[Thrift连接池优化](http://blog.csdn.net/chen7253886/article/details/52779471)

标签： [thrift](http://www.csdn.net/tag/thrift)[优化](http://www.csdn.net/tag/%e4%bc%98%e5%8c%96)

2016-10-10 17:06 638人阅读 [评论](http://blog.csdn.net/chen7253886/article/details/52779471#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chen7253886/article/details/52779471#report)

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

thrift rpc（7） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

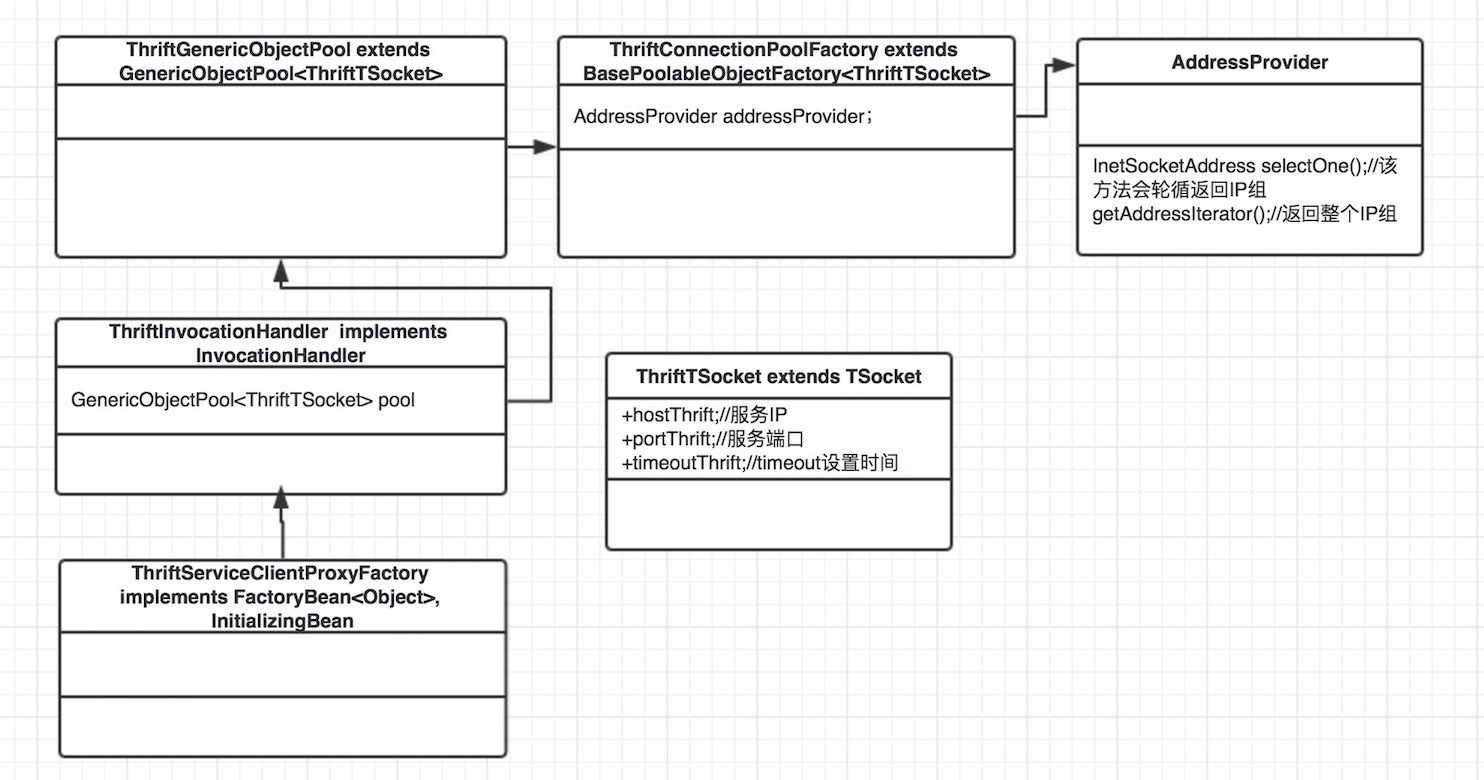
目录[(?)[+]](http://blog.csdn.net/chen7253886/article/details/52779471)

背景

        众所周知，thrift是一款很优秀的rpc框架，公司今年在部门间推行thrift框架来提高部门间的通信效率，作者本人的工作内容主要是作为客户端(本人所在组为服务端，对于提供服务的其他部门来说是客户端)调用其他部门的接口，在工作过程中发现thrift有个较大的弊端，一般情况下服务端会向客户端提供一组服务IP，所有的负载均衡工作，连接是否可用等工作都需要客户端自己来维护，而apache本身提供的thrift、GenericObjectPool 并没有提供相应的机制来保证当服务端部分机器服务不可用时及时切换，TSocket.isOpen()方法只能检测到连接是否打开，在已打开的情况下然后网络中断或服务宕机，该方法是无法检测到连接的可用性，所以整个工作需要我们自己来维护，本文作者将介绍下自己在开发过程中碰到的问题以及改进方式，欢迎有更好的处理方式的同学分享经验。   
        本文前几节主要介绍优化过程中的策略，最后一节会用最新的代码来展示最新的策略实现。

