作业1 自己画出RPC框架设计图！

作业2 不看资料，自己手画Spring Cloud底层原理图

**优秀作业1:**

学员：顾飞  
 点评：思考的非常深入，技术选型有自己的见解

作业1：

1、动态代理层：RPC框架的一切调用细节都是动态代理方法里实现的。可选用Java原生Proxy类方法或Spring的Cglib实现。生成接口类的代理类，重写方法，实现真实调用。

2、注册中心：提供者启动时进行注册，消费者对某些服务进行订阅，服务发现。

① zookeeper， dubbo service-A service-B provider cosumer provider consumer IP+port IP+port IP+port IP+port IP+port IP+port IP+port IP+port 如上图设计好的树状结构来存储;

Providers启动时在Zookeeper中某个服务节点下创建临时子节点，节点名称使用IP、端口、权重等信息拼接字符串，这样实现服务注册。

Consumer监听某个服务节点服务的子节点，并且拉取子节点列表。实现服务订阅和发现。

当Provider上线或下线时，临时节点发生新增或删除。所以Consumer那边监听会收到通知，拉取最新的注册表数据，更新本地注册表缓存，这样实现服务发现。

② 也可以采用Eureka的实现方式，采用纯内存ConcurrentHashMap来存储注册表，Provider启动时将IP、端口、权重等信息注册。实现服务注册。

Consumers定时去拉取注册表缓存在本地。实现服务发现。然后各服务采用定时心跳的方式来告诉注册中心自己是否存活。 实现故障感知。

比如用Map<String,Map<String,Lease<InstanceInfo>>>这样的结构来保存，服务名为key，Value Map的key为实例ID，Lease中维护最近的心跳时间，InstanceInfo中保存实例的IP、端口等信息。这是内存注册表。

并且可以加入多级缓存来优化并发读写冲突。 配置好合适的注册表拉取时间和心跳时间，比如30秒，则可以单机承载大规模系统日千万级的访问量。 （1秒钟差不多100次的调用）。

综上：Zookeeper方式显然更加实时感知集群变动。而Eureka的方式因为是定时拉取，所以消费端对集群变更感知会有一定延时性。

3、Cluster层：实现路由和负载均衡、集群事件处理，比如故障切换等。必备轮询、加权轮询、随机、加权随机、一致性Hash等路由或负载均衡方式。

加权可以根据权重增加虚拟节点实现。一致性Hash，采用Hash环方式，并加入一定数量的虚拟节点来使节点更加均衡的分布在Hash环上，顺时针旋转时，继而请求分布更加均匀。可以采用ConcurrentSkipListMap来实现。

4、数据发送部分：Exchange，Portocol协议、序列化反序列化机制、Transport网络IO。 Exchange封装为Request/Response，结合BlockingQueue将Netty异步通信转化同步。

协议可以是HTTP、dubbo等 序列化加入多种选择，比如Json、Protobuf、Protostuff、Hessian、Kryo等、Java序列化等等。

5、网络IO部分，可以使用BIO、NIO。BIO性能极差！推荐使用NIO框架Netty来实现，并处理好数据粘包问题。

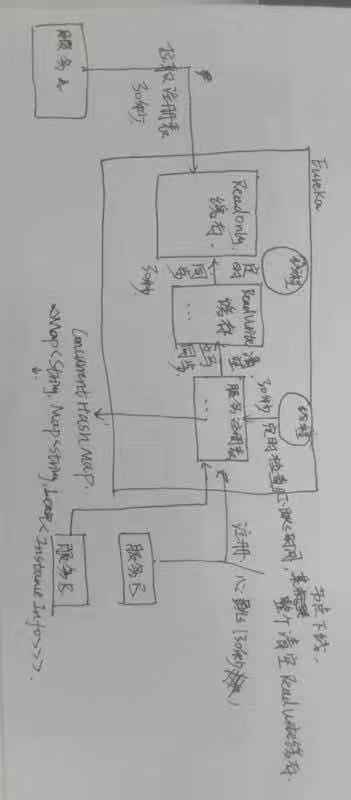
使用多路复用IO机制，服务端有一组Acceptor线程，ServerSocketChannel监听某个端口，而Acceptor线程通过多路复用器Selector来轮询监听ServerSocketChannel的Accept事件。

