Zookeeper部署及典型应用

讲师：董西成
博客：dongxicheng.org
微信二维码见右。

Open Passion Value

目录

1. Zookeeper是什么
2. Zookeeper基本原理
3. Zookeeper应用场景
4. Zookeeper安装部署
5. Zookeeper客户端设计
6. 总结

目录

1. Zookeeper是什么
2. Zookeeper基本原理
3. Zookeeper应用场景
4. Zookeeper安装部署
5. Zookeeper客户端设计
6. 总结

Zookeeper是什么？

- 是一个针对大型分布式系统的可靠协调系统；
- 提供的功能包括：配置维护、名字服务、分布式同步、组服务等；
- 目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户；
- Zookeeper已经成为Hadoop生态系统中的基础组件。

Zookeeper特点

- 最终一致性：为客户端展示同一视图，这是zookeeper最重要的功能。
- 可靠性：如果消息被到一台服务器接受，那么它将被所有的服务器接受。
- 实时性：Zookeeper不能保证两个客户端能同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用sync()接口。
- 等待无关（wait-free）：慢的或者失效的client不干预快速的client的请求。
- 原子性：更新只能成功或者失败，没有中间状态。
- 顺序性：所有Server，同一消息发布顺序一致。

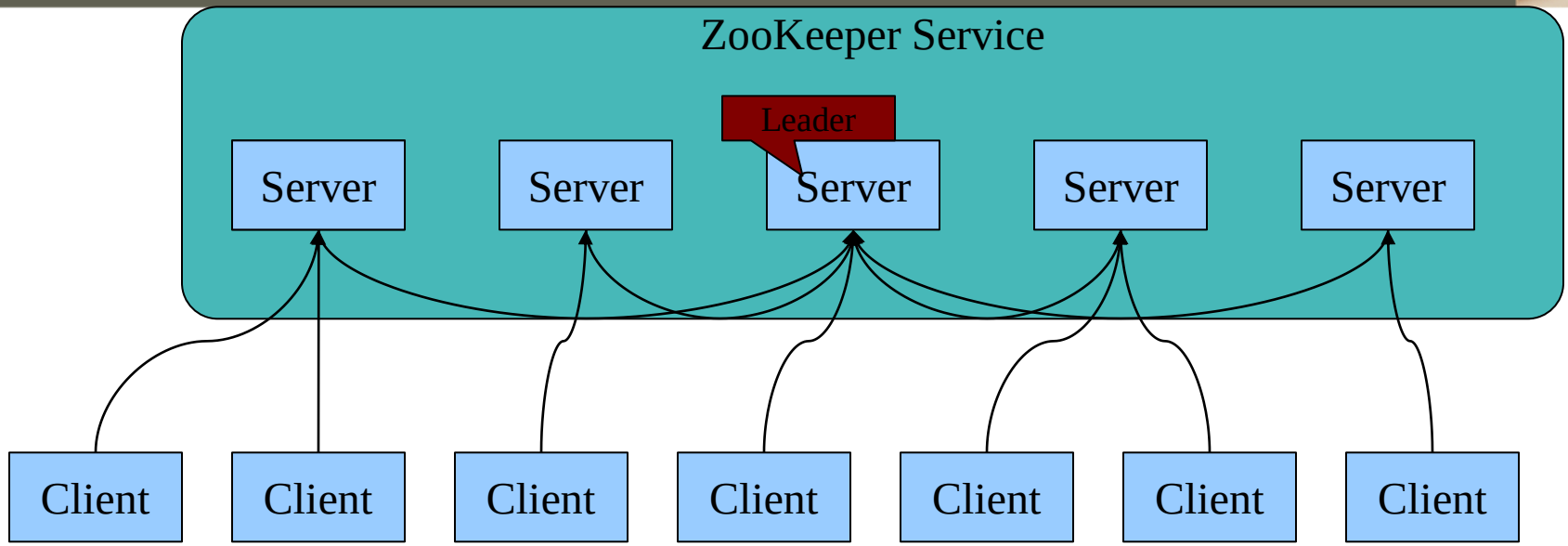
哪些系统用到了Zookeeper

- **HDFS**
- **YARN**
- **Storm**
- **HBase**
- **Flume**
- **Dubbo（阿里巴巴）**
- **metaq阿里巴巴）**

