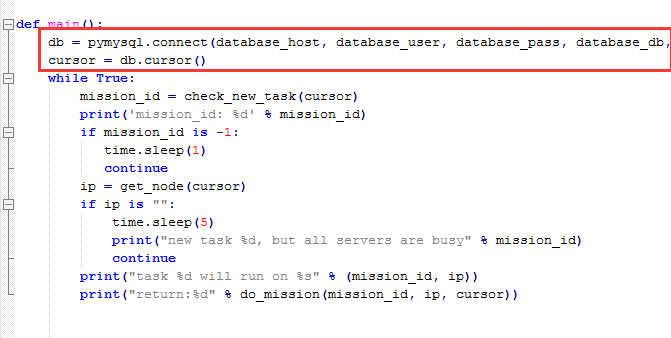
4）脚本不能正常的给下一个任务分配机器，一直显示机器被占用

打印日志发现，这里的data不为空，但执行对应的sql查库，内容为空。



怀疑是游标cursor没有更新的感觉，两种修改方法：

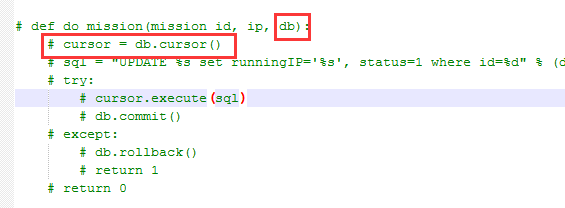
1）把连库操作放在while循环里，



2）发现check\_new\_task和get\_node函数，都是参数传的cursor，

而do\_mission函数，参数传入的是db，在函数内建的cursor，

即三个函数用的不是同一个cursor，



将do\_mission函数的入参也改成cursor，即保证三个函数用的是一个cursor，解决了问题。

3）几天后，发现2的修改也不生效了，把建立db和cursor放到了while循环里解决了问题，为了避免建立连接过多，每次还需要关闭连接，可能对db性能有一定影响。

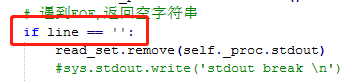
4）后来发现，db和cursor可以声明为全局变量，然后在每次sql操作完之后，都执行一下db.commit() 就可以解决问题了。理论上只有在修改数据库内容的时候才需要执行db.commit()，单纯的select查询是不需要提交的，但是在这个场景下提交一下可以解决不生效的问题。

## 5、asycommands.py问题

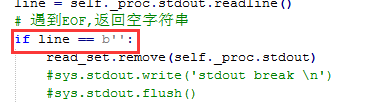
这个模块是良辉基于subprocess的一个封装，可以异步执行命令并返回命令输出的数据。从py2改为py3的过程中，遇到了一个问题：调用asycommands执行命令之后，如果超时时间还没到，则不会退出。

原因是在判断是否是EOF的时候，py2和py3不同：

Py2:



Py3:



## 6、pexpect问题

在同步test\_data和test\_conf时，使用的是scp，这里涉及到输入密码等交互操作，代码中用pexpect实现。参考testcache\_runner.py的用法，会有EOF的exception。最后是参考了之歌的autoqps\_runner.py：



1）在pexpect.spawn的时候try一下，而不是对整体try exception；

2）expect最后返回0表示匹配到了所需的关键字, 如果后面的匹配关键字是一个列表的话，就会返回一个数字表示匹配到了列表中第几个关键字，从 0 开始计算。关键字一旦匹配，就会返回0表示匹配成功，但是如果一直匹配不到呢？默认是会一直等下去，但是如果设置了 timeout 的话就会超时；

3）第一个if块是针对第一个expect的处理，执行scp命令后，无非有三种情况：

* 直接提示输入密码
* 提示输入yes/no
* 退出或者超时；

第一个if块正是对这三种情况的处理；

4）第二个if块是针对第一个if块中1、2两种情况expect的处理：

* 输入yes/no之后，预期会提示输入密码，所以当except\_result==0时，sendline密码，同时会再expect一下；
* 当except\_result==2时，表示用户名或者密码输入错误，会出现please try again；
* 当except\_result==3，表示子进程结束，对应的是第一个if块中，直接输入正确的密码，子进程执行结束；
* 当except\_result==4时，表示超时；

