**RPC**

1. **选用Thrift的原因：**

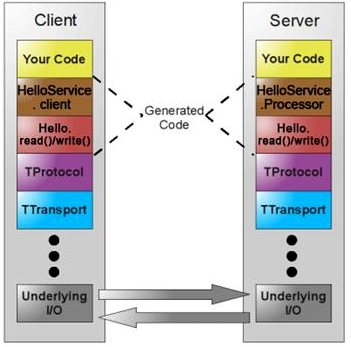
Thrift作为Facebook开源的RPC框架, 通过IDL（IDL是Interface description language的缩写，指接口描述语言，是[CORBA](https://baike.baidu.com/item/CORBA/2776997" \t "_blank)规范的一部分，是[跨平台](https://baike.baidu.com/item/%E8%B7%A8%E5%B9%B3%E5%8F%B0/8558902" \t "_blank)开发的基础）中间语言, 并借助代码生成引擎生成各种主流语言的rpc框架服务端/客户端代码，主要特点：开发速度快，接口维护简单高效，学习成本低，多语言/跨语言支持，成熟稳定。

······ 备用方案dubbo，开始搭建框架。

1. **基础知识**

**2.1.Thrift软件栈**

Thrift对软件栈的定义非常的清晰, 使得各个组件能够松散的耦合, 针对不同的应用场景, 选择不同是方式去搭建服务。



* + Transport: 传输层，定义数据传输方式，可以为TCP/IP传输，内存共享或者文件共享等
  + protocol: 协议层, 定义数据传输格式，可以为二进制或者XML等
  + Processor: 处理层, 这部分由定义的idl来生成, 封装了协议输入输出流, 并委托给用户实现的handler进行处理.
  + Server: 服务层, 整合上述组件, 提供网络模型(单线程/多线程/事件驱动), 最终形成真正的服务.

**2.2. Thrift 对语言的支持**

Thrift和Google Protobuf相比, 都提供了可扩展序列化机制, 不但兼容性好而且压缩率高. 两者在这块各有长短, 性能上PB稍有优势. 但在语言的支持度上, Protobuf只支持c++/java/python这三种主流的语言, Thrift则几乎覆盖了大部分的语言, 从这点上来说, Thrift的优势非常的明显.

**3.基本步骤**

**3.1.thrift生成代码**

先手动创建编写thrift文件，然后使用官方提供的编译工具，可以生成各种代码。由于我是在Java语言上测试的，所以我编译成了Java代码。Thrift文件中详细的格式介绍可见官方文档: <http://thrift.apache.org/static/files/thrift-20070401.pdf>。

看下thrift生成的类有：

//Thrift生成的RpcService代码, 省略了函数和具体实现

public class RpcService {

//接口类Iface, 同步接口

public interface Iface{}

// 接口类AsyncIface, 异步接口

public interface AsyncIface{}

//具体类, 同步Client

public static class Client{}

//具体类, 异步Client

public static class AsyncClient{}

//同步处理器Processor

public static class Processor<I extends Iface>{}

//异步处理器AsyncProcessor

public static class AsyncProcessor<I extends AsyncIface>{}

//定义了一个对象，里面封装了 对外暴露的接口方法，

public static class rpc\_args{}

//实际调用 我们的业务方法并返回结果

public static class rpc\_result{}

**3.2实现接口**

**3.3编写相应的服务器和客户端的代码**

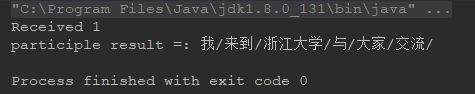
**4.基本测试**

**对应工程Thrift-test.**

**服务器开始结果：**



**客户端开始结果：**



其间还解决了log4j的报错：工程中使用了多个版本的log4j。

在第一次作业检查时，发现好多人的结果截图都是有这个错误的，都没有意识到，这是个错误。因为网上的；例程都是存在这个错误的—总结：网上的例程也是有可能会存在常识性错误的，要多留个心眼。对自己后期再此基础上的扩展是有好处的。

**5.** **TthreadedSelectorServer实现**

ThreadedSelectorServer分离了Accept和Read/Write的Selector线程, 同时引入Worker工作线程池. 它也是种Half-sync/Half-async的服务模型