目录

1. Zookeeper是什么
2. Zookeeper基本原理
3. Zookeeper应用场景
4. Zookeeper安装部署
5. Zookeeper客户端设计
6. 总结

Zookeeper 架构



- 每个Server在内存中存储了一份数据；
- Zookeeper启动时，将从实例中选举一个leader（Paxos协议）；
- Leader负责处理数据更新等操作（Zab协议）；
- 一个更新操作成功，当且仅当大多数Server在内存中成功修改数据。

Zookeeper角色

角色		描述
领导者（Leader）		领导者负责进行投票的发起和决议，更新系统状态
学习者（Learner）	跟随者（Follower）	Follower 用于接收客户请求并向客户端返回结果，在选主过程中参与投票
	观察者（ObServer）	ObServer 可以接收客户端连接，将写请求转发给 leader 节点。但 ObServer 不参加投票过程，只同步 leader 的状态。ObServer 的目的是为了扩展系统，提高读取速度
客户端（Client）		请求发起方

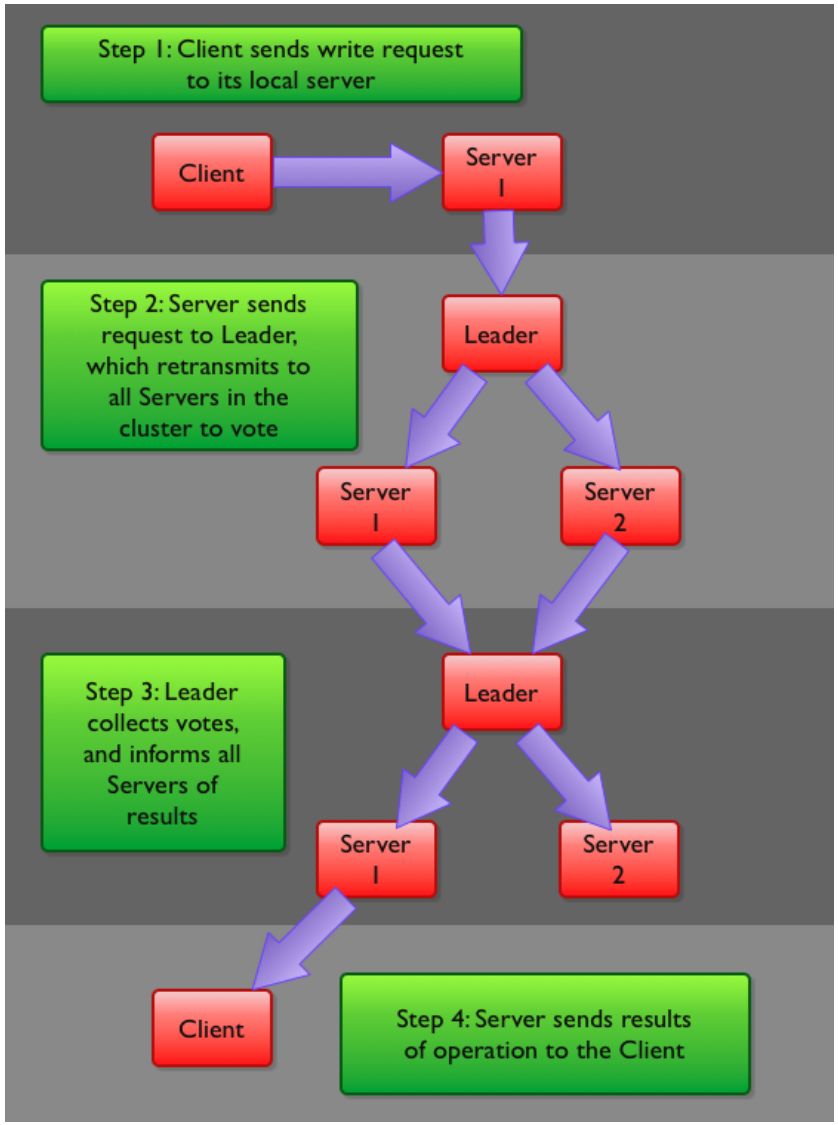
3.3.0版本新增角色Observer

- Zookeeper需保证高可用和强一致性;
- 为了支持更多的客户端，需要增加更多Server;
- Server增多，投票阶段延迟增大，影响性能;
- 权衡伸缩性和高吞吐率，引入Observer
  - ✓ Observer不参与投票;
  - ✓ Observers接受客户端的连接，并将写请求转发给leader节点;
  - ✓ 加入更多Observer节点，提高伸缩性，同时不影响吞吐率。

