# 数据库设计

1. accounts集合

文档结构：{\_id:account\_id , pwd: account\_pwd , player: player\_id}



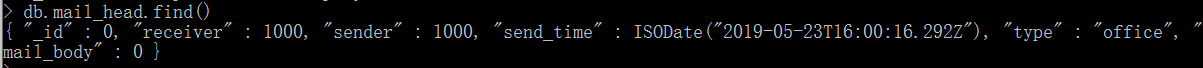
1. players集合

文档结构：{\_id:player\_id , name: player\_name , sex: player\_sex , mail\_head:[mh\_id…]}



1. mail\_head集合

文档结构:{\_id:mh\_id , receiver:player\_id , sender:player\_id , send\_time:time , type:type, check:1/0 , index:index,mail\_body:mb\_id}



1. mail\_body集合

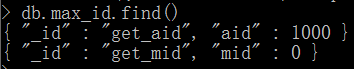
文档结构:{\_id:mb\_id , content:content ,extra\_type: extra\_type ,extra\_data:extra\_data, index:index,}



1. max\_id集合

文档结构{ "\_id" : "get\_aid", "aid" : aid }

{ "\_id" : "get\_mid", "mid" : mid }



# 系统设计

socket

network

log\_in

register

mail\_manager

account\_manager

log\_in

server

no

player\_manager

yes

send

rec

find

delete

update\_name

delete

find\_player

log\_out

mail

player

Load

mongo

## 系统简述

**account**数据在服务器启动时从数据库加载进内存，验证通过后建立player。此次项目为一个account对应一个player，所以两者的id都是一一对应的，并且是相同的。

* 注册功能仅仅注册账号，并不连带的建立好玩家角色，需要玩家登陆后再自行创建玩家角色
* 登陆时如有重复登陆，则对头一个登陆者踢下线
* Player的删除操作将删除数据库对应的信息，并且将account下的player置为null，邮件的索引相应减1.
* 改名操作修改数据库相对应的信息；
* 查找操作根据提供的id从数据库返回player信息；
* 登出操作则断开player与sock的联系。

加入mail后player的属性需要添加多一个mail\_head属性，存放所有邮件头编号。

**mail**在概念上分为两部分，mail\_head和mail\_body。

mail\_head部分由mh\_id , receiver , sender , send\_time , type, index， checked,mail\_body组成，玩家登陆时通过自身的mail\_head属性找到响应的邮件头信息，加载并常驻在内存。

mail\_body部分存放邮件信息部分，由mb\_id , index ，content ,extra\_type , extra\_data组成，当玩家请求查看信息时才会提取数据库的mail\_body信息。

邮件的整体如下：

|  |  |
| --- | --- |
| Head | Mh\_id  Receiver  Sender  send\_time  type,  mail\_body  checked  index |
| Body | mb\_id  content  extra\_type  extra\_data  index |

邮件的查找链：player.mh\_id->mail\_head.mb\_id->mail\_body.content

邮件依赖于玩家角色，即玩家角色存在的时候，才有邮件功能

* 发件操作玩家仅需提供玩家id（sender）和发送内容（content）即可，如果有附件的话需提供extra\_type，extra\_data，否则默认为空,其它信息由系统分配。
* 收件功能在下线情况下仍然要可以使用，具体实现为其他玩家发送邮件后在数据库产生数据，当玩家上线时拉取数据库信息，即可收到来件信息。在线玩家登陆时会将自身的pid和sock加载进内存，其他玩家发送邮件时会先在内存里找目标玩家的pid所对应的sock，使用此sock给目标玩家发送来邮提示。
* 查看操作根据player属性里md\_id，查看所有自己收发的所有邮件
* 删除操作则删除已发送或已收到的邮件，由于每封邮件都有自己的ID，玩家提供邮件id即可删除player里的mail\_head，删除操作会相应的减少该邮件的索引值，当索引值为0时，则删除数据库的保存的邮件。

# 节点安排

前2节点用于写项目需求和项目设计，所以剩下的按8个节点分配。

* 节点1：为重新添加进来的mial模块设置数据库，并对原有的模块进行调整，并且试验，保证原有的模块不受影响
* 节点2-3：撰写maill代码，并且单独测试功能是否合理和正常。如果顺利就话费2个节点，不顺利则话费再加节点。
* 节点4-5：撰写mial\_manager代码，联通server和mail，并进行测试
* 节点6：整体的测试程序，消灭bug
* 节点7-8：测试程序，也备用其他节点进展不顺利时。

# UML类图

|  |
| --- |
| Server |
| +send\_rec\_obj  +mongo,network  +account\_manager  +players\_manager  +mail\_manager |
| +on\_data(data,sock)  +listen\_connect() |

|  |  |  |
| --- | --- | --- |
| Account\_manager | Player\_manager | Mail\_manager |
| +cru\_max\_aid  +all\_accounts  +mongo  +send\_rec\_obj  +online | +cru\_max\_aid  +mongo  +send\_rec\_obj | +mongo  +cru\_max\_aid  +send\_rec\_obj  +online |
| +log\_in(aid\_pwd,sock)  +create\_account(data,sock) | +init\_player(sock)  +create\_player(data,sock) | +init\_mail(sock) |

|  |  |  |
| --- | --- | --- |
| Account | Player | Mail |
| +id  +pwd  +player  +send\_rec\_obj  +mongo  +sock | +id  +name +sex  +mongo  +send\_rec\_obj  +sock | +cru\_max\_aid  +pid  +type  +all\_mail\_head  +unread\_mail\_head  +readed\_mail\_head  +sended\_mail\_head  +mongo  +sock  +send\_rec\_obj |
| +log\_out()  +update\_account(operate) | +update\_name(new\_name)  +find\_player(pid)  +delete\_player() | +send\_mail(data)  +load\_my\_mail()  +check\_mail(option)  +show\_mail\_detail(mid)  +delete\_one\_mail(mid) |