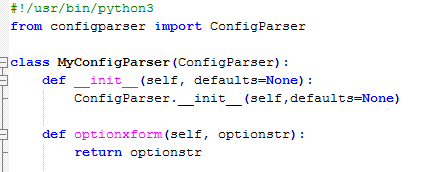
5）第三个if块是对第二个if块中expect的处理：输入密码后，如果except\_result==3，表示输入的密码正确，子进程执行结束；

6）这里要注意的是这种if判断的逻辑，每一个if块是对上一次expect的处理，不要在一个if块中处理所有可能的情况，那样逻辑比较混乱；

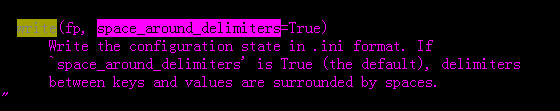
## 7、configparser使用

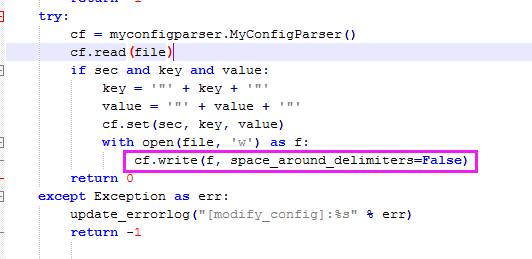
对summary进行稳定性测试时，需要修改备库、监听端口、缓存大小等配置，这里使用了configparser模块，遇到以下两个问题：

1）大小写问题：configparser处理后的字符串均为小写，可以通过重载optionxform函数解决：



2）修改完文件write之后，key和value的分隔符“=”，左右两侧新增了空格。看了configparser的源码，write函数有一个参数项space\_around\_delimiters，设置为False即可，默认是True。





## 8、try…except的使用

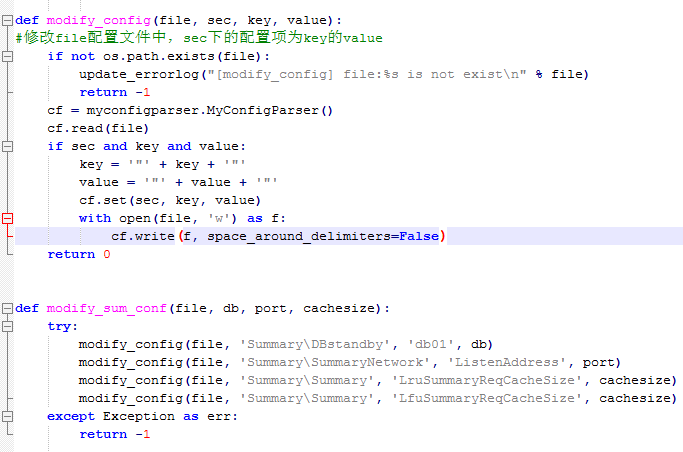
在修改summary配置文件的时候，涉及到两个函数：

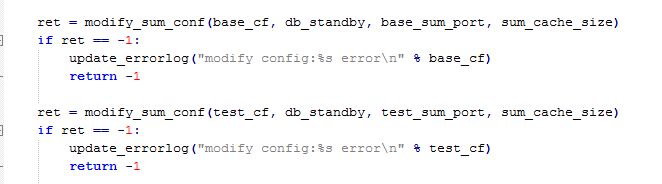
1）modify\_config：将file文件中的sec下的key配置项的值设置为value；

2）modify\_sum\_conf：传入具体的备库地址、端口、缓存大小的值，进行配置修改；

我们在modify\_config中不使用try…except，而是在调用它的函数modify\_sum\_conf中使用，这样如果modify\_config崩了，其异常可以被上层的modify\_sum\_conf捕获，其会return -1。在调用modify\_sum\_conf的地方，可以判断其返回值是否是-1并打印相关日志。