Zookeeper Server数目一般为奇数

- Leader选举算法采用了Paxos协议；
- Paxos核心思想：当多数Server写成功，则任务数据写成功
  - ✓ 如果有3个Server，则两个写成功即可；
  - ✓ 如果有4或5个Server，则三个写成功即可。
- Server数目一般为奇数（3、5、7）
  - ✓ 如果有3个Server，则最多允许1个Server挂掉；
  - ✓ 如果有4个Server，则同样最多允许1个Server挂掉
  - ✓ 既然如此，为啥要用4个Server?

Zookeeper 数据写流程



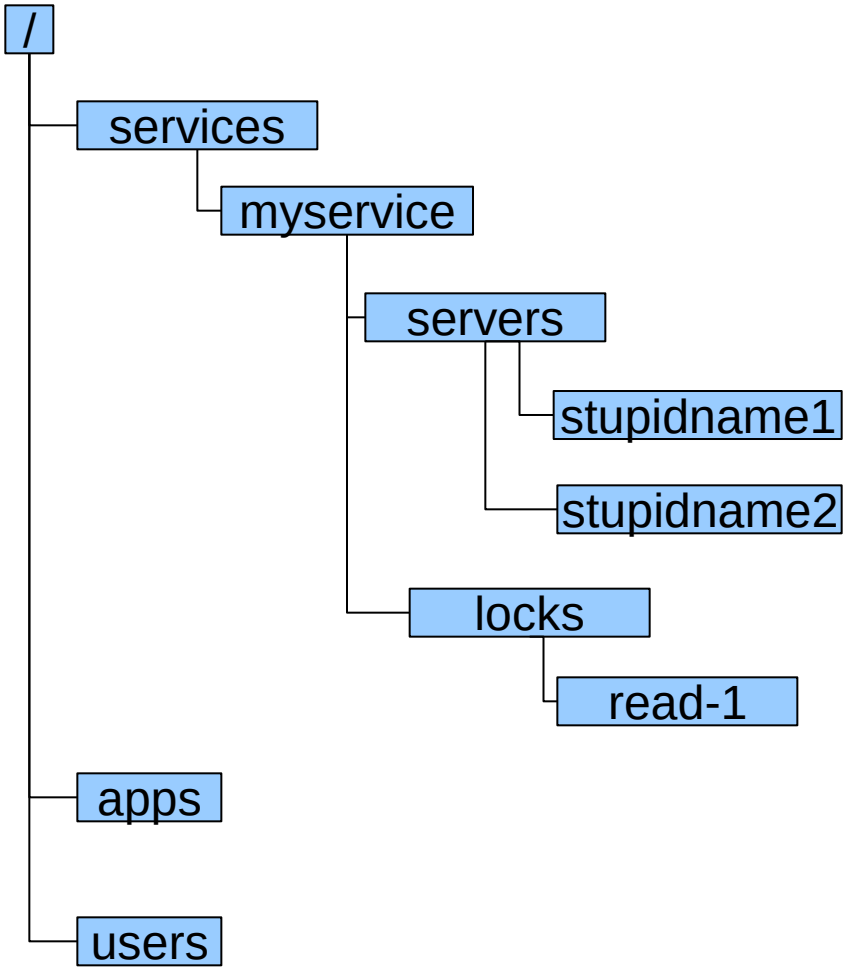
Zookeeper数据模型

- 层次化的目录结构，命名符合常规文件系统规范；
- 每个节点在zookeeper中叫做znode,并且其有一个唯一的路径标识；
- 节点Znode可以包含数据和子节点（EPHEMERAL类型的节点不能有子节点）；
- Znode中的数据可以有多个版本，比如某一个路径下存有多个数据版本，那么查询这个路径下的数据需带上版本；
- 客户端应用可以在节点上设置监视器（Watcher）；
- 节点不支持部分读写，而是一次性完整读写。

