使用 ZooKeeper 实现服务发现(1)

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程

服务发现

服务发现主要应用于微服务架构和分布式架构场景下。在这些场景下，一个服务通常需要松耦合的多个组件的协同才能完成。服务发现就是让组件发现相关的组件。服务发现要提供的功能有以下3点：

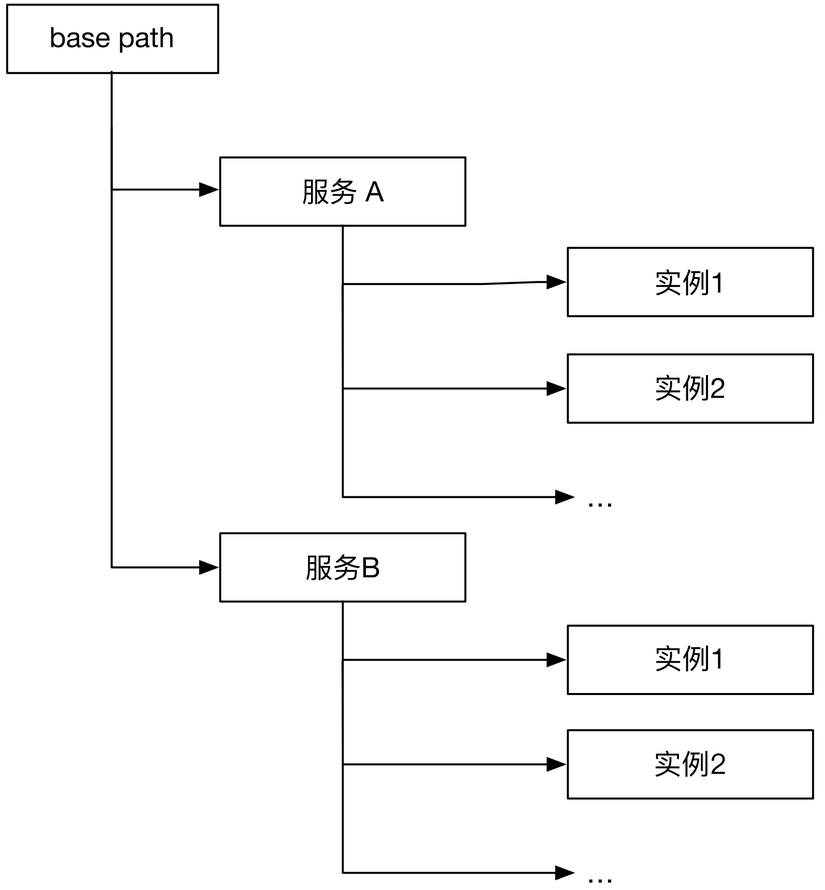
- 服务注册。
- 服务实例的获取。
- 服务变化的通知机制。

Curator 有一个扩展叫作 curator-x-discovery 。curator-x-discovery 基于 ZooKeeper 实现了服务发现。

curator-x-discovery 设计

使用一个 base path 作为整个服务发现的根目录。在这个根目录下是各个服务的的目录。服务目录下面是服务实例。实例是服务实例的 JSON 序列化数据。服务实例对应的 znode 节点可以根据需要设置成持久性、临时性和顺序性。

