[**Zookeeper开发常见问题**](http://tech.uc.cn/?p=1189)

Posted by

[*陈, 超*](http://tech.uc.cn/?author=33)

 on 2013 年 5 月 30 日

Share

**背景与目的**

Zookeeper开发过程中遇到一些常见问题，为了后续开发不犯同样的错误，总结一下此类问题，并进行分析和解决。

**适合人员**

主要适合zookeeper开发、测试及运维相关人员。

**问题与解决**

**一、 关于zookeeper\_init函数的使用**

**问题描述：**

开发人员在调用zookeeper\_init函数时，若返回一个非空句柄zhandle\_t  \*zh，则认为初始化成功，这样可能会导致后续操作失败。

**问题分析：**

zhandle\_t  \*zookeeper\_init(const char \*host, watcher\_fn fn, int recv\_timeout,const   clientid\_t \*clientid, void \*context, int flags) 函 数     返回一个zookeeper客户端与服务器通信的句柄，通常我们仅仅根据返回句柄情况来判断zookeeper 客户端与zookeeper服务器是否 建立连接。如果句柄为空则认为是失败，非空则成功。其实不然，zookeeper\_init创建与ZooKeeper服务端通信的句柄以及对应于此句柄的会话，而会话的创建是一个异步的过程，仅当会话建立成功，zookeeper\_init才返回一个可用句柄。

**问题解决：**

如何正确判断zookeepr\_init初始化成功，可通过以下三种方式  
1、判断句柄的state是否为ZOO\_CONNECTED\_STATE状态，通过zoo\_state(zh)判断状态值是否为ZOO\_CONNECTED\_STATE。



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void ensureConnected()  {      pthread\_mutex\_lock(&lock);      while (zoo\_state(zh)!=ZOO\_CONNECTED\_STATE)      {          pthread\_cond\_wait(&cond, &lock);      }      pthread\_mutex\_unlock(&lock);  } |

2、 在zookeeper\_init中设置watcher，当zookeeper client与server会话建立后，触发watcher，当 watcher 的state = 3 （ZOO\_CONNECTED\_STATE）， type = -1（ZOO\_SESSION\_EVENT）时，确认 会话成功建立，此时zookeeper client 初始化成功，可进行后续操作。  
3、业务上可以做保证，调用zookeeper\_init返回句柄zh，通过该句柄尝试做zoo\_exists()或zoo\_get\_data（）等操作，根据操作结果来判断是否初始化成功。

**二、 如何解决session失效问题**

**问题描述：**

session失效，导致注册的watcher全部丢失。

**问题分析：**

如果zookeeper client与server在协商的超时时间内仍没有建立连接，当client与server再次建立连接时，由于session失效了，所有watcher已经被服务器端删除，从而导致所有的watcher需要重新注册。  
session 失效，zookeeper client与server重连后所有watcher都会收到两次触发，第一次 wathetr state = 1，type = -1（state = 1表示正在连接中，type = -1 表示session事件）；第二次 watcher state = -112，type = -1（state = -112表示session失效）。

**问题解决：**

可以通过以下两种方法解决session失效问题  
1、获取触发session失效watcher后，业务重新注册所有的watcher。  
2、不能根本解决，但是可以减小session失效的概率。通过zookeeper client 与server设置更长的session超时时间。（参考下一问题）

**三、 为什么zookeeper\_init设置recv\_timeout较长却没有效果**

**问题描述：**

zookeeper\_init设置recv\_timeout 100000ms，但客户端与服务端断开连接30s就session失效了。

**问题分析：**

关于session超时时间的确定：zookeeper\_init中设置的超时时间并非真正的session超时时间，session超时时间需要 server与client协商，业务通过zoo\_recv\_timeout(zhandle\_t\* zh)获取server与client协商后的超时时间。服务端: minSessionTimeout (默认值为：tickTime \* 2) ,  maxSessionTimeout(默认值为:tickTime \* 20), ticktime的默认值为2000ms。所以session范围为4s ~ 40s 。客户端：  sessionTimeout， 无默认值，创建实例时设置recv\_timeout 值。经常会认为创建zookeeper客户端时设置了sessionTimeout为100s，而没有改变server端的配置，默认值是 不会生效的。 原因： 客户端的zookeeper实例在创建连接时，将sessionTimeout参数发送给了服务端，服务端会根据对应的  minSession/maxSession Timeout的设置，强制修改sessionTimeout参数，也就是修改为4s~40s 返回的参数。所以服务端不一定会以客户端的sessionTImeout做为session expire管理的时间。

**问题解决：**

增加zookeeper\_init recv\_timeout大小的同时，需要配置tickTime的值。  
tickTime设置是在 conf/zoo.cfg 文件中

|  |
| --- |
| # The number of milliseconds of each ticktickTime=2000  （默认） 注： tickTime 心跳基本时间单位毫秒，ZK基本上所有的时间都是这个时间的整数倍。 |