Zookeeper数据模型

- Znode有两种类型，短暂的（ephemeral）和持久的（persistent）；
- Znode的类型在创建时确定并且之后不能再修改；
- 短暂znode的客户端会话结束时，zookeeper会将该短暂znode删除，短暂znode不可以有子节点；
- 持久znode不依赖于客户端会话，只有当客户端明确要删除该持久znode时才会被删除；
- Znode有四种形式的目录节点，PERSISTENT、PERSISTENT_SEQUENTIAL、EPHEMERAL、EPHEMERAL_SEQUENTIAL。

Zookeeper数据模型

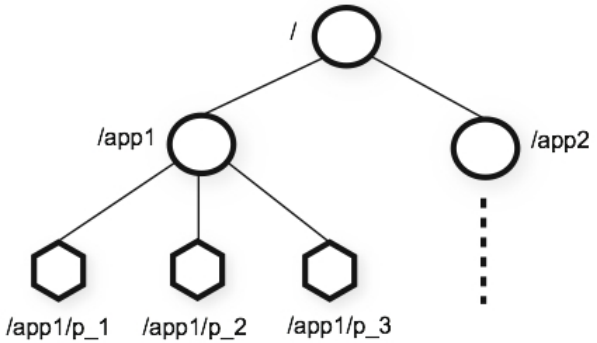


总结

- Zookeeper是什么
- Zookeeper基本原理
- Zookeeper应用场景
- Zookeeper安装部署
- Zookeeper客户端设计
- 总结

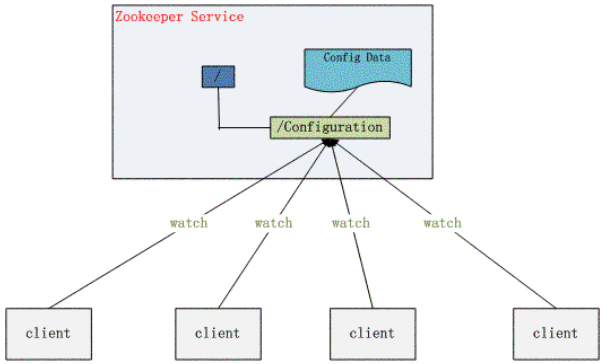
统一命名服务

- 分布式环境下，经常需要对应用/服务进行统一命名，便于识别不同服务；
  - ✓类似于域名与ip之间对应关系，域名容易记住；
  - ✓通过名称来获取资源或服务的地址，提供者等信息
- 按照层次结构组织服务/应用名称
  - ✓ 可将服务名称以及地址信息写到Zookeeper上，客户端通过Zookeeper获取可用服务列表类



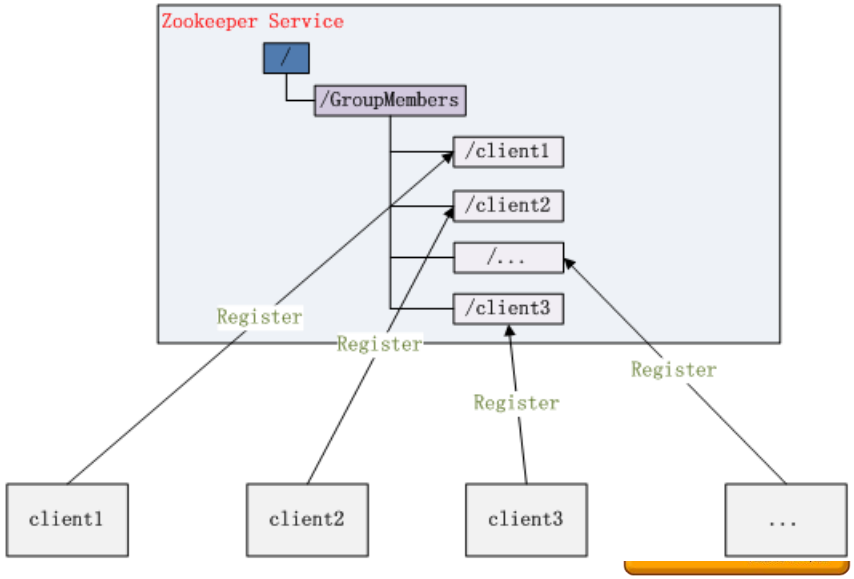
配置管理

- **分布式环境下，配置文件管理和同步是一个常见问题；**
  - ✓一个集群中，所有节点的配置信息是一致的，比如Hadoop；
  - ✓对配置文件修改后，希望能够快速同步到各个节点上
- **配置管理可交由Zookeeper实现；**
  - ✓ 可将配置信息写入Zookeeper的一个znode上；
  - ✓ 各个节点监听这个znode
  - ✓ 一旦znode中的数据被修改，zookeeper将通知各个节点