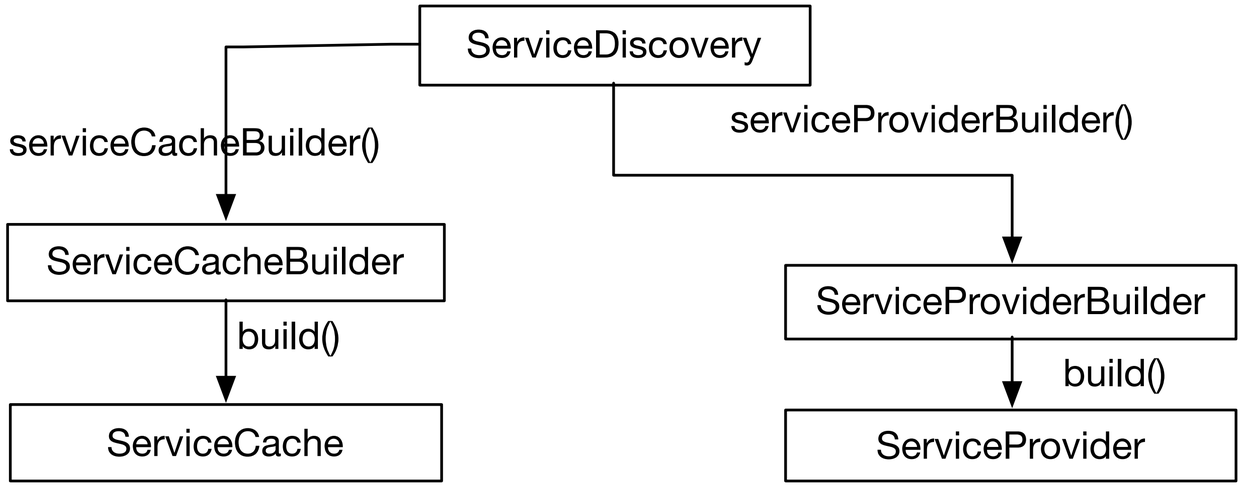
znode数据模型



核心接口

左图列出了服务发现用户代码要使用的 curator-x-discovery 接口。最主要的有以下三个接口：

- ServiceProvider :在服务 cache 之上支持服务发现操作，封装了一些服务发现策略。
- ServiceDiscovery :服务注册，也支持直接访问 ZooKeeper 的服务发现操作。
- ServiceCache :服务 cache 。

