**[zookeeper之监听事件总结](http://blog.csdn.net/liu857279611/article/details/70495413)**

2017-04-23 12:17 1765人阅读 [评论](http://blog.csdn.net/liu857279611/article/details/70495413#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/liu857279611/article/details/70495413#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

大数据（36） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

1.  zookeeper的Watch机制

一个zk的节点可以被监控，包括这个目录中存储的数据的修改，子节点目录的变化，一旦变化可以通知设置监控的客户端，这个功能是zookeeper对于应用最重要的特性，通过这个特性可以实现的功能包括配置的集中管理，集群管理，分布式锁等等。

       watch机制官方说明：**一个Watch事件是一个一次性的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。**

2. zookeeper机制的特点

   1)  一次性的触发器（one-time trigger）

       当数据改变的时候，那么一个Watch事件会产生并且被发送到客户端中。但是客户端只会收到一次这样的通知，如果以后这个数据再次发生改变的时候，之前设置Watch的客户端将不会再次收到改变的通知，因为Watch机制规定了它是一个一次性的触发器。

         当设置监视的数据发生改变时，该监视事件会被发送到客户端，例如，如果客户端调用了 getData("/znode1", true) 并且稍后 /znode1 节点上的数据发生了改变或者被删除了，客户端将会获取到 /znode1 发生变化的监视事件，而如果 /znode1 再一次发生了变化，除非客户端再次对 /znode1 设置监视，否则客户端不会收到事件通知。

  2)发送给客户端（Sent to the client）

       这个表明了Watch的通知事件是从服务器发送给客户端的，是异步的，这就表明不同的客户端收到的Watch的时间可能不同，但是ZooKeeper有保证：当一个客户端在看到Watch事件之前是不会看到结点数据的变化的。例如：A=3，此时在上面设置了一次Watch，如果A突然变成4了，那么客户端会先收到Watch事件的通知，然后才会看到A=4。

       Zookeeper 客户端和服务端是通过 Socket 进行通信的，由于网络存在故障，所以监视事件很有可能不会成功地到达客户端，监视事件是异步发送至监视者的，Zookeeper 本身提供了保序性(ordering guarantee)：即客户端只有首先看到了监视事件后，才会感知到它所设置监视的 znode 发生了变化(a client will never see a change for which it has set a watch until it first sees the watch event). 网络延迟或者其他因素可能导致不同的客户端在不同的时刻感知某一监视事件，但是不同的客户端所看到的一切具有一致的顺序。

   3)被设置Watch的数据（The data for which the watch was set）

         这意味着 znode 节点本身具有不同的改变方式。你也可以想象 Zookeeper 维护了两条监视链表：

       数据监视和子节点监视(data watches and child watches)

       getData() and exists() 设置数据监视，getChildren() 设置子节点监视。 或者，你也可以想象 Zookeeper 设置的不同监视返回不同的数据，getData() 和 exists() 返回 znode 节点的相关信息，而 getChildren() 返回子节点列表。

       因此， setData() 会触发设置在某一节点上所设置的数据监视(假定数据设置成功)，而一次成功的 create() 操作则会出发当前节点上所设置的数据监视以及父节点的子节点监视。一次成功的 delete() 操作将会触发当前节点的数据监视和子节点监视事件，同时也会触发该节点父节点的child watch。

3.各种watch触发的情况总结

可以注册watcher的方法：getData、exists、getChildren。

       可以触发watcher的方法：create、delete、setData。连接断开的情况下触发的watcher会丢失。

       一个Watcher实例是一个回调函数，被回调一次后就被移除了。如果还需要关注数据的变化，需要再次注册watcher。

         New ZooKeeper时注册的watcher叫default watcher，它不是一次性的，只对client的连接状态变化作出反应。

**什么样的操作会产生什么类型的事件:**

|  |  |  |
| --- | --- | --- |
|  | event For “/path” | event For “/path/child” |
| create(“/path”) | EventType.NodeCreated | 无 |
| delete(“/path”) | EventType.NodeDeleted | 无 |
| setData(“/path”) | EventType.NodeDataChanged | 无 |
| create(“/path/child”) | EventType.NodeChildrenChanged（getChild） | EventType.NodeCreated |
| delete(“/path/child”) | EventType.NodeChildrenChanged（getChild） | EventType.NodeDeleted |
| setData(“/path/child”) | 无 | EventType.NodeDataChanged |

