

33 | Cluster组件：Tomcat的集群通信原理

2019-07-25 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 08:29 大小 7.78M



为了支持水平扩展和高可用，Tomcat 提供了集群部署的能力，但与此同时也带来了分布式系统的一个通用问题，那就是如何在集群中的多个节点之间保持数据的一致性，比如会话（Session）信息。

要实现这一点，基本上有两种方式，一种是把所有 Session 数据放到一台服务器或者一个数据库中，集群中的所有节点通过访问这台 Session 服务器来获取数据。另一种方式就是在集群中的节点间进行 Session 数据的同步拷贝，这里又分为两种策略：第一种是将一个节点的 Session 拷贝到集群中其他所有节点；第二种是只将一个节点上的 Session 数据拷贝到另一个备份节点。

对于 Tomcat 的 Session 管理来说，这两种方式都支持。今天我们就来看看第二种方式的实现原理，也就是 Tomcat 集群通信的原理和配置方法，最后通过官网上的一个例子来了

解下 Tomcat 集群到底是如何工作的。

集群通信原理

要实现集群通信，首先要知道集群中都有哪些成员。Tomcat 是通过**组播**（Multicast）来实现的。那什么是组播呢？为了理解组播，我先来说说什么是“单播”。网络节点之间的通信就好像是人们之间的对话一样，一个人对另外一个人说话，此时信息的接收和传递只在两个节点之间进行，比如你在收发电子邮件、浏览网页时，使用的就是单播，也就是我们熟悉的“点对点通信”。

如果一台主机需要将同一个消息发送多个主机逐个传输，效率就会比较低，于是就出现组播技术。组播是**一台主机向指定的一组主机发送数据报包**，组播通信的过程是这样的：每一个 Tomcat 节点在启动时和运行时都会周期性（默认 500 毫秒）发送组播心跳包，同一个集群内的节点都在相同的**组播地址**和**端口**监听这些信息；在一定的时间内（默认 3 秒）不发送**组播报文**的节点就会被认为已经崩溃了，会从集群中删去。因此通过组播，集群中每个成员都能维护一个集群成员列表。

集群通信配置


有了集群成员的列表，集群中的节点就能通过 TCP 连接向其他节点传输 Session 数据。Tomcat 通过 SimpleTcpCluster 类来进行会话复制（In-Memory Replication）。要开启集群功能，只需要将 server.xml 里的这一行的注释去掉就行：

```
<!--  
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>  
-->
```

