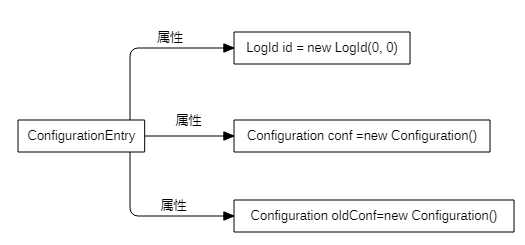
# 选举

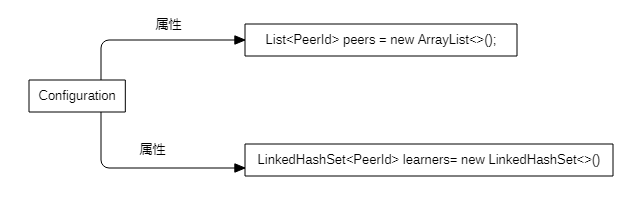
## ConfigurationEntry分析



1. boolean isStable()  
   1.1 功能：判断属性oldConf是否为空
2. boolean isEmpty()  
   2.1 功能：判断属性conf是否为空
3. Set<PeerId> listPeers()  
   3.1 功能：将属性conf和oldConf放入Set集合进行去重
4. boolean isValid()  
   4.1 功能：  
    （1）如果属性conf包含learners集合中元素，则返回false

（2）如果属性conf和属性oldConf包含learners集合中元素，则返回false

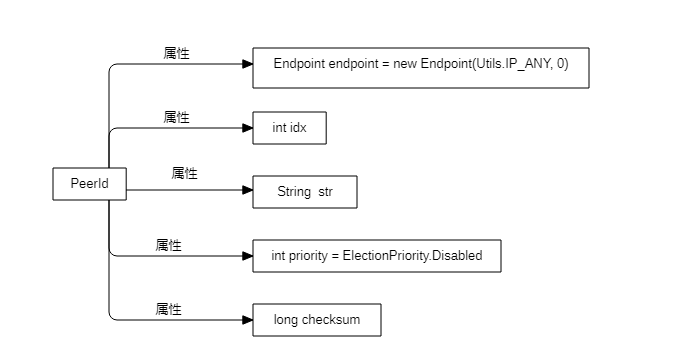
## Configuration类分析



1. boolean parse(final String conf)  
 1.1 参数conf：多个配置用逗号分割  
 1.2 功能：  
 （1）如果Learner，则配置示例为：localhost:8081/learner  
 🡪添加到learners属性中  
 （2）如果不是Learner，则添加到peers属性中  
 🡪最终解析到PeerId#parse(final String s)

2. boolean isValid()  
2.1 功能：返回peers里是否包含learners集合中元素

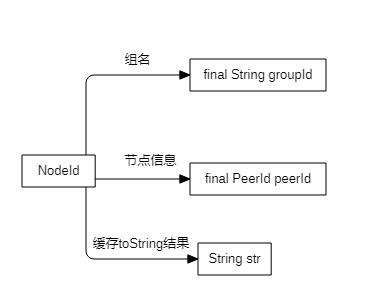
## PeerId类分析



Boolean parse(final String s)

1.1 支持的参数格式  
 （1）a:b  
 （2）a:b:c  
 （3）a:b::d  
 （4）a:b:c:d（ip:port:idx:priority）  
 1.2 功能：将参数分别解析到endpoint、idx、priority属性中

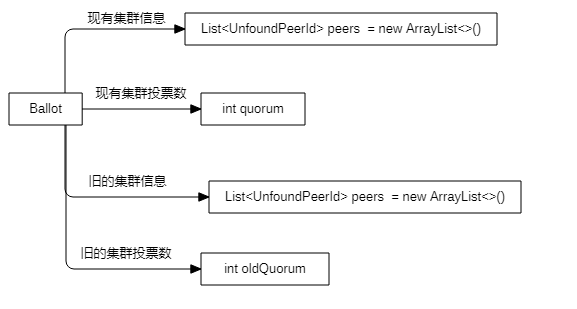
## NodeId类分析



## NodeImpl类分析

1. 构造方法：NodeImpl(final String groupId,final PeerId serverId)  
   1.1 初始化属性：groupId、serverId、state、currTerm

## Ballot类分析（投票的选票）



# RPC

## AbstractClientService(详情见mdj文件中的RPC-AbstractClientService)

### 静态块加载协议消息工厂

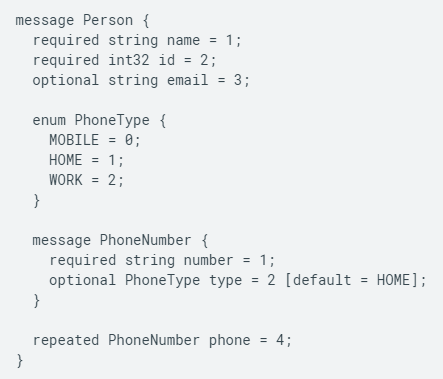
1. 采用[protocolbuffers](https://github.com/protocolbuffers/protobuf)