第一版大概设计

        首先确定了连接池整体框架如下图：



* ThriftTSocket：该类继承自TSocket，创建该类而不使用原生TSocket有两点原因，1.原生的TTransport无法获取到Socket对象，进而无法获取到某条连接的ip,port信息，并且Socket在关闭时也获取不到ip,port信息，所以创建ThriftTSocket，里面添加ip,port,timeout属性,方便负载均衡时使用。2.虽然thrift底层支持多种TTransport协议，但业务中只使用TSocket，所以不必支持所有协议，连接池只支持保存ThriftTSocket。
* AddressProvider：用于保存服务端的ip和port信息，对外提供selectOne()方法，该方法轮循所有IP并按队列形式返回其中一个IP，用于负载均衡，将连接平均地创建到服务端机器。
* ThriftConnectionPoolFactory：连接创建工厂，从AddressProvider获取一个IP并创建ThriftTSocket，当创建一次失败时，会轮循一遍IP列表尝试创建。
* ThriftGenericObjectPool：连接池，直接继承自GenericObjectPool，用于保存ThriftTSocket，对构造方法有小改动。
* ThriftInvocationHandler：负责从连接池获取ThriftTSocket，转为可调用的thrift client端后执行方法调用，并进行连接响应时间统计等其他工作。
* ThriftServiceClientProxyFactory：代理工厂，提供给业务层调用，会加载thrift服务的Client Factory相关类，并创建ThriftInvocationHandler。

        在确定了整体框架后，设计具体策略如下：

* ThriftGenericObjectPool：连接池参数设置maxActive=maxIdle=minIdle；连接的保存形式采用队列；其他参数默认即可
* ThriftInvocationHandler：当一条连接执行中发生timeout异常时（对方服务出现问题，网络问题可能性较小），将该连接close并返回给连接池，下次获取到尝试再open，并不从连接池中remove
* ThriftInvocationHandler：对每条连接对应的IP做响应时间统计，当某个IP的平均响应时间超过阀值时，将该IP标记为“切服”状态，下次从连接池中取出该IP的连接时，会直接返回null，一定时间后“切服”状态自动清除

        制定策略的依据在于：虽然ThriftConnectionPoolFactory在创建连接时会轮循对方的服务IP，但是当部分服务IP宕机时，会创建新的连接集中打到正常的机器上，恢复后的机器无法收到任何请求，所以考虑到负载均衡问题，连接池采用maxActive=maxIdle=minIdle策略，项目启动时便均衡的创建到每个IP的连接，不remove有问题的连接来保证请求均匀打到对方机器，为了防止对方服务有问题造成我方大量线程阻塞的情况，又加入了“切服”机制进行降级处理。   
        采用此种策略后，在压测过程中发现，当后端服务宕机时，并不影响客户端机器的运行，并且达到了负载均衡的效果，后端可以均匀的收到的请求，但产生的较大的一个问题是：由于部分IP进入“切服”状态，到该IP的连接处理的请求的速度会非常快（直接return null比调用thrift接口快得非常多），进而导致绝大数请求拿到了空数据，而不是按照“切服”IP连接在连接池中的比例数取不到数据。