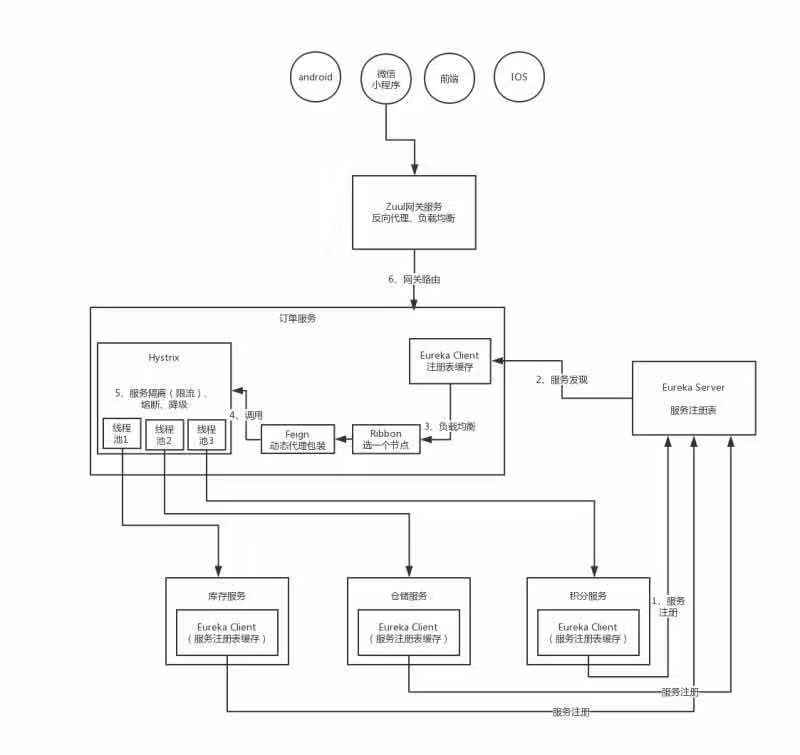
当客户端的netty请求连接时，此时会创建对应的SocketChannel。被Processor线程组通过Selector轮询监听，来处理实际的IO。

当客户端发来数据时，服务端SocketChannel触发事件，由Processor线程去解析请求，经历反序列化和Exchange、Protocol层层解析后，经由代理类找到对应的服务实现方法处理。再经过SocketChannel返回给客户端。

客户端同样是processor线程组监听到SocketChannel的事件，读取响应给程序。

综上所述： 划分层次的话，就是10层 Service、Config、Proxy、Registry、Cluster、Protocol、Exchage、Transport、Serialize、Monitor





**优秀作业2：**

学员：carson

作业1: 设计一个rpc框架画出设计图

1: 首先需要service层，提供接口，给提供方和消费端实现。

2: 然后提供一个注册中心，用于服务注册与发现，将服务方配置信息注册到注册中心进行管理。

3: 如果消费方调用生产者，先通过动态代理，生成代理对象。

4: 通过clustor组件中（directory 获取服务配置信息 router获取路由信息）。

5:通过负载均衡策略（random loadbalance、roundrobin loadbalance、leastactive loadbalance、consistanthash loadbalance）选择一台机器进行服务调用。

