主讲老师：Fox

课前须知：

1.Zookeeper的应用场景依赖于znode节点特性和watch监听机制，不清楚的同学补一下前面的课。

2.分布式锁常见的两种实现方案redis和zookeeper的设计思路务必掌握

3.下一个专题是mq，中间会停两节课给大家复习之前的专题，有精力的同学可以去看看Zookeeper源码分析课

```
文档：3. Zookeeper典型使用场景实战.note
链接：http://note.youdao.com/noteshare?id=3479e31a40bf0ee83faaaeb14d8439e6&sub=43B421531F464846ACC85F0DAE5589BE
```


Zookeeper 分布式锁实战

什么是分布式锁

在单体的应用开发场景中涉及并发同步的时候，大家往往采用Synchronized（同步）或者其他同一个JVM内Lock机制来解决多线程间的同步问题。在分布式集群工作的开发场景中，就需要一种更加高级的锁机制来处理跨机器的进程之间的数据同步问题，这种跨机器的锁就是分布式锁。

目前分布式锁，比较成熟、主流的方案：

（1）基于数据库的分布式锁。db操作性能较差，并且有锁表的风险，一般不考虑。

（2）基于Redis的分布式锁。适用于并发量很大、性能要求很高而可靠性问题可以通过其他方案去弥补的场景。

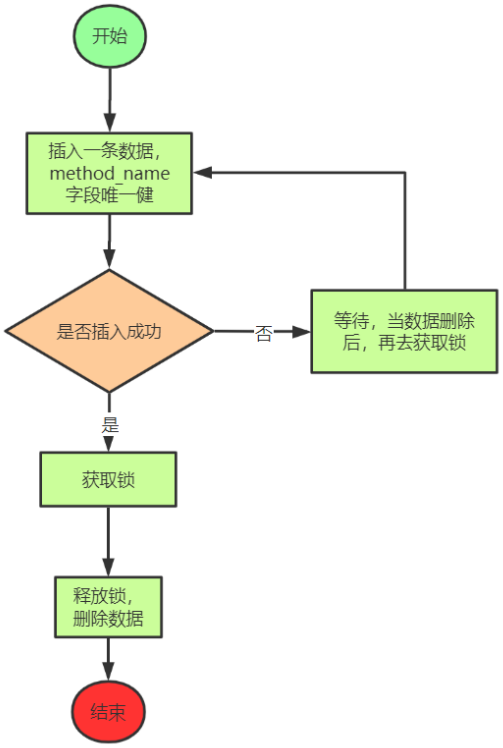
（3）基于ZooKeeper的分布式锁。适用于高可靠（高可用），而并发量不是太高的场景。

