1. 自我介绍
   1. 如何学习java
      1. 通过网络视频和java相关书籍学习基础知识。
      2. 通过阅读博客相关技术主题和阅读开源源码以及分析开源代码设计理念来提升知识深度和广度。
   2. 工作经历
      1. 2014年开始第一份java工作，当时参与普通的oa系统模块开发。
      2. 2015-2017上半年开始从事java爬虫工作，在这期间参与并熟悉了网络垂直类数据分布式采集系统的开发和架构，在此也衍生出了分布式任务调度，分布式节点管理，RPC通信等基础类组件，并且部分结合学习开源相关基础组件的设计思想，并做了个人的相关开源实现。
      3. 2017下半年到现在，主要从事金融信贷风控平台的工作，目前负责风控平台外部数据接入工作。
2. 基础能力

2.1 数据结构和算法

2.2 TCP相关知识

3. 架构设计

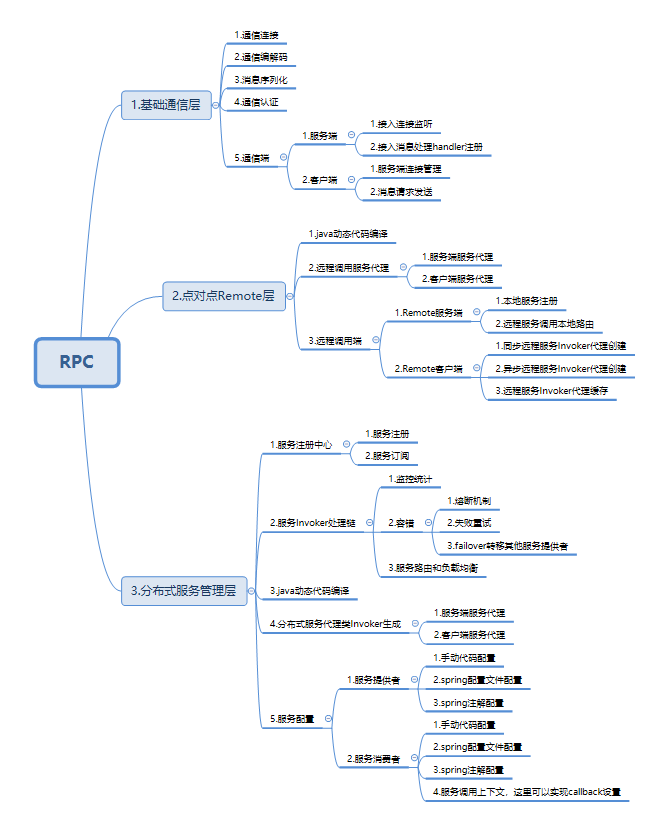
纵观了好多设计理念，凡是遵守纵向分层，横向分模块功能设计都是好的设计

2.3 JAVA相关知识

1. 项目经验(目前开源的这些基础类组件设计理念都差不多，我主要讲下大体的架构设计)

3.1 RPC通信

3.1.1 架构设计



3.1.2 细节和改进

1.通信层容易上手的api提供。

2.因为消息都是某个固定长度读取的，通过包装类封装多个缓存byte[]对外提供读写，减少缓存来回复制，比如netty缓存零拷贝实现（这个有区别于系统层面的内存零拷贝）。

3.合理粘包拆包处理。

4.nio实现，比如netty。

5.高性能序列化采用，比如protuf或者自定义消息实现read和write方法。

6.通信基础类库的采用，比如netty,mina等。

7.dubbo请求参数是通过Object[]封装的，这里存在2个问题:

1). 基本类型封箱拆箱

2). 基本类型参数同复杂类型参数一起序列。

针对这个问题，我们可以采用动态代码生成来优化例如:



3.1.2 容错

3.1.2.1 心跳检测机制(心跳包可以是某个固定的静态块)。为什么不采用tcp keepalive,它的弊端如下:

1. 它不是 TCP 的标准协议, 并且是默认关闭的.
2. TCP keepalive 机制依赖于操作系统的实现, 默认的 keepalive 心跳时间是 两个小时, 并且对 keepalive 的修改需要系统调用(或者修改系统配置), 灵活性不够.
3. TCP keepalive 与 TCP 协议绑定, 因此如果需要更换为 UDP 协议时, keepalive 机制就失效了.