集群管理

- **分布式环境中，实时掌握每个节点的状态是必要的；**
  - ✓可根据节点实时状态作出一些调整；
- **可交由Zookeeper实现；**
  - ✓ 可将节点信息写入Zookeeper的一个znode上；
  - ✓ 监听这个znode可获取它的实时状态变化
- **典型应用**
  - ✓ Hbase中Master状态监控与选举



分布式通知/协调

- **分布式环境中，经常存在一个服务需要知道它所管理的子服务的状态；**
  - ✓NameNode须知道各DataNode的状态
  - ✓JobTracker须知道各TaskTracker的状态
- **心跳检测机制可通过Zookeeper实现；**
- **信息推送可由Zookeeper实现（发布/订阅模式）。**

分布式锁

- **Zookeeper是强一致的；**
  - ✓多个客户端同时在Zookeeper上创建相同znode，只有一个创建成功。
- **实现锁的独占性**
  - ✓ 多个客户端同时在Zookeeper上创建相同znode ，创建成功的那个客户端得到锁，其他客户端等待。
- **控制锁的时序**
  - ✓ 各个客户端在某个znode下创建临时znode （类型为CreateMode.EPHEMERAL_SEQUENTIAL），这样，该znode可掌握全局访问时序。

分布式队列

- 两种队列；
  - ✓当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列。
  - ✓队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型。（可通过分布式锁实现）
- 同步队列
  - ✓ 一个job由多个task组成，只有所有任务完成后，job才运行完成。
  - ✓ 可为job创建一个/job目录，然后在该目录下，为每个完成的task创建一个临时znode，一旦临时节点数目达到task总数，则job运行完成。

总结

- Zookeeper是什么
- Zookeeper基本原理
- Zookeeper应用场景
- Zookeeper安装部署
- Zookeeper客户端设计
- 总结

Zookeeper部署（单机模式）

- 下载ZooKeeper
  - ✓http://zookeeper.apache.org/releases.html#download
- 解压
  - ✓tar xzf zookeeper-3.4.5.tar.gz
- 配置。在conf目录下创建一个配置文件zoo.cfg:

```
tickTime=2000
dataDir=/user/local/zookeeper/data
dataLogDir=/user/local/zookeeper/dataLog
clientPort=2181
```

- 启动/关闭ZooKeeper Server
  - ✓启动： bin/zkServer.sh start
  - ✓关闭： bin/zkServer.sh stop

Zookeeper部署（伪分布式模式）

- 建了3个文件夹，**server1 server2 server3**，在每个文件夹中解压**zookeeper**的下载包
- 创建**myid**文件：进入**data**目录，创建名为**myid**的文件，分别写入一个数字，**server1~server3**对应的**myid**文件分别写**1~3**
- 在**conf**目录下创建一个配置文件**zoo.cfg:**

```
tickTime=2000
dataDir=/usr/local/zookeeper/data
dataLogDir=/usr/local/zookeeper/server1/
clientPort=2181
initLimit=5
syncLimit=2
server.1=server1:2888:3888
server.2=server2:2889:3889
server.3=server3:2890:3890
```

Zookeeper部署（分布式模式）

- 创建myid文件
  - server1~ server3机器的内容为:1~3
- 在conf目录下创建一个配置文件zoo.cfg,

```
tickTime=2000
dataDir=/usr/local/zookeeper/data
dataLogDir=/usr/local/zookeeper/dataLog
clientPort=2181
initLimit=5
syncLimit=2
server.1=server1:2888:3888
server.2=server2:2888:3888
server.3=server3:2888:3888
```

Zookeeper部署（Observer配置）

- 修改zoo.cfg中的两个配置：
  - ✓peerType=observer
  - ✓server.1:localhost:2181:3181:observer

总结

- Zookeeper是什么
- Zookeeper基本原理
- Zookeeper应用场景
- Zookeeper安装部署
- Zookeeper客户端设计
- 总结

