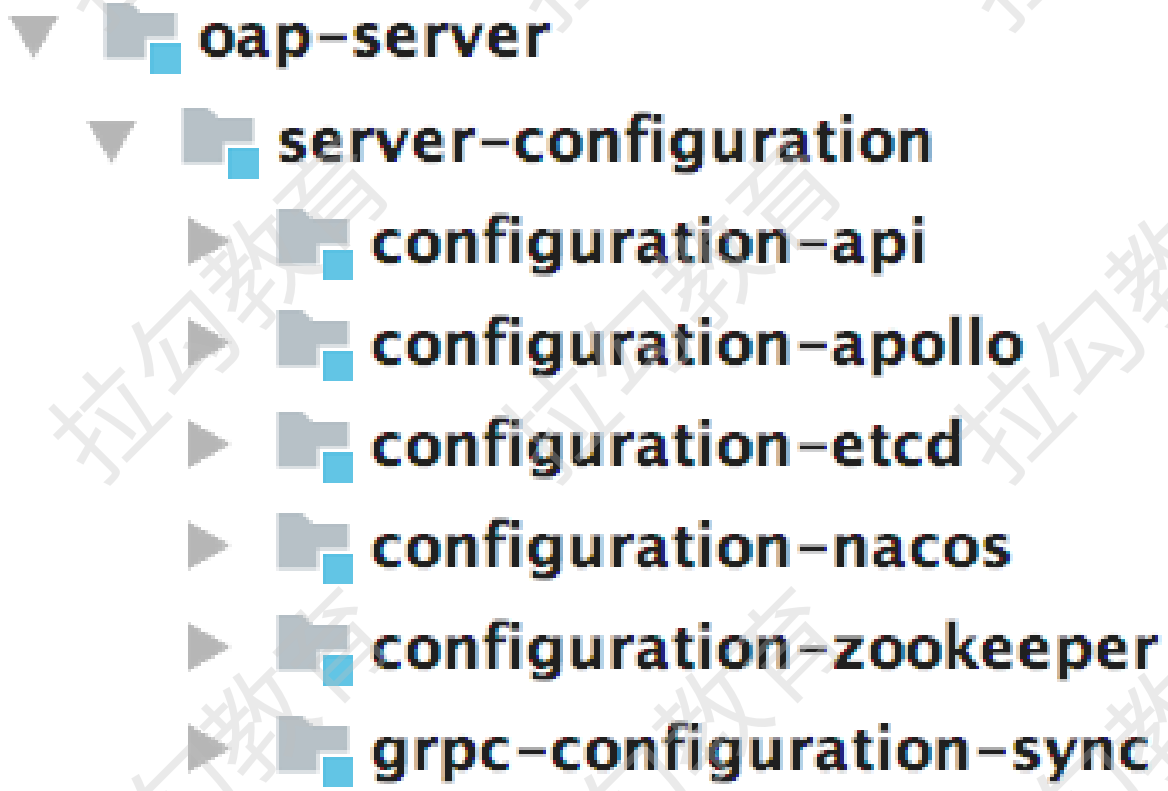
《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

第20讲：深入剖析 Configuration 插件 实现可插拔接入多种配置中心

configuration-api 模块



configuration-api 模块


```
configuration:
  none:
#   apollo:
#     apolloMeta: http://106.12.25.204:8080
#     apolloCluster: default
#     # apolloEnv: # defaults to null
#     appId: skywalking
#     period: 5
#   nacos:
#     # Nacos Server Host
#     serverAddr: 127.0.0.1
#     # Nacos Server Port
#     port: 8848
#     # Nacos Configuration Group
#     group: 'skywalking'
#     # Unit seconds, sync period. Default fetch every 60 seconds.
#     period : 5
#     # the name of current cluster, set the name if you want to upstream system known.
#     clusterName: "default"
```