**四、 zoo\_get\_children内存泄露问题**

**问题描述：**

调用zoo\_get\_children函数出现内存泄露问题。

**问题分析：**

通过查看代码发现问题在于 ZOOAPI int zoo\_get\_children(zhandle\_t \*zh, const char \*path, int watch,struct String\_vector \*strings)， 该API中String\_vector \*strings结构体定义如下：



|  |  |
| --- | --- |
| 1  2  3  4  5 | struct String\_vector  {      int32\_t count;      char \* \*data;  }; |

Zookeeper API将getchildren结果通过String\_vector结构体返回时，malloc分配内存，将子节点所有目录存放在data中，而释放内存需要有客户端来做处理。

**问题解决：**

调用zoo\_get\_children(zh, path, watch, strings)；需要通过调用zookeper提供的释放内存的方法：deallocate\_String\_vector(strings)。



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | int deallocate\_String\_vector(struct String\_vector \*v){      if (v->data)      {          int32\_t i;          for(i = 0; i < v->count; i++)          {              deallocate\_String(&v->data[i]);          }          free(v->data);          v->data = 0;      }      return 0;  } |

**五、 如何正确设置和获取watcher**

Watcher 设置是开发中最常见的，需要搞清楚watcher的一些基本特征，对于exists、getdata、getchild对于节点的不同操作会收到不同的 watcher信息。对父节点的变更以及孙节点的变更都不会触发watcher，而对watcher本身节点以及子节点的变更会触发watcher，具体参照下表。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作 | 方法 | 触发watcher | watcher state | watcher type | watcher path |
| Create当前节点 | getdata | × | × | × | × |
| getchildren | √ | 3 | 4 | √ |
| exists | × | × | × | × |
| set当前节点 | getdata | √ | 3 | 3 | √ |
| getchildren | × | × | × | × |
| exists | √ | 3 | 3 | √ |
| delete当前节点 | getdata | √ | 3 | 2 | √ |
| getchildren | √ | 3 | 2 | √ |
| exists | √ | 3 | 2 | √ |
| create子节点 | getdata | × | × | × | × |
| getchildren | √ | 3 | 4 | √ |
| exists | × | × | × | × |
| set子节点 | getdata | × | × | × | × |
| getchildren | × | × | × | × |
| exists | × | × | × | × |
| delete子节点 | getdata | × | × | × | × |
| getchildren | √ | 3 | 4 | √ |
| exists | × | × | × | × |
| 恢复连接 | getdata | √ | 1 | -1 | × |
| getchildren | √ | 1 | -1 | × |
| exists | √ | 1 | -1 | × |
| 恢复连接session未超时 | getdata | √ | -112 | -1 | × |
| getchildren | √ | -112 | -1 | × |
| exists | √ | -112 | -1 | × |
| 恢复连接session超时 | getdata | √ | 3 | -1 | × |
| getchildren | √ | 3 | -1 | × |
| exists | √ | 3 | -1 | × |

**注： state = 2 表示删除事件；state = 3表示节点数据变更；state =4表示子节点事件；state = -1表示session事件。 type = -112表示session失效；type = 1表示session建立中；tpye = = 3表示session建立成功。×表示否，√表示是。**

**六、 zookeeper连接数问题**

**问题描述：**

zookeeper服务器都运行正常，而客户端连接异常。

**问题分析：**

这是由于zookeeper client连接数已经超过了zookeeper server获取的配置最大连接数。所以导致zookeeper client连接失败。

**解决方法：**

修改zookeeper安装目录下 conf/zoo.cfg文件。将maxClientCnxns参数改成更大的值。

**七、 zookeeper服务器安装相关问题**

**问题一：是否可以只装一个zookeeper服务器。**

**问题解答:**

可以安装,此时没有leader,follow，此时zookeeper server状态为standalone。

|  |
| --- |
| ./zkServer.sh status JMX enabled by default Using config: /home/bbs/zookeeper-3.4.5/bin/../conf/zoo.cfg Mode: standalone |

关于zoo.cfg配置:

|  |
| --- |
| tickTime=2000initLimit=10 syncLimit=5 dataDir=../data clientPort=4181 |

**问题二：开发没有足够机器，一台机子上是否装三个zookeeper服务器集群。**

**问题解答:**

这种安装模式只能说是一种伪集群模式。三个zookeeper服务器都安装在同一个服务器（platform）上，需保证clientPort不相同。  
将zookeeper安装包分别解压在三个目录server1，server2，server3下，配置文件zoo.cfg  
Server1配置文件 zoo.cfg，server1在data目录下增加文件myid内容为1。

|  |
| --- |
| dataDir=../datadata LogDir=../dataLog clientPort=5181 server.1=platform:5888:6888 server.2= platform:5889:6889 server.3= platform:5890:6890 |

Server2配置文件 zoo.cfg，server1在data目录下增加文件myid内容为2。

|  |
| --- |
| dataDir=../datadata LogDir=../dataLog clientPort=6181 server.1=platform:5888:6888 server.2= platform:5889:6889 server.3= platform:5890:6890 |

Server3配置文件 zoo.cfg，server1在data目录下增加文件myid内容为3。

|  |
| --- |
| dataDir=../datadata LogDir=../dataLog clientPort=7181 server.1=platform:5888:6888 server.2= platform:5889:6889 server.3= platform:5890:6890 |