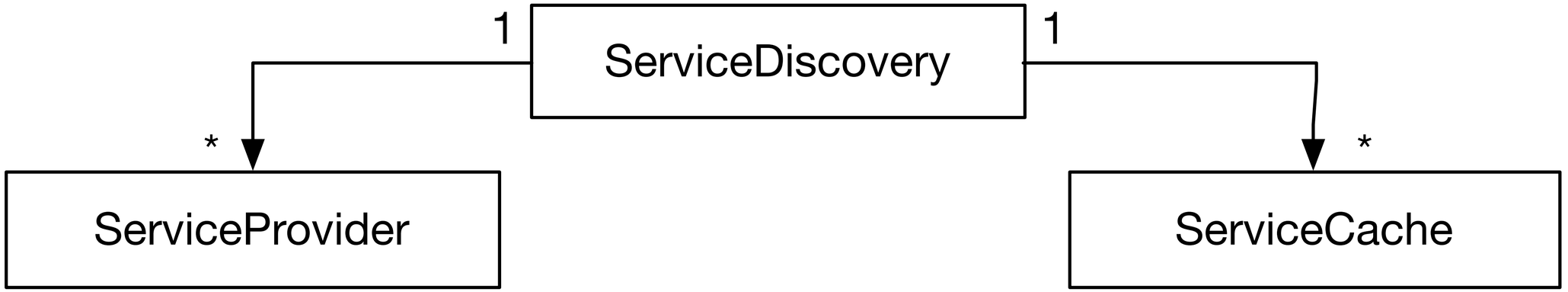


ServiceInstance

用来表示服务实例的 POJO，除了包含一些服务实例常用的成员之外，还提供一个 payload 成员让用户存自定义的信息。

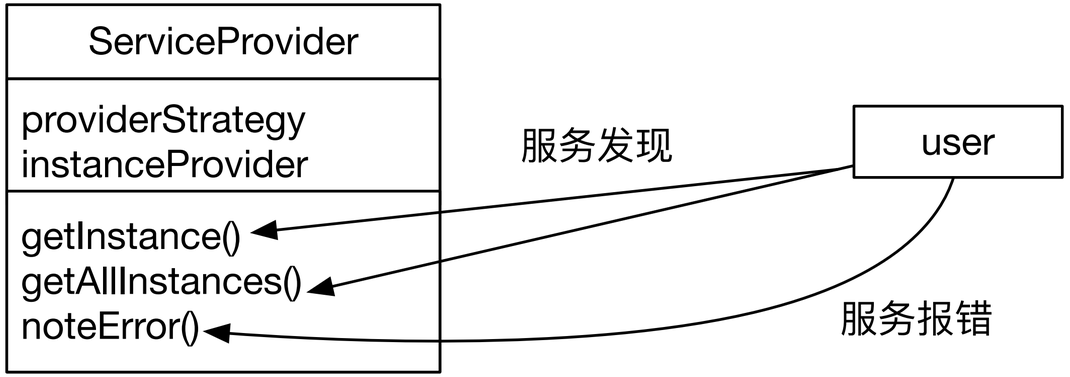
ServiceDiscovery

从一个 ServiceDiscovery ， 可以创建多个 ServiceProvider 和多个 ServiceCache 。



ServiceProvider

ServiceProvider 提供服务发现 high-level API 。ServiceProvider 是封装 ProviderStraegy 和 InstanceProvider 的 facade 。 InstanceProvider 的数据来自一个服务 Cache 。服务 cache 是 ZooKeeper 数据的一个本地 cache ，服务 cache 里面的数据可能会比 ZooKeeper 里面的数据旧一些。ProviderStraegy 提供了三种策略： 轮询, 随机和 sticky 。



ServiceProvider 除了提供服务发现的方法(getInstance 和 getAllInstances)以外，还通过 noteError 提供了一个让服务使用者把服务使用情况反馈给 ServiceProvider 的机制。

ServiceCache

展示代码。

ZooKeeper 交互

ServiceDiscovery 提供的服务注册方法是对 znode 的更新操作，服务发现方法是 znode 的读取操作。同时它也是最核心的类，所有的服务发现操作都要从这个类开始。

另外服务 Cache 会接受来自 ZooKeeper 的更新通知，读取服务信息（也就是读取 znode 信息）。

ServiceDiscovery、ServiceCache、ServiceProvider 说明

- 都有一个对应的 builder。这些 builder 提供一个创建这三个类的 fluent API 。
- 在使用之前都要调用 start 方法。
- 在使用之后都要调用 close 方法。close 方法只会释放自己创建的资源，不会释放上游关联的资源。例如 ServiceDiscovery 的 close 方法不会去调用 CuratorFramework 的 close 方法。

服务发现调用代码

使用 ZooKeeper 实现服务发现(2)

Node Cache

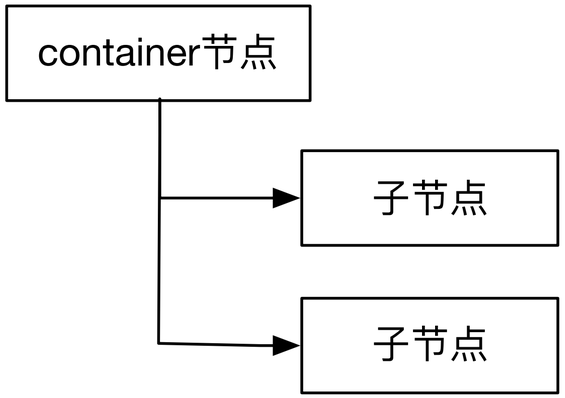
Node Cache 是 curator 的一个 recipe，用来本地 cache 一个 znode 的数据。Node Cache 通过监控一个 znode 的 update / create / delete 事件来更新本地的 znode 数据。用户可以在 Node Cache 上面注册一个 listener 来获取 cache 更新的通知。

