

ice切换及ice对象池维护优化

目前此部分功能机制：

新建守护进程控制池中对象的动态平衡；

守护进程中每5秒执行3个方法

```
class MonitorThread extends Thread {
    @Override
    public void run() {
        while (!thread.isInterrupted()) {
            try {
                TimeUnit.SECONDS.sleep(5);
                initIpAddress();
                FindAvailableAddr();
                balanceObjectNum();
            } catch (Exception e) {
                e.printStackTrace();
                log.error("MonitorThread run failed!The reason is {}", e);
            }
        }
    }
}
```

`initIpAddress()` 方法：将配置文件中配置的ip地址提取到 `listAll` 地址集合中。

```
public void initIpAddress() {
    String masterProperty = "dataProcI:tcp -h " + iceMasterIp + " -p 10011";
    if (!listAll.contains(masterProperty)) {
        listAll.add(masterProperty);
    }
    if (iceOtherIp.trim().length() > 0) {
        String otherIpArray[] = iceOtherIp.split(",");
        for (String ip : otherIpArray) {
            String property = "dataProcI:tcp -h " + ip + " -p 10011";
            if (!listAll.contains(property)) {
                listAll.add(property);
            }
        }
    }
}
```

`FindAvailableAddr()` 方法：遍历 `listAll` 中地址，若地址可用 放入可用地址集合 `listUsAble` 中且创建一个该地址对象放入对象池中；若地址不可用且对象池子中存在该不可用地址创建的对象，则清除这个对象且从 `listUsAble` 中删除该地址。

```
public void FindAvailableAddr() {
```

```

        for (String s : listAll) {
            DataProcPrx proxy = (DataProcPrx) createIceProxy(communicator, s,
DataProcPrx.class);
            try {
                // 插入到可用地址集合中
                proxy.ice_ping();
                // 确保不重复放入
                if (!listUsAble.contains(s)) {
                    listUsAble.add(s);
                    // 此处将可用对象直接放入池中
                    addObjectInPool(s);
                    log.info("找到可用地址" + s + ",并生成可用对象到线程池中");
                }
            } catch (Exception e) {
                log.info("The current address " + s + " is not available. " + e);
                // 清除池中该地址对应的对象(强硬移除, 不管当前此对象状态是否为忙, 意思就是调用方丢弃
                // 此对象或对象池丢弃此对象都行)
                if (objects != null) {
                    Enumeration<PooledObject> enumerate = objects.elements();
                    while (enumerate.hasMoreElements()) {
                        PooledObject pObj = enumerate.nextElement();
                        if (s.equals(pObj.getIp())) {
                            objects.removeElement(pObj);
                            listUsAble.remove(s);
                            log.info(s + "地址不可用,清除池子中的对应对象");
                        }
                    }
                }
            }
        }
    }
}

```

`balanceObjectNum()` 方法: 平衡对象池子中的空闲对象; 若对象池中对象的数量大于设置的数量5且空闲的数量大于配置文件设置的数值3时, 删除池中多余的空闲对象。

```

public void balanceObjectNum() {
    if (objects.size() > numObjects) {
        int count = 0;
        Enumeration<PooledObject> enumerate = objects.elements();
        while (enumerate.hasMoreElements()) {
            PooledObject pObj = enumerate.nextElement();
            if (pObj.isBusy() == false) {
                // 空闲数量+1
                count++;
                if (count > Idlenum) {
                    objects.removeElement(pObj);
                }
            }
        }
        log.info("池子中的对象数量为" + objects.size() + "大于设置数量" + numObjects + "且当前
                空闲对象数量为" + count + "若大于保活空闲数量" + Idlenum
                + "会进行空闲对象回收");
    }
}

```

```
}  
}
```

对外提供接口获取对象 `getObject ()` :从对象池中获取一个空闲地可用对象, 无的话进行扩容后获取

```
public static synchronized PooledObject getObject() {  
    //      log.info("*****获取一个可用的PooledObject对象  
    *****");  
    // 从对象中获得一个可用的对象  
    PooledObject obj = findFreeObject();  
    // 进行扩容  
    if (obj == null) {  
        capacityExpansion();  
        // 重新从池中查找是否有可用对象  
        obj = findFreeObject();  
    }  
    return obj;  
}
```

扩容方法 `capacityExpansion()` :先判断是否存在可用地址, 后根据对象池中目前对象进行判断: 先整合对象中的ip,提取使用数量最少的ip,用使用数量最少的ip进行扩容。

```
public static synchronized void capacityExpansion() {  
    // 当前无可用地址, 则不需要进行扩容  
    if (listUsAble.size() != 0) {  
        Map<String, Long> collect = objects.stream()  
            .collect(Collectors.groupingBy(PooledObject::getIp,  
Collectors.counting()));  
        log.info("(扩容)当前可用地址集合的size为" + listUsAble.size() + ",池子中的对象数量为" +  
objects.size() + ",当前对象池中每个地址的计数统计为"  
            + collect);  
        Long value = collect.values().stream().sorted().findFirst().get();  
        List<String> result = collect.entrySet().stream()  
            .filter(kvEntry -> Objects.equals(kvEntry.getValue(),  
value)).map(Map.Entry::getKey)  
            .collect(Collectors.toList());  
        for (String s : result) {  
            addObjectInPool(s);  
        }  
    }  
}
```