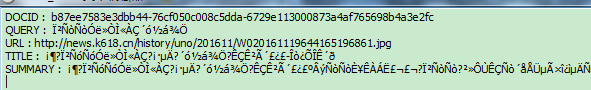
如果在modify\_config中使用了try…except，except的时候返回-1，那么在modify\_sum\_conf中就需要对每一次调用modify\_config来判断返回值是否是-1，比较麻烦。



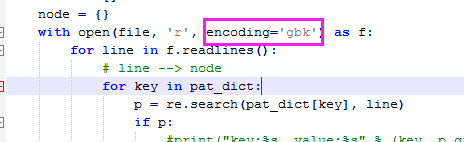


## 9、summary diff 结果入mysql库乱码

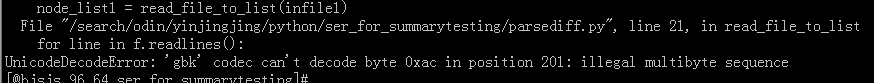
在heidiSQL中看到的数据如下，中文显示乱码：

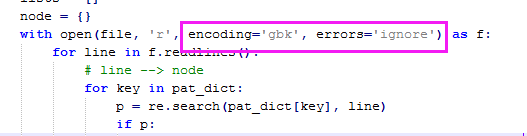


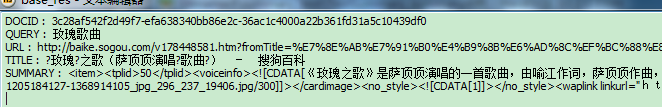
用notepad++查看网页summary的err日志，发现是gbk编码的，而我程序内部都是utf8编码的，所以需要在读入文件的时候，指定编码方式为gbk：



程序运行还是会报错，如下，需要加上errors=’ignore’来解决，库里的数据就正常了







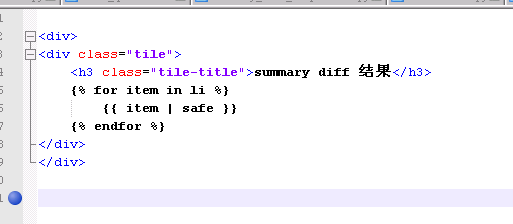
## 10、前端展现summary diff结果

1、在views.py中增加diff\_detail函数，在用filter函数将某个task对应的summarydiff结果从mysql中读出；

2、这里获取到的是summarydiff对象的列表，需要对列表中的元素进行遍历，要获取base\_res或test\_res字段，则使用obj.base\_res获得；

3、使用make\_tables函数，一个字符是一行，不符合需求，故使用make\_file函数；

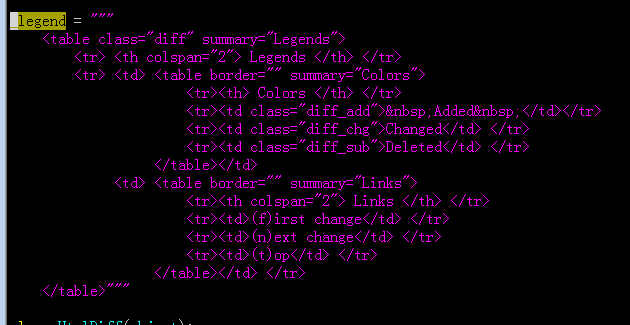
4、make\_file生成的结果就是html文本，且带了一定的样式，所以在diff\_detail.html中直接应用就可以，diff\_detail.html整个页面内容如下：



5、展现结果中有LengendS内容，从html页面源码中找到对应的内容，准备在生成diff结果的时候，把这部分结果替换为空，但是实际不生效，应该是多行文本替换不生效。



看了下difflib的源码，生成如上html片段的变量是\_legend，如下，在源码里将其定义为空即可：



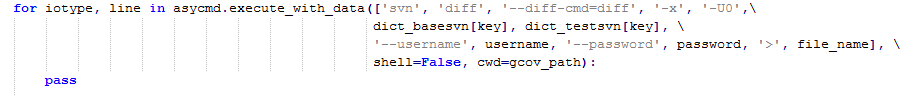
## 11、gcov功能中遇到的问题

1）os.popen()不是阻塞的，导致没拷贝完数据就执行编译，导致报错。如果想要阻塞的，使用subprocess，调用wait()；

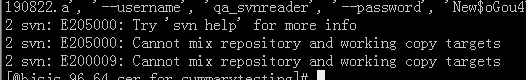
2）Asycommands是生成器，必须用for iotype, line in …的方式调用；

