[**zookeeper事件监听实战**](http://workman666.iteye.com/blog/2346879)

**博客分类：**

* [zookeeper](http://workman666.iteye.com/category/355031)

[zookeeper](http://www.iteye.com/blogs/tag/zookeeper)[事件监听](http://www.iteye.com/blogs/tag/%E4%BA%8B%E4%BB%B6%E7%9B%91%E5%90%AC)

一:事件监听

    原生zookeeper的事件监听采用Watcher实现,不过Watcher监听是一次性的,如果需要继续监听该事件,必须重新注册。Curator中采用cache来封装了对事件的监听,包括监听节点,监听子节点等,下面分别做介绍  
1.1 NodeCache  
    NodeCache主要用来监听节点本身的变化,当节点的状态发生变更后,回调NodeCachaListener

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** NodeCacheListener{
2. /\*\*
3. \* Called when a change has occurred
4. \*/
5. **public** **void**     nodeChanged() **throws** Exception;
6. }

    看一个例子:

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** NodeCacheExample {
3. **private** **static** **final** String PATH = "/nodeCache";
5. **private** **static** CountDownLatch latch = **new** CountDownLatch(1);
7. **static** NodeCache nodeCache;
9. **static** CuratorFramework client;
11. **static** {
13. client = CuratorFrameworkFactory.newClient(
14. "host:2181", 5000, 5000, **new** ExponentialBackoffRetry(
15. 1000, 3));
16. client.start();
17. }
19. **public** **static** **void** initCache() **throws** Exception {
20. client.create().forPath(PATH);
21. client.setData().forPath(PATH, "节点的初始值".getBytes());
22. nodeCache = **new** NodeCache(client, PATH);
23. EnsurePath ensurePath = client.newNamespaceAwareEnsurePath(PATH);
24. ensurePath.ensure(client.getZookeeperClient());
25. //设置成true,那么nodeCache在第一次启动的时候就会到zookeeper上去获取节点的数据内容，并保存在cache中
26. nodeCache.start(**true**);
27. startCache(nodeCache);
28. }
30. **private** **static** **void** startCache(**final** NodeCache cache) **throws** Exception {
32. ChildData data = cache.getCurrentData();
34. System.out.println("第一次启动获取到的内容:"  + **new** String(data.getData()));
36. cache.getListenable().addListener(**new** NodeCacheListener() {
37. @Override
38. **public** **void** nodeChanged() **throws** Exception {
39. System.out.println("NodeCache changed,data is "
40. + **new** String(cache.getCurrentData().getData()));
41. latch.countDown();
42. }
43. });
44. Thread.sleep(2000);
45. **if**(client.checkExists().forPath(PATH) != **null**){
46. System.out.println("node is exist,准备给节点设置新的内容");
47. client.setData().forPath(PATH, "节点新内容".getBytes());
48. }
49. }
51. **public** **static** **void** main(String[] args) **throws** Exception {
52. initCache();
53. latch.await();
54. }
55. }

   运行结果如下:

**Java代码  [收藏代码](javascript:void())**

1. 第一次启动获取到的内容:节点的初始值
2. node is exist,准备给节点设置新的内容
3. NodeCache changed,data is 节点新内容

  1.2 PathChildrenCache  
    PathChildrenCache主要用来监听子节点,它有几个构造函数,参数最多的是下面这个

**Java代码  [收藏代码](javascript:void())**

1. **public** PathChildrenCache(CuratorFramework client, String path, **boolean** cacheData, **boolean** dataIsCompressed, **final** ExecutorService executorService)
2. {
3. **this**(client, path, cacheData, dataIsCompressed, **new** CloseableExecutorService(executorService));
4. }

    参数解释:  
    client：一个Curator客户端实例  
    path：监听路径  
    cacheData：boolean类型,如果是true,那么curator客户端在请求服务端的时候,会将监听节点的内容保存起来。而zookeeper原生的watcher监听是不会返回节点的内容的,只会返回节点状态,路径等  
    dataIsCompressed：表示是否对数据进行压缩  
    executorService:说明可以使用线程池来处理事件通知,当子节点数据发生变化时,会回调