该机制存在的问题:

以实际运行过程日志为例子 (一直存在获取对象使用对象)

```

l : 初始化Ice连接器完毕
ll 132 : 找到可用地址dataProcIttcp -h 200.200.220.48 -p 10011,并生成可用对象到线程池子 1
ll 132 : 找到可用地址dataProcIttcp -h 200.200.220.49 -p 10011,并生成可用对象到线程池子 2
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为2,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=1, dataProcIttcp -h 200.200.220.49 -p 10011=1) 3
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为4,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=2, dataProcIttcp -h 200.200.220.49 -p 10011=2) 4
ll 137 : 池子中的对象数量为大于设置数量5且当前空闲对象数量为5若大于保活空闲数量3会进行空闲对象回收 5
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为4,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=1) 6
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=2) 7
ll 137 : 池子中的对象数量为大于设置数量5且当前空闲对象数量为5若大于保活空闲数量3会进行空闲对象回收 8
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为4,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=1) 9
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=2) 10
ll 135 : The current address dataProcIttcp -h 200.200.220.49 -p 10011 is not available. Ice.ConnectFailedException 11
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象 12
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象 13
!IceClientPool 218 : (扩容)当前可用地址集合的size为1,池子中的对象数量为4,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=1)14
!IceClientPool 218 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=2)15
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象 16
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象 17
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象 18
!IceClientPool 218 : (扩容)当前可用地址集合的size为1,池子中的对象数量为4,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=1) 19
!IceClientPool 218 : (扩容)当前可用地址集合的size为1,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcIttcp -h 200.200.220.48 -p 10011=3, dataProcIttcp -h 200.200.220.49 -p 10011=2)
ll 135 : The current address dataProcIttcp -h 200.200.220.49 -p 10011 is not available. Ice.ConnectFailedException
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象
ll 144 : dataProcIttcp -h 200.200.220.49 -p 10011地址不可用,清除池子中的对应对象

```

- 1、ip48可用，创建1个ip为48的对象到池子中（目前池子中 48:1个）
- 2、ip49可用（目前池子中 48:1 49:1）
- 3、由于一直存在获取对象使用对象，池子中2个不够用，将进行扩容。根据扩容机制，池子中使用ip最少的是1个，所以48和49都会扩容1个。由于日志是在创建对象前打印，所以第3条日志后实际对象池子情况（48:2 49:2）
- 4、（目前池子中 48:3 49:3）
- 5、由于池子中对象数量为6大于设置的平衡数量5，将判断6个中空闲对象的数量并最多留3个空闲的对象（目前池子中 48:3 49:1）
- 6、添加了一个49的对象（48:3 49:2）

....

- 11、手动退出49的ice (48:3 49:3)
 - 12和13、原则上需要将池子中49的对象都删了，实际只删了2个（最后实际上池子中48:3 49:1）
- 导致后续扩容中添加的都是49的对象。获取对象也可能是49的对象。

只删除2个的原因：

```

Enumeration<PooledObject> enumerate = objects.elements();
while (enumerate.hasMoreElements()) {
    PooledObject pObj = enumerate.nextElement();
    if (s.equals(pObj.getIp())) {
        objects.removeElement(pObj);
        listUsAble.remove(s);
        log.info(s + "地址不可用,清除池子中的对应对象");
    }
}

```

在使用Enumeration遍历的过程中删除元素。

当我们在 Enumeration 中使用 nextElement() 来获取下一个元素时，如果当前元素被移除，这样可能会跳过一个元素，导致部分匹配的 PooledObject 实例没有被检查。换句话说，删除当前元素后，Enumeration 的状态并没有更新，因此可能会错过后续的匹配项。

扩容只找使用最少的ip扩容，存在的问题是：若当前有2个ip可用，但池中因为平衡空闲对象将其中1个ip49的对象都删了，只留下了另外一个ip48的对象，这时ip48地址不可用了的话，池中ip48的对象都被删除，后续池中就没对象了，扩容也加不了。此时若ip49也不可用的话，因为池中无对象，从可用地址集中删除ip49的步骤也执行不了；一直会记录ip49的地址还可用。

```
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=4, dataProcI:tcp -h 200.200.220.49 -p 10011=1)
tPool 201 : 池子中的对象数量为5大于设置数量s且当前空闲对象数量为s若大于保活空闲数量3会进行空闲对象回收
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=4, dataProcI:tcp -h 200.200.220.49 -p 10011=1)
tPool 201 : 池子中的对象数量为5大于设置数量s且当前空闲对象数量为s若大于保活空闲数量3会进行空闲对象回收
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=5)
tPool 201 : 池子中的对象数量为5大于设置数量s且当前空闲对象数量为s若大于保活空闲数量3会进行空闲对象回收
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为5,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=5)
tPool 201 : 池子中的对象数量为5大于设置数量s且当前空闲对象数量为s若大于保活空闲数量3会进行空闲对象回收
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为7,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=7)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为7,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=8)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为9,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=9)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为10,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=10)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为11,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=11)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为12,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=12)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为13,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=13)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为14,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=14)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为15,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=15)
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为2,池子中的对象数量为16,当前对象池中每个地址的计数统计为(dataProcI:tcp -h 200.200.220.48 -p 10011=16)
tPool 130 : The current address dataProcI:tcp -h 200.200.220.48 -p 10011 is not available. Ice.ConnectFailedException
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
tPool 148 : dataProcI:tcp -h 200.200.220.48 -p 10011地址不可用,清除池子中的对应对象
IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
Ice.IceClientPool 232 : (扩容)当前可用地址集合的size为1,池子中的对象数量为0,当前对象池中每个地址的计数统计为()
```

