# 1、读任务表，检查是否有master更新 & need\_check=1

如果没有，循环检测；

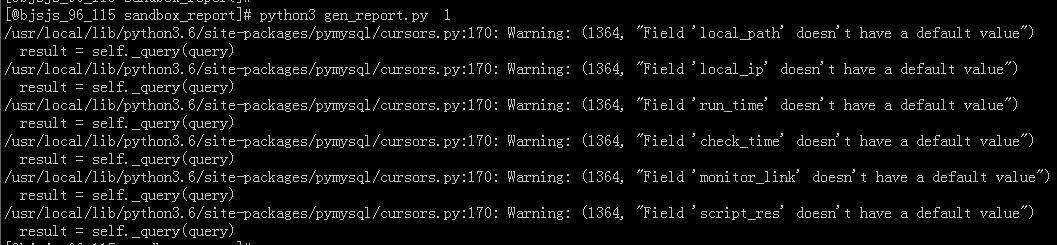
如果有，用pid + run\_tim + check\_time取接口数据，查看是否运行正常，并对is\_check\_ok、fail\_reason和need\_rollback置位；

# 2、读任务表，检查是否有需要发报告的任务

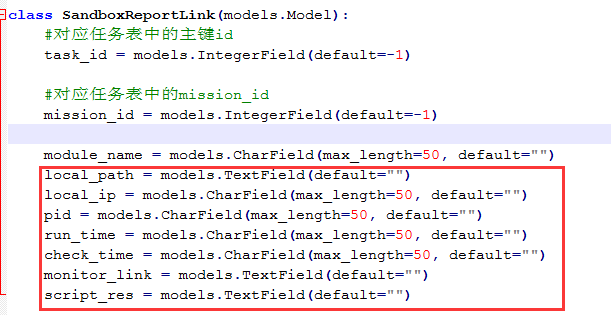
如果没有，循环检测；

如果有，在reslut表里插入数据，根据从各接口拿到的数据做入库；

# 3、从费晓的任务表中拉取要出报告的数据，插入到报告表中时，会有如下报错，说如下字段没有默认值。



但是在models.py中是指定了默认值的：



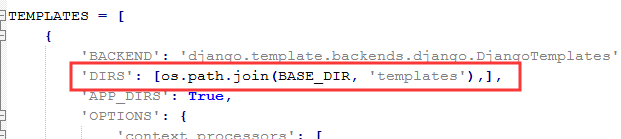
问了金铭，需要加上blank=True, null=True

default=””就相当于是blank=True， 表示的是内容为空

null=True相当于是这一列数据可以为空

# 4、写前端django页面时，访问<http://10.144.96.115:5555/sandbox_report>，报TemplateDoesNotExist错误，解决：

1）在总setting.py中的TEMPLATES下，增加如下内容：



2）在sandbox\_report APP下，增加templates（注意拼写，不要丢掉s）目录，里面放该app需要加载的html页面

# 5、在django中，点击跳转到性能检测平台的时候，发现跳转有误，原因是：

原始数据：

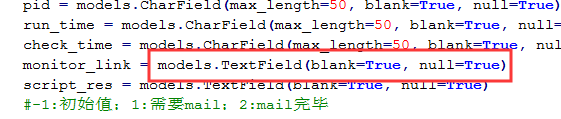
http://bzw.sogou.gd.ted/jarvan?pid=2&pn=网页前台测试组&ip=quickpcsh02.web.sjs.ted&ana=7,8,4,9&nt=&purpose=one&viewId=1537&monitorrids=678,679,680,681&pf=3&tid=619&starttime=2019-11-19 20:20:00&endtime=2019-11-20 00:20:00#show.jarvan.monitor.logsMonitor

如上数据中，startime和endtime中有空格，如果不进行urlencode的话，在href跳转的时候，空格后面的内容就丢失了，所以需要进行urlencode，使用金铭的urlhandle.py。

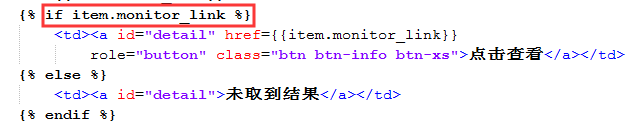
# 6、当获取不到结果链接时的处理：

（<http://www.liujiangblog.com/course/django/146>）

当从接口获取不到结果链接时，因为在models里设置了允许为空，所以task.monitor\_link值为“None”



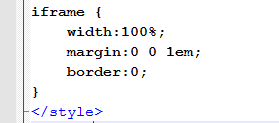
在djiango前端，需要判断task.monitor\_link是否为空，不为空可以跳转，为空显示“未获取到结果”。判断是否为空用如下方法：



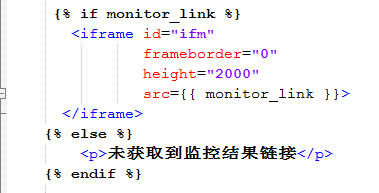
# 7、使用iframe嵌入性能监控结果

性能测试结果的展现，不是采用点击跳转的方案，而是采用嵌入到当前页面的方式，这里就用到iframe：

1）在html（per\_mission\_detail.html）的head中加入iframe的css：



2）在html的body中加入iframe：



# 8、使用nginx+uwsig实现django多进程访问

(<https://segmentfault.com/a/1190000014361352>)