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** PathChildrenCacheListener
2. {
3. /\*\*
4. \* Called when a change has occurred
5. \*
6. \* @param client the client
7. \* @param event describes the change
8. \* @throws Exception errors
9. \*/
10. **public** **void**     childEvent(CuratorFramework client, PathChildrenCacheEvent event) **throws** Exception;
11. }

    并且不会对二级节点进行监听,来看一个例子

**Java代码  [收藏代码](javascript:void())**

1. **import** java.util.concurrent.CountDownLatch;
2. **import** org.apache.curator.framework.CuratorFramework;
3. **import** org.apache.curator.framework.CuratorFrameworkFactory;
4. **import** org.apache.curator.framework.recipes.cache.ChildData;
5. **import** org.apache.curator.framework.recipes.cache.PathChildrenCache;
6. **import** org.apache.curator.framework.recipes.cache.PathChildrenCacheEvent;
7. **import** org.apache.curator.framework.recipes.cache.PathChildrenCacheListener;
8. **import** org.apache.curator.retry.ExponentialBackoffRetry;
10. /\*\*
11. \*
12. \* Path Cache用来监控一个ZNode的子节点. 当一个子节点增加， 更新，删除时， Path Cache会改变它的状态，
13. \*  会包含最新的子节点， 子节点的数据和状态
14. \*
15. \* @author tanjie
16. \*
17. \*/
18. **public** **class** PathCacheExample {
20. **private** **static** **final** String PATH = "/cache";
22. **static** PathChildrenCache cache;
24. **static** CuratorFramework client;
26. **static** CountDownLatch latch = **new** CountDownLatch(1);
28. **static** CountDownLatch coutCountDownLatch = **new** CountDownLatch(5);
30. **static** {
31. client = CuratorFrameworkFactory.newClient(
32. "host:2181", 5000, 5000, **new** ExponentialBackoffRetry(
33. 1000, 3));
34. client.start();
35. }

38. **private** **static** **void** startCache()
39. **throws** Exception {
40. cache = **new** PathChildrenCache(client, PATH, **true**);
41. cache.start();
42. // 给当前节点创建子节点
43. **for** (**int** i = 1; i <= 5; i++) {
44. String newPath = PATH + "/child\_" + i;
45. String childNodeName = "child\_" + i;
46. **if** (client.checkExists().forPath(PATH) != **null**) {
47. client.create().creatingParentsIfNeeded().forPath(newPath, childNodeName.getBytes());
48. }
49. coutCountDownLatch.countDown();
50. }
51. coutCountDownLatch.await();
52. addlistener(cache);
53. **for** (**final** ChildData childData : cache.getCurrentData()) {
54. System.out.println("pathChildCache:" + "路径:" + childData.getPath()
55. + ",内容:" + **new** String(childData.getData()));
56. }
57. System.out.println("对父节点进行设置新值不会有通知事件发生........start");
58. client.setData().forPath(PATH,"哈哈".getBytes());
59. System.out.println("对父节点设置值 end..........");
60. //改变子节点的数据
61. System.out.println("准备删除子节点........start");
62. client.delete().forPath( PATH + "/child\_1");
63. System.out.println("删除子节点........end,会有节点事件通知返回");
64. Thread.sleep(2000);
65. //改变二级节点的内容
66. **for**(**int** j=1;j<=2;j++){
67. String newPath = PATH + "/child\_2/" + j;
68. String childNodeName = "child\_2" + j;
69. **if** (client.checkExists().forPath(PATH) != **null**) {
70. client.create().forPath(newPath, childNodeName.getBytes());
71. }
72. }
73. addlistener(cache);
74. System.out.println("准备删除二级节点......start");
75. client.delete().forPath( PATH + "/child\_2/" + 1);
76. System.out.println("删除二级节点......end..不会有事件监听返回");
77. latch.countDown();
79. }
81. **private** **static** **void** addlistener(**final** PathChildrenCache cache) {
82. **final** PathChildrenCacheListener pathChildrenCacheListener = **new** PathChildrenCacheListener() {
83. @Override
84. **public** **void** childEvent(CuratorFramework curatorClient,
85. PathChildrenCacheEvent event) **throws** Exception {
86. System.out.println("childNode path:"
87. + event.getData().getPath() + ",childNode data : "
88. + **new** String(event.getData().getData()));
89. }
90. };
91. cache.getListenable().addListener(pathChildrenCacheListener);
92. }
94. **public** **static** **void** main(String[] args) **throws** Exception {
95. startCache();
96. latch.await();
97. }
98. }