变成这样：

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

虽然只是简单的一行配置，但这一行配置等同于下面这样的配置，也就是说 Tomcat 给我们设置了很多默认参数，这些参数都跟集群通信有关。

 复制代码

1 <!--

2 SimpleTcpCluster 是用来复制 Session 的组件。复制 Session 有同步和异步两种方式：

```

3  同步模式下，向浏览器的发送响应数据前，需要先将 Session 拷贝到其他节点完；
4  异步模式下，无需等待 Session 拷贝完成就可响应。异步模式更高效，但是同步模式
5  可靠性更高。
6  同步异步模式由 channelSendOptions 参数控制，默认值是 8，为异步模式；4 是同步模式。
7  在异步模式下，可以通过加上 " 拷贝确认 "（Acknowledge）来提高可靠性，此时
8  channelSendOptions 设为 10
9  -->
10 <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
11     channelSendOptions="8">
12     <!--
13     Manager 决定如何管理集群的 Session 信息。
14     Tomcat 提供了两种 Manager: BackupManager 和 DeltaManager。
15     BackupManager—集群下的某一节点的 Session，将复制到一个备份节点。
16     DeltaManager— 集群下某一节点的 Session，将复制到所有其他节点。
17     DeltaManager 是 Tomcat 默认的集群 Manager。
18
19     expireSessionsOnShutdown—设置为 true 时，一个节点关闭时，
20     将导致集群下的所有 Session 失效
21     notifyListenersOnReplication—集群下节点间的 Session 复制、
22     删除操作，是否通知 session listeners
23
24     maxInactiveInterval—集群下 Session 的有效时间（单位:s）。
25     maxInactiveInterval 内未活动的 Session，将被 Tomcat 回收。
26     默认值为 1800(30min)
27     -->
28 <Manager className="org.apache.catalina.ha.session.DeltaManager"
29     expireSessionsOnShutdown="false"
30     notifyListenersOnReplication="true"/>
31
32     <!--
33     Channel 是 Tomcat 节点之间进行通讯的工具。
34     Channel 包括 5 个组件: Membership、Receiver、Sender、
35     Transport、Interceptor
36     -->
37 <Channel className="org.apache.catalina.tribes.group.GroupChannel">
38     <!--
39     Membership 维护集群的可用节点列表。它可以检查到新增的节点，
40     也可以检查没有心跳的节点
41     className—指定 Membership 使用的类
42     address—组播地址
43     port—组播端口
44     frequency—发送心跳（向组播地址发送 UDP 数据包）的时间间隔（单位:ms）。
45     dropTime—Membership 在 dropTime(单位:ms) 内未收到某一节点的心跳，
46     则将该节点从可用节点列表删除。默认值为 3000。
47     -->
48 <Membership className="org.apache.catalina.tribes.membership.
49     McastService"
50     address="228.0.0.4"
51     port="45564"
52     frequency="500"
53     dropTime="3000"/>
54

```

```

55 <!--
56 Receiver 用于各个节点接收其他节点发送的数据。
57 接收器分为两种：BioReceiver(阻塞式)、NioReceiver(非阻塞式)
58
59 className—指定 Receiver 使用的类
60 address—接收消息的地址
61 port—接收消息的端口
62 autoBind—端口的变化区间，如果 port 为 4000，autoBind 为 100，
63 接收器将在 4000-4099 间取一个端口进行监听。
64 selectorTimeout—NioReceiver 内 Selector 轮询的超时时间
65 maxThreads—线程池的最大线程数
66 -->
67 <Receiver className="org.apache.catalina.tribes.transport.nio.
68 NioReceiver"
69 address="auto"
70 port="4000"
71 autoBind="100"
72 selectorTimeout="5000"
73 maxThreads="6"/>
74
75 <!--
76 Sender 用于向其他节点发送数据，Sender 内嵌了 Transport 组件，
77 Transport 真正负责发送消息。
78 -->
79 <Sender className="org.apache.catalina.tribes.transport.
80 ReplicationTransmitter">
81 <!--
82 Transport 分为两种：bio.PooledMultiSender(阻塞式)
83 和 nio.PooledParallelSender(非阻塞式)，PooledParallelSender
84 是从 tcp 连接池中获取连接，可以实现并行发送，即集群中的节点可以
85 同时向其他所有节点发送数据而互不影响。
86 -->
87 <Transport className="org.apache.catalina.tribes.
88 transport.nio.PooledParallelSender"/>
89 </Sender>
90
91 <!--
92 Interceptor : Cluster 的拦截器
93 TcpFailureDetector—TcpFailureDetector 可以拦截到某个节点关闭
94 的信息，并尝试通过 TCP 连接到此节点，以确保此节点真正关闭，从而更新集
95 群可用节点列表
96 -->
97 <Interceptor className="org.apache.catalina.tribes.group.
98 interceptors.TcpFailureDetector"/>
99
100 <!--
101 MessageDispatchInterceptor—查看 Cluster 组件发送消息的
102 方式是否设置为 Channel.SEND_OPTIONS_ASYNCHRONOUS，如果是，
103 MessageDispatchInterceptor 先将等待发送的消息进行排队，
104 然后将排好队的消息转给 Sender。
105 -->
106 <Interceptor className="org.apache.catalina.tribes.group.