3）在生成svndiff报告时，遇到的问题及解决方法：

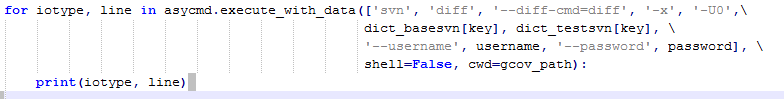
* 方法一：用Asycommands来实现，直接重定向到文件



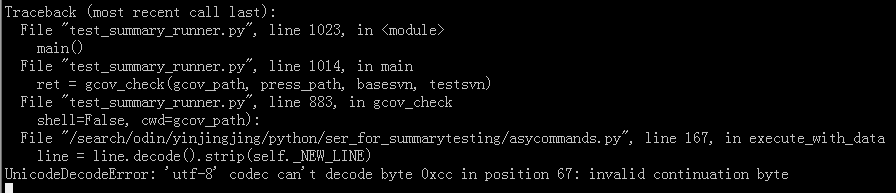
问题：有如下报错：



* 方法二：使用Asycommands来实现，不直接重定向到文件，可以考虑在print的时候定向

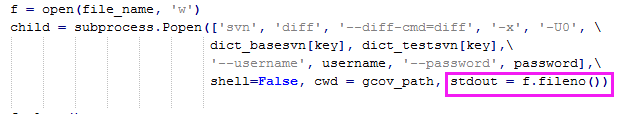


问题：有如下报错：



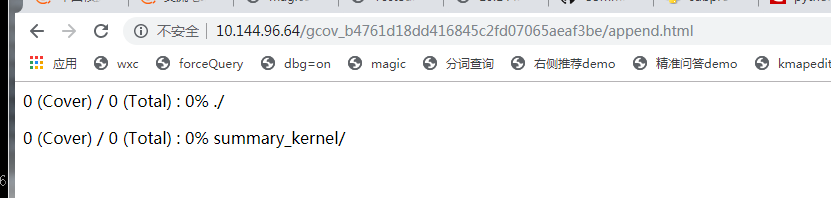
* 方法三：使用subprocess，但是如何将标准输出重定向的文件，这里花费的时间较多。

参考文章：<https://www.jianshu.com/p/8e582146bd4c>



注意stdout = f.fileno()，不少文章里都是直接传入打开的file对象f，但这样输出文件为空。这里需要传入f.fileno()，**fileno()** 方法返回一个整型的文件描述符(file descriptor FD 整型)，可用于底层操作系统的 I/O 操作。

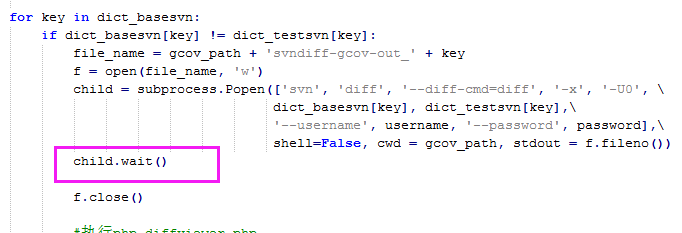
4）代码运行时，一直跑不出正确的结果，结果为：



后来单步运行，发现是在生成svndiff文件+执行php diffviewer.php时有问题，应该是svndiff文件还没有生成完全，就开始执行php diffviewer.php，所以生成的报告是空报告。