Nginx：是一个高性能的HTTP和反向代理服务器；

Django：是一个开放源代码的Web应用框架；

WSGI：是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。用于连接Web服务器和Web应用框架；

**工作原理：**

1. 一般来说，客户端发起的http/https请求首先到达了我们的Web服务器（在这里我们使用Nginx作为我们的服务器）。

2. Web服务器收到请求后，对请求进行解析处理（一般来说如果是静态请求，直接返回静态页面（纯HTML页面）），然后将请求发送给uWSGI。

3. uWSGI对来自Nginx的请求进行处理，将处理后的结果发送给Web应用框架（在这里我们使用Django）。

4. Web应用框架收到请求之后，Web应用对请求进行处理（例如，Django通过某个函数进行处理），将结果返回给uWSGI。

5. uWSGI收到Web应用框架的处理结果后，对该结果进行处理打包，发送给Web服务器。

6. Web服务器将收到的结果返回给客户端。

这三者之间信息的传递涉及到数据格式以及协议的转换。

1）pip安装uwsgi

命令：pip3 install uwsgi

2）测试uwsgi运行状态

新建test.py文件：

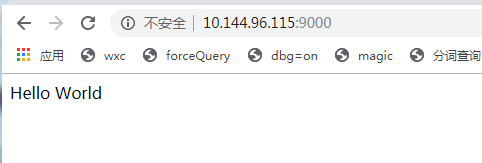
**def** **application**(env, start\_response):

start\_response('200 OK', [('Content-Type','text/html')])

**return** [b"Hello World"]

运行：uwsgi --http :8000 --wsgi-file test.py

访问页面：<http://10.144.96.115:9000/>，如下结果表示uwsgi运行正常



3）在django的最外层目录（和manage.py同级目录），新建uwsgi.ini文件，内容如下（标蓝的是需要修改的内容）：

[uwsgi]

socket = 127.0.0.1:8032

master = true

pidfile = uwsgi.pid

#启动多少个django进程

workers = 8

reload-mercy = 30

# the base directory (full path)

chdir = /search/odin/yinjingjing/python/sandbox\_report/django\_sandbox

wsgi-file = django\_sandbox/wsgi.py

daemonize = uwsgi.log

buffer-size = 32768

post-buffering = 32768

desable-logging = true

enable-threads = true

harakiri = 30

disable-logging = true

#plugin = python

stats = 127.0.0.1:8033

stats-http = true

procname = django\_sandbox

procname-master = main\_master

4）启动uwsgi，使用run脚本，内容如下：

启动后，top可以看到启动了8个django\_sandbox



#! /usr/bin/env bash

export PYTHONUNBUFFERED=1

NAME="uwsgi.ini"

process="main\_[pm]\*"

stop\_uwsgi(){

if [ ! -n "$NAME" ];then

echo "no arguments"

exit;

fi

echo $process

ID=`ps -ef | grep "$process" | grep -v "$0" | grep -v "grep" | awk '{print $2}'`

echo $ID

echo "################################################"

for id in $ID

do

kill -9 $id

echo "kill $id"

done

echo "################################################"

}

if [ $# -ne 1 ]

then

echo "parameter err. must only 1."

exit

fi

if [ "$1" == "start" ]

then

uwsgi "$NAME"

#uwsgi -x "$NAME"

exit

fi

if [ "$1" == "stop" ]

then

stop\_uwsgi

exit

fi

if [ "$1" == "restart" ]

then

kill -9 `ps aux|grep vrTool|grep -v grep|awk '{print $2}'`

sleep 0.5

uwsgi "$NAME"

#uwsgi -x "$NAME"

exit

fi

if [ "$1" == "reload" ]