TThreadedSelectorServer模式是目前Thrift提供的最高级的模式，它内部有如果几个部分构成：

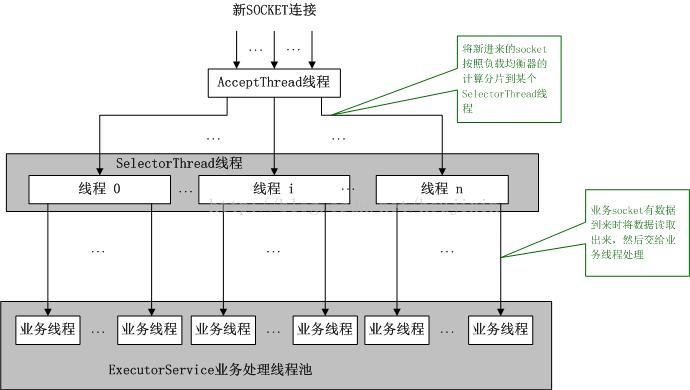
（1） 一个AcceptThread线程对象，专门用于处理监听socket上的新连接；

（2） 若干个SelectorThread对象专门用于处理业务socket的网络I/O操作，所有网络数据的读写均是有这些线程来完成；

（3） 一个负载均衡器SelectorThreadLoadBalancer对象，主要用于AcceptThread线程接收到一个新socket连接请求时，决定将这个新连接请求分配给哪个SelectorThread线程。

（4） 一个ExecutorService类型的工作线程池，在SelectorThread线程中，监听到有业务socket中有调用请求过来，则将请求读取之后，交个ExecutorService线程池中的线程完成此次调用的具体执行；主要用于处理每个rpc请求的handler回调处理(这部分是同步的).

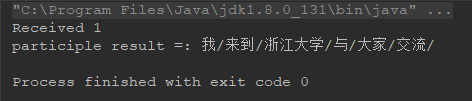
工作模式图：



**服务器开始结果：**



**客户端开始结果：**



此处只是进行了单个客户端的使用测试，后面打算采用循环不断开启客户端来测试。

**6．加入了连接池，并用spring集合了Thrift**

**6.1.连接池**是创建和管理一个连接的缓冲池的技术，这些连接准备好被任何需要它们的线程使用。我采用了commons-pool2提供的连接池。

**优点：**

1.资源重用

避免了频繁创建、释放Thrift连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片）。

2.更快的系统响应速度

连接池在初始化过程中，往往已经创建了若干Thrift连接置于池中备用。此时Thrift连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用Thrift连接，避免了连接初始化和释放过程的时间开销，从而缩减了系统整体响应时间。