第二版优化

        为了解决部分IP进入“切服”导致的绝大多数请求拿不到数据的情况，优化策略如下：

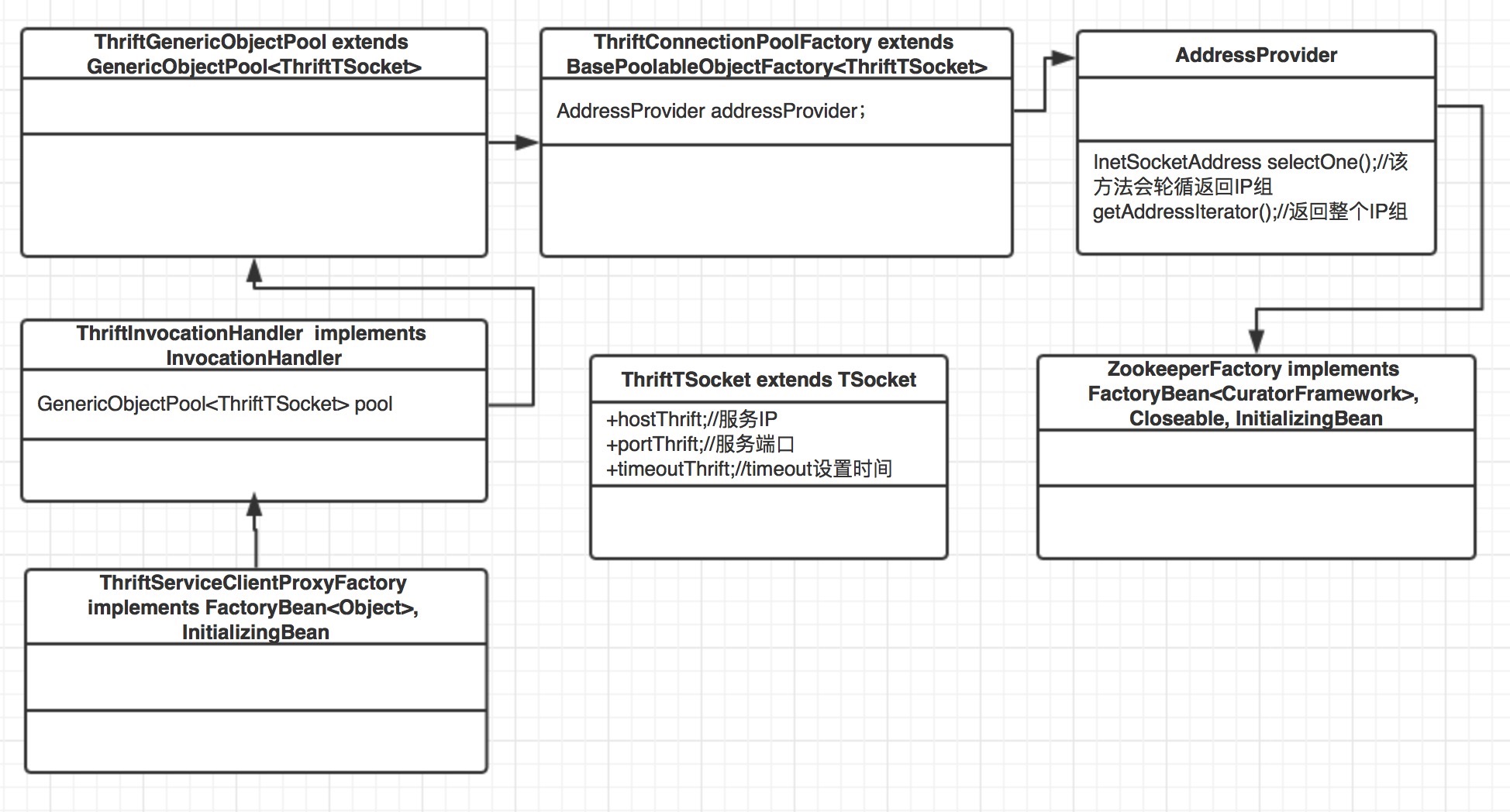
* ThriftGenericObjectPool：连接池参数设置maxActive=maxIdle，minIdle=0；连接的保存形式使用栈结构；启用监控线程，每隔1分钟清理一次空闲连接；设置获取连接策略为阻塞time时间后抛出异常
* ThriftInvocationHandler：当某一条连接发生timeout时，认为该条连接不可用(初始化timeout值较大，所以发生 timeout时可认为服务不用)，并从连接池中remove掉该条连接
* ThriftInvocationHandler： “切服”逻辑改为针对整个服务，不再针对某一IP，当获取连接时获取到一定量的阻塞抛出异常情况，认为整个服务不可用，对其“切服”，所有的请求均return null，一定时间后“切服”状态自动清除
* ThriftConnectionPoolFactory:删除轮循所有IP行为，由于设置了阻塞时间，这里的创建过程用时久，代码执行不到。   
          优化策略的依据：当连接发生timeout时从连接池中remove掉，并创建新的可用连接，这样可保证所有请求都会取得数据。另外，为了方便连接的重建，设置minIdle=0，在半夜QPS较低的时候可将连接回收，白天QPS提高时重新均衡地创建连接到所有服务器上。由于发生timeout时将该条连接清除(表明该IP已不可用)，所以后续是无法创建到该IP的连接，进而无法统计该IP的响应时间，也就不会进入“切服”逻辑，所以调整为针对整个服务进行“切服”，例如：6台机器3台宕机，但可以支撑起服务，就认为其服务可用，并且起到一定的负载均衡作用，当6台机器都宕机时，认为其服务不可用，进入“切服”，宕掉的机器在“切服”期间都恢复后，“切服”状态清除后，又可均衡的创建连接到6台机器。   
          优化后的策略可以提高请求的正确性，也达到了一定的负载均衡效果，但还是存在以下两个问题：   
  1. 当服务端需要添加服务器IP或者删除服务器IP时，需要人工通知客户端修改配置文件并重启服务
  2. 晚上QPS较高时，如果部分机器宕机后恢复，由于连接已经稳定，无法新创建到恢复后的机器连接，只能等到晚上QPS下降后清除连接，白天QPS上升才能均匀创建连接

第三版优化

        目前优化的最后一版，额外添加了zookeeper监控服务，zookeeper由运维提供给客户端，作者这里只负责调用zookeeper集群，具体策略如下：

* AddressProvider：添加zookeeper客户端进行实时监控，当集群注册的服务有更改时，及时修改本地的IP列表。
* ThriftInvocationHandler：在远程方法调用结束后，加入连接统计功能，统计连接池中每个IP的连接数，并将其与本地的IP列表进行对比，发现负载不均衡时及时删除当前连接，并创建合适的连接。

        加入zookeeper监控和统计连接功能后，能够达到实时负载均衡的功能，同时保证了本服务的高可用性，目前统计连接功能还未实现，后面的代码只包括zookeeper监控功能，整个框架图改进如下：



        ZookeeperFactory：用于生成一个CuratorFramework客户端，维护更新AddressProvider的服务列表。

代码实现

1.连接类 ThriftTSocket

package省略

import省略

public class ThriftTSocket extends TSocket {

private String hostThrift;//连接的ip

private int portThrift;//连接的port

private int timeoutThrift;//连接设置的timeout时长

public ThriftTSocket(String host, int port, int timeout) throws TTransportException {

super(host, port, timeout);

this.hostThrift = host;

this.portThrift = port;

this.timeoutThrift = timeout;