3.1.2.2 断开重连。(比如Netty连接断开后会触发channelInactive调用)

3.1.2.3 调用超时机制

3.1.2.4 熔断机制

3.1.2.5 负载均衡机制

3.1.2.6 失败failover重试机制

3.1.2.7 合理的异常定义

1) 服务端服务拒绝异常

2) 服务端服务不可用异常

3) 服务服务执行异常

4) 客户端创建连接异常

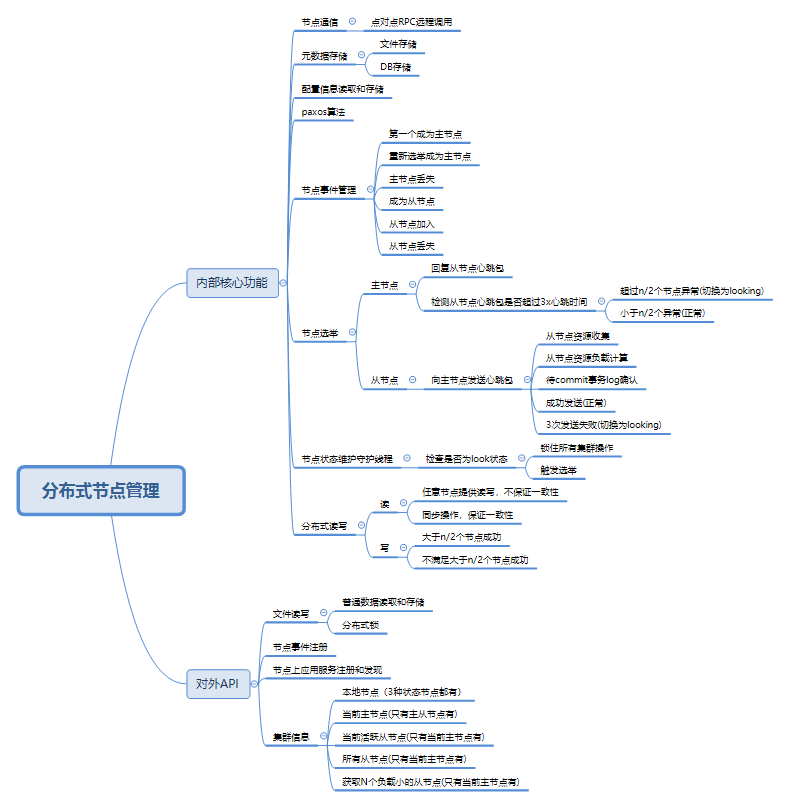
5) 客户端发送超时异常

6) 客户端发送失败异常

7) 其他未知异常

3.2 分布式节点管理

3.2.1 架构设计



3.2.2 细节和改进

1) 需要有效的封装内部节点状态维护，避免非法修改。

2) 不能达到节点状态变化绝对实时，但是尽可能达到近实时。

3) 对内部核心功能进行合理的分层分模块拆分，达到高内聚低耦合。

3.2.3 容错

1) 节点选举通过节点版本>节点id等维度选举出大于n/2个同意的master。

2) 当从节点向主节点发送心态失败3次后切换状态为looking重新触发选举。

3) 当主节点检测从节点上次心跳时间超过心跳时间\*3后并且这种情况的节点数 小于n/2个的话，那么主节点认为这个从节点离线，并且产生从节点离线事件。

4) 当主节点检测从节点上次心跳时间超过心跳时间\*3后并且这种情况的节点数 大于n/2个的话，那么主节点设置自身为looking状态重新触发选举。

5) 保证当节点状态looking时所有有影响的操作都认为是非法操作。

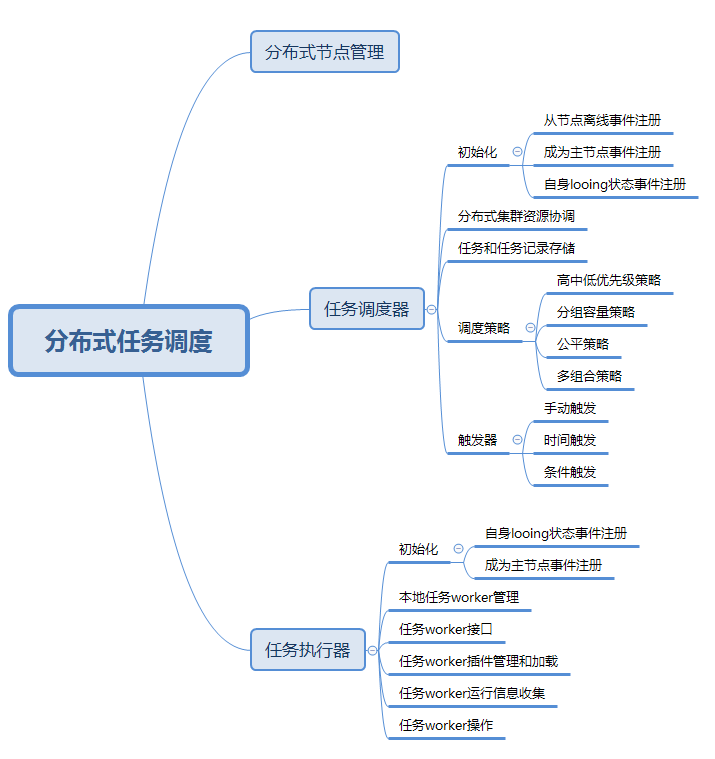
6) 所有的写操作必须发现master节点，然后主节点向所有的从节点发送写提议，