Path Cache

Path Cache 和 Node Cache 一样，不同之处是在于 Path Cache 缓存一个 znode 目录下所有子节点。

container 节点

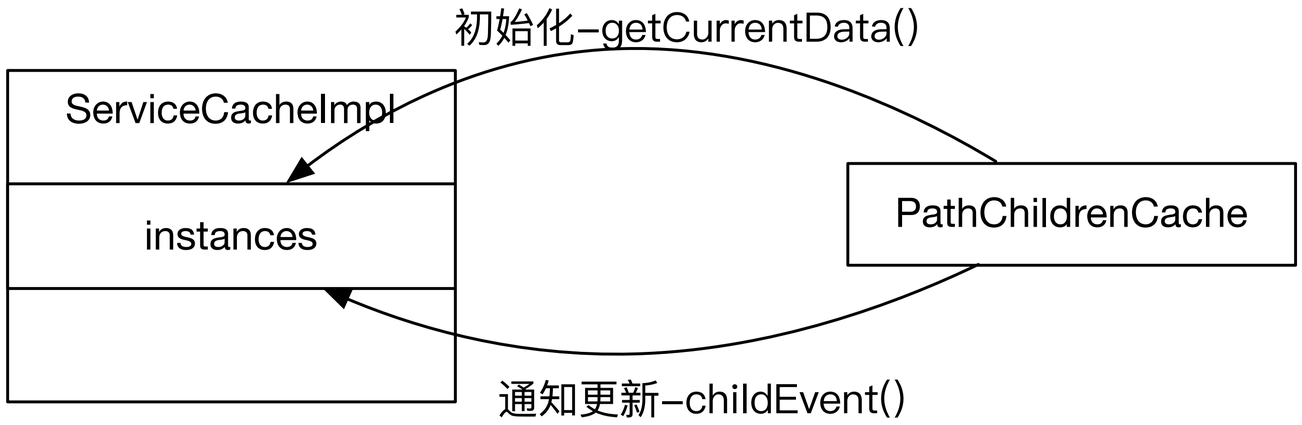
container 节点是一种新引入的 znode，目的在于下挂子节点。当一个 container 节点的所有子节点被删除之后，ZooKeeper 会删除掉这个 container 节点。服务发现的 base path 节点和服务节点就是 containe 节点。



ServiceDiscoveryImpl

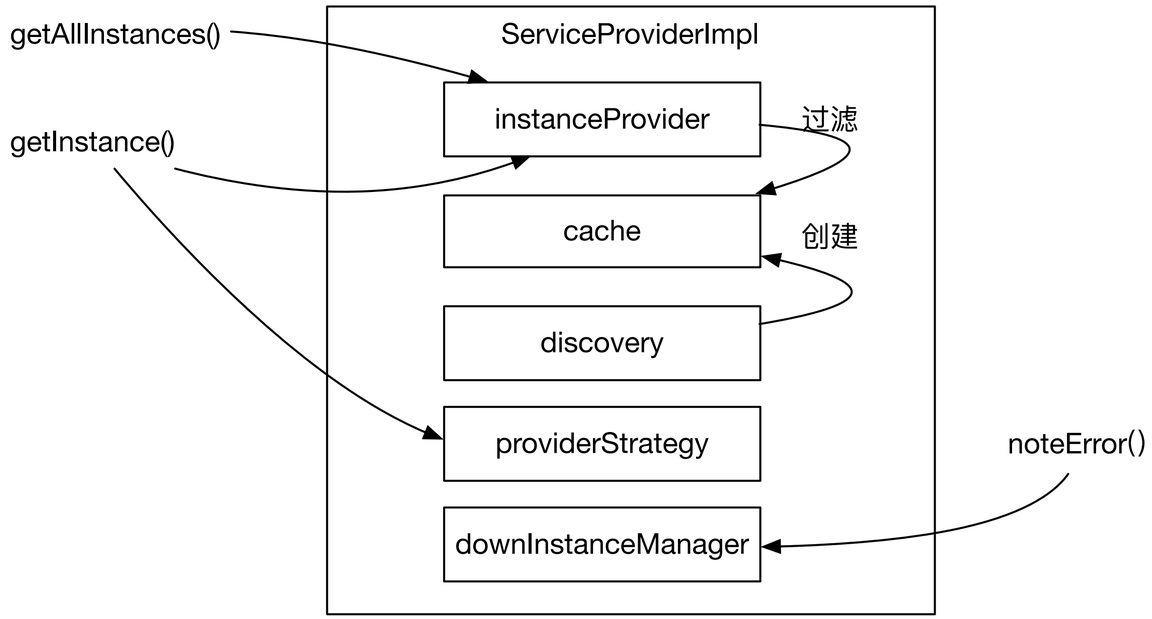
ServiceCacheImpl

ServiceCacheImpl 使用一个 PathChildrenCache 来维护一个 instances 。这个 instances也是对 znode 数据的一个 cache 。