   运行结果如下:

**Java代码  [收藏代码](javascript:void())**

1. pathChildCache:路径:/cache/child\_1,内容:child\_1
2. pathChildCache:路径:/cache/child\_2,内容:child\_2
3. pathChildCache:路径:/cache/child\_3,内容:child\_3
4. pathChildCache:路径:/cache/child\_4,内容:child\_4
5. pathChildCache:路径:/cache/child\_5,内容:child\_5
6. 对父节点进行设置新值不会有通知事件发生........start
7. 对父节点设置值 end..........
8. 准备删除子节点........start
9. childNode path:/cache/child\_1,childNode data : child\_1
10. 删除子节点........end,会有节点事件通知返回
11. 准备删除二级节点......start
12. 删除二级节点......end..不会有事件监听返回

1.3 TreeCache  
    TreeCache即能监听节点也能监听节点的子节点变更

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** TreeCacheExample {
3. **private** **static** **final** String PATH = "/treeNodeCache";
5. **static** TreeCache  treeCache;
7. **static** CuratorFramework client;
9. **static** {
10. client = CuratorFrameworkFactory.newClient(
11. "host:2181", 5000, 5000, **new** ExponentialBackoffRetry(
12. 1000, 3));
13. client.start();
14. }
16. **public** **static** **void**  initCache() **throws** Exception {
17. treeCache = **new** TreeCache(client, PATH);
18. client.create().forPath(PATH);
19. treeCache.start();
20. treeCache.getListenable().addListener(**new** TreeCacheListener() {
21. @Override
22. **public** **void** childEvent(CuratorFramework client, TreeCacheEvent event)
23. **throws** Exception {
24. **if**(**null** != event.getType()){
25. System.out.println("节点状态发送了改变.内容为:" + **new** String(event.getData().getData()));
26. }
27. }
28. });
30. //添加子节点
31. **for**(**int** i=1;i<=5;i++){
32. String newPath = PATH + "/child\_" + i;
33. client.create().forPath(newPath,("child\_"+i).getBytes());
34. }
35. Thread.sleep(3000);
36. client.setData().forPath(PATH+ "/child\_2", "我是给第二个子节点赋值".getBytes());
37. **if**(**null** != client.checkExists().forPath(PATH)){
38. client.setData().forPath(PATH,"我是来对父节点赋值的".getBytes());
39. }
40. Thread.sleep(1000);
41. client.setData().forPath(PATH,"我是来对父节点第二次赋值的".getBytes());
42. }
44. **public** **static** **void** main(String[] args) **throws** Exception {
45. initCache();
46. Thread.sleep(Integer.MAX\_VALUE);
47. }
49. }

   运行结果如下:

**Java代码  [收藏代码](javascript:void())**

1. 节点状态发送了改变.内容为:child\_1
2. 节点状态发送了改变.内容为:child\_2
3. 节点状态发送了改变.内容为:child\_3
4. 节点状态发送了改变.内容为:child\_4
5. 节点状态发送了改变.内容为:child\_5
6. 节点状态发送了改变.内容为:我是给第二个子节点赋值
7. 节点状态发送了改变.内容为:我是来对父节点赋值的
8. 节点状态发送了改变.内容为:我是来对父节点第二次赋值的

    可以看到,只有监听的节点状态发送了变化,监听事件就会执行回调,并且也能监听器父节点且不用反复注册监听