基于Redis实现分布式锁

https://vip.tulingxueyuan.cn/detail/p_602e53b3e4b035d3cdb8f856/6

基于数据库设计思路

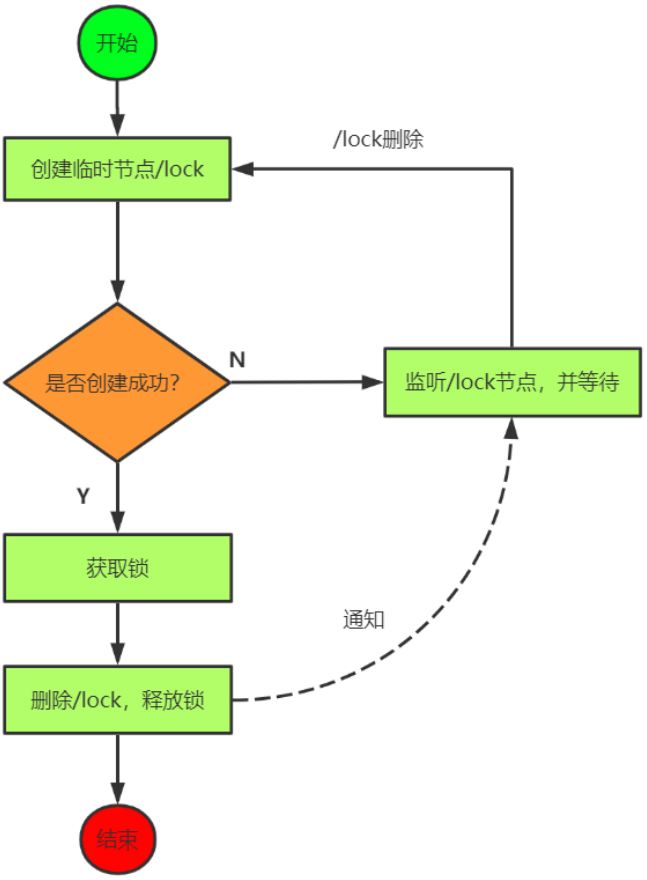
可以利用数据库的唯一索引来实现，唯一索引天然具有排他性



思考：基于数据库实现分布式锁存在什么问题？

基于Zookeeper设计思路一

使用临时 znode 来表示获取锁的请求，创建 znode成功的用户拿到锁。



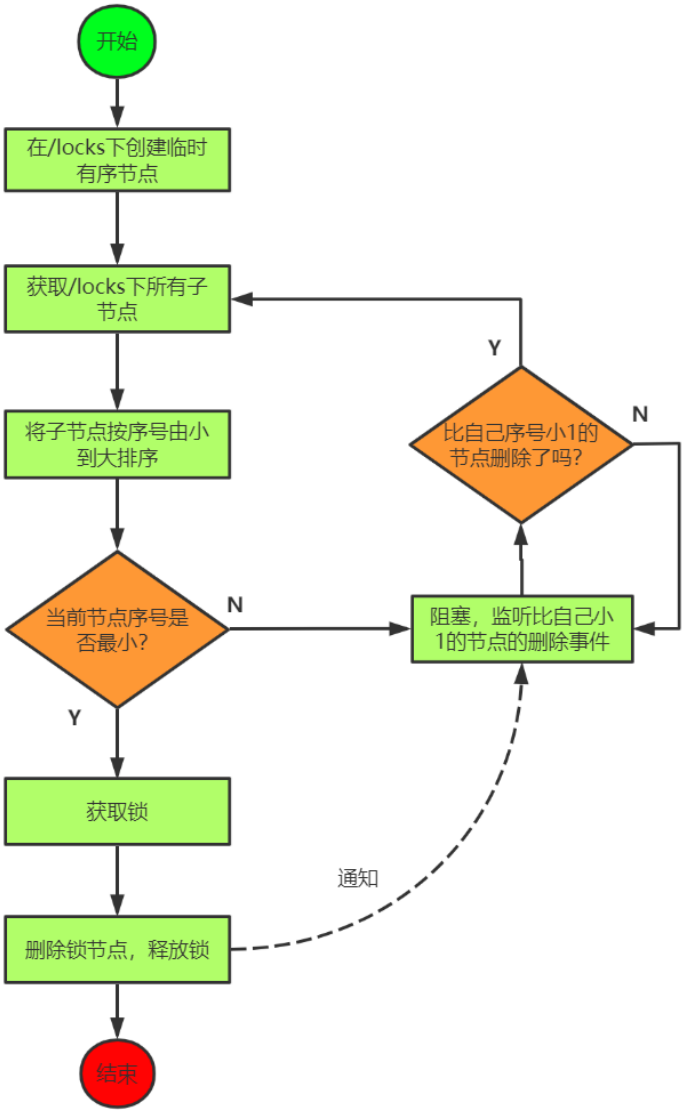
思考：上述设计存在什么问题?

