# 超时与重试

服务容错的设计基本原则，Design for Failure，软件设计时考虑各种边界场景和对于服务间调用出现的异常或延迟情况，实现以下目标：

* 一个依赖服务故障不会严重破坏用户体验
* 系统会自动或半自动处理故障，具备自我恢复能力

其中超时与重试是最常用模式，也是Hadoop系统目前使用的容错方式

1. 超时模式，常见的设置网络连接超时时间，RPC的响应时间等，在分布式服务调用的场景中，其主要解决当前依赖服务出现网络连接或者响应延迟时，不用无线等待的问题，调用方可以根据事先设计的超时时间中断调用及时释放关键资源。避免整个系统资源耗尽出现拒绝对外提供服务的情况
2. 重试模式，与超时结合使用，适用于对下游服务的数据强依赖的场景，通过重试来保证数据的可靠性或一致性，常用于因网络抖动而导致服务调用出现超时的场景。与超时设置结合使用后，考虑接口响应时间的分布式请求，超时时间可设置为依赖服务接口99.5%响应时间的值，重试次数一般1~2次为宜，否则会导致请求响应时间延长，延迟整个系统

# Hadoop超时机制

客户端从集群中读写数据时，可以设置读超时和写超时时间，参数如下:

*dfs.datanode.socket.write.timeout*

*dfs.client.socket-timeout*

在DFSClientConf中这两个参数值的获取如下：

*datanodeSocketWriteTimeout = conf.getInt(*

*DFS\_DATANODE\_SOCKET\_WRITE\_TIMEOUT\_KEY,*

*HdfsConstants.WRITE\_TIMEOUT); //默认8 \* 60 \* 10000*

*socketTimeout = conf.getInt(DFS\_CLIENT\_SOCKET\_TIMEOUT\_KEY,*

*HdfsConstants.READ\_TIMEOUT); //默认 60 \* 1000*

客户端端口执行查询的源码如下：

*SocketIOWithTimeout#*

*static void connect(SocketChannel channel,*

*SocketAddress endpoint, int timeout) throws IOException {*

*boolean blockingOn = channel.isBlocking();*

*if (blockingOn) {*

*channel.configureBlocking(false);*

*}*

*try {*

*long timeoutLeft = timeout;*

*long endTime = (timeout > 0) ? (Time.now() + timeout): 0;*

*while (true) {*

*int ret = selector.select((SelectableChannel)channel,*

*SelectionKey.OP\_CONNECT, timeoutLeft);*

*if (ret > 0 && channel.finishConnect()) { //在timeout时间内返回，结果正常*

*return;*

*}*

*if (ret == 0 || //timeout，连接超时，则抛出SocketTimeoutException*

*(timeout > 0 && (timeoutLeft = (endTime - Time.now())) <= 0)) {*

*throw new SocketTimeoutException(*

*timeoutExceptionString(channel, timeout, SelectionKey.OP\_CONNECT));*

*}}}......}*

最终调用是JDK NIO框架中的SelectorImpl.select

*private int lockAndDoSelect(long var1) throws IOException {*

*synchronized(this) {*

*//坚持selector是否已经打开了*

*if(!this.isOpen()) {*

*throw new ClosedSelectorException();*

*} else {*

*Set var4 = this.publicKeys;*

*int var10000;*

*//这里用了双重锁来实现同步访问，双重锁可能引起死锁。*

*synchronized(this.publicKeys) {*

*Set var5 = this.publicSelectedKeys;*

*synchronized(this.publicSelectedKeys) {*

*var10000 = this.doSelect(var1);*

*}*

*}*

*return var10000;*

*}*

*}*

*}*

访问超时后，抛出SocketTimeoutException，捕获该异常进行处理。

https://blog.csdn.net/threewaterxzm/article/details/73087928

# Hadoop系统重试机制

HDFS客户端支持Automatic failver，即NameNode开启高可用时，访问某个NN失败后自动切换到另外的NN上，其配置如下：

*<property>*

*<name>dfs.ha.automatic-failover.enabled</name>*

*<value>true</value>*

*</property>*

*<property>*

*<name>dfs.client.failover.proxy.provider.nn</name> <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>*