configuration-api 模块对应的配置

configuration-apollo 模块相关的配置

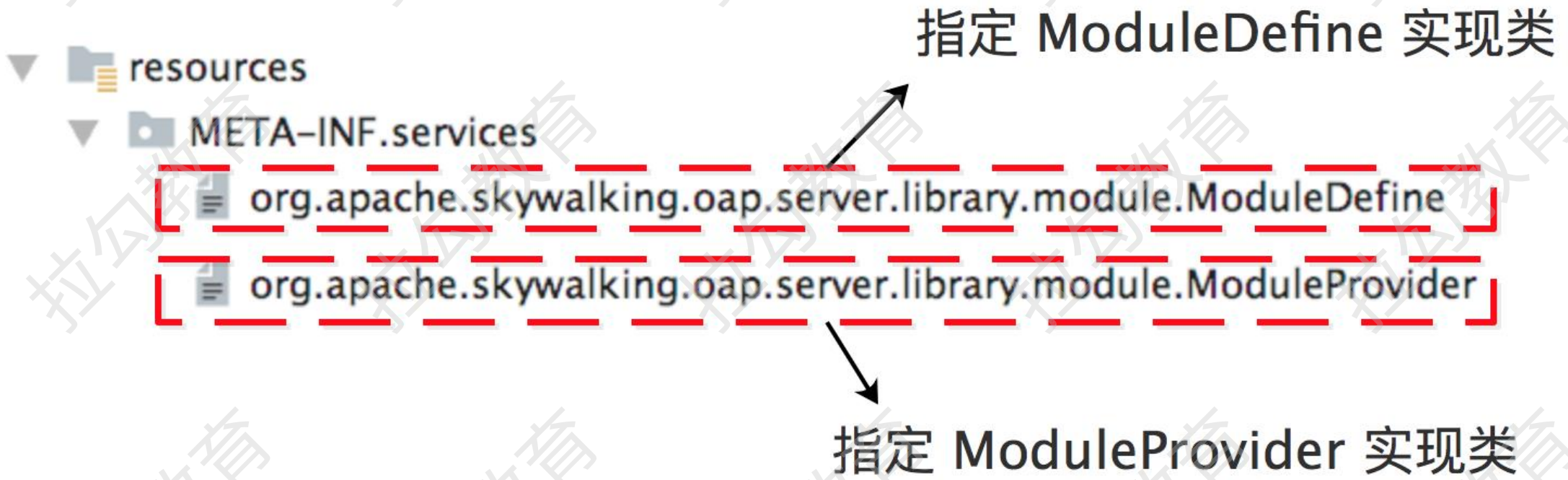
configuration-nacos 模块相关的配置

configuration-api 模块


```
#  nacos:
#    # Nacos Server Host
#    serverAddr: 127.0.0.1
#    # Nacos Server Port
#    port: 8848
#    # Nacos Configuration Group
#    group: 'skywalking'
#    # Unit seconds, sync period. Default fetch every 60 seconds.
#    period : 5
#    # the name of current cluster, set the name if you want to upstream system known.
#    clusterName: "default"
#  zookeeper:
#    period : 60 # Unit seconds, sync period. Default fetch every 60 seconds.
#    nameSpace: /default
#    hostPort: localhost:2181
#    #Retry Policy
#    baseSleepTimeMs: 1000 # initial amount of time to wait between retries
#    maxRetries: 3 # max number of times to retry
```

configuration-zookeeper 模块相关的配置

configuration-api 模块



configuration-api 模块


```
public Class[] services() {
    return new Class[] {DynamicConfigurationService.class};
}
```

configuration-api 模块

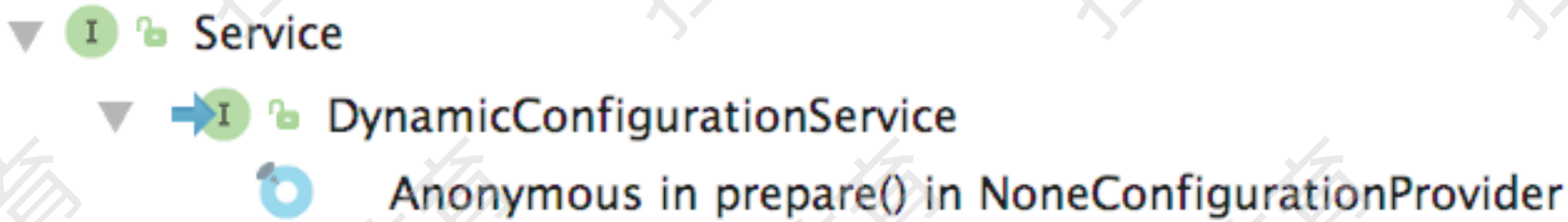

```
public Class[] services(){
    return new Class[] {DynamicConfigurationService.class};
}
```

DynamicConfigurationService 接口中只定义了一个
registerConfigChangeWatcher 方法
用于注册 ConfigChangeWatcher 这个监听器

```
public interface DynamicConfigurationService extends
Service {
    void registerConfigChangeWatcher(ConfigChangeWatcher
watcher);
}
```

configuration-api 模块


```
public interface DynamicConfigurationService extends
Service {
   void registerConfigChangeWatcher(ConfigChangeWatcher
watcher);
}
```