解决方案

遍历容器的同时删除元素导致没完全删除需要删除的元素：

```
//地址不可用删除池子中对象
log.info("The current address " + s + " is not available. " + e);
listUsAble.remove(s);
// 清除池中该地址对应的对象(强硬移除,不管当前此对象状态是否为忙,意思就是调用方丢弃此对象或对象池丢弃此对象都行)
if (objects != null) {
    List<PooledObject> rmpObjList = new ArrayList<>();
    Enumeration<PooledObject> enumerate = objects.elements();
    while (enumerate.hasMoreElements()) {
        PooledObject pObj = enumerate.nextElement();
        if (s.equals(pObj.getIp())) {
            rmpObjList.add(pObj);
        }
    }
    for (PooledObject pooledObject : rmpObjList){
        objects.removeElement(pooledObject);
        log.info(s + "地址不可用,清除池子中的对应对象");
    }
}
```

```

//平衡池中空闲对象
    if (objects.size() > numObjects) {
        int count = 0;
        List<PooledObject> rmpObjList = new ArrayList<>();
        Enumeration<PooledObject> enumerate = objects.elements();
        while (enumerate.hasMoreElements()) {
            PooledObject pObj = enumerate.nextElement();
            if (pObj.isBusy() == false) {
                // 空闲数量+1
                count++;
                if (count > Idlenum) {
                    rmpObjList.add(pObj);
                }
            }
        }
        log.info("池子中的对象数量为" + objects.size() + "大于设置数量" + numObjects + "且当前
        空闲对象数量为" + count + "若大于保活空闲数量" + Idlenum
                + "会进行空闲对象回收");
        for (PooledObject pooledObject : rmpObjList){
            objects.removeElement(pooledObject);
            log.info("平衡池中空闲对象,清除池中ip:" + pooledObject.getIp() + " 的1个空闲对
            象");
        }
    }
}

```

遍历的时候先把需要删除的对象存到 `rmpObjList` 中，先整理需要删除的对象，遍历完后续一起删除。

扩容时只扩使用最少次数的ip对象：

```

// 判断可用地址集合中所有的地址是否在池子中都用到
boolean hasAvailableIp = listUsAble.stream().allMatch(ip ->
collect.containsKey(ip));
    if (hasAvailableIp) {
        //可用地址都有用到
        Long minCount = collect.values().stream().sorted().findFirst().get();
        List<String> result = collect.entrySet().stream()
                .filter(kvEntry -> Objects.equals(kvEntry.getValue(),
minCount)).map(Map.Entry::getKey)
                .collect(Collectors.toList());
        for (String s : result) {
            log.info("扩容添加使用次数最少的地址: " + s);
            addObjectInPool(s);
        }
    } else {
        //存在可用地址没有用到
        for (String s : listUsAble) {
            if (objects.stream().noneMatch(obj -> obj.getIp().equals(s))) {
                log.info("扩容添加未用到的可用地址: " + s);
                addObjectInPool(s);
            }
        }
    }
}

```

先判断可用地址集合 `listUsAble` 中的所有地址是否都有对象在池子中，若都有则扩容添加使用次数最少的；若有 ip 未用到则扩容未用到的 ip 的对象。

对象池中**没有对象**，此时有 ip 地址不可用，删除不了可用地址集合 `listUsAble` 中该地址的问题：

寻找可用地址方法 `FindAvailableAddr()` 中将删除地址的操作放到判断对象池是否为 null 之前

```
    } catch (Exception e) {
        log.info("The current address " + s + " is not available. " + e);
        listUsAble.remove(s); //有地址不可用，先在listUsAble中删除该地址
        // 清除池中该地址对应的对象(强硬移除，不管当前此对象状态是否为忙，意思就是调用方丢弃
        // 此对象或对象池丢弃此对象都行)
        if (objects != null) {
            List<PooledObject> rmpObjList = new ArrayList<>();
            Enumeration<PooledObject> enumerate = objects.elements();
            while (enumerate.hasMoreElements()) {
                PooledObject pObj = enumerate.nextElement();
                if (s.equals(pObj.getIp())) {
                    rmpObjList.add(pObj);
                }
            }
            for (PooledObject pooledObject : rmpObjList){
                objects.removeElement(pooledObject);
                log.info(s + "地址不可用,清除池子中的对应对象");
            }
        }
    }
}
```