

简介

- 孙海龙，时光网运维DBA主管，ORACLE DBA后转MYSQL DBA，目前主要工作运维MYSQL/MONGODB/REDIS/ORACLE数据库，参与自动化平台建设
- 主题：一个运维DBA对类的理解和使用
- 分享内容：
 - 1、我从学习类到使用类的历程；
 - 2、从一个简单的例子谈谈类的设计；
 - 3、从MONGODB上线平台的设计谈谈类再工作中的应用；
- 目标：

以前工作没有使用过类的同学通过今天的分享开始在工作中开始尝试使用类

简介

你该怎么听，听什么？

- 没有使用类经验的小白同学？重点听我怎么从函数演化到类！
- 用过一些不太熟练，也写过python脚本，没有过专门的代码开发，重点听我mongo项目的需求分析和设计阶段！
- 希望重点了解mongo部分的，可以重点听听我对这个项目的需求分析！
- 有经验的老鸟多给我批评指正，在下不胜感激！

目录

- 学习历程
- 一个简单的例子谈类设计
- MONGODB上线平台谈类使用

**声明：今天说的类和面向对象基本是
一码事！**

你是否也曾从入门到放弃，然后就“
望类生畏”过呢？

目标

- 不再害怕类
- 在自己的项目中开始有意识的尝试使用类
- 遇到代码组织问题时可以想到类

学习历程

- 从入门到放弃阶段；
- 跟开发同学讨论、学习阶段；
- 跟张老师学写狮子王；
- 自己编写类阶段；

类是什么？

- 函数是什么？
- 类是什么？
- 一定要有类吗？

类是什么？

- 函数、类都是一种组织代码的形式，都是一种编程思想！
- 谁需要编程思想，谁不需要编程思想
 - 我觉得写简单的脚本不需要编程思想，shell的简单罗列也可以解决问题嘛！
 - 但是代码多了你可能需要使用函数了，如果逻辑复杂你可能需要使用类了！
- 最直接的好处就是代码量少了，好维护了
 - 判断代码好坏的标准，写完后过一个月自己看下，感觉如何？！
- 我近期优化了一个view从200多行到100行左右

人类简史

需求：实现人

- 两只眼睛、两条腿、心会跳、大脑会思考
- 能吃、能睡、会用火、玩游戏
- 能工作、能说话、能学习、跑、会控制别人的心脏跳动频率
- 呼吸....
- （要是可以你是不是还想改性别啊。。。）

class Humankind(object):

```
def __init__(self):    两只眼睛、两条腿
def 跑(): ....
def 说话(): .....
def 心跳(): .....
def 玩(): ...
def 吃():....
def 睡(): ...
def 呼吸(): ...
def 控制心跳(): ...
def 工作(): ...
.....
```

猪类简史

需求：猪

- 两只眼睛、两条腿、心会跳、大脑会思考
- 能吃、能睡、
- 能说话、能学习、跑
- 呼吸....

class Pig(object):

```
    def __init__(self):      两只眼睛、两条腿
    def 跑(): ....
    def 说话(): .....
    def 心跳(): .....
    def 玩(): ...
    def 吃():....
    def 睡(): ...
    def 呼吸(): ...
    .....
```

猪的没落

需求：实现人养猪

- 两只眼睛、两条腿、心会跳、大脑会思考
- 能吃、能睡、
- 能工作、能说话、能学习、跑
- 呼吸....

```
class Humankind(object):  
    def 养猪(): ...
```

```
class Pig(object): ...  
    def __init__(): 体重
```

现在的问题是，人和猪的区别不大，都会吃喝睡、心会跳、嘴会说，这些代码重复了两次，有没有什么好办法呢？ 抽象父类，以后再复杂抽象祖先类好了！

未来简史

继续抽象

Class 动物(object): ...

Class 哺乳动物(动物): ...

Class 智人(哺乳动物): ...

Class 猪 (哺乳动物) : ...

Class 智神 (智人) : ...