```

```

107         interceptors.MessageDispatchInterceptor"/>
108     </Channel>
109
110     <!--
111         Valve : Tomcat 的拦截器，
112         ReplicationValve 在处理请求前后打日志；过滤不涉及 Session 变化的请求。
113     -->
114     <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
115         filter=""/>
116     <Valve className="org.apache.catalina.ha.session.
117         JvmRouteBinderValve"/>
118
119     <!--
120         Deployer 用于集群的 farm 功能，监控应用中文件的更新，以保证集群中所有节点
121         应用的一致性，如某个用户上传文件到集群中某个节点的应用程序目录下，Deployer
122         会监测到这一操作并把文件拷贝到集群中其他节点相同应用的对应目录下以保持
123         所有应用的一致，这是一个相当强大的功能。
124     -->
125     <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
126         tempDir="/tmp/war-temp/"
127         deployDir="/tmp/war-deploy/"
128         watchDir="/tmp/war-listen/"
129         watchEnabled="false"/>
130
131     <!--
132         ClusterListener : 监听器，监听 Cluster 组件接收的消息
133         使用 DeltaManager 时，Cluster 接收的信息通过 ClusterSessionListener
134         传递给 DeltaManager，从而更新自己的 Session 列表。
135     -->
136     <ClusterListener className="org.apache.catalina.ha.session.
137         ClusterSessionListener"/>
138
139 </Cluster>


```



从上面的参数列表可以看到，默认情况下 Session 管理组件 DeltaManager 会在节点之间拷贝 Session，DeltaManager 采用的一种 all-to-all 的工作方式，即集群中的节点会把 Session 数据向所有其他节点拷贝，而不管其他节点是否部署了当前应用。当集群节点数比较少时，比如少于 4 个，这种 all-to-all 的方式是不错的选择；但是当集群中的节点数量比较多时，数据拷贝的开销成指数级增长，这种情况下可以考虑 BackupManager，BackupManager 只向一个备份节点拷贝数据。

在大体了解了 Tomcat 集群实现模型后，就可以对集群作出更优化的配置了。Tomcat 推荐了一套配置，使用了比 DeltaManager 更高效的 BackupManager，并且通过 ReplicationValve 设置了请求过滤。

这里还请注意在一台服务器部署多个节点时需要修改 Receiver 的侦听端口，另外为了在节点间高效地拷贝数据，所有 Tomcat 节点最好采用相同的配置，具体配置如下：

 复制代码

```
1 <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
2         channelSendOptions="6">
3
4     <Manager className="org.apache.catalina.ha.session.BackupManager"
5             expireSessionsOnShutdown="false"
6             notifyListenersOnReplication="true"
7             mapSendOptions="6"/>
8
9     <Channel className="org.apache.catalina.tribes.group.
10    GroupChannel">
11
12     <Membership className="org.apache.catalina.tribes.membership.
13    McastService"
14         address="228.0.0.4"
15         port="45564"
16         frequency="500"
17         dropTime="3000"/>
18
19     <Receiver className="org.apache.catalina.tribes.transport.nio.
20    NioReceiver"
21         address="auto"
22         port="5000"
23         selectorTimeout="100"
24         maxThreads="6"/>
25
26     <Sender className="org.apache.catalina.tribes.transport.
27    ReplicationTransmitter">
28         <Transport className="org.apache.catalina.tribes.transport.
29        nio.PooledParallelSender"/>
30    </Sender>
31
32     <Interceptor className="org.apache.catalina.tribes.group.
33    interceptors.TcpFailureDetector"/>
34
35     <Interceptor className="org.apache.catalina.tribes.group.
36    interceptors.MessageDispatchInterceptor"/>
37
38     <Interceptor className="org.apache.catalina.tribes.group.
39    interceptors.ThroughputInterceptor"/>
40 </Channel>
41
42 <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
43     filter=".*\.(gif|.*\.(js|.*\.(jpeg|.*\.(jpg|.*\.(png|.*\
44     .htm|.*\.(html|.*\.(css|.*\.(txt"/>
45
46 <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
```



```
47     tempDir="/tmp/war-temp/"
48     deployDir="/tmp/war-deploy/"
49     watchDir="/tmp/war-listen/"
50     watchEnabled="false"/>
51
52     <ClusterListener className="org.apache.catalina.ha.session.
53     ClusterSessionListener"/>
54 </Cluster>
```