接收到提议的从节点将提议写入待commit事务log中后回复主节点写入待commit 事务log成功，主节点只要判断大于n/2个节点数成功既认为成功，然后向写入待 commit 事务log成功的从节点发送commit。最后回复客户端成功。采用待commit 事务log主要是避免从节点回复主节点失败后，待commit 事务log中的记录可以 后续在从节点向主节点发送心跳包时得到确认。

1. 任意节点提供读操作，除master节点外，不保证一致性，但保证最终一致性。
2. 对上层应用提供多个节点事件watch机制

3.3 分布式任务调度

3.3.1 架构设计



3.3.2 细节和改进

1) 需要引入JobBoss概念，减小主节点监控所有从节点上任务运行的压力。

2) 优化调度策略算法，提升集群资源的使用率，并且保证集群的负载均衡。

3) 给出更合理的从节点执行任务资源算法。

4) 根据现存任务运行记录分析，给出天，周，月等时间维度上任务分配优化， 避免某个时间点上任务分配过热。

3.3.3 容错

1) 保证任务不能重复执行。

2) 任务执行faliover重试机制。

3) 对于短期任务给出minTimeout和maxTimeout机制。当某个任务被某个节点 运行任务超出maxTimeout，根据任务配置主节点可以重新调度另一个正常从 节点执行该任务在某个节点上的任务分片。优先采取先完成的任务分片结果。

4) 主节点检测到从节点离线时，根据离线从节点上一次心态汇报记录，重新调度 其他正常从节点接手此节点的工作。

5) 从节点检测自身为looking状态后，应立马暂停当前节点上的任务，并等待正 常从节点状态后，询问master是否继续运行和结束。

6) 从节点检测自身为looking状态后，应立马暂停当前节点上的任务，并等待正 常主节点状态后，停止当前节点上的任务，并做重新分配，初始化Master角 色工作。

7) 主节点检测自身为looking状态后，应当立马暂停当前所有的调度角色工作， 并等待正常状态后，根据相应的节点状态做初始化。

8) 任务运行快照机制。

9) 任务运行记录统计机制。

1. 面试岗位要求对应是