Zookeeper API

- 有点像“一个可提供强一致性保证的分布式小文件系统”
  - ✓**String create (path, data, acl, flags)**
  - ✓**void delete (path, expectedVersion)**
  - ✓**Stat setData (path, data, expectedVersion)**
  - ✓**byte[] getData (path, watch)**
  - ✓**Stat exists (path, watch)**
  - ✓**String[] getChildren (path, watch)**
  - ✓**void sync (path)**

Zookeeper Watcher

- **Watcher 在 ZooKeeper 是一个核心功能**
  - ✓可以监控目录节点的数据变化以及子目录的变化；
  - ✓一旦状态发生变化，服务器就会通知所有设置在这个目录节点上的 Watcher；
- **基本特点**
  - ✓一次设置对应一次触发
  - ✓异步触发
  - ✓顺序触发
- **可以设置观察的操作：exists getChildren getData**
- **可以触发观察的操作：create delete setData**

写操作与内部事件对应关系

	event For "/path"	event For "/path/child"
create("/path")	EventType.NodeCreated	NA
delete("/path")	EventType.NodeDeleted	NA
setData("/path")	EventType.NodeDataChanged	NA
create("/path/child")	EventType.NodeChildrenChanged	EventType.NodeCreated
delete("/path/child")	EventType.NodeChildrenChanged	EventType.NodeDeleted
setData("/path/child")	NA	EventType.NodeDataChanged