**事件类型与watcher的对应关系：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| event For “/path” | Default Watcher | exists(“/path”) | getData(“/path”) | getChildren(“/path”) |
| EventType.None | √ | √ | √ | √ |
| EventType.NodeCreated |  | √ | √ |  |
| EventType.NodeDeleted |  | √ | √ |  |
| EventType.NodeDataChanged |  | √ | √ |  |
| EventType.NodeChildrenChanged |  |  |  | √ |

   本表总结：exits和getData设置数据监视，而getChildren设置子节点监视

**操作与watcher的对应关系：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | exits("/path") | getData(“/path”) | getChildren(“/path”) | exits("/path/child") | getData(“/path/child”) | getChildren(“/path/child”) |
| create(“/path”) | √ | √ | 会报错 |  |  |  |
| delete(“/path”) | √ | √ | √（这个要注意） |  |  |  |
| setData(“/path”) | √ | √ |  |  |  |  |
| create(“/path/child”) |  |  | √ | √ | √ |  |
| delete(“/path/child”) |  |  | √ | √ | √ | √ |
| setData(“/path/child”) |  |  |  | √ | √ |  |
|  |  |  |  |  |  |  |

值得注意的是：getChildren("/path")监视/path的子节点，如果（/path）自己删了，也会触发NodeDeleted事件。

4.实现永久监听

  由于zookeeper是一次性监听，所以我们必须在wather的process方法里面再设置监听。一个方法如下：

       以下逻辑是实现的是生产者和消费者模型，消费者监听某一路径下面子节点的变化，当生产者有消息发送过来的时候，在该节点下面创建一个子节点，然后把消息放到该子节点里面，这会触发消费者的process方法被调用，然后消费者取到该节点下面的子节点(顺便设置一个再监听该节点的子节点)，然后取出子节点的内容，做处理，然后删除该子节点。

public void process(WatchedEvent event) {

   // TODO Auto-generated method stub

   if (event.getState() == KeeperState.SyncConnected) {

      System.out.println("watcher received event");

      countDownLatch.countDown();

   }

   System.out.println("回调watcher1实例： 路径" + event.getPath() + " 类型："+ event.getType());

   // 事件类型，状态，和检测的路径

   EventType eventType = event.getType();

   KeeperState state = event.getState();

   String watchPath = event.getPath();

   switch (eventType) {

      case NodeCreated:

        break;

      case NodeDataChanged:

        break;

      case NodeChildrenChanged:

         try {

              //处理收到的消息

              handleMessage(watchPath);

         } catch (UnsupportedEncodingException e) {

              // TODO Auto-generated catch block

              e.printStackTrace();

         } catch (KeeperException e) {

             // TODO Auto-generated catch block

             e.printStackTrace();

         } catch (InterruptedException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

         }

        break;

       default:

         break;

       }

}

public void handleMessage(String watchPath) throws KeeperException,InterruptedException, UnsupportedEncodingException {

    System.out.println("收到消息");

    //再监听该子节点

    List<String> Children = this.getChildren(watchPath);

    for (String a : Children) {

        String childrenpath = watchPath + "/" + a;

        byte[] recivedata = this.getData(childrenpath);

        String recString = new String(recivedata, "UTF-8");

        System.out.println("receive the path:" + childrenpath + ":data:"+ recString);

        //做完了之后，删除该节点

        this.deletNode(childrenpath, -1);

    }

}

public List<String> getChildren(String path) throws KeeperException,InterruptedException {

     //监听该节点子节点的变化情况

     return this.zooKeeper.getChildren(path, this);

}

public Stat setData(String path, byte[] data, int version)throws KeeperException, InterruptedException {

     return this.zooKeeper.setData(path, data, version);

}