gfirefly dbentrust

**dbentrust介绍**

dbentrust是gfirefly的数据处理部分，主要封装了memcache和关系型数据库的操作，封装了使用memcache对关系型数据库查询数据的缓存，关系型数据库读写分离等。

**MemObj类介绍**

MemObj是dbentrust中的核心基础类，基础他可以实现自定义的映射到memcache数据的类。从下面的例子我们可以看到MemObj类的作用。

from gfirefly.dbentrust.memclient import memcached\_connect

memcached\_connect(["127.0.0.1:11211"])

from gfirefly.dbentrust.memfields import MemObj,MemFields

class Mcharacter(MemObj):

def \_\_init\_\_(self,name):

MemObj.\_\_init\_\_(self, name)

self.id = MemFields()

self.level = MemFields()

self.profession = MemFields()

self.nickname = MemFields()

self.guanqia = MemFields()

self.initFields()

mcharacter = Mcharacter('character:1')

mcharacter.nickname='someone'

mc\_other = Mcharacter('character:1')

mc\_other2 = Mcharacter('character:2')

print "mcharacter.nickname",mcharacter.nickname

print "mc\_other.nickname",mc\_other.nickname

print "mc\_other2.nickname",mc\_other2.nickname

mcharacter.mdelete()

得到的输出结果为

mcharacter.nickname someone

mc\_other.nickname someone

mc\_other2.nickname None

Mcharacter类继承了MemObj类，创建一个名称为character:1的Mcharacter的实例mcharacter，修改他的nickname为someone。再创建一个名称同为character:1的实例mc\_other和一个名称为mc\_other2的实例。从输出的结果可以看出mc\_other与名称相同的mcharacter对象的nickname的值相同。这个过程的数据同步是通过memcache完成的。

**MemModel**与**MemAdmin类介绍**

MemModel继承了MemObj类，主要作用是实现了对象到memcache再到关系型数据库的数据同步。

from gfirefly.dbentrust.dbpool import dbpool

from gfirefly.dbentrust.memclient import memcached\_connect

memcached\_connect(["127.0.0.1:11211"])

from gfirefly.dbentrust.memmodel import MemAdmin

import time

aa = {"default":{'host':"localhost",'user':'root',

'passwd':'111',

'db':'legend',

'port':3306,

'charset':'utf8'},

}

dbpool.initPool(aa)

ma = MemAdmin('tb\_role\_info','id',incrkey='id')

t1 = time.time()

mm = ma.getObj(1)

mm.data['vip\_exp'] = 123

mm.save()

print "mm.data",mm.data

data = dict(mm.data)

del data['id']

mm\_new = ma.new(data)

print mm\_new.data['id']

下面是数据表的创建语句

CREATE TABLE `tb\_role\_info` (

`id` int(20) NOT NULL AUTO\_INCREMENT COMMENT '角色的ID',

`username` varchar(20) COLLATE utf8\_unicode\_ci DEFAULT NULL COMMENT '用户名',

`role\_name` varchar(20) COLLATE utf8\_unicode\_ci DEFAULT NULL COMMENT '角色昵称',

`profession` int(20) DEFAULT '1' COMMENT '角色的职业',

`hp` int(20) DEFAULT '100' COMMENT '当前血量',

`mp` int(20) DEFAULT '100' COMMENT '当前魔法值',

`level` int(20) DEFAULT '1' COMMENT '当前等级',

`exp` int(20) DEFAULT '0' COMMENT '当前经验值',

`scene` int(20) DEFAULT '1' COMMENT '当前所在的地图',

`position\_x` int(20) DEFAULT '400' COMMENT '当前所在的x坐标',

`position\_y` int(20) DEFAULT '400' COMMENT '当前所在的y坐标',

`vip` int(20) DEFAULT '0' COMMENT '当前vip等级',

`vip\_exp` int(20) DEFAULT NULL COMMENT '当前vip经验',

`createdate` date DEFAULT NULL COMMENT '创建时间',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=12 DEFAULT CHARSET=utf8 COLLATE=utf8\_unicode\_ci;

Ma.getObj(1)初次获取id为1的数据返回对象mm，这个过程memcache中缓存了这条记录的话则记录直接从memcache中读取，在修改mm中data的vip\_exp并且调用mm.save()后memcache中的数据将会更新，并且更新到数据库中。由于框架采用的是gevent这个过程将是异步的。

