|  |
| --- |
| Netease |
| **Juggle说明文档** |
|  |

|  |
| --- |
| 吴铭  2015/2/2 |

Juggle说明文档

[Juggle说明文档 2](#_Toc410655372)

[Juggle简介 3](#_Toc410655373)

[Juggle使用说明 4](#_Toc410655374)

[Juggle设计文档 8](#_Toc410655375)

[使用uuid确保消息的唯一性 11](#_Toc410655376)

[使用内存池管理service中的对象 12](#_Toc410655377)

[使用无锁队列优化多线程中的消息传递 13](#_Toc410655378)

[使用协程实现同步非阻塞接口 14](#_Toc410655379)

[使用codegen提高工作效率 15](#_Toc410655380)

Juggle简介

Juggle是一个基于dsl语言的可配置网络数据封包协议的rpc框架，基于codegen提供了c++的服务器构建工具。

同类产品:

Protobuf: https://code.google.com/p/protobuf/

Thrift: http://thrift.apache.org/

Juggle使用说明

Juggle dsl是强类型的中间语言，通过模板，支持了仅原生类型的泛型。

Juggle dsl关键字:

module 对应c++中的class，用于定义一个供客户端访问的服务

array 对应c++中的std：：vector

struct 对应c++中的class，用于用户自定义数据

string 字符串,对应c++中的std::string

float 浮点数，对应c++中的double

int 整数，对应c++中的int64\_t

bool 布尔值, 对应c++中的bool

juggle的一个简单例子:

module juggle{

string login(string argv3);

string test(string argv3);

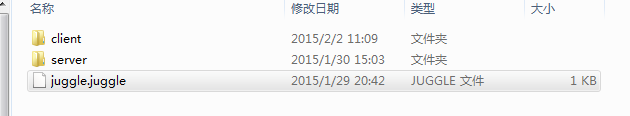
}

使用juggle rpcmake生成代码

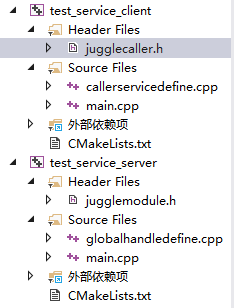


参数分别为juggle代码目录，生成代码目标目录，协议封包对象, 协议封包对象头文件

生成代码如下:



之后即可基于juggle生成的代码，创建服务器工程



客户端直接使用生成的远程调用对象即可

sync::juggle j(ch.get());

std::cout << j.login("i am login").c\_str() << std::endl;

std::cout << j.test("i am test").c\_str() << std::endl;

服务器端生成了一组协议响应代码，并且定义了rpc接口的虚函数以及创建module的



用户需要实现对应的接口即可

class juggleimpl : public juggle{

public:

virtual std::string login(std::string argv3){

printf("juggleimpl login %s\n", argv3.c\_str());

return "login sucess";

}

virtual std::string test(std::string argv3){

printf("juggleimpl test %s\n", argv3.c\_str());

return "test sucess";

}

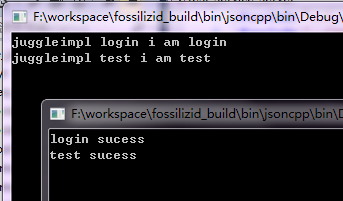
};

juggle \* create\_juggle(){

return new juggleimpl();

}

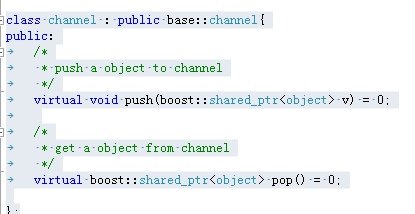
测试用例执行如下:



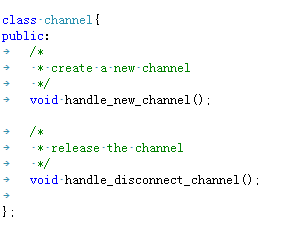
Juggle设计文档

Juggle基于协程和函数对象设计了清晰了同步非阻塞访问接口，并且在设计上实现了通信层和协议层、消息处理的解耦合。

网络层部分juggle定义了一组最简的通信接口:



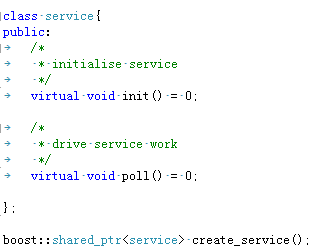
和一组注册channle到事务层的接口



协议层部分juggle定义了一个虚基类的打包接口定义,用户需要基于juggle给出的接口实现自己的打包协议:



事务层juggle完成了一个单线程的整个事务处理的全部流程，玩家只需要定义juggle service的接口，并且调用对应的初始化接口和驱动接口，完成整个事件循环，然后在此基础上编写消息响应代码即可。

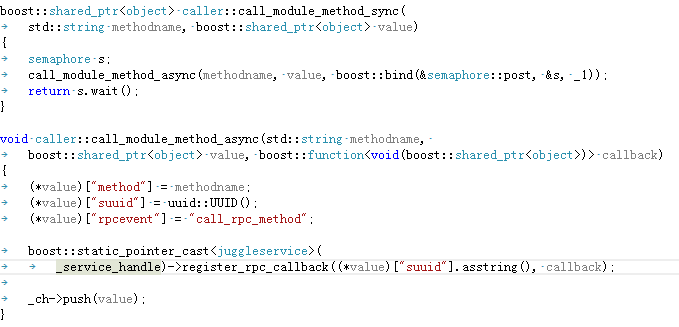




使用uuid确保消息的唯一性

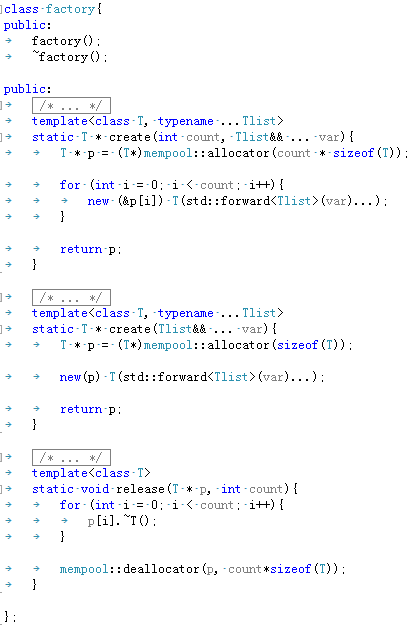
UUID是一种通过算法来确保在一个有限的范围内分布式唯一的技术。

在juggle中，为每条消息生成了uuid，并且通过uuid为消息注册自己的回调函数，以此来实现了对同一类事件即可同步访问亦可异步访问。



使用内存池管理service

Fossilizid提供了接口优雅的内存池，juggle以此配合shared\_ptr来管理service中的对象。



使用无锁队列优化多线程中的消息传递

无锁数据结构主要用于为多线程环境提供一个可以高效访问的容器。

代表性的有CDS: <http://libcds.sourceforge.net/>

michael在这个领域的论文:<http://www.research.ibm.com/people/m/michael/>

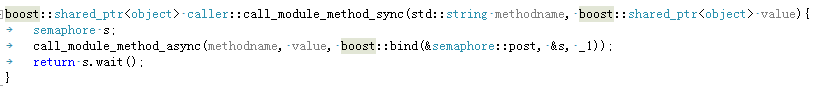
Fossilizid提供了几组简单的无锁队列，并且用于网络层的消息传递



使用协程实现同步非阻塞接口

一般认识上远程访问一般是异步的，因为对一个远端的请求之后，客户端需要等待服务器的返回。有次引出了消息响应，状态保存等一系列策略。

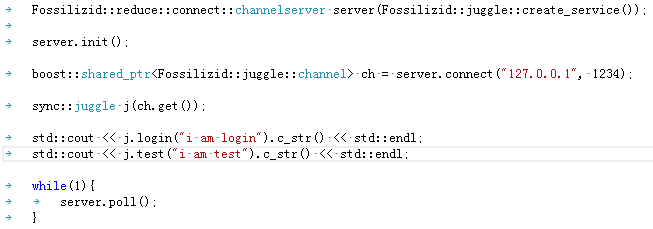
但是基于协程有用户层控制切换的特性，我们可以在提出一个请求后将此协程挂起，并且将此消息的uuid注册到协程管理器，然后等待远端服务器的返回，收到返回消息后调度执行此协程。



得益于c++提供的函数对象等机制。

我们只需为此消息注册特殊的回调函数即可。

见用例，可以看见我们对远端的访问是同步的。



使用codegen提高工作效率

一般意义上的消息响应代码都是趋同的：

注册消息响应函数->

消息收发->

消息unpack->

调用消息响应函数

由此我们可以编写一些代码生成工具，以此完成这些重复的工作。这亦是juggle中最核心的部分。

通过对juggle脚本的分析，rpcmake就会自动生成对应的代码

关于代码生成的说明: <http://km.netease.com/kp_blog/view?article_id=165054>