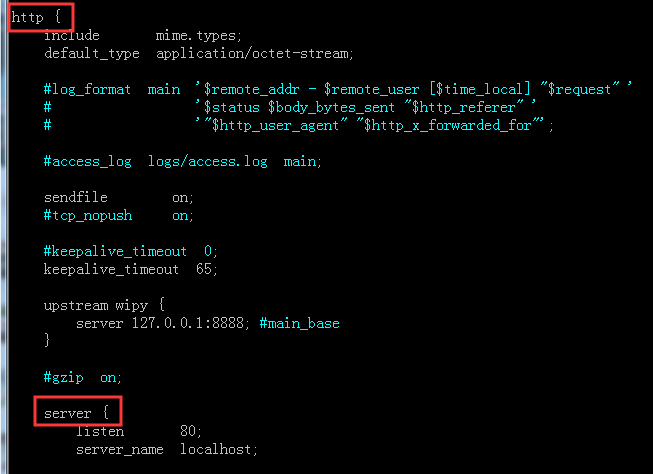
then

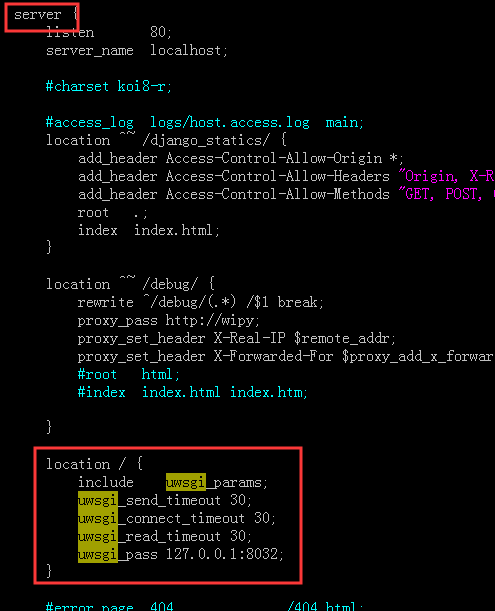
uwsgi --reload uwsgi.pid

fi

5）准备nginx，并启动nginx

6）在nginx配置文件/search/odin/daemon/nginx/conf/nginx.conf中，在http/server中增加关于uwsgi的配置：





其中，uwsgi\_params是一个文件，和nginx.conf在同级目录下，内容如下：

uwsgi\_param QUERY\_STRING $query\_string;

uwsgi\_param REQUEST\_METHOD $request\_method;

uwsgi\_param CONTENT\_TYPE $content\_type;

uwsgi\_param CONTENT\_LENGTH $content\_length;

uwsgi\_param REQUEST\_URI $request\_uri;

uwsgi\_param PATH\_INFO $document\_uri;

uwsgi\_param DOCUMENT\_ROOT $document\_root;

uwsgi\_param SERVER\_PROTOCOL $server\_protocol;

uwsgi\_param REQUEST\_SCHEME $scheme;

uwsgi\_param HTTPS $https if\_not\_empty;

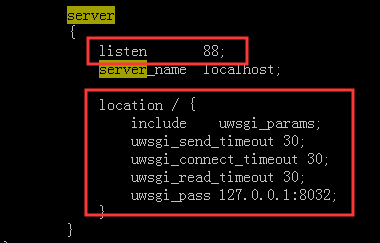
uwsgi\_param REMOTE\_ADDR $remote\_addr;

uwsgi\_param REMOTE\_PORT $remote\_port;

uwsgi\_param SERVER\_PORT $server\_port;

uwsgi\_param SERVER\_NAME $server\_name;

注意：在server中，需要指定监听端口和server\_name。因为10.144.96.115上的nginx配置用的是wap前端的配置，所以已经占用了80端口。所以需要新建一个server，内容如下：



7）访问<http://10.144.96.115:88/sandbox_report/mission_list/>正常，这个访问就是通过nginx、uwsgi打到django的。

这里不需要用manage.py启动django了，通过uwsgi已经启动了8个django进程。通过前端nginx10.144.96.115:88访问的请求，都打到了这8个django上。但是如果调试的话，还是通过manage.py启动的django好调试。

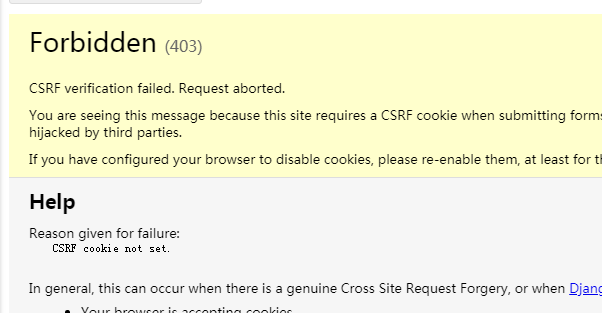
# 9、从postman发post请求，提示403错误

<https://blog.csdn.net/liweiblog/article/details/77800446>

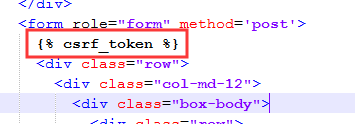
在views.py中，兼容了对POST请求的处理，但是从postman发POST请求的时候，会有403错误。增加@csrf\_exempt装饰器可以解决该问题。

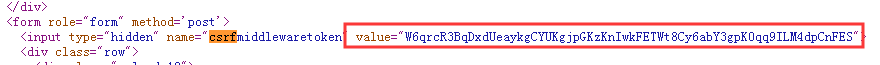
csrf\_exempt：取消csrftoken验证

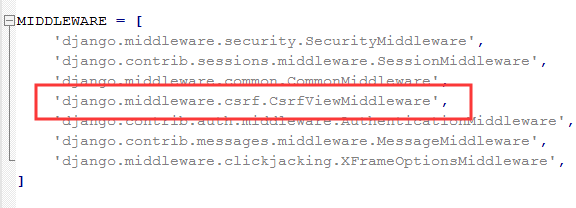




模板template中的{% csrf\_token %}，是用来生成csrf\_token，服务器端进行验证，服务器端验证的工作通过'django.middleware.csrf.CsrfViewMiddleware'这个中间层来完成。在setting.py中进行设置。







在views.py中，如果一个接口要对外开放，需要用@csrf\_exempt取消csrftoken验证。