MemModel的实例通过MemAdmin来创建。

**RouterBase数据库读写分离**

数据库的读写分离通过继承RouterBase，重写db\_for\_read和db\_for\_write来实现，如下面的实例

from gfirefly.dbentrust.dbpool import dbpool,RouterBase

class router(RouterBase):

def db\_for\_read(self, \*\*kw):

return "default"

def db\_for\_write(self, \*\*kw):

return "master"

然后绑定router

dbpool.bind\_router(router)

下面是完整的示例

from gfirefly.dbentrust.dbpool import dbpool,RouterBase

from gfirefly.dbentrust.memclient import memcached\_connect

memcached\_connect(["127.0.0.1:11211"])

from gfirefly.dbentrust.memmodel import MemAdmin

aa = {"default":{'host':"localhost",'user':'root',

'passwd':'111',

'db':'legend',

'port':3306,

'charset':'utf8'},

"master":{'host':"localhost",'user':'root',

'passwd':'111',

'db':'test',

'port':3306,

'charset':'utf8'}

}

dbpool.initPool(aa)

class router(RouterBase):

def db\_for\_read(self, \*\*kw):

return "default"

def db\_for\_write(self, \*\*kw):

return "master"

dbpool.bind\_router(router)

ma = MemAdmin('tb\_role\_info','id',incrkey='id')

mm = ma.getObj(1)

mm.data['vip\_exp'] = 123

print "mm.data",mm.data

data = dict(mm.data)

del data['id']

mm\_new = ma.new(data)

print mm\_new.data['id']

通过运行可以发现读取数据是从default中读取的，而写入数据是数据被写入了master。

**MemObj 并发读写**

应用场景：

比如一个MemObj中对应memcache中其中一个字段a的值为A，现在一个客户端C1去修改获取到的值是A，另一个客户端C2获取到A，C1基于a现有的值在后面加上B变成AB，同时C2基于获取到的现有值在后面加上C结果是AC，最终的结果是AC，这并不是理想的结果，我们需要的结果实际上市ABC。这样问题就出现。

为了避免这个问题，MemObj引入了对象锁，这个锁的超时是2秒钟，2秒后锁会自动释放掉。如果配置了MemObj的cas为True，当这个对象实例化的时候，对象就被锁定，其他地方再获取值的时候是获取不到的，会尝试获取10次，直到锁被释放才会获取到当前的值，并且在这个对象中被锁定，否则获取不到值。

这个释放的过程是需要手动释放的，obj.release()，但是在MemModel中调用save后会释放掉锁。

可以看下面的两个例子：

from memclient import memcached\_connect

memcached\_connect(["127.0.0.1:11211"])

class Mcharacter(MemObj):

def \_\_init\_\_(self,name,\*\*kw):

MemObj.\_\_init\_\_(self, name,\*\*kw)

self.id = MemFields(1)

self.level = MemFields(0)

self.profession = MemFields(9)

self.nickname = MemFields(u"")

self.guanqia = MemFields(100)

self.initFields()

mcharacter = Mcharacter('character:1',cas=True)

mcharacter.nickname = 'lanjinmin'

mcharacter.insert()

print "mcharacter.nickname",mcharacter.nickname

mcharacter.release()

mc\_other = Mcharacter('character:1',cas=True)

mc\_other2 = Mcharacter('character:2',cas=True)

print "mc\_other.nickname",mc\_other.nickname

print "mc\_other2.nickname",mc\_other2.nickname

print "mc\_other.guanqia",mc\_other.guanqia

print type(mc\_other.guanqia)

del mc\_other,mc\_other2

下面一个例子：

from dbpool import dbpool

from memclient import memcached\_connect

memcached\_connect(["127.0.0.1:11222"])

from memclient import mclient

import time

aa = {"default":{'host':"localhost",'user':'root',

'passwd':'111',

'db':'legend',

'port':3306,

'charset':'utf8'},

}

dbpool.initPool(aa)

ma = MemAdmin('tb\_role\_info','id',incrkey='id',cas=True,cache=False)

t1 = time.time()

ma.load()

mm = ma.getObj(1)

print time.time()-t1

mm.data['vip\_exp'] = 123

print "mm.data",mm.data

print "use time :",time.time()-t1

data = dict(mm.data)

print "mc get",mclient.get("tb\_role\_info:1:data")

del data['id']

mm\_new = ma.new(data)

print "mc get",mclient.get("tb\_role\_info:1:data")

print "mm\_new.data",mm\_new.data