# 前言：

本手册目的在于方便新来开发人员了解系统，迅速进入开发状态。

由于时代的发展，本手册中所提到内容可能会有过时的，使用的时候千万注意。

如果可能，还是希望有人维护一下本手册~

考虑到新人接触系统以及具体工作由浅入深的客观步骤，本手册分为两部分，第一部分只介绍实现一个简单策划案所需要的系统。第二部分没有特定顺序和逻辑，可以用来做查询用。

本手册假设你已经更新到了所有的代码。

# 热血精灵派系统实现的简要介绍

热血的业务逻辑实现是按照简单的前端发包，后台接包，进行逻辑处理，然后给前端返包的顺利进行。

# 拿到案子，不知道如何下手？

热血的业务逻辑实现是按照简单的前端发包，后台接包，进行逻辑处理，然后给前端返包的顺序进行。因此，要完成一个案子，最基本的方式就是将案子从逻辑上面分解为一个个步骤，比如前端点击一个按钮，后台做些处理，然后给前端返回，这样的。

而具体的实现步骤基本上就是：

进入/home/rudy/blitz/proto/client/define路径，找到command.xml，内容如下：

<command id="10351" name="cs\_year\_charge\_get\_bean" des="获取兑换精灵豆" />

按照里面的格式，自己定一个id（注意svn更新和提交）。然后运行一下make命令，就会自动生成一个枚举值，在这里就是cli\_cmd\_ cs\_year\_charge\_get\_bean，注意，在代码中只允许使用枚举值，而不准使用硬编码的id哦。

然后在此路径下面找到某个\*.proto来定义前后端交互的protobuf包的格式，具体哪个proto，怎么命名，请咨询组长或导师。

然后进入/home/rudy/blitz/online/src路径，打开server.cpp找到init\_processors()，

函数的内容为:g\_proto\_processor->register\_command(cli\_cmd\_cs\_year\_charge\_get\_bean, new YearChargeGetBeanCmdProcessor());

照着这个样子，注册你自己的处理对象。其中YearChargeGetBeanCmdProcessor你现在还没有，先写上去一个，应该如何命名应该一看就懂了吧。

然后找到一个.h和.cpp文件来实现YearChargeGetBeanCmdProcessor，目前是使用的/operation-date/activity-processor-date.\*，具体应该写在哪里请咨询组长或导师。

写上自己的对象，比如：

class LuoLuoSkillGetCmdProcessor : public CmdProcessorInterface

{

public:

//接受客户端发包

int proc\_pkg\_from\_client(player\_t\* player, const char\* body, int bodylen);

//接受服务器端发包

int proc\_pkg\_from\_serv(player\_t\* player, const char\* body, int bodylen);

};

注意继承CmdProcessorInterface。

编译一下，看能否通过。

如果通过了，恭喜你这个步骤就基本完成了，剩下的就只是往proc\_pkg\_from\_client和proc\_pkg\_from\_serv里面添加内容实现业务逻辑了。系统会根据你定义的cmd-id调用你的proc\_pkg\_from\_client，来进行处理。

这里是一个实例：

class YouFriendSetPetCmdProcessor : public CmdProcessorInterface

{

public:

int proc\_pkg\_from\_client(player\_t\* player, const char\* body, int bodylen);

int proc\_pkg\_from\_serv(player\_t\* player, const char\* body, int bodylen)

{

return 0;//不需要接收其他服务器的发包，因此不做处理

}

private:

onlineproto::cs\_you\_friend\_set\_pet cli\_in\_;//这两个就是你在\*.proto里面定义的包的格式

onlineproto::sc\_you\_friend\_set\_pet cli\_out\_;

};

int YouFriendSetPetCmdProcessor::proc\_pkg\_from\_client(

player\_t\* player, const char\* body, int bodylen)

{

cli\_in\_.Clear();//清除掉数据

if (parse\_message(body, bodylen, &cli\_in\_) != 0) {

return send\_err\_to\_player(player, player->wait\_cmd,

cli\_err\_proto\_format\_err);

}//将前端发过来的protobuf序列进行反序列化，获得数据

//以下为各种具体的业务逻辑

Pet\* pet = PetUtils::get\_pet\_in\_loc(player, cli\_in\_.petid(), PET\_LOC\_BAG);

if (pet == NULL) {

return send\_default\_cmd\_err(player, cli\_err\_pet\_not\_exist);

}

player->temp\_info.you\_friend\_pet\_chosen = cli\_in\_.petid();

return send\_msg\_to\_player(player,cli-cmd-cs-\*\*\*, cli\_out\_);//将数据返回给前端，完成这个流程

}

几个重要api：

parse\_message:将前端发过来的protobuf反序列。

send-msg-to-player，将结果返回给前端。

Async-server的接口