**6.2.Spring**是一个开放源代码的设计层面框架，他解决的是业务逻辑层和其他各层的松耦合问题，因此它将面向接口的编程思想贯穿整个系统应用。

**优点：**

1.使用Spring的IOC容器，将对象之间的依赖关系交给Spring，降低组件之间的耦合性，让我们更专注于应用逻辑

2.可以提供众多服务，事务管理，WS等。

3.AOP的很好支持，方便面向切面编程。

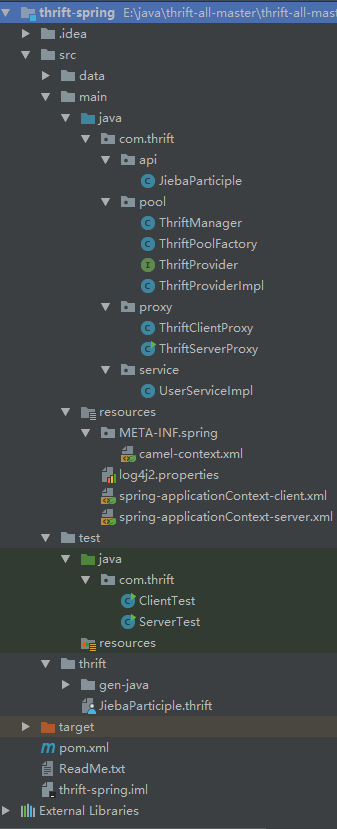
4.对主流的框架提供了很好的集成支持

5.Spring DI机制降低了业务对象替换的复杂性。

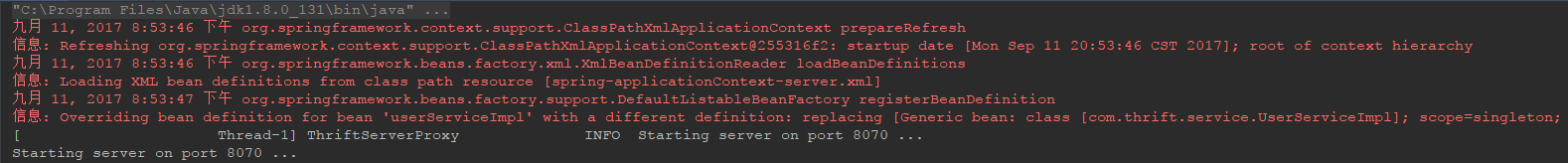
6.Spring属于低侵入，代码污染极低。

7.Spring的高度可开放性，并不强制依赖于Spring，开发者可以自由选择Spring部分或全部

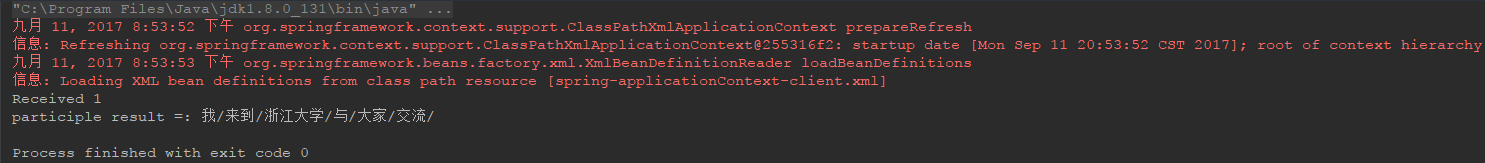
**结果：（代码量太多，只展示下工程目录）**



**服务器运行：**



**客户端运行：**



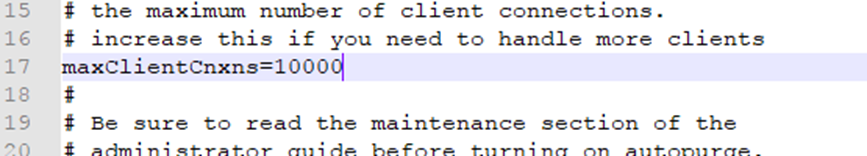
**7.Zookeeper**

ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

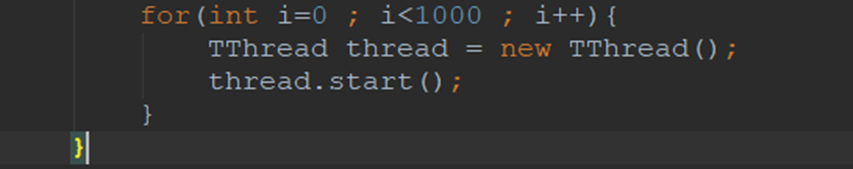
实现: Curator是Netflix公司开源的一个Zookeeper客户端，与Zookeeper提供的原生客户端相比，Curator的抽象层次更高，简化了Zookeeper客户端编程。