*</property>*

测试如下，当nn1为Standby后，则客户端连接到该节点后抛出异常，则自动跳转到nn2，执行如下：

*$hdfs dfs -ls /*

*ipc.Client: Connecting to cmhhost1.novalocal/10.139.4.82:8020*

*ipc.Client: IPC Client (1095088856) connection to cmhhost1.novalocal/10.139.4.82:8020 from yarn: starting, having connections 1*

*ipc.Client: IPC Client (1095088856) connection to cmhhost1.novalocal/10.139.4.82:8020 from yarn sending #0 org.apache.hadoop.hdfs.protocol.ClientProtocol.getFileInfo*

*ipc.Client: IPC Client (1095088856) connection to cmhhost1.novalocal/10.139.4.82:8020 from yarn got value #0*

*retry.RetryInvocationHandler: org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.ipc.StandbyException): Operation category READ is not supported in state standby.*

*DEBUG ipc.Client: Connecting to cmhhost2.novalocal/10.139.4.83:8020*

*DEBUG ipc.ProtobufRpcEngine: Call: getListing took 3ms*

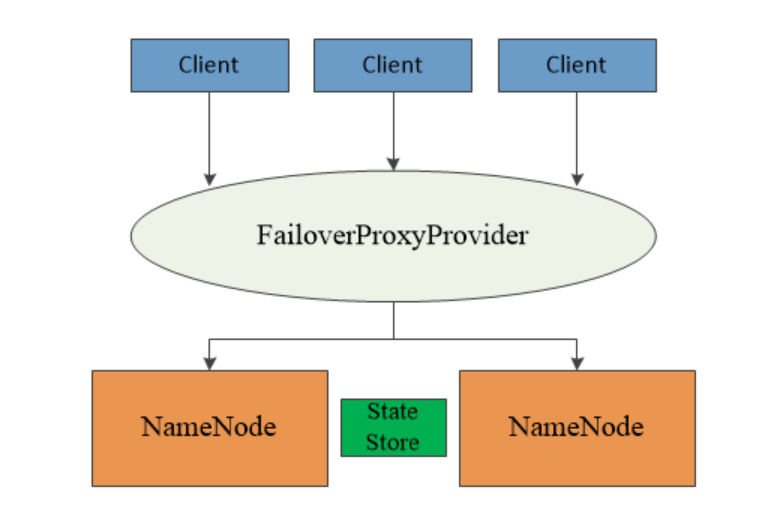
*Found 7 items*

*drwxrwxrwt - yarn hadoop 0 2018-11-16 22:24 /app-logs*

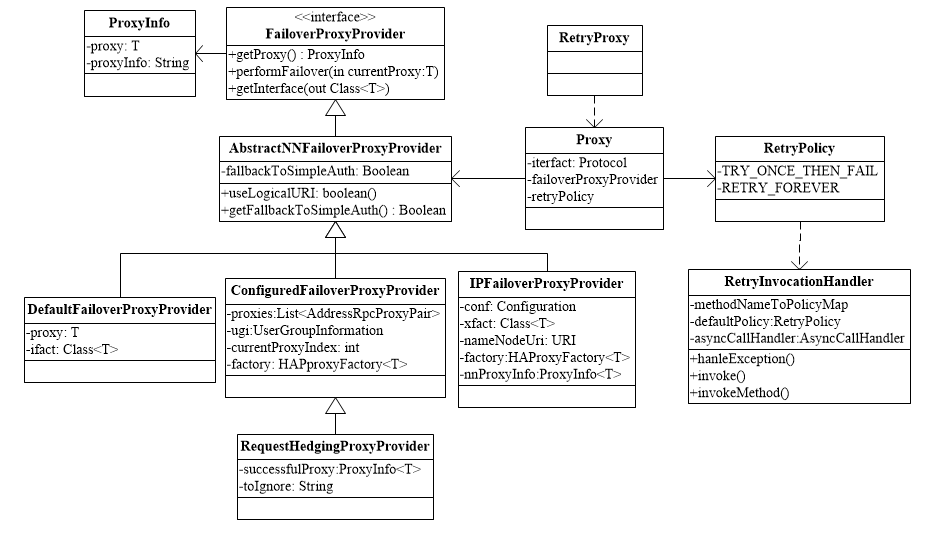
*drwxr-xr-x - yarn hadoop 0 2018-11-13 09:30 /ats*