6: 选择一种通信协议（dubbo、http、rmi）进行服务之间交互。

7: 将请求参数通过exchange组件封装。

8: 通过网络通信框架netty将数据进行序列化发送到指定的端口。

9: 服务方监听发送过来的请求并处理。

10: 解析请求对象进行反序列化。

11: 选择一种通信协议生成动态代理调用接口。

12:结果处理通过动态代理进行返回。

作业2：画Spring Cloud底层原理图

Erueka

1: 首先有个服务接口，IP地址127.0.0.1，端口8080，服务注册并暴露。

2: spring Cloud Eureka注册中心有个注册服务表记录服务注册上来的服务地址端口等信息。

3: Eureka注册表registry会立马向读写缓存readWriteCache同步注册信息。

4: readWriteCache会通过一个线程定时（30s）向readOnlyCache同步一次注册配置信息。

5: 消费方会从readOnlyCache中定时同步注册信息到本地缓存注册表。

6: 消费方会通过本地缓存route信息通过负载均衡Ribbon选择一台机器。

7: 选择完机器后就通过feign组件调用服务接口进行交互。

8: eureka还有检测服务心跳与故障功能，会定时扫描registry服务注册上来的心跳，如果90s内没有心跳，认为服务已挂，会remove注册表信息。

9: 故障的注册配置信息清空后也会立马同步给readWriteCache缓存，缓存做清空操作。

10: 此时readOnlyCache可能并没有和readWriteCache同步，导致消费者调用故障地址所在的服务调用失败。

11: 当readOnlyCache定时同步比对writeOnlyCache时候，发现readWriteCache已经被清空，那么readOnlyCache也被清空。

12: 消费者来readaOnlyCache已经清空则去readWriteCache同步，发现还是为空，则去register拉去注册服务，然后同步给readWriteCache和readOnlyCache缓存。

13：消费者通过正常机器的地址去调用服务接口正常。

提问： 老师，您好，关于今天讲解的Erueka原理有个疑问：就是某个接口所在的服务器故障后，register感知后，则把该故障的配置移除，然后立马同步给readWriteCache

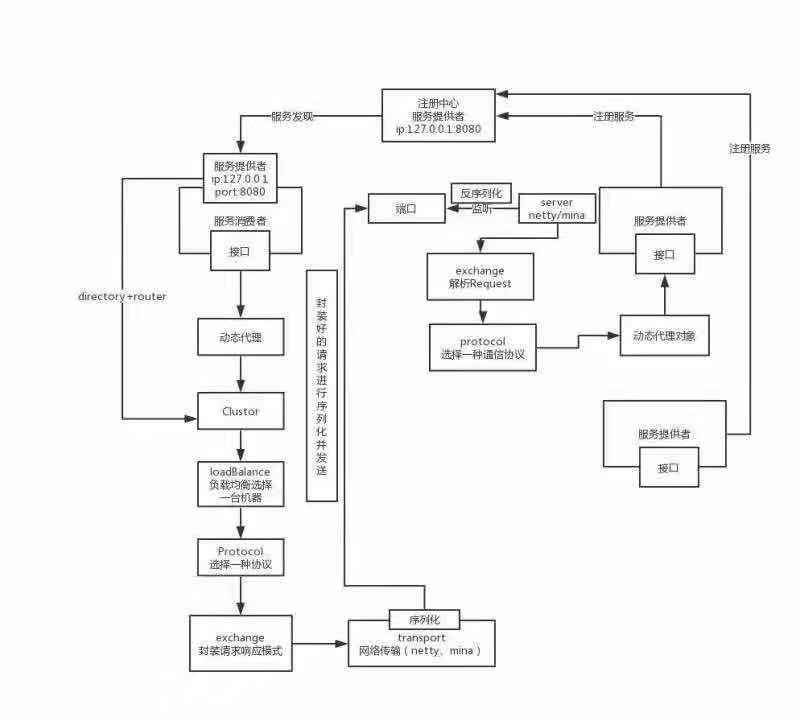
这个过程中，readWriteCache会做全量清空的操作,后续的readOnlyCache定时比对readWriteCache后，也会做全量清空的操作?

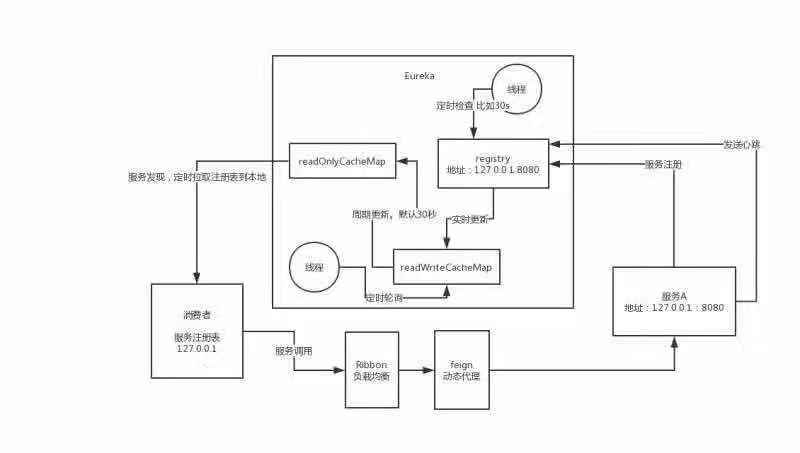
为什么这几个缓存直接的数据同步不做增量同步的操作呢?哪些不一致就清空哪些不可以么?这种设计的优点会是什么?麻烦请教下老师 谢谢!