}

public String getHostThrift() {

return this.hostThrift;

}

public int getPortThrift() {

return this.portThrift;

}

public int getTimeoutThrift() {

return this.timeoutThrift;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

2.第三方服务状态

package省略

import省略

public class ThriftServiceStatus {

private static Log serviceStatusLoger = LogFactory.getLog("serviceStatusLoger");

private static Log apiSwitchMonitor = LogFactory.getLog("apiSwitchMonitorLog");

private static final int INTERFACE\_TOTAL\_COUNT = 30;// 服务访问次数达到该值进行切服计算

private static final long INTERFACE\_RECORD\_COUNT\_RESET\_TIME = 1 \* 60 \* 1000;// 服务统计量重置时间间隔

private static final long INTERFACE\_AUTO\_NORMAL\_TIME = 5 \* 60 \* 1000;// 接口自动恢复时间5分钟

/\*\*

\* 第三方服务异常次数

\*/

private int count;

/\*\*

\* 第三方服务开始计数时间

\*/

private long recordStartTime;

/\*\*

\* 第三方服务最近一次关闭时间

\*/

private long closeTime;

/\*\*

\* 第三方服务名称

\*/

private String serviceName;

private Lock lock;

public ThriftServiceStatus(String serviceName) {

this.recordStartTime = System.currentTimeMillis();

this.count = 0;

this.closeTime = 0;

this.lock = new ReentrantLock();

this.serviceName = serviceName;

}

/\*\*

\* 在切服时间内，服务处于不可用状态

\* @return

\*/

public boolean ifServiceUsable() {

return (System.currentTimeMillis() - this.closeTime) > INTERFACE\_AUTO\_NORMAL\_TIME;

}

public void checkThriftServiceStatus() {

this.lock.lock();

try {

this.count++;

serviceStatusLoger.info("[this service " + this.serviceName + "]" + " has exceptions. count:["

+ this.count + "] recordStartTime:[" + TimeUtil.timestamp2date(recordStartTime) + "] nowTime:["

+ TimeUtil.timestamp2date(System.currentTimeMillis()) + "]");

// 服务异常次数 超过统计阀值时 进行 切服

if (this.count >= INTERFACE\_TOTAL\_COUNT) {

this.closeTime = System.currentTimeMillis();// 更新关闭时间

apiSwitchMonitor.info("[" + this.serviceName + "]" + " close time:["

+ TimeUtil.timestamp2date(this.closeTime) + "]" + "average response time:[10000]");

}

// 1分钟后 重置计数

if ((System.currentTimeMillis() - recordStartTime) > INTERFACE\_RECORD\_COUNT\_RESET\_TIME) {

this.count = 0;

this.recordStartTime = System.currentTimeMillis();

}

} finally {

this.lock.unlock();

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76

3.CuratorFramework客户端，连接zookeeper集群

package省略

import省略

public class ZookeeperFactory implements FactoryBean<CuratorFramework>, Closeable, InitializingBean {

private static Logger LOGGER = LoggerFactory.getLogger(ZookeeperFactory.class);

/\*\*

\* zookeeper集群地址

\*/

private String zookeeperHosts;

// session超时

private int sessionTimeout = 3000;

private int connectionTimeout = 3000;

private CuratorFramework zkClient;

// 第三方未提供，所以暂时用不到

private String namespace;

public void setZookeeperHosts(String zookeeperHosts) {

this.zookeeperHosts = zookeeperHosts;

}

public void setSessionTimeout(int sessionTimeout) {

this.sessionTimeout = sessionTimeout;

}

public void setConnectionTimeout(int connectionTimeout) {

this.connectionTimeout = connectionTimeout;

}

public void setNamespace(String namespace) {

this.namespace = namespace;

}

public void setZkClient(CuratorFramework zkClient) {

this.zkClient = zkClient;

}

@Override

public CuratorFramework getObject() throws Exception {

return this.zkClient;

}

@Override

public void afterPropertiesSet() throws Exception {

if (StringUtil.isBlank(this.zookeeperHosts)) {

return;

}

this.zkClient = this.create(zookeeperHosts, sessionTimeout, connectionTimeout, namespace);

this.zkClient.start();

}

private CuratorFramework create(String connectString, int sessionTimeout, int connectionTimeout, String namespace) {

try {

CuratorFrameworkFactory.Builder builder = CuratorFrameworkFactory.builder();

return builder.connectString(connectString).sessionTimeoutMs(sessionTimeout).connectionTimeoutMs(30000)

.canBeReadOnly(true).namespace(namespace)

.retryPolicy(new ExponentialBackoffRetry(1000, Integer.MAX\_VALUE)).defaultData(null).build();

} catch (Exception e) {

LOGGER.error("ZookeeperFactory create error", e);

throw e;

}

}

public void close() {

if (zkClient != null) {

zkClient.close();

}

}

@Override

public Class<?> getObjectType() {

return CuratorFramework.class;

}

@Override

public boolean isSingleton() {

return true;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85

4.保存服务IP类

package省略

import省略

public class AddressProvider {

private static Logger LOGGER = LoggerFactory.getLogger(AddressProvider.class);