这样就基本就差不多了！

遥控心跳的问题

- 还记得人有个方法叫：会控制别人的心脏跳动频率

```
class Humankind(object):
```

```
    ...  
    def 控制心率(): ....  
    ...
```

其实我不需要这个方法，只要心跳方法有个参数可以控制频率就可以了！
这样我就能控制别人的心脏了，有意思吧！！！看谁不爽让他快跳几下....
但是我们的代码这样玩就不安全了，心脏是个内部的方法，外部用不到，

我们这个方法是内部使用的不想别人随便可以用，所以我们用下**私有方法、私有属性** python中的私有属性也是可以被外部调用的就像医生一样

不好的代码

```
def get_metadata(hosttag, flag, tname=''):
    dbname = Db_name.objects.get(dbtag=hosttag).dbname
    #get table list
    if flag ==1:
        if len(tname)>0:
            sql = "select TABLE_NAME, TABLE_TYPE, ENGINE, TABLE_COLLATION, TABLE_COMMENT from information_schema.tables where table_schema='"+dbname+"' and TABLE_NAME like '%" + tname + "%'"
        else:
            sql = "select TABLE_NAME, TABLE_TYPE, ENGINE, TABLE_COLLATION, TABLE_COMMENT from information_schema.tables where table_schema='"+dbname+"'"
        results, col, tar_dbname = get_data(hosttag, sql)
        return results, col, tar_dbname
    #get column list
    elif flag==2:
        sql = "SELECT ORDINAL_POSITION AS POS, COLUMN_NAME, COLUMN_TYPE, COLUMN_DEFAULT, IS_NULLABLE, CHARACTER_SET_NAME, COLLATION_NAME, COLUMN_KEY, EXTRA, COLUMN_COMMENT FROM information_schema.columns where table_schema='"+dbname+"' and COLUMN_NAME like '%" + tname + "%'"
        results, col, tar_dbname = get_data(hosttag, sql)
        return results, col, tar_dbname
    #get indexes list
    elif flag==3:
        sql = "SELECT INDEX_NAME, NON_UNIQUE, SEQ_IN_INDEX, COLUMN_NAME, COLLATION, CARDINALITY, SUB_PART, PACKED, NULLABLE, INDEX_TYPE, COMMENT, INDEX_COMMENT FROM information_schema.statistics where table_schema='"+dbname+"' and INDEX_NAME like '%" + tname + "%'"
        results, col, tar_dbname = get_data(hosttag, sql)
        return results, col, tar_dbname
    #table details
    elif flag == 4:
        sql = "select * from information_schema.tables where TABLE_SCHEMA='"+dbname+"' and TABLE_NAME='"+tname+"'"
        results, col, tar_dbname = get_data(hosttag, sql)
        return results, col, tar_dbname
    elif flag == 5:
        sql = "show create table " + tname
        results, col, tar_dbname = get_data(hosttag, sql)
```

又一段

```
def get_his_meta(dbtag, flag):
    if flag == 1:
        if dbtag == 'all':
            sql = "select * from mon_tbsize order by `TOTAL(M)` desc limit 50"
        else:
            sql = "select * from mon_tbsize where DBTAG='" + dbtag + "' order by `TOTAL(M)` desc "
    elif flag == 2:
        if dbtag == 'all':
            sql = "select * from mon_autoinc_status order by AUTO_INCREMENT/MAX_VALUE desc limit 20"
        else:
            sql = "select * from mon_autoinc_status where DBTAG='" + dbtag + "' order by AUTO_INCREMENT/MAX_VALUE desc limit 20"

    elif flag == 3:
        if dbtag == 'all':
            sql = "SELECT * FROM (select a.DBTAG, a.TABLE_SCHEMA, \
a.TABLE_NAME, a.`TOTAL(M)` - b.`TOTAL(M)` AS 'inc_size(M)', \
    (UNIX_TIMESTAMP(a.update_time) - UNIX_TIMESTAMP(b.update_time))/3600 as 'DIF(h)', \
    a.update_time as 'LAST_CHECKTIME' from\
mon_tbsize a join mon_tbsize_last b using (DBTAG, TABLE_NAME)) B order by 4 desc limit 20;"
        else:
            sql = "SELECT * FROM (select a.DBTAG, a.TABLE_SCHEMA, \
a.TABLE_NAME, a.`TOTAL(M)` - b.`TOTAL(M)` AS 'inc_size(M)', \
    (UNIX_TIMESTAMP(a.update_time) - UNIX_TIMESTAMP(b.update_time))/3600 as 'DIF(h)', \
    a.update_time as 'LAST_CHECKTIME' from\
mon_tbsize a join mon_tbsize_last b using (DBTAG, TABLE_NAME) where a.DBTAG='" + dbtag + "' ) B order by 4 desc limit 20;"

    elif flag == 4:
        #top 10 DBsize
        sql = "select DBTAG, sum(`TOTAL(M)`) as 'TOTAL(M)', sum(`DATA(M)`) as 'DATA(M)' \
, sum(`INDEX(M)`) as 'INDEX(M)' from mon_tbsize group by DBTAG order by 2 desc limit 10 ;"

    elif flag == 5:
```