问题解答：

缓存最麻烦的就是缓存更新操作，缓存增量更新会稍微复杂一点；

其次在生产环境下服务很少会出现下线等操作，所以清空缓存实际不会那么频繁，直接删除缓存就好了，等下次拉取注册表用到的时候再进行缓存





**优秀作业3：**

学员：线程池

Spring Cloud 基本原理图 服务注册和发现流程：

1. 服务提供方会注册到 eureka 上，写入注册表中。

2. 注册表内容有变化，会将 writeOnly 清空。

3. 有一个定时线程，每隔 30 秒将 writeOnly 的内容同步至 readOnly。

4. 有一个心跳定时检测线程，每隔一定时间移除没有心跳的节点信息。更新注册表，然后再走 2 的步骤。

5. eureka client 会每隔 30 秒读取 readOnly，保存至本地。 采用了 readOnly 和 writeOnly 是为了减少锁的竞争，让服务快速注册上 eureka，而不用等待锁阻塞。

这里有一个疑问：老师，您在视频讲解的时候说，Eureka 启动后，服务第一次注册时，会立马把注册表数据同步给 writeOnly。

但是如果有一个节点重启完成后，注册表上会有一个新的节点，此时注册表是清空 writeOnly。

我的疑问是 只有Eureka 启动后，服务第一次注册会立马同步数据给 writeOnly吗？此后，注册表变动，都是清空 writeOnly 吗？

服务调用流程：

1. Feign 会对加了注解的接口生成一个动态代理对象。

2. Ribbon 会从 eureka client 注册表通过负载均衡策略，拿到一个请求地址。

3. Feign 拿到 Ribbon 负载均衡后得到的请求地址，并发起 HTTP 请求。

RPC 框架架构设计图 服务注册发现阶段：

1. 服务提供者向注册中心注册服务。

2. 注册中心保存这个服务提供者的节点信息。

3. 服务调用者向注册中心订阅服务，并获取节点信息。

4. 当节点信息变化时，注册中心要主动通知给服务调用者，服务调用者及时同步本地注册表。

服务调用阶段：

1. 对服务提供者的接口生成动态代理对象。

2. 这个动态代理对象中有 “熔断降级 Mock 等机制” ，“服务注册表” 和 “负载均衡”。

3. 动态代理对象会根据是否熔断降级和 Mock 走相应的逻辑。如果是真实请求，那么通过负载均衡机制从注册表中选一个请求地址和端口，并发起请求。

4. 请求后，过滤器链中会记录请求的行为，做好监控。

5. 接下来协议层进行封装协议。

6. 传输层将序列化后的数据发给服务提供者。

消息接收阶段:

1. 服务提供者拿到数据进行反序列化和解析协议。

2. 通过过滤器链统计请求处理信息，进行监控。

3. 提取出请求数据里面的 impl 类名和方法参数，进行反射调用。

Netty 基本流程说明：

1. bossGroup 中取一个 nioEventLoop 把 ServerSocketChannel 注册到 selector 上。

2. nioEventLoop 通过 ServerSocketChannel 绑定端口然后注册 OP\_ACCEPT 事件。

3. 当有新连接进入的时候，ServerSocketChannel 将新连接交给 childGroup 来处理。

4. childGroup 选一个 nioEventLoop 来负责这个 SocketChannel。

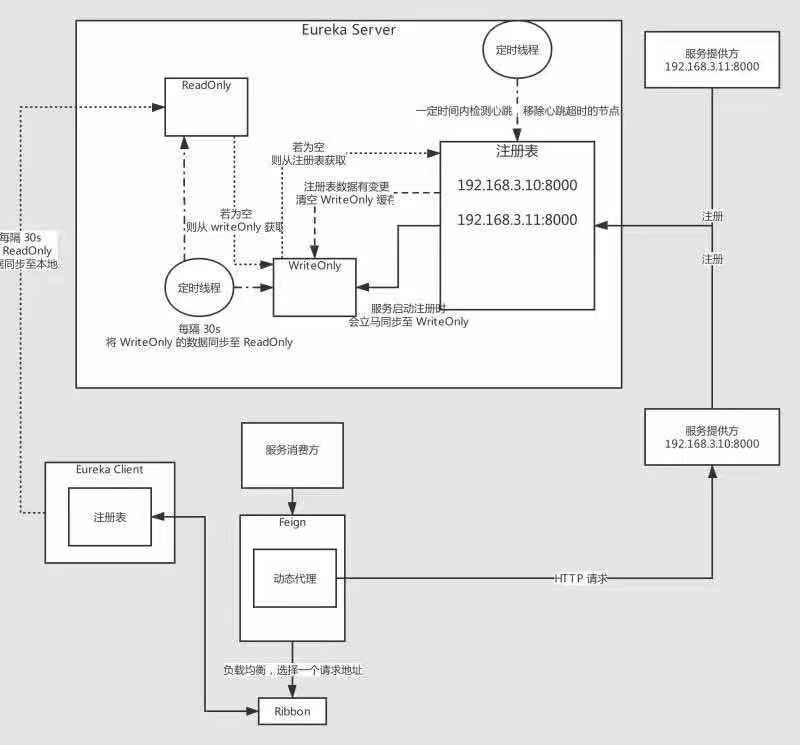
5. 接下来，nioEventLoop 把 SocketChannel 注册到 selector 上，并关注 OP\_READ 事件。