**结果：**

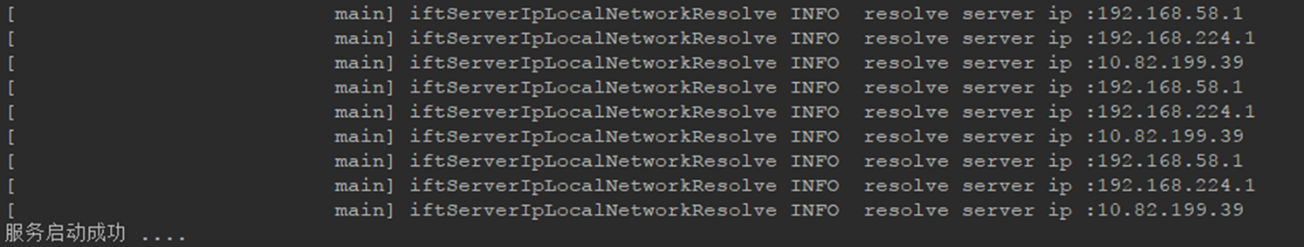
Zookeeper配置10000次调用：



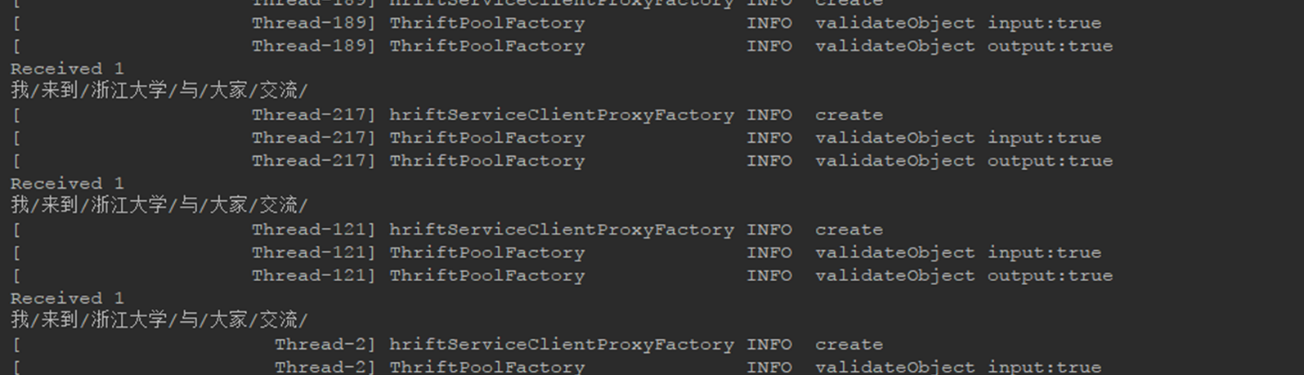
客户端开启1000个线程（10000个运行缓慢）：



服务器开启结果：



客户端测试部分结果：



**8.进步**

对maven和spring有了更深的了解，学习了thrift，dubbo, zookeeper。

**9.不足**

1. 分词并未优化。

进度不是很快，项目时间有限，并没有花时间去优化完善分词那一块。只是简单调用了一下相关功能包。

2. Zookeeper使用并不成熟。

试过测试10000次的调用，发现运行速度不是很快，而且运行到一半时，16G的内存就已经满负荷了。虽然可是自身设备性能有限的原因。但是由于刚接触zookeeper，所以应该是有很多值得优化的地方，并没有意识，需待后期，认真学习优化。

**Sql**

**1.csv导入到Mysql中**

先将csv文件导入到了Mysql中。

**代码：**

load data infile '/data/database.csv' into database

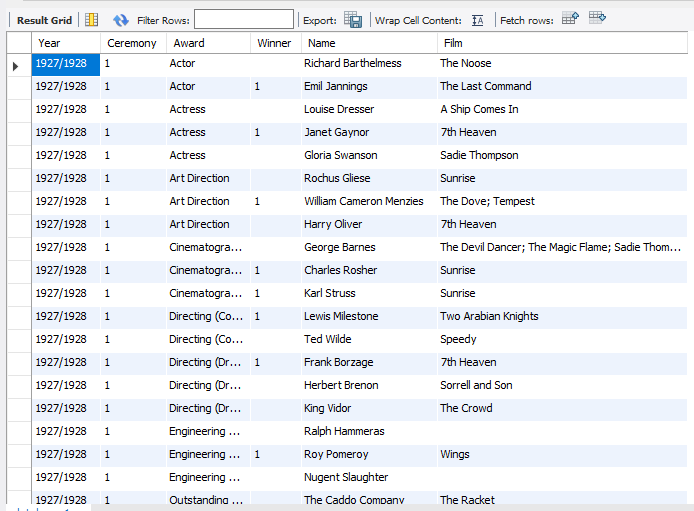
character set utf8

fields terminated by ','

enclosed by '"'

lines terminated by '\r\n';

**结果:**



2.初步提炼数据

每个奖项下，每个人的提名次数(用ct表示)和获奖次数(用wct表示)。

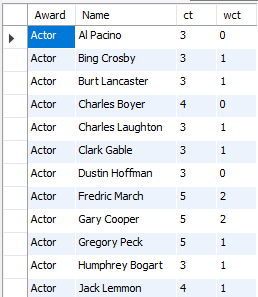
统计提名次数(ct)的原因：一开始只统计获奖次数时，发现获奖次数不少于3的只有三个人，那这个题的后面的排序就没有意义了，所以，需要把获得提名也包含到获奖里面。

统计获奖次数(wct)的原因：只统计提名次数时发现次数相同的太多，对只提前三名不是很有利，于是又添加了一个新的排序标准，最后结合ct和wct来排序。

**代码:**



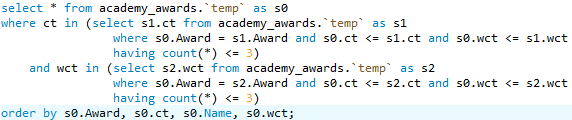
**结果:**



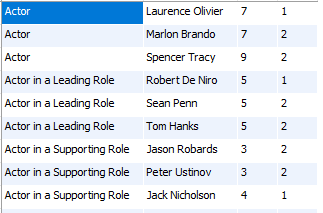
3.排序

在新生成的temp表里面排序。

**代码：**



**结果：**



上面是结果的一部分，不过，用了两个字段进行排序后发现，还是有一个奖项里面出现了并排第三名的现象。但目前，就已有的数据，我暂时不能想出别的方法来排除其中一个人。