MONGO上线平台的需求

- 1、开发人员可以提交语句
- 2、提交时要做一定的校验
- 3、运维人员可以执行语句
- 4、执行前要备份像inception那样
- 5、备份数据可恢复

页面原型

上线申请

请选择--

请选择环境--

db.test.insert({"name":"haha"})

业务场景描述

提交审核

MongoDB上线申请列表

序号	申请人	数据库	SQL语句	申请时间	状态	操作
1	孙海龙	test	db.test.insert()	2017-04-01 18:50:42	审核通过	执行 删除
2	孙海龙	test	db.test.insert()	2017-04-01 18:50:42	审核通过	执行 删除

一步步设计

- 需求分析
- 最核心的操作是：执行mongo语句；
- 好办，封装一个mongo执行语句的类（Mymongo），调用个execute方法即可！
- 执行语句前要做数据备份；
- 有点麻烦，首先至少我需要生产备份语句，调用Mymongo执行备份
- 但是注意备份库和执行语句的库不一致；至少两个Mymongo实例吧！！
- 分析上线操作insert、update、remove、ensureIndex；
- Insert -> none **update->insert** **remove->insert** ensureIndex -> none
- 可能需要个反向操作的字典保存映射关系
- 还有最麻烦的是处理mongo语句了，需要正则表达式。。。

一步步设计

- 需求分析
- **需要支持的语句:**
 - `db.menu.update({"_id":1}, {$set: {'leaflds': [NumberInt(121)]}});`
`db.menu.remove({"_id":119});`
`db.getCollection('ID').insert({"_id" : NumberLong(100000),"sn" : NumberLong(100000)})`
- **解析语句**
 - 需要把`db.getCollection('ID').insert({"_id" : NumberLong(100000),"sn" : NumberLong(100000)})` 解析为:
 - `ID`、`insert`、`({"_id" : NumberLong(100000),"sn" : NumberLong(100000)})`

一步步设计

- **流程图**
 - VISIO
 - 参考：流程分析文档
- **ER图**
 - POWER DESINGER中画一下
- **开始编码**

一步步设计

- **编码**

- 一种情况：不知道怎么拆，怎么抽象，索性都写到一个函数里！
- 随着编码的进行有些函数就可以独立出来了
- 比如我们可能需要多次用到连接mongo数据库，不管你增删改查，备份都需要连接数据库，是不是我要写个mongo的封装工具类呢？毋庸置疑肯定是要的！
- 你可能在实现查询影响的行数、备份功能中需要对mongo语句进行多次解析，或用到其中的数据，拆函数吧！
- 逐渐查分你可能发现有一堆的函数很类似，还有函数间的调用，管理很麻烦，这时把他们放到一个类中是合适的！！
- 现在我们需要两个工具类，mongo连接工具类，mongo语句处理工具类
- **问题：不调整或者说不抽象我就一个函数行不行？**

一步步设计

- 编码
- 另一种情况：大致知道拆分成至少两个类：

```
class Mymongoconn:
```

```
    def __init__(self): ...
```

```
    def execute(self, sqltext):...      # 关键  执行传入的语句
```

```
class Mongo_cmd_handler:
```

```
    """
```

```
    主要处理前端输入的mongo命令
```

```
    """
```

```
    def __split_cmd_str(self): ....
```

```
    # 关键，给定语句，可以解析出想要的内容
```

```
    def backup_data(self): ....
```

```
    # 关键，执行备份
```

一步步设计

两个类:

```
class Mymongoconn:
```

```
    def __init__(self): ...
```

```
    def getConnection(self):....    # 辅助__init__
```

```
    def execute(self, sqltext):...    # 关键
```

```
class Mongo_cmd_handler:
```

```
    """
```

```
    主要处理前端输入的mongo命令
```

```
    """
```

```
    def __split_cmd_str(self): ....    # 关键，给定语句，可以解析出想要的内容
```

```
    # 这个函数解析出的内容是不想让外面访问，所以私有化了，但是外面需要查询的还是要给的，怎么给呢？
```

```
    def backup_data(self): ....    # 关键，执行备份
```

```
    def get_query_sql(self): ...    # 查询出来备份，配合备份的方法
```