其中一个实现是在 NoneConfigurationProvider 中定义的匿名类（该实现为空实现）

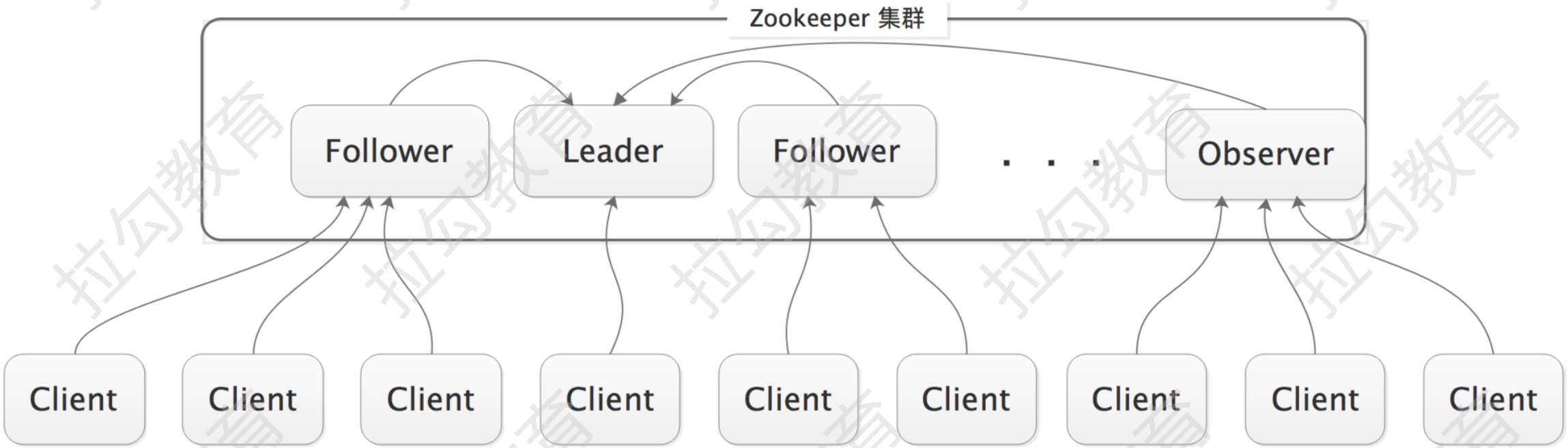
Zookeeper 基础速读

Apache ZooKeeper

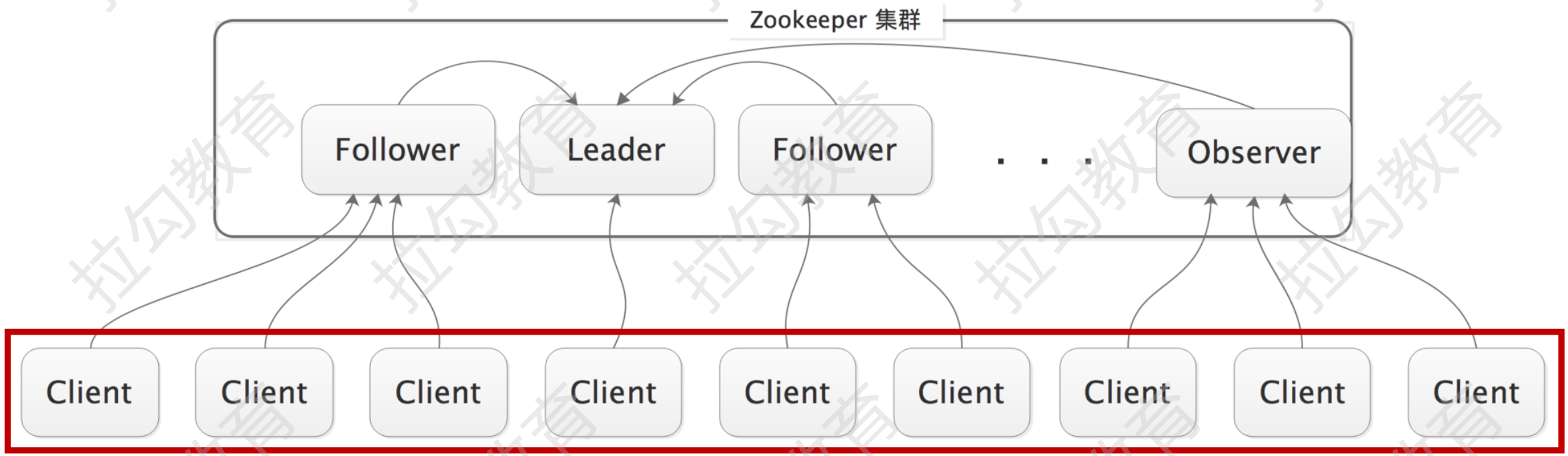
是一个针对分布式系统的、可靠的、可扩展的协调服务

通常作为统一命名服务、统一配置管理、分布式集群管理（注册中心）、分布式锁服务、Leader 选举服务等角色出现