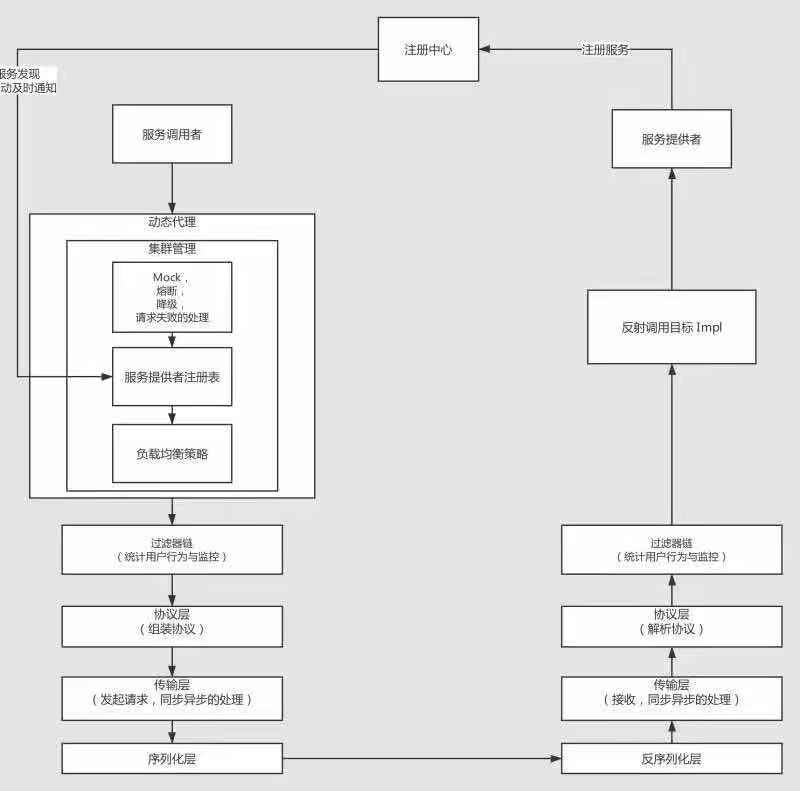
6. 之后有数据读事件来时，SocketChannel 就可以来处理了。

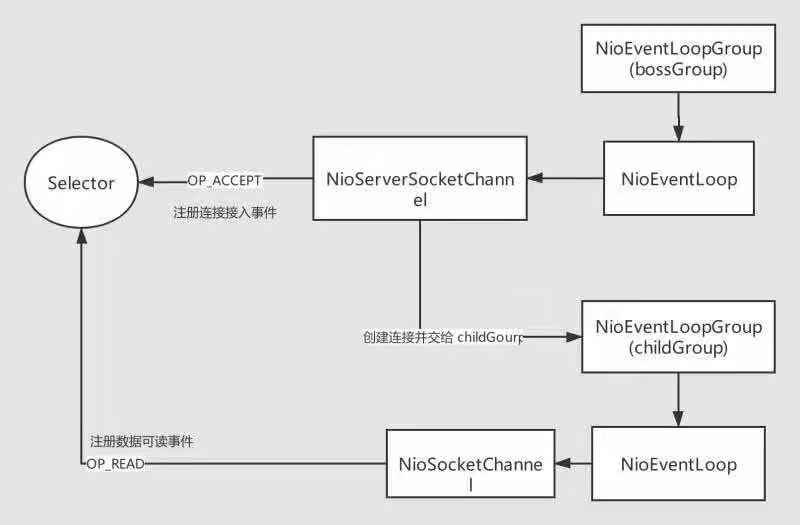
老师，我的作业中有一个疑问，麻烦留意一下，帮忙回答一下哦。

问题回答：

服务实例发生注册、下线、故障的时候，也就是注册表发生变动时都会去刷新readWriteCacheMap，将对应的缓存给过期掉







**优秀作业4:**

学员：逐日

老师好，我说下我自己设计RPC框架的话，大体的思路：

1、框架讲究能够配置的都提取为配置，然后框架再根据配置去自动做很多事情，先说配置，配置可以选择在xml中配置，或者提供元数据注解的方式。在这里，RPC框架的话，主要是配置我们要调用的接口，接口的方法。

2、在真正要进行调用的时候（或者是程序启动时），此时需要生成RPC接口的动态代理，由于是对接口生成代理，直接使用jdk的动态代理即可，动态代理中，即是我们的核心逻辑；

3、传给动态代理的参数，包括了要调用的接口、接口的方法、以及参数，我们就可以根据接口名，去服务注册表拉取服务注册数据，拉到的数据可能是一个list，里面保护了实现了该接口的服务的ip、端口等；

4、拿到的是列表，但我们只需要调用其中一台，这里需要进行负载均衡，如果简单实现，可以直接轮询，要做成动态扩展的话，考虑SPI机制

即由我们作为框架作者，指定负载均衡的接口，框架使用者可以实现我们的接口，并像SPI机制一样，在META-INF目录下的文件中指定接口的实现

我们框架可以自动扫描类路径下的这类文件，如果发现框架使用者自定义了实现，即使用，否则使用框架默认的；

5、在第四步中，获取到一台服务器后，开始构造请求，然后丢给下一步；

6、网络通信层，这一层需要进行对要发送的请求，进行编码，编好后交给网络通信框架发送出去，可以选择使用netty等成熟框架；

这一层层次不是特别清晰，因为如果使用netty的话，编解码部分一般会直接使用netty的encoder，并在pipeline中直接发送出去。

这里需要考虑的是，是否要考虑长连接，如果使用长连接的话，应该需要设计连接池，避免每次RPC调用时，都重建连接，最好直接从连接池获取。

还要考虑的是，像netty是nio读写，写操作不会阻塞，但是我们还需要等待rpc调用返回，才能将结果返回上层。 这一步的设计，还需要参考各netty客户端框架的实现。

7、收到rpc响应后，返回给上层等。

有个问题想咨询老师：客户端采用netty实现的框架，在业务线程等待服务端返回时，线程此时不能返回给上层，因为数据还没回来

那么此时，业务线程是要阻塞吗？一般数据回来是在io线程，io线程再去notify业务线程？

这块比较困惑，希望老师解答一下

问题解答：

是的，dubbo是通过类似Future.get()方法来实现阻塞的，底层是通过ReentrantLock的Condition的await()和notify来实现阻塞和唤醒的。