/\*\*

\* 最新的服务器IP列表，由zookeeper来维护更新

\*/

private List<InetSocketAddress> serverAddresses = new CopyOnWriteArrayList<InetSocketAddress>();

/\*\*

\* 没有配置zookeeper时使用原来配置文件中的IP列表

\*/

private List<InetSocketAddress> backupAddresses = new LinkedList<InetSocketAddress>();

/\*\*

\* 轮循队列，获取IP时使用

\*/

private Queue<InetSocketAddress> loop = new LinkedList<InetSocketAddress>();

private Lock loopLock = new ReentrantLock();

/\*\*

\* zookeeper 监控

\*/

private PathChildrenCache cachedPath;

public AddressProvider() {

}

public AddressProvider(String backupAddress, CuratorFramework zkClient, String zookeeperPath) throws Exception {

// 默认使用配置文件中的IP列表

this.backupAddresses.addAll(this.transfer(backupAddress));

this.serverAddresses.addAll(this.backupAddresses);

Collections.shuffle(this.backupAddresses);

Collections.shuffle(this.serverAddresses);

// 配置zookeeper时，启动客户端

if (!StringUtil.isBlank(zookeeperPath) && zkClient != null) {

this.buildPathChildrenCache(zkClient, zookeeperPath, true);

cachedPath.start(StartMode.POST\_INITIALIZED\_EVENT);

}

}

public InetSocketAddress selectOne() {

loopLock.lock();

try {

if (this.loop.isEmpty()) {

this.loop.addAll(this.serverAddresses);

}

return this.loop.poll();

} finally {

loopLock.unlock();

}

}

public Iterator<InetSocketAddress> addressIterator() {

return this.serverAddresses.iterator();

}

/\*\*

\* 初始化 cachedPath，并添加监听器，当zookeeper上任意节点数据变动时，更新本地serverAddresses

\* @param client

\* @param path

\* @param cacheData

\* @throws Exception

\*/

private void buildPathChildrenCache(final CuratorFramework client, String path, Boolean cacheData) throws Exception {

final String logPrefix = "buildPathChildrenCache\_" + path + "\_";

cachedPath = new PathChildrenCache(client, path, cacheData);

cachedPath.getListenable().addListener(new PathChildrenCacheListener() {

@Override

public void childEvent(CuratorFramework client, PathChildrenCacheEvent event) throws Exception {

PathChildrenCacheEvent.Type eventType = event.getType();

switch (eventType) {

case CONNECTION\_RECONNECTED:

LOGGER.info(logPrefix + "Connection is reconection.");

break;

case CONNECTION\_SUSPENDED:

LOGGER.info(logPrefix + "Connection is suspended.");

break;

case CONNECTION\_LOST:

LOGGER.warn(logPrefix + "Connection error,waiting...");

return;

case INITIALIZED:

LOGGER.warn(logPrefix + "Connection init ...");

default:

}

// 任何节点的时机数据变动,都会rebuild,此处为一个"简单的"做法.

cachedPath.rebuild();

rebuild();

}

private void rebuild() throws Exception {

List<ChildData> children = cachedPath.getCurrentData();

if (CollectionUtils.isEmpty(children)) {

// 有可能所有的thrift server都与zookeeper断开了链接

// 但是 thrift client与thrift server之间的网络是良好的

// 因此此处是否需要清空serverAddresses,是需要多方面考虑的.

// 这里我们认为zookeeper的服务是可靠的，数据为空也是正确的

serverAddresses.clear();

LOGGER.error(logPrefix + "server ips in zookeeper is empty");

return;

}

List<InetSocketAddress> lastServerAddress = new LinkedList<InetSocketAddress>();

for (ChildData data : children) {

String address = new String(data.getData(), "utf-8");

lastServerAddress.add(transferSingle(address));

}

// 更新本地IP列表

serverAddresses.clear();

serverAddresses.addAll(lastServerAddress);

Collections.shuffle(serverAddresses);

}

});

}

/\*\*

\* 将String地址转换为InetSocketAddress

\* @param serverAddress

\* 10.183.222.59:1070

\* @return

\*/

private InetSocketAddress transferSingle(String serverAddress) {

if (StringUtil.isBlank(serverAddress)) {

return null;

}

String[] address = serverAddress.split(":");

return new InetSocketAddress(address[0], Integer.valueOf(address[1]));

}

/\*\*

\* 将多个String地址转为InetSocketAddress集合

\* @param serverAddresses

\* ip:port;ip:port;ip:port;ip:port

\* @return

\*/

private List<InetSocketAddress> transfer(String serverAddresses) {

if (StringUtil.isBlank(serverAddresses)) {

return null;

}

List<InetSocketAddress> tempServerAdress = new LinkedList<InetSocketAddress>();

String[] hostnames = serverAddresses.split(";");

for (String hostname : hostnames) {

tempServerAdress.add(this.transferSingle(hostname));

}

return tempServerAdress;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152

5.连接创建工厂

package省略

import省略

public class ThriftConnectionPoolFactory extends BasePoolableObjectFactory<ThriftTSocket> {

private static Logger LOGGER = LoggerFactory.getLogger(ThriftConnectionPoolFactory.class);

private final AddressProvider addressProvider;

private int timeout = 2000;

protected ThriftConnectionPoolFactory(AddressProvider addressProvider) throws Exception {

this.addressProvider = addressProvider;