如果所有的锁请求者都 watch 锁持有者，当代表锁持有者的 znode 被删除以后，所有的锁请求者都会通知到，但是只有一个锁请求者能拿到锁。这就是羊群效应。

基于Zookeeper设计思路二

使用临时顺序 znode 来表示获取锁的请求，创建最小后缀数字 znode 的用户成功拿到锁。

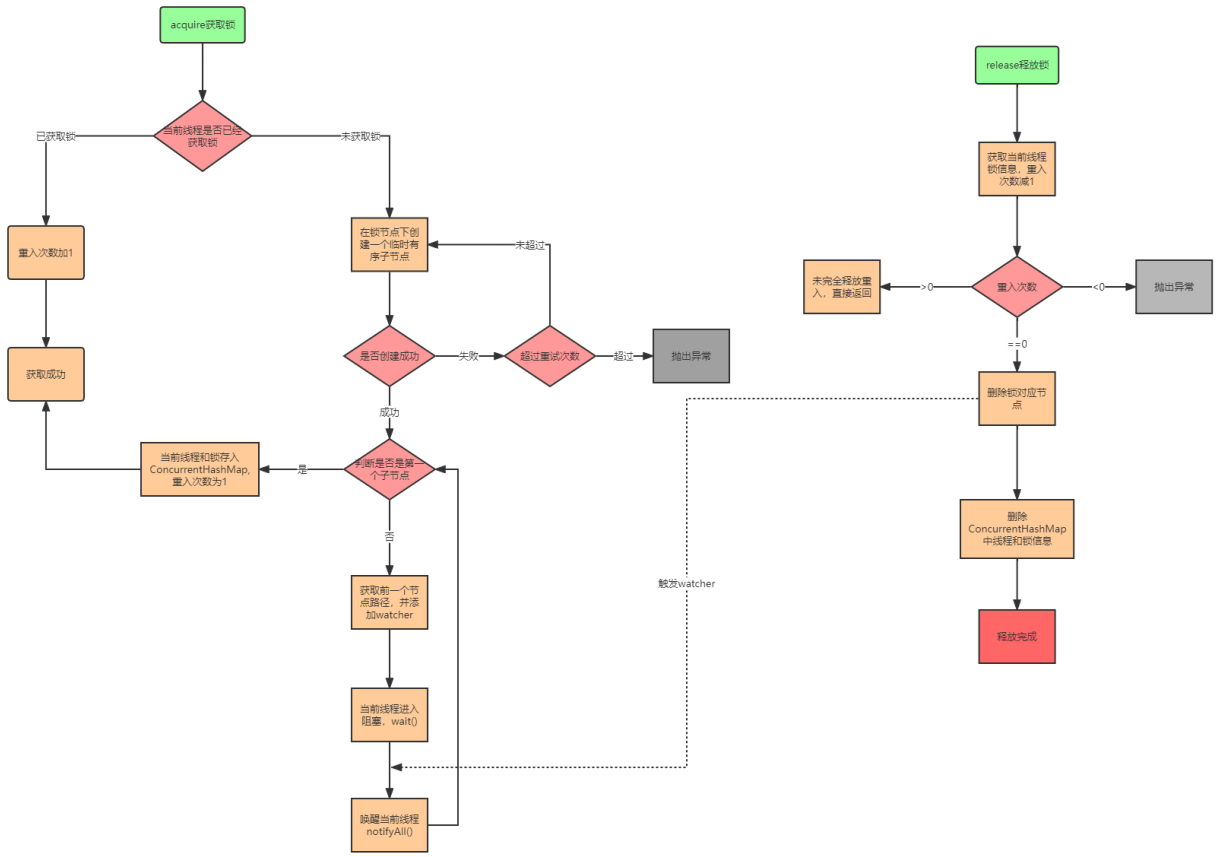
公平锁的实现



在实际的开发中，如果需要使用到分布式锁，不建议去自己“重复造轮子”，而建议直接使用Curator客户端中的各种官方实现的分布式锁，例如其中的InterProcessMutex可重入锁。

Curator 可重入分布式锁工作流程

https://www.processon.com/view/link/5cadacd1e4b0375afbef4320



总结

优点：ZooKeeper分布式锁（如InterProcessMutex），具备高可用、可重入、阻塞锁特性，可解决失效死锁问题，使用起来也较为简单。

缺点：因为需要频繁的创建和删除节点，性能上不如Redis。

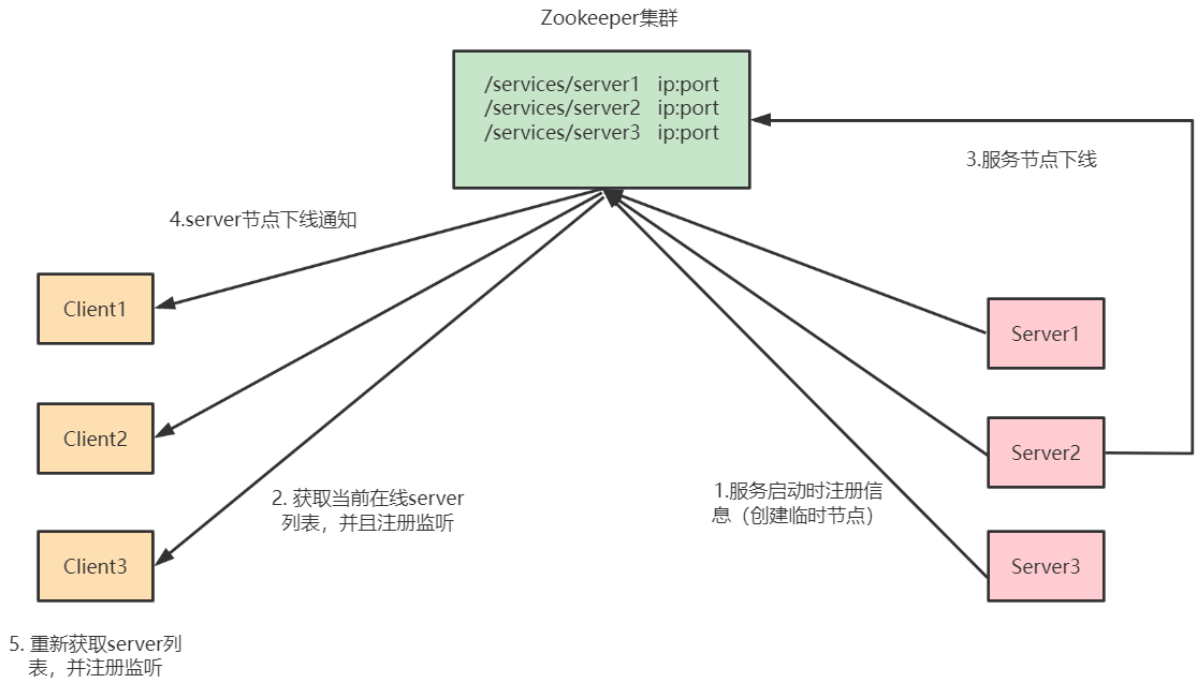
在高性能、高并发的应用场景下，不建议使用ZooKeeper的分布式锁。而由于ZooKeeper的高可用性，因此在并发量不是太高的应用场景中，还是推荐使用ZooKeeper的分布式锁。