集群工作过程

Tomcat 的官网给出了一个例子，来说明 Tomcat 集群模式下是如何工作的，以及 Tomcat 集群是如何实现高可用的。比如集群由 Tomcat A 和 Tomcat B 两个 Tomcat 实例组成，按照时间先后顺序发生了如下事件：

1. Tomcat A 启动

Tomcat A 启动过程中，当 Host 对象被创建时，一个 Cluster 组件（默认是 SimpleTcpCluster）被关联到这个 Host 对象。当某个应用在 `web.xml` 中设置了 Distributable 时，Tomcat 将为此应用的上下文环境创建一个 DeltaManager。SimpleTcpCluster 启动 Membership 服务和 Replication 服务。

2. Tomcat B 启动（在 Tomcat A 之后启动）

首先 Tomcat B 会执行和 Tomcat A 一样的操作，然后 SimpleTcpCluster 会建立一个由 Tomcat A 和 Tomcat B 组成的 Membership。接着 Tomcat B 向集群中的 Tomcat A 请求 Session 数据，如果 Tomcat A 没有响应 Tomcat B 的拷贝请求，Tomcat B 会在 60 秒后 time out。在 Session 数据拷贝完成之前 Tomcat B 不会接收浏览器的请求。

3. Tomcat A 接收 HTTP 请求，创建 Session 1

Tomcat A 响应客户请求，在把结果发送回客户端之前，ReplicationValve 会拦截当前请求（如果 Filter 中配置了不需拦截的请求类型，这一步就不会进行，默认配置下拦截所有请求），如果发现当前请求更新了 Session，就调用 Replication 服务建立 TCP 连接将 Session 拷贝到 Membership 列表中的其他节点即 Tomcat B。在拷贝时，所有保存在当前 Session 中的可序列化的对象都会被拷贝，而不仅仅是发生更新的部分。

4. Tomcat A 崩溃

当 Tomcat A 崩溃时，Tomcat B 会被告知 Tomcat A 已从集群中退出，然后 Tomcat B 就会把 Tomcat A 从自己的 Membership 列表中删除。并且 Tomcat B 的 Session 更新时不再往 Tomcat A 拷贝，同时负载均衡器会把后续的 HTTP 请求全部转发给 Tomcat B。在此过程中所有的 Session 数据不会丢失。

5. Tomcat B 接收 Tomcat A 的请求

Tomcat B 正常响应本应该发往 Tomcat A 的请求，因为 Tomcat B 保存了 Tomcat A 的所有 Session 数据。

6. Tomcat A 重新启动

Tomcat A 按步骤 1、2 操作启动，加入集群，并从 Tomcat B 拷贝所有 Session 数据，拷贝完成后开始接收请求。

7. Tomcat A 接收请求，Session 1 被用户注销

Tomcat 继续接收发往 Tomcat A 的请求，Session 1 设置为失效。请注意这里的失效并非因为 Tomcat A 处于非活动状态超过设置的时间，而是应用程序执行了注销的操作（比如用户登出）而引起的 Session 失效。这时 Tomcat A 向 Tomcat B 发送一个 Session 1 Expired 的消息，Tomcat B 收到消息后也会把 Session 1 设置为失效。

8. Tomcat B 接收到一个新请求，创建 Session 2

同理这个新的 Session 也会被拷贝到 Tomcat A。

9. Tomcat A 上的 Session 2 过期

因超时原因引起的 Session 失效 Tomcat A 无需通知 Tomcat B，Tomcat B 同样知道 Session 2 已经超时。因此对于 Tomcat 集群有一点非常重要，**所有节点的操作系统时间必须一致**。不然会出现某个节点 Session 已过期而在另一节点此 Session 仍处于活动状态的现象。