一步步设计

两个类:

```
class Mymongoconn:
```

```
.....
```

```
class Mongo_cmd_handler:
```

```
"""
```

```
主要处理前端输入的mongo命令
```

```
"""
```

```
def __split_cmd_str(self): .... # 关键，给定语句，可以解析出想要的内容
```

```
    # 这个函数解析出的内容是不想让外面访问，所以私有化了，但是外面需要查询的还是要给的，怎么给呢？ 写个方法return出来如何？！
```

```
def backup_data(self): .... # 关键，执行备份
```

```
def get_query_sql(self): ... # 查询出来备份，配合备份的方法
```

```
def get_collection_name(self): .... # 给外面查询私有属性用的
```

一步步设计

两个类:

```
class Mymongoconn:
```

```
.....
```

```
class Mongo_cmd_handler:
```

```
def __split_cmd_str(self): .... # 关键，给定语句，可以解析出想要的内容
```

这个函数解析出的内容是不想让外面访问，所以私有化了，但是外面需要查询的还是要给的，怎么给呢？ 写个方法return出来如何？！

```
def backup_data(self): .... # 关键，执行备份
```

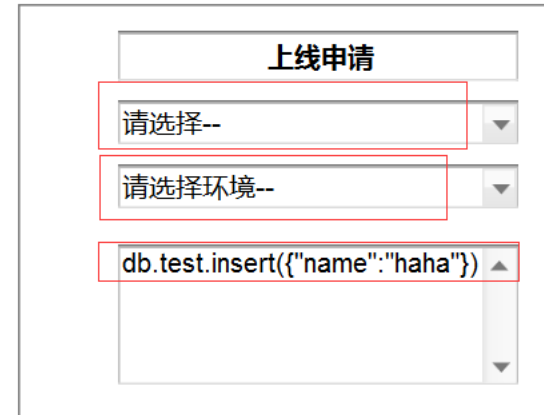
```
def get_query_sql(self): ... # 查询出来备份，配合备份的方法
```

```
def get_collection_name(self): .... # 给外面查询私有属性用的
```

代码写的过程中发现我需要校验用户输入的数据是否一致，于是增加了。。

```
def inputcmd_selectcmd_diff(self): ...
```

校验红框选择的数据和输入的数据是否一致



封装的一个MONGO工具类

```
class Mymongo:

    def __init__(self, DB_HOST='192.168.51.123', DB_PORT=123456, DB_USER='', DB_PWD='', DB_NAME=''):

        self.DB_HOST = DB_HOST
        self.DB_PORT = DB_PORT
        self.DB_USER = DB_USER
        self.DB_PWD = DB_PWD
        self.DB_NAME = DB_NAME

        self.conn = self.getConnection()

    def getConnection(self):

        host=self.DB_HOST
        port=self.DB_PORT
        user=self.DB_USER
        passwd=self.DB_PWD
        dbname=self.DB_NAME
        client = pymongo.MongoClient( host, port )

        if user and passwd:

            if dbname:
                # 判断参数中是否配置了数据库名称
```

Mongo_cmd_handler类

```
class Mongo_cmd_handler:
```

```
    """
```

```
    主要处理前端输入的mongo命令
```

```
    """
```

```
SUPPORT_OPER_LIST = ['insert', 'update', 'remove', 'dropindex', 'ensureindex']
```

```
BACKUP_MAP_RULE= {'update':'insert', 'remove':'insert'}
```

```
def __init__(self, cmd_str):
```

```
    self.__cmd_str = cmd_str
```

```
    self.__error = ''
```

```
    self.__cmd_db, self.__cmd_colname, self.__oper_type, self.__cmd_json_list_str, self.__cmd_filter_str = self.__split_cmd_str()
```

```
    self.backup_instance = Mymongoconn(DB_HOST='127.0.0.1', DB_PORT=27019)
```

```
def __split_cmd_str(self):...
```

```
def backup_data(self, backup_inst, exec_inst, collection_name):...
```

```
def get_query_sql(self):...
```

```
def inputcmd_selectcmd_diff(self, select_cmd_dbname, select_oper_type):...
```

观点分享

- 好架构是进化出来的，好代码是迭代出来的；
- 永远不要追求一步到到位；
- 需求分析重于一切；
- 多写代码，你才有感觉，这暗合吴老师的实战班思想；
以前听说评价java程序员水平高低用代码量来评估的！
- 书看多了也没用！够用就好！

感谢知数堂！

感谢张老师！给我这次会！

感谢大家花了这么久的时间听我分享！

祝知数堂越办越好！