}

@Override

public ThriftTSocket makeObject() throws Exception {

String logPrefix = "makeObject\_";

ThriftTSocket thriftTSocket = null;

InetSocketAddress address = null;

Exception exception = null;

try {

address = this.addressProvider.selectOne();

thriftTSocket = new ThriftTSocket(address.getHostName(), address.getPort(), timeout);

thriftTSocket.open();

LOGGER.info(logPrefix + "connect server:[" + address.getHostName() + ":" + address.getPort() + "] success");

} catch (Exception e) {

LOGGER.error(logPrefix + "connect server[" + address.getHostName() + ":" + address.getPort() + "] error: ",

e);

exception = e;

thriftTSocket = null;// 这里是为了下面连接其他服务器

}

// 轮循所有ip

if (thriftTSocket == null) {

String hostName = address.getHostName();

int port = address.getPort();

Iterator<InetSocketAddress> addressIterator = this.addressProvider.addressIterator();

while (addressIterator.hasNext()) {

try {

address = addressIterator.next();

// 不再尝试连接之前已经连接失败的主机

if (address.getHostName().equals(hostName) && address.getPort() == port) {

continue;

}

thriftTSocket = new ThriftTSocket(address.getHostName(), address.getPort(), timeout);

thriftTSocket.open();

LOGGER.info(logPrefix + "connect server:[" + address.getHostName() + ":" + address.getPort()

+ "] success");

break;

} catch (Exception e) {

LOGGER.error(logPrefix + "connect server[" + address.getHostName() + ":" + address.getPort()

+ "] error: ", e);

exception = e;

thriftTSocket = null;

}

}

}

// 所有服务均无法建立连接时抛出异常

if (thriftTSocket == null) {

throw exception;

}

return thriftTSocket;

}

@Override

public void destroyObject(ThriftTSocket tsocket) throws Exception {

if (tsocket != null) {

try {

tsocket.close();

} catch (Exception e) {

}

}

}

@Override

public boolean validateObject(ThriftTSocket tsocket) {

if (tsocket == null) {

return false;

}

// 在成功创建连接后，将网络断掉，这里调用还是true

return tsocket.isOpen();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83

6.连接池

package省略

import省略

public class ThriftGenericObjectPool extends GenericObjectPool<ThriftTSocket> {

public ThriftGenericObjectPool(AddressProvider addressProvider, int maxActive, int maxIdle, int minIdle,

long maxWait) throws Exception {

/\*\*

\* 池策略：最大连接数，最大等待时间，最大空闲数，最小空闲数由人工配置，

\* 最大连接数尽量=最大空闲数，最小空闲数尽量为0，以便清除无用线程

\* 其他参数写死：

\* GenericObjectPool.WHEN\_EXHAUSTED\_BLOCK：获取连接时阻塞超时时抛出异常

\* GenericObjectPool.DEFAULT\_TEST\_ON\_BORROW

\* GenericObjectPool.DEFAULT\_TEST\_ON\_RETURN

\* 60\*1000，检测空闲线程60秒运行一次

\* 5，检测空闲线程运行一次检测 5条连接

\* 60\*10\*1000，空闲线程最小空闲时间10分钟，超过10分钟后会被检测线程 remove

\* GenericObjectPool.DEFAULT\_TEST\_WHILE\_IDLE

\* GenericObjectPool.DEFAULT\_SOFT\_MIN\_EVICTABLE\_IDLE\_TIME\_MILLIS

\* GenericObjectPool.DEFAULT\_LIFO 后入先出队列，保证不是所有的连接都在使用，及时被清除

\*/

super(new ThriftConnectionPoolFactory(addressProvider), maxActive, GenericObjectPool.WHEN\_EXHAUSTED\_BLOCK,

maxWait, maxIdle, minIdle, GenericObjectPool.DEFAULT\_TEST\_ON\_BORROW,

GenericObjectPool.DEFAULT\_TEST\_ON\_RETURN, 60 \* 1000, 5, 60 \* 10 \* 1000,

GenericObjectPool.DEFAULT\_TEST\_WHILE\_IDLE,

GenericObjectPool.DEFAULT\_SOFT\_MIN\_EVICTABLE\_IDLE\_TIME\_MILLIS, GenericObjectPool.DEFAULT\_LIFO);

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

7.执行器

package省略

import省略

public class ThriftInvocationHandler implements InvocationHandler {

private static Logger LOGGER = LoggerFactory.getLogger(ThriftInvocationHandler.class);

private static Log httpClientUtilLogger = LogFactory.getLog(HttpClientUtil.class);

private GenericObjectPool<ThriftTSocket> pool; // 连接池

private TServiceClientFactory<TServiceClient> tServiceClientFactory = null;

private Integer protocol;

private ThriftServiceStatus thriftServiceStatus;// 服务状态

private AddressProvider addressProvider;

public ThriftInvocationHandler(GenericObjectPool<ThriftTSocket> pool,

TServiceClientFactory<TServiceClient> tServiceClientFactory, Integer protocol, String serviceName,

AddressProvider addressProvider) {

this.pool = pool;

this.tServiceClientFactory = tServiceClientFactory;

this.protocol = protocol;

this.thriftServiceStatus = new ThriftServiceStatus(serviceName);

this.addressProvider = addressProvider;