ServiceProviderImpl

如下图所示，ServiceProviderImpl 是多个对象的 facade 。



ProviderStrategy

DownInstancePolicy

使用 ZooKeeper 实现服务发现(3)

curator-x-discovery-server 扩展

curator-x-discovery-server 是基于 curator-x-discovery 实现的对外提供服务发现的 HTTP API 。他的 HTTP API 是基于 JAX-RS 研发的。因为国内很少有人用 JAX-RS ， 我用 Spring Boot 重新实现了一下这个扩展。

代码展示

总结

curator-x-discovery 在系统质量和影响力和 ZooKeeper 相比还是有很大差距的，但是提供的服务发现的功能还是很完备的。如果我们的服务发现场景和 curator-x-discovery 匹配，就可以直接用它或者扩展它。curator-x-discovery-server 本身实现的功能很少，不建议使用，完全可以自己实现类似的功能。

进行 ZooKeeper API 开发，我个人建议以下的 SDK 使用优先顺序：

curator recipes -> curator framework -> ZooKeeper API 。

Kafka 是如何使用 ZooKeeper 的

ZooKeeper 在 Kafka 中的使用

Kafka 使用 ZooKeeper 实现了大量的协同服务。如果我们检查一个 Kafka 使用的 ZooKeeper，会发现大量的 znode：

```
ls /
[cluster, controller_epoch, controller, brokers, zookeeper, admin, isr_change_noti
fication, consumers, log_dir_event_notification, latest_producer_id_block, config]
```

ZooKeeper Directories 对一些关键的 ZooKeeper 使用有一个说明。Broker Node Registry 是用来保存 Kafka 集群的 Kafka 节点，是典型的组成员管理协同服务。我们下面会把这个协同服务做一个详细的讨论。

安装一个 Kafka 集群

使用 Confluent 的发行版

- 从 https://www.confluent.io/download/ 下载 confluent-community-5.3.0-2.12.tar.gz
- 在一个节点上的 etc/kafka/zookeeper.properties 文件中配置一下 dataDir，在 etc/kafka/server.properties 配置 log.dirs 、broker.id 和 zookeeper.connect 。
- 把 confluent-5.3.0 目录 rsync 到其他节点。在其他节点上更新 broker.id，创建 dataDir 目录和 log.dirs 目录。

broker 注册演示

- 启动 standalone 的 ZooKeeper 服务。
- 打开 zookeeper-shell，运行 ls /。
- 启动一个 Kafka 服务，使用 zookeeper-shell 检查 /brokers/ids 下的内容。
- 再启动一个 Kafka 服务，使用 zookeeper-shell 检查 /brokers/ids 下的内容。
- 杀掉一个 Kafka 服务，使用 zookeeper-shell 检查 /brokers/ids 下的内容。

multi API

ZooKeeper 的 multi 方法提供了一次执行多个 ZooKeeper 操作的机制。多个 ZooKeeper 操作作为一个整体执行，要么全部成功，要么全部失败。

另外 ZooKeeper 还提供了一个 builder 风格的 API 来使用 multi API 。

配置 Kafka 源代码

以下是在 Mac 上面配置 Kafka 源代码的步骤：

- git clone https://github.com/apache/kafka.git
- brew install gradle
- git fetch --all --tags --prune
- git checkout tags/2.3.0 -b 2.3.0
- gradle idea
- 使用 Idea 打开 Kafka 项目

Kafka 代码展示

总结

Kafka 在逐渐减少对 ZooKeeper 的依赖：

- 在老的版本中，committed offsets 是保存在 Kafka 中的。
- 未来 Kafka 还有计划完全移除对 ZooKeeper 的依赖。

Kafka 使用 ZooKeeper 的方式值得我们依赖。在我们刚开始一个分布式系统的时候，我们可以把协同数据都给 ZooKeeper 来管理，迅速让系统上线。如果在系统的后续使用中要对协同数据进行定制化处理，我们可以研发自己的协同数据机制来代替 ZooKeeper 。

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程