本期精华


今天我谈了 Tomcat 的集群工作原理和配置方式，还通过官网上的一个例子说明了 Tomcat 集群的工作过程。Tomcat 集群对 Session 的拷贝支持两种方式：DeltaManager 和 BackupManager。

当集群中节点比较少时，可以采用 DeltaManager，因为 Session 数据在集群中各个节点都有备份，任何一个节点崩溃都不会对整体造成影响，可靠性比较高。

当集群中节点数比较多时，可以采用 BackupManager，这是因为一个节点的 Session 只会拷贝到另一个节点，数据拷贝的开销比较少，同时只要这两个节点不同时崩溃，Session 数据就不会丢失。

课后思考

在 Tomcat 官方推荐的配置里，ReplicationValve 被配置成下面这样：

 复制代码

```
1 <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"  
2     filter=".*\.(gif|.*\.(js|.*\.(jpeg|.*\.(jpg|.*\.(png|.*\  
3         .htm|.*\.(html|.*\.(css|.*\.(txt"/>
```

你是否注意到，filter 的值是一些 JS 文件或者图片等，这是为什么呢？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。

深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双

eBay 技术主管



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 32 | Manager组件：Tomcat的Session管理机制解析

下一篇 特别放送 | 如何持续保持对学习的兴趣？

精选留言 (7)

写留言



QQ怪

2019-07-25

感觉这种方式应该在生产环境用的很少吧，大多数都是用redis集群来保存session

作者回复: 对的，Redis有额外安装维护开销，小的集群可以用Tomcat原生方案



1

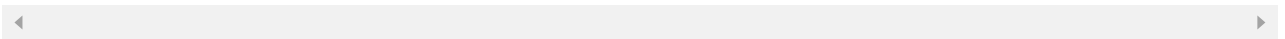


chon

2019-07-25

生产中，如果机器多的话，很少用session复制吧

作者回复: 对的



1



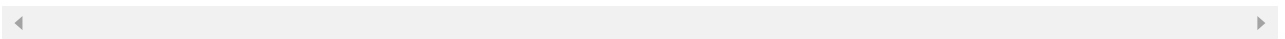
-W.LI-

2019-07-26

好老师哈。那些操作一般不会涉及session变化。BackupManager实现高可用，和好多中间件的原理差不多。以前都是接入层一致性hash，没有启用session集群。这个session集群同步开销高么？一次只同步一个session还是批量打包的？Tomcat支持把session放在redis么？我项目是token+redis。

展开 ▾

作者回复: Tomcat支持把Session存Redis的



Liam

2019-07-26

避免复制的时候提及文件这类大数据吧？



门窗小二

2019-07-25

思考题应该是静态资源不会更新session值吧！请问老师我也有跟neohope一样的一问，集群分裂的情况tomcat的处理方式



neohope

2019-07-25

老师您好，有两个问题想咨询一下：

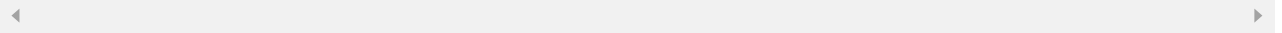
1、采用DeltaManager模式后，如果主节点挂掉，存在新的主节点选举的这个过程吗？如果有的话，Tomcat是如何防止产生集群分裂（脑裂）的呢？

...

展开 ∨

作者回复: 1, Tomcat集群其实没有Zookeeper那样的选主机制, 一台挂了就将请求发到用备份节点

2, 小的集群可以用Tomcat原生方案, 大集群还是用Redis



a、

2019-07-25

今天的问题:我觉得因为一般静态资源不会涉及到session更新, 所以就不需要拦截。还有个问题我想问下老师, 如果我有四台机器A,B,C,D, 设置了BackupManager, 那A的备份机器会不会是B, B的备份机器是C, C的备份机器是D, D的备份机器是A? 还是说如果A的备份机器是B, 那C只能选择D做备份机器?

展开 ∨

作者回复: A的备份机器是B, B的备份机器也可以是A。A和B只要不全挂掉就行。