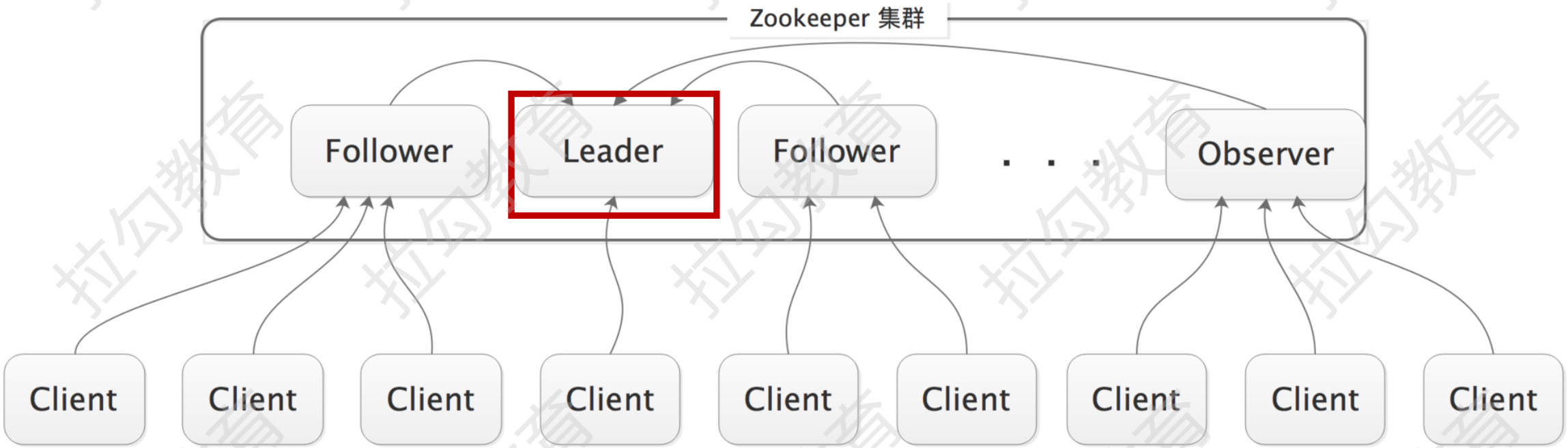
Zookeeper 基础速读



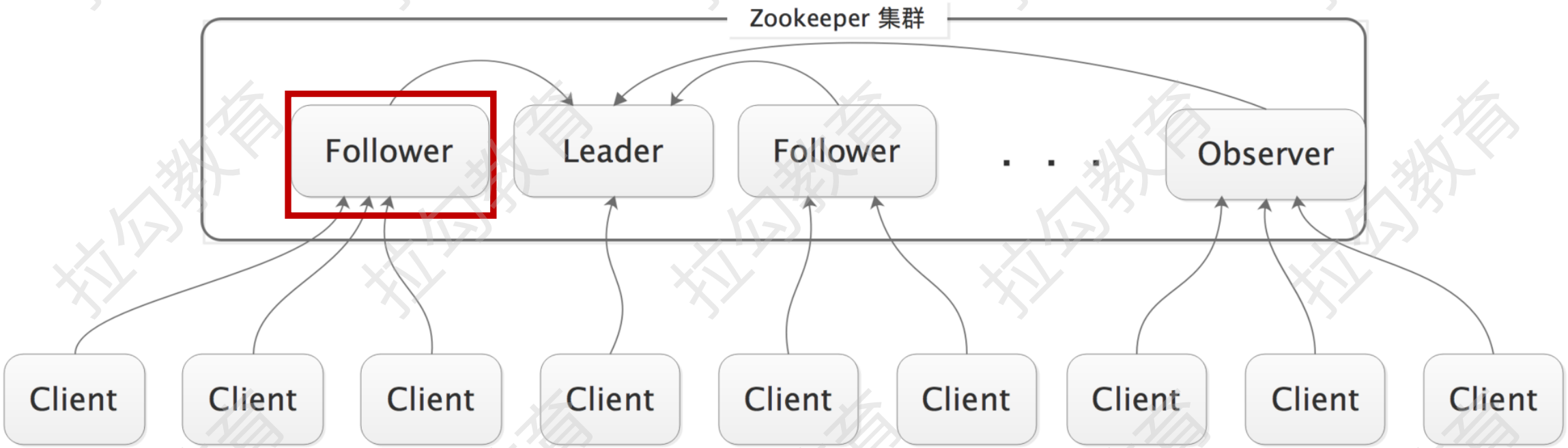
Zookeeper 基础速读



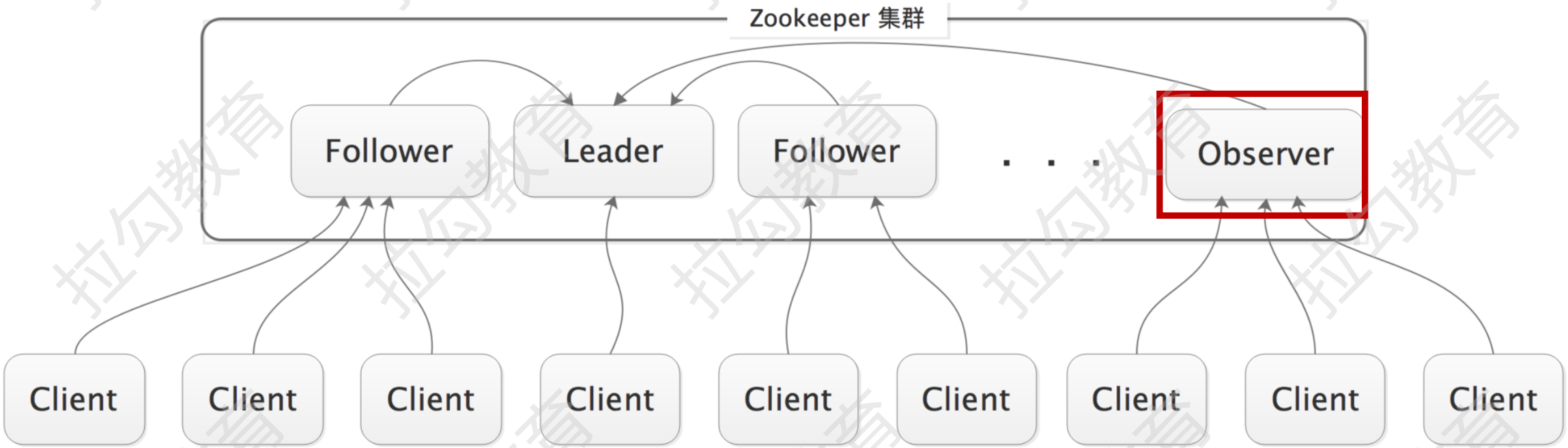
Zookeeper 基础速读



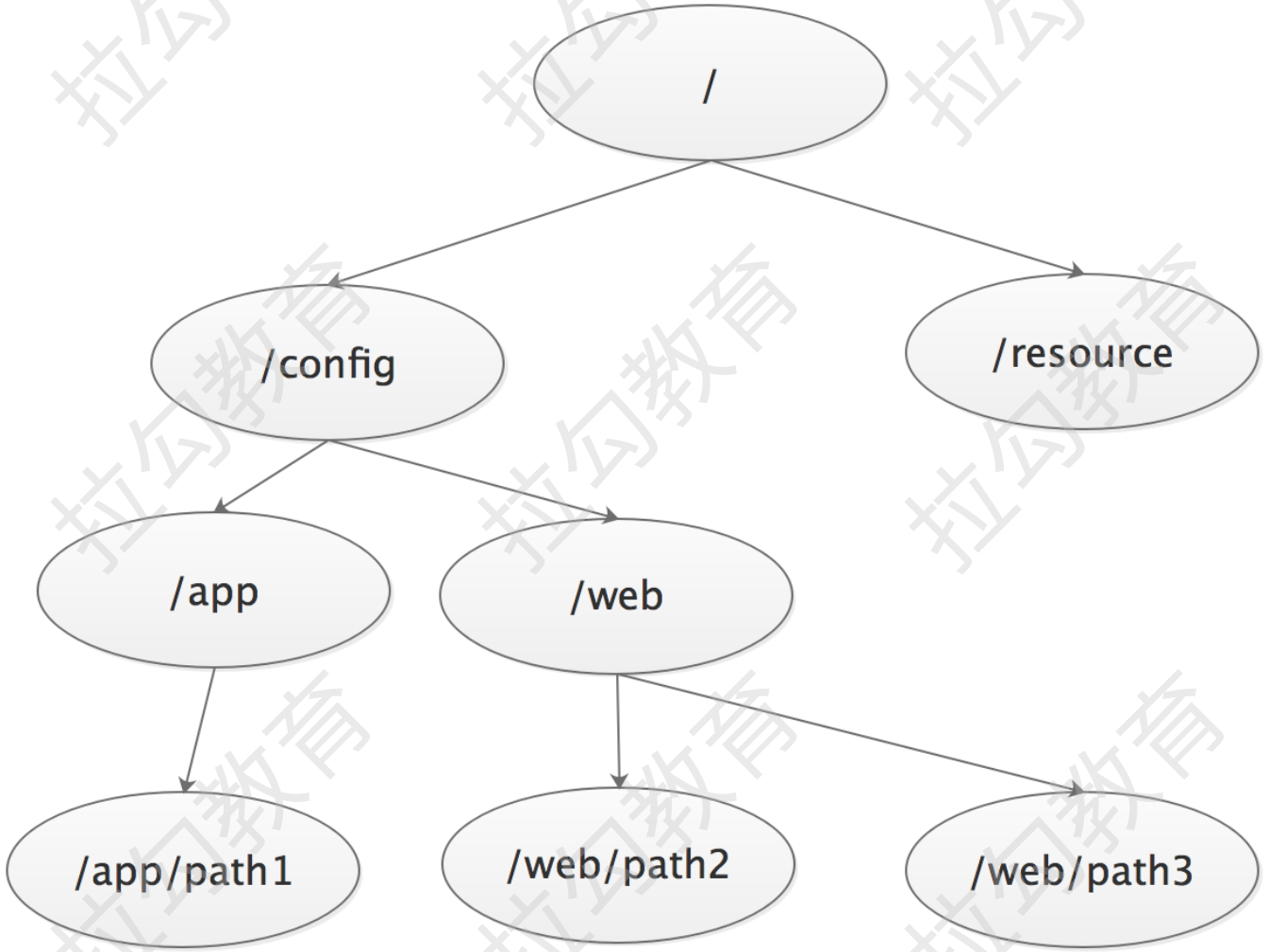
Zookeeper 基础速读



Zookeeper 基础速读



Zookeeper 基础速读



Zookeeper 基础速读

ZNode 节点类型有如下四种：

- 持久节点：节点创建后，会一直存在，不会因创建该节点的 Client 会话失效而删除
- 持久顺序节点：基本特性与持久节点一致，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
- 临时节点：创建该节点的 Client 的会话失效后，该节点会被自动删除
  另外，在临时节点下面不能创建子节点
- 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