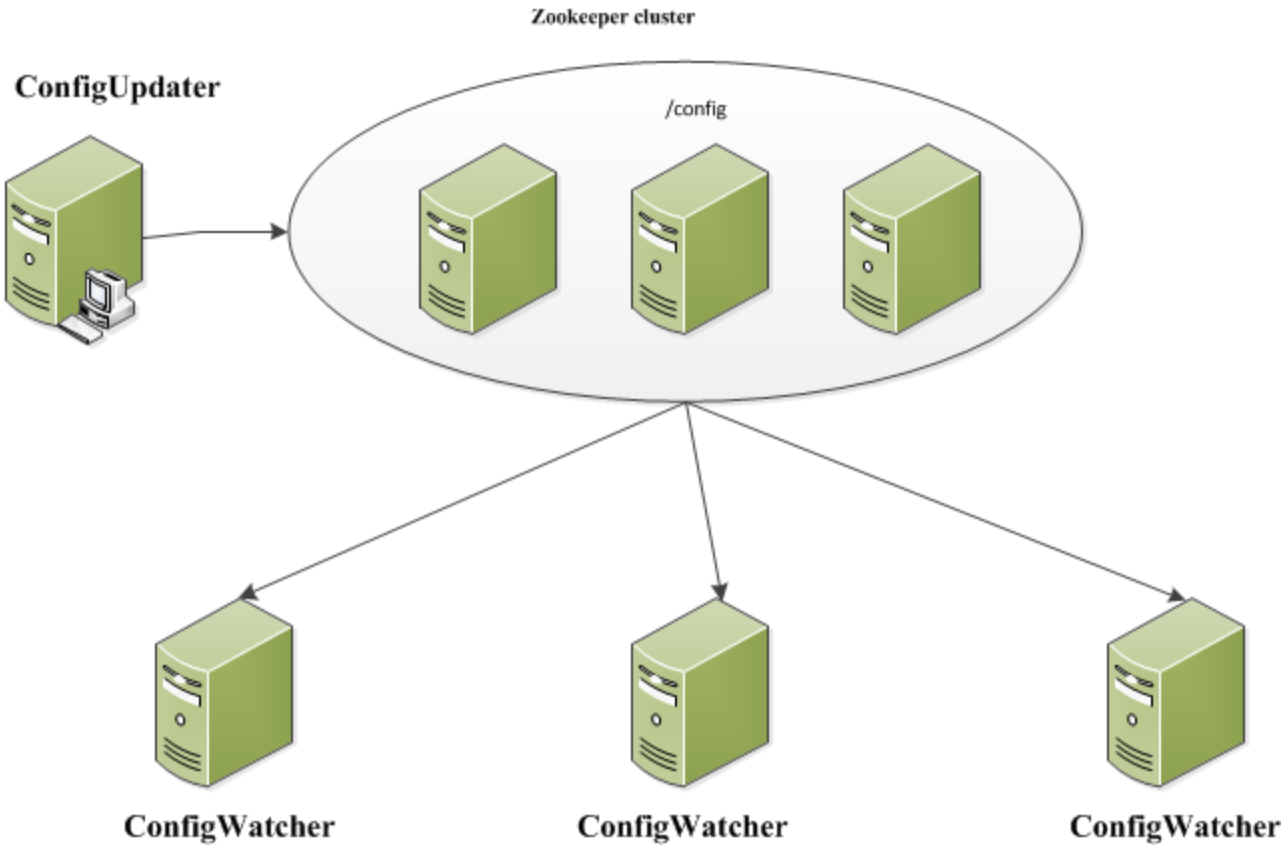
Watcher与内部事件对应关系

event For "/path"	defaultWatcher	exists ("/path")	getData ("/path")	getChildren ("/path")
EventType.None	√	√	√	√
EventType.NodeCreated		√	√	
EventType.NodeDeleted		√(不正常)	√	
EventType.NodeDataChanged		√	√	
EventType.NodeChildrenChanged				√

写操作与Watcher对应关系

	"/path"			"/path/child"		
	exists	getData	getChildren	exists	getData	getChild
create("/path")	√	√				
delete("/path")	√	√	√			
setData("/path")	√	√				
create("/path/child")			√	√	√	
delete("/path/child")			√	√	√	√
setData("/path /child")				√	√	

Zookeeper案例—配置管理



Zookeeper案例—配置管理（Updater设计）

```
public class ConfigUpdater {

    public static final String PATH = "/config";

    private ActiveKeyValueStore _store;
    private Random _random = new Random();

    public ConfigUpdater(String hosts) throws IOException, InterruptedException {
        _store = new ActiveKeyValueStore();
        _store.connect(hosts);
    }

    public void run() throws InterruptedException, KeeperException {
        //noinspection InfiniteLoopStatement
        while (true) {
            String value = _random.nextInt(100) + "";
            _store.write(PATH, value);
            System.out.printf("Set %s to %s\n", PATH, value);
            TimeUnit.SECONDS.sleep(_random.nextInt(10));
        }
    }

    public static void main(String[] args) throws IOException, InterruptedException, KeeperException {
        ConfigUpdater updater = new ConfigUpdater(args[0]);
        updater.run();
    }

}
```

Zookeeper案例—配置管理（Wacher设计）

```
public class ConfigWatcher implements Watcher {

    private ActiveKeyValueStore _store;

    public ConfigWatcher(String hosts) throws InterruptedException, IOException {
        _store = new ActiveKeyValueStore();
        _store.connect(hosts);
    }

    public void displayConfig() throws InterruptedException, KeeperException {
        String value = _store.read(ConfigUpdater.PATH, this);
        System.out.printf("Read %s as %s\n", ConfigUpdater.PATH, value);
    }


    @Override
    public void process(WatchedEvent event) {
        System.out.printf("Process incoming event: %s\n", event.toString());
        if (event.getType() == Event.EventType.NodeDataChanged) {
            try {
                displayConfig();
            }
            catch (InterruptedException e) {
                System.err.println("Interrupted. Exiting");
                Thread.currentThread().interrupt();
            }
            catch (KeeperException e) {
                System.err.printf("KeeperException: %s. Exiting.\n", e);
            }
        }
    }

    public static void main(String[] args) throws IOException, InterruptedException, KeeperException {
        ConfigWatcher watcher = new ConfigWatcher(args[0]);
        watcher.displayConfig();

        Thread.sleep(Long.MAX_VALUE);
    }

}
```

Zookeeper 实例

- 更多实例
  - ✓https://github.com/sleberknight/zookeeper-samples/tree/master/src/main/java/com/nearinfinity/examples/zookeeper
- **NetFlix贡献的Curator ZooKeeper客户端成为Apache顶级项目 http://curator.apache.org/index.html**

总结

- Zookeeper是什么
- Zookeeper基本原理
- Zookeeper应用场景
- Zookeeper安装部署
- Zookeeper客户端设计
- 总结

总结

- Zookeeper基本概念
- Zookeeper设计架构与基本原理
- Zookeeper应用场景
- Zookeeper安装部署
- Zookeeper客户端设计