}

@Override

public Object invoke(Object proxy, Method method, Object[] args) throws Exception {

String logPrefix = "ThriftInvocationHandler\_";

ThriftTSocket thriftTSocket = null;

boolean ifBorrowException = true;

try {

// 服务处于“切服”状态时 直接返回null

if (!this.thriftServiceStatus.ifServiceUsable()) {

return null;

}

// 当第三方服务不可用时，会阻塞在这里一定时间后抛出异常，并进行服务状态统计

thriftTSocket = this.pool.borrowObject();

ifBorrowException = false;

String interfaceWholeName = this.getInterfaceName(method) + "&ip=" + thriftTSocket.getHostThrift() + ":"

+ thriftTSocket.getPortThrift();

LOGGER.info(logPrefix + interfaceWholeName + " borrowed:" + this.pool.getNumActive() + " idle:"

+ this.pool.getNumIdle() + " total :" + (this.pool.getNumActive() + this.pool.getNumIdle()));

long startTime = System.currentTimeMillis();

long costTime;

Object o = null;

try {

o = method.invoke(this.tServiceClientFactory.getClient(this.getTProtocol(thriftTSocket)), args);

costTime = System.currentTimeMillis() - startTime;

httpClientUtilLogger.info(this.getUrl(interfaceWholeName, args) + "|200|0|" + costTime + "|0");

} catch (Exception e) {

costTime = System.currentTimeMillis() - startTime;

httpClientUtilLogger.error(this.getUrl(interfaceWholeName, args) + "|000|0|" + costTime + "|1");

// 抛出异常的连接认为不可用，从池中remove掉

this.pool.invalidateObject(thriftTSocket);

thriftTSocket = null;

o = null;

}

return o;

} catch (Exception e) {

LOGGER.error("thrift invoke error", e);

if (ifBorrowException) {

this.thriftServiceStatus.checkThriftServiceStatus();

}

return null;

} finally {

if (thriftTSocket != null) {

this.pool.returnObject(thriftTSocket);

}

}

}

private String getInterfaceName(Method method) {

String interfaceName = method.getDeclaringClass().toString();

interfaceName = interfaceName.substring(10, interfaceName.length());

return interfaceName + "$" + method.getName();

}

private String getUrl(String service, Object[] args) {

StringBuilder wholeUrl = new StringBuilder("thrift://");

wholeUrl.append(service.substring(service.lastIndexOf("$") + 1, service.indexOf("&ip="))).append("/?")

.append("service=").append(service);

if (args != null) {

wholeUrl.append("&allParams=[ ");

for (Object object : args) {

wholeUrl.append(object);

}

wholeUrl.append(" ]");

}

return wholeUrl.toString();

}

private TProtocol getTProtocol(TSocket tSocket) {

// 服务端均为非阻塞类型

TTransport transport = new TFramedTransport(tSocket);

TProtocol tProtocol = null;

switch (this.protocol) {

case 1:

tProtocol = new TBinaryProtocol(transport);

break;

case 2:

tProtocol = new TCompactProtocol(transport);

break;

case 3:

tProtocol = new TJSONProtocol(transport);

break;

case 4:

tProtocol = new TSimpleJSONProtocol(transport);

break;

default:

tProtocol = new TBinaryProtocol(transport);

}

return tProtocol;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117

8.代理工厂

package省略

import省略

public class ThriftServiceClientProxyFactory implements FactoryBean<Object>, InitializingBean {

/\*\*

\* 接口的完整路径

\*/

private String service;

/\*\*

\* 连接池

\*/

private ThriftGenericObjectPool thriftGenericObjectPool;

private Object proxyClient;

private Class<Object> objectClass;

/\*\*

\* 传输协议

\* 1.TBinaryProtocol – 二进制格式.

\* 2.TCompactProtocol – 压缩格式

\* 3.TJSONProtocol – JSON格式

\* 4.TSimpleJSONProtocol –提供JSON只写协议, 生成的文件很容易通过脚本语言解析。

\*/

private Integer protocol;

private AddressProvider addressProvider;

@SuppressWarnings("unchecked")

@Override

public void afterPropertiesSet() throws Exception {

// 加载第三方提供的接口和Client.Factory类

ClassLoader classLoader = Thread.currentThread().getContextClassLoader();

this.objectClass = (Class<Object>) classLoader.loadClass(this.service + "$Iface");

Class<TServiceClientFactory<TServiceClient>> tServiceClientFactoryClass = (Class<TServiceClientFactory<TServiceClient>>) classLoader

.loadClass(this.service + "$Client$Factory");

// 设置创建handler

InvocationHandler clientHandler = new ThriftInvocationHandler(this.thriftGenericObjectPool,

tServiceClientFactoryClass.newInstance(), this.protocol, this.service, this.addressProvider);

this.proxyClient = Proxy.newProxyInstance(classLoader, new Class[] { this.objectClass }, clientHandler);

}

@Override

public Object getObject() throws Exception {

return this.proxyClient;

}

@Override

public Class<?> getObjectType() {

return this.objectClass;

}

@Override

public boolean isSingleton() {

return true;

}

public void setProtocol(Integer protocol) {

this.protocol = protocol;

}

public void setService(String service) {

this.service = service;