Zookeeper 基础速读

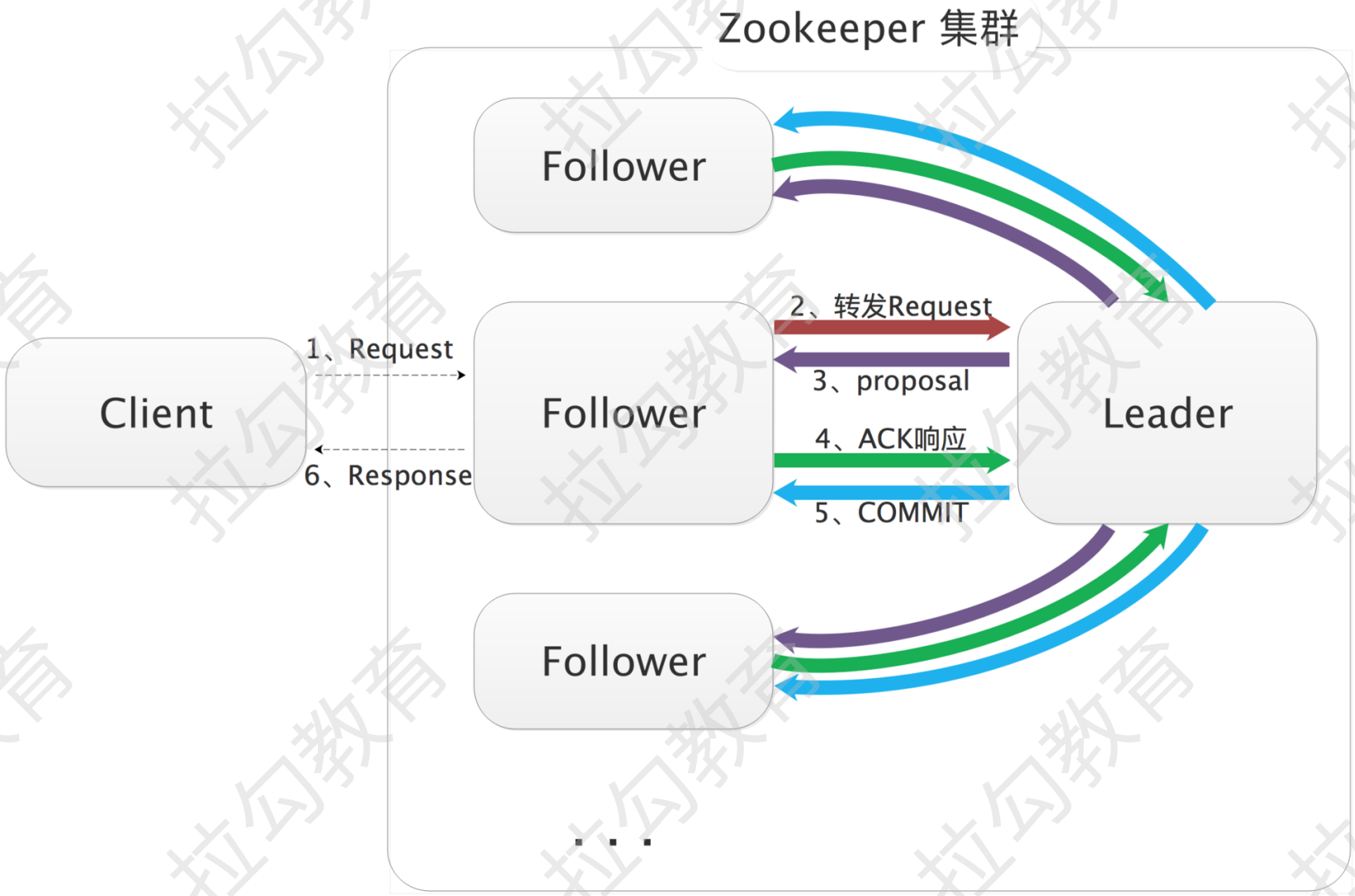
序号	属性	数据结构	描述	
1	czxid	long	节点被创建的Zxid值	事务ID可以识别出请求的全局顺序
2	mzxid	long	节点被修改的Zxid值	
3	pzxid	long	子节点最有一次被修改时的事务ID	
4	ctime	long	节点被创建的时间	
5	mtime	long	节点最后一次被修改的时间	
6	versoin	long	节点被修改的版本号,	基于CAS理论保证分布式数据原子性操作
7	cversion	long	节点的所拥有子节点被修改的版本号	
8	aversion	long	节点的ACL被修改的版本号	
9	emphemeralOwner	long	如果此节点为临时节点，那么它的值为这个节点 拥有者的会话ID；否则，它的值为0	
10	dataLength	int	节点数据域的长度	
11	numChildren	int	节点拥有的子节点个数	

Zookeeper 基础速读

- **主动推送**：Watcher 被触发时，由 ZooKeeper 集群主动将更新推送给客户端，而不需要客户端轮询
- **一次性**：数据变化时，Watcher 只会被触发一次

  如果客户端想得到后续更新的通知，必须要在 Watcher 被触发后重新注册一个 Watcher
- **可见性**：如果一个客户端在读请求中附带 Watcher，Watcher 被触发的同时再次读取数据，客户端在得到 Watcher 消息之前肯定不可能看到更新后的数据。换句话说，更新通知先于更新结果
- **顺序性**：如果多个更新触发了多个 Watcher ，那 Watch 被触发的顺序与更新顺序一致

基本工作原理概述

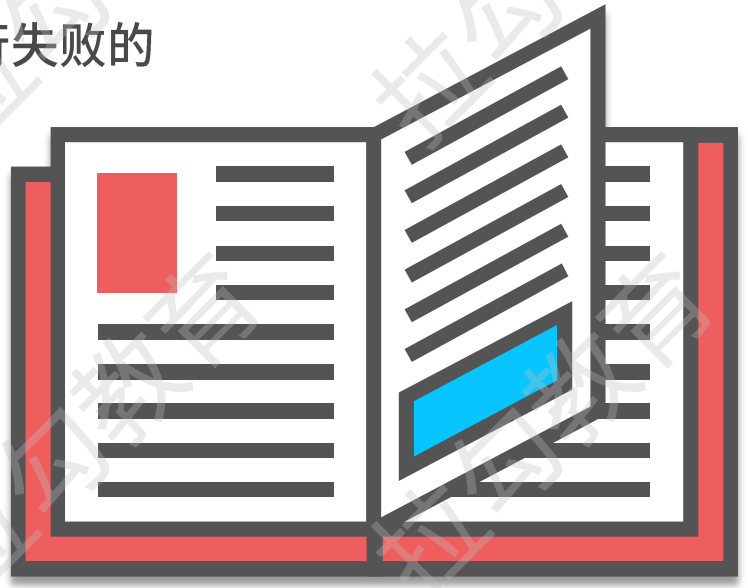


基本工作原理概述

- 当 Leader 节点收到半数以上 Follower 节点的 ACK 响应之后，向各个 Follower 节点广播 COMMIT 命令

  同时也会在本地执行 COMMIT 并向连接的客户端进行响应

  如果在各个 Follower 在收到 COMMIT 命令前 Leader 就宕机了，导致剩下的服务器并没有执行这条消息
- 当 Leader 节点生成 proposal 之后就宕机了，而其他 Follower 并没有收到此 proposal（或者只有一小部分 Follower 节点收到了这条 proposal），那么此次写操作就是执行失败的