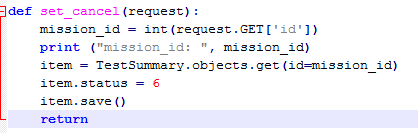
解决方法是在subprocess进行wait()，等待svndiff完全生成。

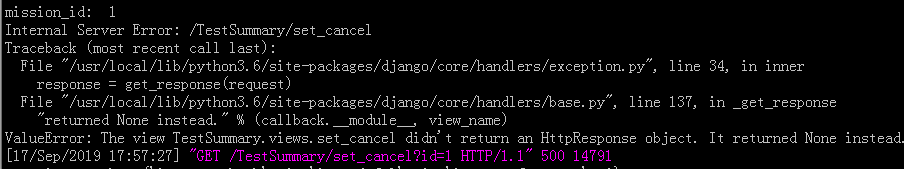
和os.popen()不是阻塞的是同一类问题。



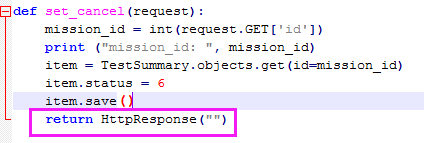
## 12、cancel功能

在cancel功能中，只需把对应任务的status值为canceling（status=6）即可，并不需要返回内容渲染页面，所以函数返回值为None。这样虽然对功能没有影响，但是django的日志中会有一个500的报错：





原因是cancel功能一个http交互的功能，它的入参是一个http请求，所以要求返回值不能为None。对于这种情况，我们可以返回一个空的HttpResponse：

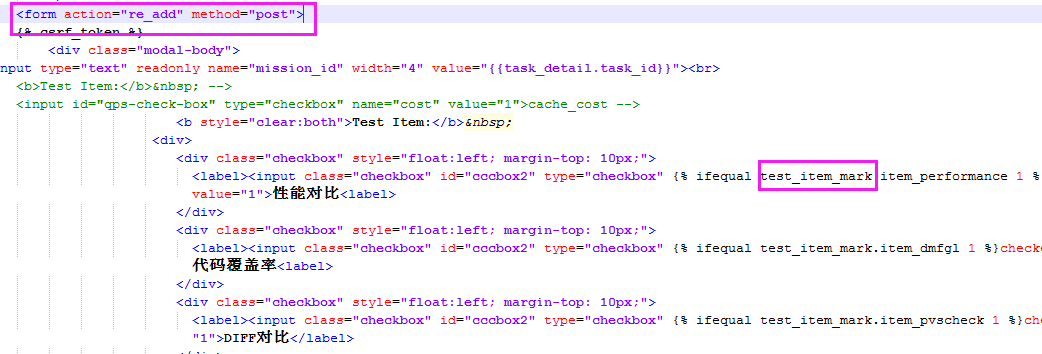


## 13、run again功能

1）当status != running，才显示run again按钮，点击后会弹出一个对话框；

2）对话框是一个form表单，提交的请求要求是post请求，这个表单是一直存在的，只是点击run again之后才会显示出来；

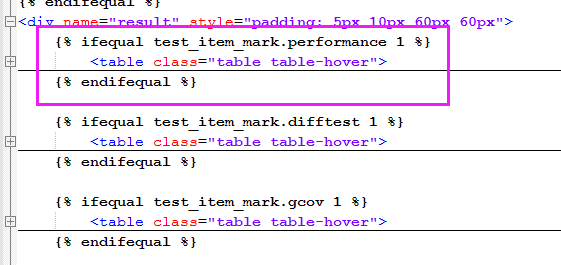
3）其中test\_item\_mark变量，表示的是创建任务时，选择的测试项目。这个变量在task\_detail函数中生成，用途是渲染如下表单中的测试项，表示该任务初始创建时，所选择的测试项目；





## 14、隐藏没有选中的测试项的内容

判断一下对应的测试项是否被选中，如果选中才展现



## 15、自动更新summary db和对应的词表

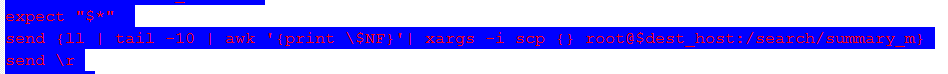
更新词表的时候，只想拷贝最近10小时的日志，由于涉及到的交互操作较多，用rsync不方便，故使用shell下的expect实现。

1）报错：/usr/bin/expect: No such file or directory

没有安装expect导致的，使用yum install安装

2）如果send的命令比较复杂，比如使用的awk语句，需要注意：

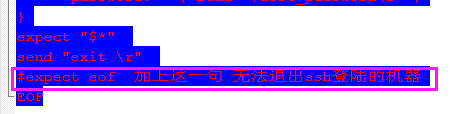
* Send语句用{}括起来；
* “$”符号需要转义，写为“\$”；
* 表示回车的操作“send \r”单起一行；