Zookeeper注册中心实战

用于服务注册和服务发现　　CP

基于 ZooKeeper 本身的特性可以实现注册中心

https://spring.io/projects/spring-cloud-zookeeper#learn



第一步：在父pom文件中指定Spring Cloud版本

```
<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.3.2.RELEASE</version>
 <relativePath/> <!-- lookup parent from repository -->
</parent>
<properties>
 <java.version>1.8</java.version>
 <spring-cloud.version>Hoxton.SR8</spring-cloud.version>
</properties>
<dependencyManagement>
 <dependencies>
 <dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-dependencies</artifactId>
 <version>${spring-cloud.version}</version>
 <type>pom</type>
 <scope>import</scope>
 </dependency>
 </dependencies>
</dependencyManagement>
```

注意： springboot和springcloud的版本兼容问题

第二步：微服务pom文件中引入Spring Cloud Zookeeper注册中心依赖

```
<!-- zookeeper服务注册与发现 -->
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-zookeeper-discovery</artifactId>
 <exclusions>
 <exclusion>
 <groupId>org.apache.zookeeper</groupId>
 <artifactId>zookeeper</artifactId>
 </exclusion>
  </exclusions>
</dependency>

<!-- zookeeper client -->
<dependency>
 <groupId>org.apache.zookeeper</groupId>
 <artifactId>zookeeper</artifactId>
 <version>3.8.0</version>
</dependency>
```

注意： zookeeper客户端依赖和zookeeper sever的版本兼容问题

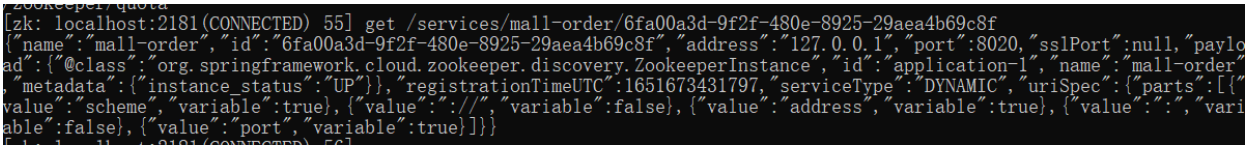
Spring Cloud整合Zookeeper注册中心核心源码入口：

ZookeeperDiscoveryClientConfiguration

第三步： 微服务配置文件application.yml中配置zookeeper注册中心地址

```
spring:
 cloud:
 zookeeper:
 connect-string: localhost:2181
 discovery:
 instance-host: 127.0.0.1
```

注册到zookeeper的服务实例元数据信息如下：



注意：如果address有问题，会出现找不到服务的情况，可以通过instance-host配置指定

第四步：整合feign进行服务调用

```
@RequestMapping(value = "/findOrderByUserId/{id}")
public R findOrderByUserId(@PathVariable("id") Integer id) {
```

```
3   log.info("根据userId:"+id+"查询订单信息");
4   //feign调用
5   R result = orderFeignService.findOrderByUserId(id);
6   return result;
7  }
```

测试：http://localhost:8040/user/findOrderByUserId/1