基本工作原理概述

当前集群中有 5 个 ZooKeeper 节点构成

sid 分别为 1，2，3，4，5，zxid 分别为 10，10，9，9，8

此时，**sid** 为 1 的节点是 Leader 节点

实际上，**zxid** 包含了 epoch 和自增计数器两部分

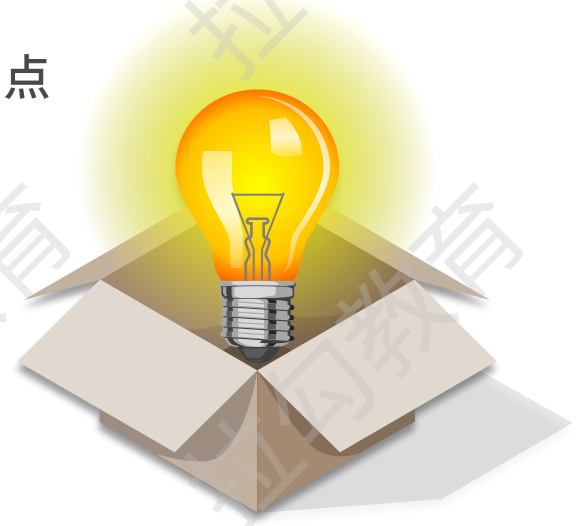
epoch 是纪元的意思，标识当前 Leader 周期，每次选举时 epoch 部分都会递增

这就防止了网络隔离之后，原始 Leader 重新连入集群造成不必要的重新选举

该示例中假设各个节点的 epoch 都相同

基本工作原理概述

- 对于节点 2 来说，接收到（3，9）、（4，9）、（5，8）的投票，对比后发现自己的 zxid 最大因此不需要做任何投票变更
- 对于节点 3 来说，接收到（2，10）、（4，9）、（5，8）的投票，对比后由于 2 的 zxid 比自己的 zxid 要大，因此需要更改投票，改投（2，10），并将改投后的票发给其他节点
- 对于节点 4 来说，接收到（2，10）、（3，9）、（5，8）的投票，对比后由于 2 的 zxid 比自己的 zxid 要大，因此需要更改投票，改投（2，10），并将改投后的票发给其他节点
- 对于节点 5 来说，也是一样，最终改投（2，10）



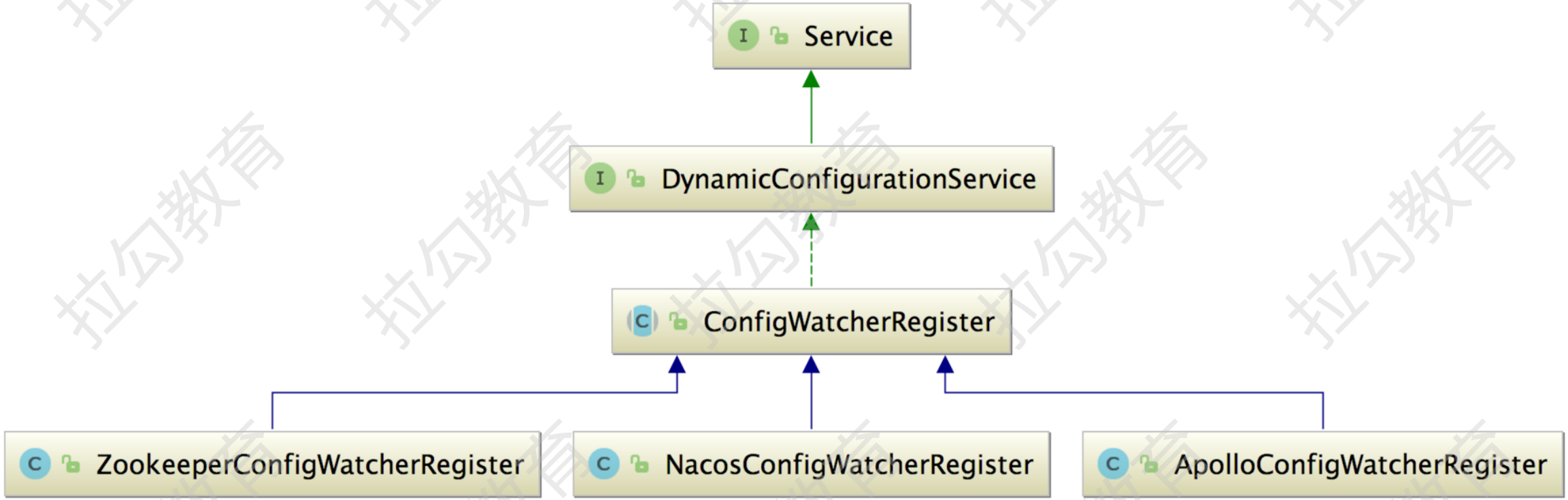
初识 Apache Curator

- ZooKeeper 的 Watcher 是一次性的，每次触发之后都需要重新进行注册
- 会话超时之后没有实现自动重连的机制
- ZooKeeper 提供的异常非常烦琐，对新手开发来说，非常不友好
- 只提供了简单的 byte[] 数组的接口，没有提供基本类型以及对象级别的序列化
- 创建节点时如果节点存在抛出异常，需要自行检查节点是否存在
- 删除节点无法实现级联删除