参考文献：<https://blog.csdn.net/su92chen/article/details/81095813>

3）拖完日志之后，进程停留在线上机器，无法退出。

发现注释掉expect eof后，可正常退出。

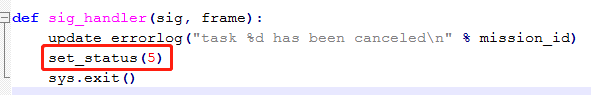


## 16、任务取消

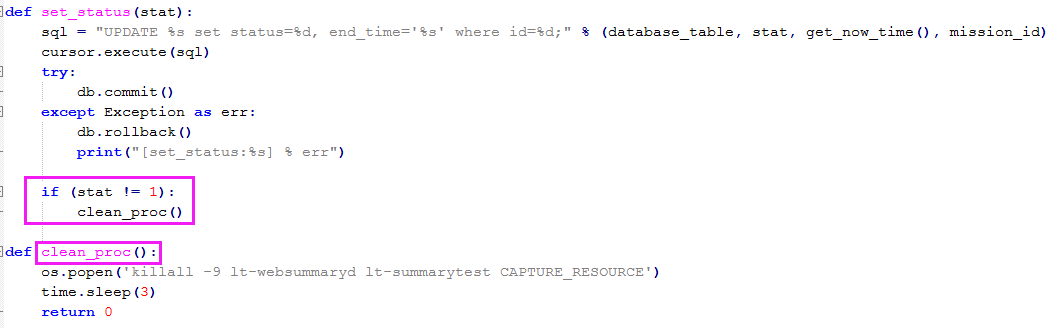
1）在平台上点击“cancle”按钮，这一步是将任务的状态设置为“待取消”（status=6）；

2）controllor.py脚本循环扫库，发现有待取消的任务，给该任务（subporcess生成的进程）发送signal信号（task\_list[cancel\_id].send\_signal(10)）；

3）在test\_summary\_runner.py中，当收到signal为10或15时，会调用sig\_handler处理该信号，即取消任务，并将状态置为“cancled”；

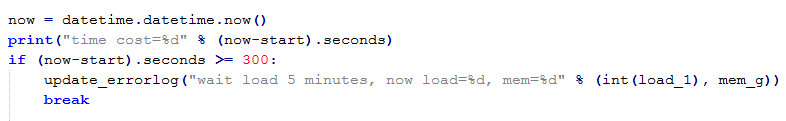


4）在set\_status中，对任务状态进行设置，除“已分配未运行”状态外，其他状态都会清理正在运行的进程，即调用clean\_proc；



## 17、wait\_to\_load无法退出

在一次测试中，遇到内存占用始终达不到50G的情况，导致wait\_to\_load函数一直在循环，无法退出。故增加逻辑：当wait\_to\_load运行达到5分钟，即使负载和内存占用没有达到要求，也退出该函数。



## 18、解决svn sw冲突问题

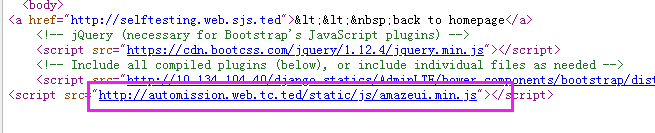
在搭建环境的过程中，有时候会遇到svn sw冲突问题，一般是手动选择tc（theirs-conflict）来解决，在自动化过程中，可以使用 --accept theirs-conflict 参数来解决。

例：svn sw --accept theirs-conflict $url

## 19、页面样式问题

开始时，任务创建页面样式有问题，有两个css和一个js没有加载成功，然后直接使用cache自测平台对应的css和js解决问题，但是页面样式还是有问题。





金铭说是缺少class="form-control"导致的，和cache自测平台对比，确实没有这个class。

Bootstrap渲染页面，是根据class来决定的，如果缺少class，bootstrap是无法渲染生效的。

在froms.py中可以直接添加class



## 20、使用域名访问django

setting.py中，在host的设置中，增加域名即可。

