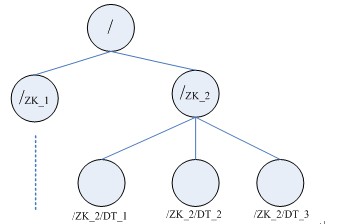
1. zookeeper的数据模型：

提供的命名空间与标准的文件系统非常相似。一个名称是由通过斜线分隔开的路径名序列所组成的。ZooKeeper中的每一个节点是都通过路径来识别。

下图是Zookeeper中节点的数据模型，这种树形结构的命名空间操作方便且易于理解。



图：ZooKeeper层次命名空间

**ZooKeeper中节点和临时节点**

ZooKeeper的节点是通过像树一样的结构来进行维护的，并且每一个节点通过路径来标示以及访问。除此之外，每一个节点还拥有自身的一些信息，包括：数据、数据长度、创建时间、修改时间等等。从这样一类既含有数据，又作为路径表标示的节点的特点中，可以看出，ZooKeeper的节点既可以被看做是一个文件，又可以被看做是一个目录，它同时具有二者的特点。为了便于表达，今后我们将使用Znode来表示所讨论的ZooKeeper节点。

具体地说，Znode维护着数据、ACL（access control list，访问控制列表）、时间戳等交换版本号等数据结构，它通过对这些数据的管理来让缓存生效并且令协调更新。每当Znode中的数据更新后它所维护的版本号将增加，这非常类似于数据库中计数器时间戳的操作方式。

另外Znode还具有原子性操作的特点：命名空间中，每一个Znode的数据将被原子地读写。读操作将读取与Znode相关的所有数据，写操作将替换掉所有的数据。除此之外，每一个节点都有一个访问控制列表，这个访问控制列表规定了用户操作的权限。

ZooKeeper中同样存在临时节点。这些节点与session同时存在，当session生命周期结束，这些临时节点也将被删除。临时节点在某些场合也发挥着非常重要的作用。

1. zookeeper的API介绍及编程：

1)ZooKeeper API 简介

ZooKeeper API 共包含 5 个包，分别为： org.apache.zookeeper ， org.apache.zookeeper.data ，org.apache.zookeeper.server ， org.apache.zookeeper.server.quorum 和org.apache.zookeeper.server.upgrade 。其中 org.apache.zookeeper 包含 ZooKeeper 类，它我们编程时最常用的类文件。

这个类是 ZooKeeper 客户端库的主要类文件。如果要使用 ZooKeeper 服务，应用程序首先必须创建一个Zookeeper 实例，这时就需要使用此类。一旦客户端和 ZooKeeper 服务建立起连接， ZooKeeper 系统将会分配给此连接回话一个 ID 值，并且客户端将会周期地向服务器发送心跳来维持会话的连接。只要连接有效，客户端就可以调用 ZooKeeper API 来做相应的处理。

它提供了表 1 所示几类主要方法 , ：

  表 1 ： ZooKeeper API 描述

|  |  |
| --- | --- |
| 功能 | 描述 |
| create | 在本地目录树中创建一个节点 |
| delete | 删除一个节点 |
| exists | 测试本地是否存在目标节点 |
| get/set data | 从目标节点上读取 / 写数据 |
| get/set ACL | 获取 / 设置目标节点访问控制列表信息 |
| get children | 检索一个子节点上的列表 |
| sync | 等待要被传送的数据 |

2)ZooKeeper API 的使用

这里，笔者通过一个例子来简单介绍，如何使用 ZooKeeper API 编写自己的应用程序，见代码清单 1 ：

代码清单 1 ： ZooKeeper API 的使用

1. import java.io.IOException;

2.

3. import org.apache.zookeeper.CreateMode;

4. import org.apache.zookeeper.KeeperException;

5. import org.apache.zookeeper.Watcher;

6. import org.apache.zookeeper.ZooDefs.Ids;

7. import org.apache.zookeeper.ZooKeeper;

8.

9. public class demo {

10.     // 会话超时时间，设置为与系统默认时间一致

11.     private static final int SESSION\_TIMEOUT=30000;

12.

13.     // 创建 ZooKeeper 实例

14.     ZooKeeper zk;

15.

16.     // 创建 Watcher 实例

17.     Watcher wh=new Watcher(){

18.            public void process(org.apache.zookeeper.WatchedEvent event)

19.            {

20.                    System.out.println(event.toString());

21.            }

22.     };

23.

24.     // 初始化 ZooKeeper 实例

25.     private void createZKInstance() throws IOException

26.     {

27.            zk=new ZooKeeper("localhost:2181",demo.SESSION\_TIMEOUT,this.wh);

28.

29.     }

30.

31.     private void ZKOperations() throws IOException,InterruptedException,KeeperException

32.     {

33.            System.out.println("/n1. 创建 ZooKeeper 节点 (znode ： zoo2, 数据： myData2 ，权限：OPEN\_ACL\_UNSAFE ，节点类型： Persistent");

34.            zk.create("/zoo2","myData2".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);

35.

36.            System.out.println("/n2. 查看是否创建成功： ");

37.            System.out.println(new String(zk.getData("/zoo2",false,null)));

38.

39.            System.out.println("/n3. 修改节点数据 ");

40.            zk.setData("/zoo2", "shenlan211314".getBytes(), -1);

41.

42.            System.out.println("/n4. 查看是否修改成功： ");

43.            System.out.println(new String(zk.getData("/zoo2", false, null)));

44.

45.            System.out.println("/n5. 删除节点 ");

46.            zk.delete("/zoo2", -1);

47.

48.            System.out.println("/n6. 查看节点是否被删除： ");

49.            System.out.println(" 节点状态： ["+zk.exists("/zoo2", false)+"]");

50.     }

51.

52.     private void ZKClose() throws  InterruptedException

53.     {

54.            zk.close();

55.     }

56.

57.     public static void main(String[] args) throws IOException,InterruptedException,KeeperException {

58.            demo dm=new demo();

59.            dm.createZKInstance( );

60.            dm.ZKOperations();

61.            dm.ZKClose();

62.     }

63.}

此类包含两个主要的 ZooKeeper 函数，分别为 createZKInstance （）和 ZKOperations （）。其中createZKInstance （）函数负责对 ZooKeeper 实例 zk 进行初始化。 ZooKeeper 类有两个构造函数，我们这里使用“ ZooKeeper （ String connectString, ， int sessionTimeout, ， Watcher watcher ）”对其进行初始化。因此，我们需要提供初始化所需的，连接字符串信息，会话超时时间，以及一个 watcher 实例。 17 行到 23 行代码，是程序所构造的一个 watcher 实例，它能够输出所发生的事件。

ZKOperations （）函数是我们所定义的对节点的一系列操作。它包括：创建 ZooKeeper 节点（ 33 行到 34 行代码）、查看节点（ 36 行到 37 行代码）、修改节点数据（ 39 行到 40 行代码）、查看修改后节点数据（ 42 行到43 行代码）、删除节点（ 45 行到 46 行代码）、查看节点是否存在（ 48 行到 49 行代码）。另外，需要注意的是：在创建节点的时候，需要提供节点的名称、数据、权限以及节点类型。此外，使用 exists 函数时，如果节点不存在将返回一个 null 值。关于 ZooKeeper API 的更多详细信息，读者可以查看 ZooKeeper 的 API 文档，如下所示：