}

public void setAddressProvider(AddressProvider addressProvider) {

this.addressProvider = addressProvider;

}

public void setThriftGenericObjectPool(ThriftGenericObjectPool thriftGenericObjectPool) {

this.thriftGenericObjectPool = thriftGenericObjectPool;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75

9.调用类

package省略

import省略

@Component

public class SearchTpDao {

private final static Logger log = LoggerFactory.getLogger(SearchTpDao.class);

@Resource

private GenericServing.Iface searchServing;

/\*\*

\* 搜索接口

\*/

public GenericServingResponse search(String from, String dt, Integer page, Integer pageSize, String word,

Integer categoryId, Integer subCategoryId, String ph, String src, Integer mix, String order,

String searchContent, String splatid, String leIds, String eid, String repo\_type, String ispay,

String albumFilter, String videoFilter, String jf, String sf, String stat, String countryArea,

String displayAppId, String displayPlatformId, CommonParam commonParam) {

String logPrefix = "search\_" + from + "\_" + dt + "\_" + page + "\_" + pageSize + "\_" + word + "\_" + categoryId

+ "\_" + ph + "\_" + src + "\_" + mix + "\_" + order + "\_" + searchContent + "\_" + splatid + "\_" + leIds

+ "\_" + eid + "\_" + repo\_type + "\_" + ispay + "\_" + albumFilter + "\_" + videoFilter + "\_" + jf + "\_"

+ sf + "\_" + stat + "\_" + countryArea + "\_" + displayAppId + "\_" + displayPlatformId;

GenericServingResponse response = null;

try {

//getSearchRequest省略

response = this.searchServing.Serve(this.getSearchRequest(from, dt, page, pageSize, word, categoryId,

subCategoryId, ph, src, mix, order, searchContent, splatid, leIds, eid, repo\_type, ispay,

albumFilter, videoFilter, jf, sf, stat, countryArea, null, displayAppId, displayPlatformId,

commonParam));

if (response != null && response.getSearch\_response() != null) {

log.info(logPrefix + " " + response.getSearch\_response().getEid());

}

} catch (Exception e) {

response = null;

log.error(logPrefix, e);

}

return response;

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39

10.配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:util="http://www.springframework.org/schema/util"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/util

http://www.springframework.org/schema/util/spring-util.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd"

xmlns:context="http://www.springframework.org/schema/context">

<context:property-placeholder location="file:///${conf.dir}/thrift.properties"/>

<!-- 搜索服务 -->

<!-- zookeeper集群 -->

<bean id="thriftZookeeper" class="com.letv.itv.v2.plugin.zookeeper.ZookeeperFactory" destroy-method="close">

<property name="zookeeperHosts" value="${search.search.server.zookeeper}"/>

<property name="connectionTimeout" value="3000" />

<property name="sessionTimeout" value="3000" />

</bean>

<!-- 保存搜索服务IP -->

<bean id="searchAddressProvider" class="com.letv.itv.v2.plugin.thrift.AddressProvider">

<constructor-arg index="0" value="${search.search.server}" />

<constructor-arg index="1" ref="thriftZookeeper"></constructor-arg>

<constructor-arg index="2" value="/service/tp/search/video/search\_engine\_rpc" />

</bean>

<!-- 搜索服务连接池 -->

<bean id="searchThriftGenericObjectPool" class="com.letv.itv.v2.plugin.thrift.ThriftGenericObjectPool">

<constructor-arg index="0" ref="searchAddressProvider"></constructor-arg>

<constructor-arg index="1" value="40" /> <!-- maxActive -->

<constructor-arg index="2" value="40" /> <!-- maxIdle -->

<constructor-arg index="3" value="1" /> <!-- minIdle-->

<constructor-arg index="4" value="1000" /> <!-- maxWait-->

</bean>

<!-- protocol参数含义

1.TBinaryProtocol – 二进制格式.

2.TCompactProtocol – 压缩格式

3.TJSONProtocol – JSON格式

4.TSimpleJSONProtocol –提供JSON只写协议, 生成的文件很容易通过脚本语言解析。 -->

<!-- 代理服务，业务层注入即可 -->

<bean id="searchServing" class="com.letv.itv.v2.plugin.thrift.ThriftServiceClientProxyFactory">

<property name="service" value="serving.GenericServing" />

<property name="thriftGenericObjectPool" ref="searchThriftGenericObjectPool" />

<property name="addressProvider" ref="searchAddressProvider" />

<property name="protocol" value="1" />

</bean>

</beans>

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46

11.thrift配置文件，

search.search.server.zookeeper=2.2.2.2:2222

search.search.server=1.1.1.1:1111

* 1
* 2
* 1
* 2

        addressProvider已传递给Handler类，后期在连接执行完加入统计[**算法**](http://lib.csdn.net/base/datastructure)，给出该IP是否需要被remove。希望有thrift负载均衡、服务发现、高可用心得的同学分享下比较厉害的框架之类的。

今天看到有在服务端做负载均衡的Haproxy和nginx相关资料：   
<http://blog.csdn.net/dengzhilong_cpp/article/details/51729918>   
<http://blog.csdn.net/ceasadan/article/details/52369045>   
<http://www.07net01.com/2015/04/819651.html>   
参考：   
<http://blog.csdn.net/zhu_tianwei/article/details/44115667/>