初识 Apache Curator

名称	描述
curator-framework	ZooKeeper API 的高层封装，简化 ZooKeeper 客户端编程 添加了 ZooKeeper 连接管理、重试机制、重复注册 Watcher 等功能
curator-recipes	ZooKeeper 典型应用场景的实现，这些实现是基于 Curator Framework 例如，Leader 选举、分布式锁、Barrier、分布式队列等等
curator-client	ZooKeeper Client的封装，用于取代原生 ZooKeeper 客户端 提供一些非常有用的客户端特性
curator-x-discovery	在 curator-framework 上构建的服务发现实现
curator-x-discoveryserver	可以和 curator-x-discovery 一起使用的 RESTful 服务器
curator-examples	各种使用 Curator 特性的案例

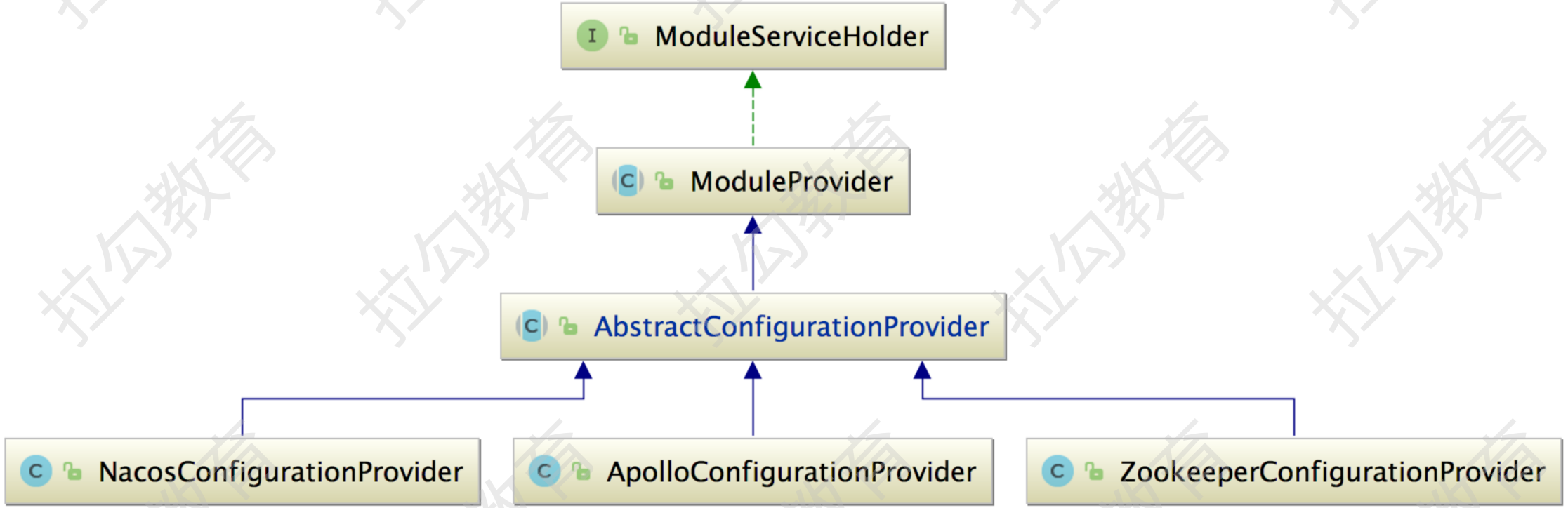
基于 ZooKeeper 的配置管理



基于 ZooKeeper 的配置管理


```
public void prepare(){
  //初始化ConfigWatcherRegister，这里的initConfigReader()是个抽象方法，
  //具体使用哪个 ConfigWatcherRegister实现类由子类决定。
   configWatcherRegister = initConfigReader();
   this.registerServiceImplementation(
    DynamicConfigurationService.class, configWatcherRegister);
}
```

基于 ZooKeeper 的配置管理



基于 ZooKeeper 的配置管理

拉勾教育
互联网人实战大学

```
public ZookeeperConfigWatcherRegister(
      ZookeeperServerSettings settings) throws Exception {
    super(settings.getPeriod()); // ????
    prefix = settings.getNameSpace() + "/" ; // ZK节点的前缀
    //根据配置创建重试策略
    RetryPolicy retryPolicy = new ExponentialBackoffRetry(
        settings.getBaseSleepTimeMs(), settings.getMaxRetries());
    //根据配置指定的地址，创建CuratorFramework客户端
    CuratorFramework client = CuratorFrameworkFactory.newClient(
      settings.getHostPort(), retryPolicy);
    client.start();
    // 创建PathChildrenCache
    this.childrenCache = new PathChildrenCache(client,
       settings.getNameSpace(), true);
    this.childrenCache.start();
}
```

基于 ZooKeeper 的配置管理

ZookeeperConfigWatcherRegister 对 readConfig() 这个抽象方法的实现也**比较简单**

直接从这个 PathChildrenCache 缓存中读取最新数据并整理成 ConfigTable 对象返回即可

60s 的定时轮训线程会执行 configSync() 方法之中完成新旧配置值的比较并通知相应的

ConfigWatcherRegister 对象

Next：第19讲《Cluster 插件剖析，你想要的集群模式它都有》

拉勾教育

—互联网人实战大学—

关注拉勾「教育公众号」
获取更多课程信息

L / A / G / O / U