*drwxr-xr-x - hdfs hdfs 0 2018-11-13 09:30 /bch*

客户端访问NN Failover执行流程如下图所示：



DFSClient访问NameNode时，其使用FailoverProxyProvider作为代理，其内维护了可用的NameNode列表，当出现异常时自动切换到其他NN。Proxy相关类图如下：



ClientProtocol对应的Proxy为RetryPolicy，初始化如下：

*proxyInfo = NameNodeProxiesClient.createProxyWithClientProtocol(conf,*

*nameNodeUri, nnFallbackToSimpleAuth);*

*this.dtService = proxyInfo.getDelegationTokenService();*

*this.namenode = proxyInfo.getProxy();*

ProxyAndInfo<ClientProtocol>初始化如下：

*public static <T> ProxyAndInfo<T> createHAProxy(*

*Configuration conf, URI nameNodeUri, Class<T> xface,*

*AbstractNNFailoverProxyProvider<T> failoverProxyProvider) {*

*DfsClientConf config = new DfsClientConf(conf);*

*T proxy = (T) RetryProxy.create(xface, failoverProxyProvider,*

*RetryPolicies.failoverOnNetworkException(*

*RetryPolicies.TRY\_ONCE\_THEN\_FAIL, config.getMaxFailoverAttempts(),*

*config.getMaxRetryAttempts(), config.getFailoverSleepBaseMillis(),*

*config.getFailoverSleepMaxMillis()));*

*......*

*return new ProxyAndInfo<>(proxy, dtService,*

*DFSUtilClient.getNNAddressCheckLogical(conf, nameNodeUri));*

*}*

ClientProtocol#getFileInfo，最终调用的是Proxy，使用的RetryInvocationHandler处理异常，如下所示：

*synchronized CallReturn invokeOnce() {*

*try {*

*if (retryInfo != null) {*

*return processWaitTimeAndRetryInfo();*

*}*

*final long failoverCount = retryInvocationHandler.getFailoverCount();*

*try {*

*return invoke();*

*} catch (Exception e) {*

*....*

*retryInfo = retryInvocationHandler.handleException(*

*method, callId, retryPolicy, counters, failoverCount, e);*

*return processWaitTimeAndRetryInfo(); //异常时重试调用*

*}*

*} catch(Throwable t) {*

*return new CallReturn(t);*

*}*

*}*

*synchronized void failover(long expectedFailoverCount, Method method,*

*int callId) {*

*// Make sure that concurrent failed invocations only cause a single*

*// actual failover.*

*if (failoverCount == expectedFailoverCount) {*

*fpp.performFailover(proxyInfo.proxy);*

*//FailoverProxyProvider执行Failover，使用可用的proxy*

*failoverCount++;*

*} else {*

*LOG.warn("A failover has occurred since the start of call #" + callId*

*+ " " + proxyInfo.getString(method.getName()));*

*}*

*proxyInfo = fpp.getProxy(); //获取当前可用的proxy，RetryInfo <=*

*}*

该例子中使用ConfiguredFailoverProxyProvider#performFailover

*synchronized void incrementProxyIndex() {*

*currentProxyIndex = (currentProxyIndex + 1) % proxies.size();*

*}*

使用下一个proxy。

备注：在客户端调用逻辑中CallHandler会启动线程，执行Call，如下

*void tryStart() {*

*final Thread current = Thread.currentThread();*

*if (running.compareAndSet(null, current)) {*

*final Daemon daemon = new Daemon() {*

*@Override*

*public void run() {*

*for (; isRunning(this);) {*

*final long waitTime = checkCalls(); //执行invokeOnce操作*

*tryStop(this);*

*try {*

*synchronized (AsyncCallHandler.this) {*

*AsyncCallHandler.this.wait(waitTime);*

*}*

*} ......*

*}*

保证最终获取到结果。

http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html#Automatic\_Failover

https://www.w3cschool.cn/hadoop/qslx1p33.html

https://data-flair.training/blogs/hadoop-namenode-automatic-failover/https://data-flair.training/blogs/hadoop-namenode-automatic-failover/

Failover