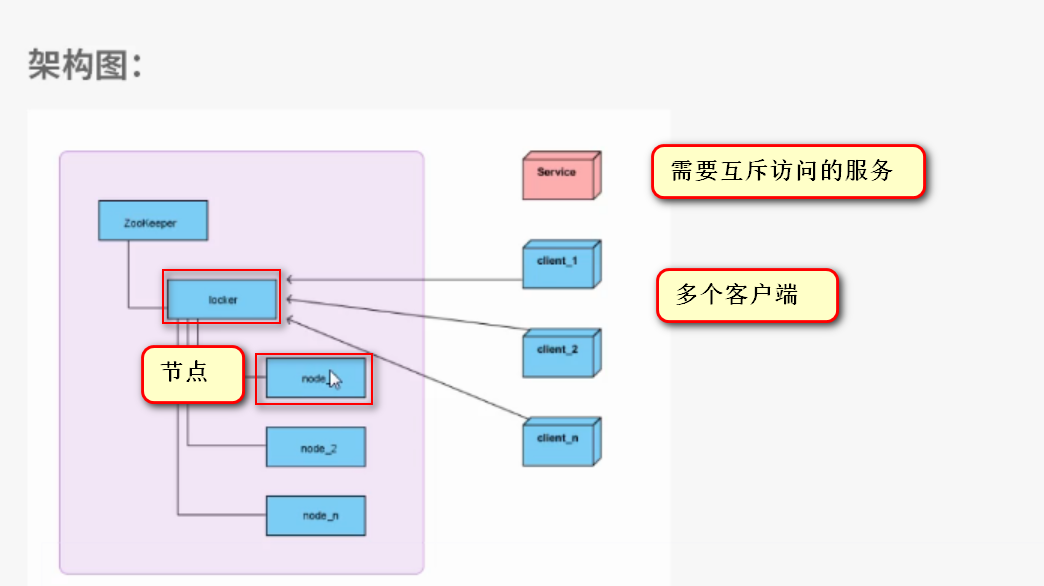
1. 分布式锁，跨进程，跨主机实现服务访问的互斥
2. 架构图



1. 核心算法。根据zookeeper创建顺序节点的特性可以由小到大进行排序，注意这里创建的节点是临时有序节点

临时顺序节点：基本特性与临时节点一致，创建节点的过程中，zookeeper会在其名字后自动追加一个单调增长的数字后缀，作为新的节点名；



这个时候通过zkclient获取locker路径下所有子节点的信息

判断 刚刚创建的节点是否是序号最小的子节点

如果是就代表自己已经获取了locker节点下的这个锁

如果不是就需要监听比自己次小的那个节点的删除事件来知道客户端是否已经释放了锁

如果已经删除就再走一遍流程

1. 核心接口

distributeLock {

public void acquire()

public Boolean acquire(long time,TimeUnit unit)

public void release()

client.delete(“path”)

}

尝试得到锁

1. 先再locker路径下创建节点

判断刚才创建的节点是否是nodeName

List.indexOf(String sequenceNodeName)

如果这个时候没有在zookeeper上找到该节点可能发生网络闪断

抛异常，让上一级进行处理（设置重试次数&记录日志）

判断 刚刚创建的节点是否是序号最小的子节点

如果是就代表自己已经获取了locker节点下的这个锁

如果不是就需要监听比自己次小的那个节点的删除事件来知道客户端是否已经释放了锁

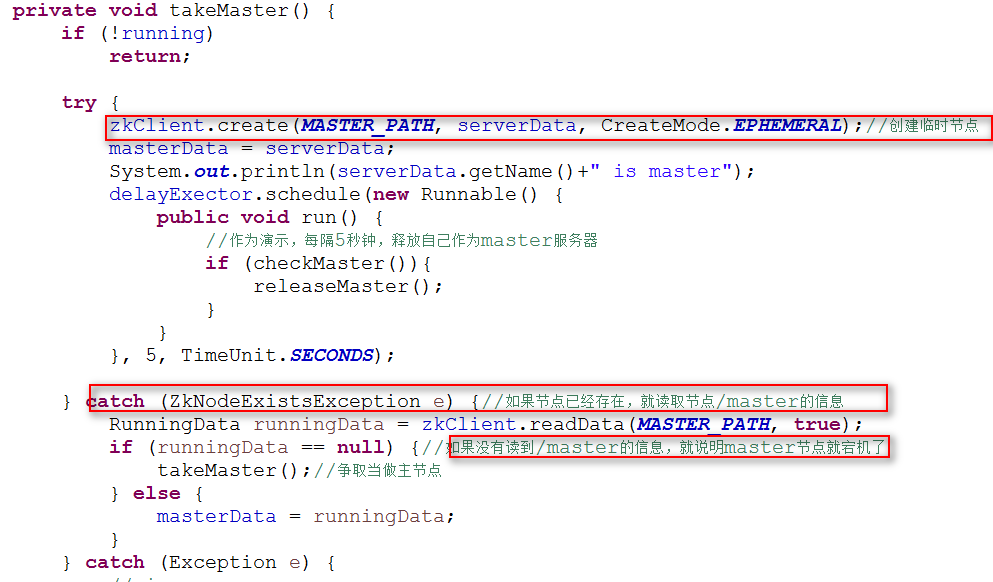
如果已经删除就再走一遍while循环



对选举模式的重新理解：

1. 选举是创建一个临时节点
2. 分布式锁是创建一个临时有序节点
3. 选举中模式是一主多从

选举中：节点调用takeMaster()方法，如果能够创建节点成功自己就是master，如果不能说明master路径下已经注册节点信息就会抛ZkNodeExistsException异常，转为获取master节点的信息



所有节点都注册master节点的删除事件，如果事件触发说明master节点宕机，调用takeMaster()