### Init(final RpcOptions)初始化属性：rpcClient、rpcExecutor、rpcOptions

# Protocolbuf

1. 协议编译器安装
   1. [下载预构建的二进制文件](https://github.com/protocolbuffers/protobuf/releases)
2. 什么是protocol buffer?
   1. 是一种用于序列化结构化数据的灵活、高效、自动化的机制  
       🡪以XML为例，但更小、更快、更简单
   2. 你想要一次构造数据的方式，然后可以使用生成的特殊源代码  
       🡪 轻松地使用各种语言在各种数据流中写入和读取结构化数据
   3. 甚至可以更新数据结构，而不会破坏已针对“旧”格式编译的已部署程序
3. 如何工作？
   1. 通过在.proto文件中定义协议缓冲区消息类型，你可以指定要序列化的信息的结构
   2. 每个协议缓冲区消息都是一个小的逻辑信息记录

🡪其中包含一系列name-value对



* 1. 每个消息类型都有一个或多个唯一编号的字段
* 2. 每个字段都有一个name和一个value类型  
   🡪 其中value类型可以是numbers(integer 或者 float)、boolean、strings、原始字节或者甚至如上所示（其他协议缓冲区消息类型）

🡪 从而可以分层地构造数据

* 3. 可以指定可选字段、必填字段和重复字段

--------------------------------- 定义消息(上述) --------------------------------------------------------

* 1. 在定义消息后，可以在.proto文件上为应用程序的语言运行协议缓冲器编译器，以生成数据访问类

3.4 将整个结构序列化为原始字节或从原始字节中解析出整个结构的方法

  
 然后读回你的消息，添加新字段，而不会破坏向后兼容性



# SPI

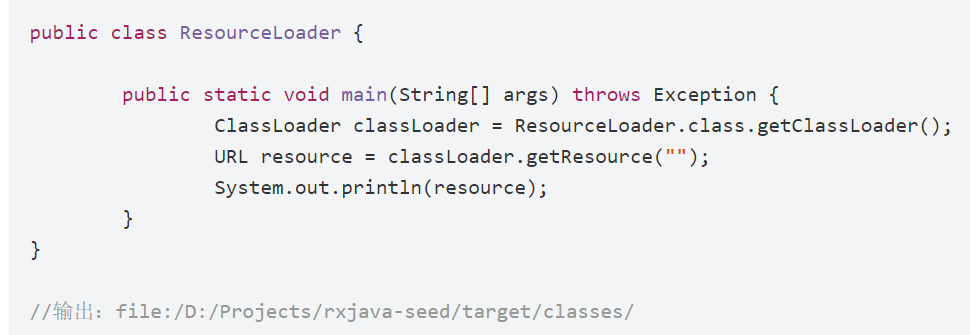
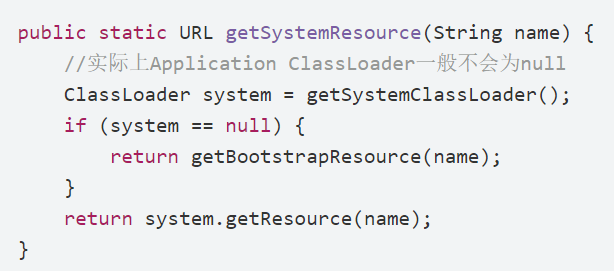
## SPI介绍

1. SPI是JDK内置的一种服务提供发现机制
2. SPI是一种动态替换发现的机制

## 类加载机制

<https://www.cnblogs.com/throwable/p/9785944.html>

### ClassLoader提供的资源加载API

1. ClassLoader#getResource(String name)方法中查找的路径  
   🡪是基于用户程序的ClassPath搜索资源
   1. 验证方法（输出结果就是当前应用的ClassPath）  
      
2. ClassLoader#getSystemResource(String name)  
   2.1 如果应用程序类加载器为null，则使用启动类加载器进行资源加载  
    🡪 如果应用程序类加载器不为null的情况下，实际上退化为ClassLoader#getResource

### 线程上下文类加载器Thread Context ClassLoader

1. 双亲委派缺陷

1.1 无法解决基础类需要回调用户的代码  
 🡪典型例子：JNDI  
 （1）JNDI的类库代码是启动类加载器加载的  
 （2）需要调用在应用的ClassPath的JNDI的SPI的代码  
 🡪 启动类加载器无法加载ClassPath下的类库

1. 这个类加载器可以通过java.lang.Thread类的setContextClassLoader()设置  
   🡪 JNDI服务可以使用线程上下文类加载器去加载所需的SPI类库