

32 | Manager组件：Tomcat的Session管理机制解析

2019-07-23 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 05:57 大小 5.46M



我们可以通过 Request 对象的 getSession 方法来获取 Session，并通过 Session 对象来读取和写入属性值。而 Session 的管理是由 Web 容器来完成的，主要是对 Session 的创建和销毁，除此之外 Web 容器还需要将 Session 状态的变化通知给监听者。


当然 Session 管理还可以交给 Spring 来做，好处是与特定的 Web 容器解耦，Spring Session 的核心原理是通过 Filter 拦截 Servlet 请求，将标准的 ServletRequest 包装一下，换成 Spring 的 Request 对象，这样当我们调用 Request 对象的 getSession 方法时，Spring 在背后为我们创建和管理 Session。

那么 Tomcat 的 Session 管理机制我们还需要了解吗？我觉得还是有必要，因为只有了解这些原理，我们才能更好的理解 Spring Session，以及 Spring Session 为什么设计成这

样。今天我们就从 Session 的创建、Session 的清理以及 Session 的事件通知这几个方面来了解 Tomcat 的 Session 管理机制。

Session 的创建

Tomcat 中主要由每个 Context 容器内的一个 Manager 对象来管理 Session。默认实现类为 StandardManager。下面我们通过它的接口来了解一下 StandardManager 的功能：

 复制代码


```
1 public interface Manager {
2     public Context getContext();
3     public void setContext(Context context);
4     public SessionIdGenerator getSessionIdGenerator();
5     public void setSessionIdGenerator(SessionIdGenerator sessionIdGenerator);
6     public long getSessionCounter();
7     public void setSessionCounter(long sessionCounter);
8     public int getMaxActive();
9     public void setMaxActive(int maxActive);
10    public int getActiveSessions();
11    public long getExpiredSessions();
12    public void setExpiredSessions(long expiredSessions);
13    public int getRejectedSessions();
14    public int getSessionMaxAliveTime();
15    public void setSessionMaxAliveTime(int sessionMaxAliveTime);
16    public int getSessionAverageAliveTime();
17    public int getSessionCreateRate();
18    public int getSessionExpireRate();
19    public void add(Session session);
20    public void changeSessionId(Session session);
21    public void changeSessionId(Session session, String newId);
22    public Session createEmptySession();
23    public Session createSession(String sessionId);
24    public Session findSession(String id) throws IOException;
25    public Session[] findSessions();
26    public void load() throws ClassNotFoundException, IOException;
27    public void remove(Session session);
28    public void remove(Session session, boolean update);
29    public void addPropertyChangeListener(PropertyChangeListener listener);
30    public void removePropertyChangeListener(PropertyChangeListener listener);
31    public void unload() throws IOException;
32    public void backgroundProcess();
33    public boolean willAttributeDistribute(String name, Object value);
34 }
```

不出意外我们在接口中看到了添加和删除 Session 的方法；另外还有 load 和 unload 方法，它们的作用分别是将 Session 持久化到存储介质和从存储介质加载 Session。


当我们调用 `HttpServletRequest.getSession(true)` 时，这个参数 `true` 的意思是“如果当前请求还没有 Session，就创建一个新的”。那 Tomcat 在背后为我们做了些什么呢？

`HttpServletRequest` 是一个接口，Tomcat 实现了这个接口，具体实现类是：`org.apache.catalina.connector.Request`。

但这并不是我们拿到的 `Request`，Tomcat 为了避免把一些实现细节暴露出来，还有基于安全上的考虑，定义了 `Request` 的包装类，叫作 `RequestFacade`，我们可以通过代码来理解一下：


 复制代码

```
1 public class Request implements HttpServletRequest {}
```

 复制代码

```
1 public class RequestFacade implements HttpServletRequest {
2     protected Request request = null;
3
4     public HttpSession getSession(boolean create) {
5         return request.getSession(create);
6     }
7 }
```

因此我们拿到的 `Request` 类其实是 `RequestFacade`，`RequestFacade` 的 `getSession` 方法调用的是 `Request` 类的 `getSession` 方法，我们继续来看 Session 具体是如何创建的：

 复制代码

```
1 Context context = getContext();
2 if (context == null) {
3     return null;
4 }
5
6 Manager manager = context.getManager();
```

```


7 if (manager == null) {
8     return null;
9 }
10
11 session = manager.createSession(sessionId);
12 session.access();

```

从上面的代码可以看出，Request 对象中持有 Context 容器对象，而 Context 容器持有 Session 管理器 Manager，这样通过 Context 组件就能拿到 Manager 组件，最后由 Manager 组件来创建 Session。

因此最后还是到了 StandardManager，StandardManager 的父类叫 ManagerBase，这个 createSession 方法定义在 ManagerBase 中，StandardManager 直接重用这个方法。

接着我们来看 ManagerBase 的 createSession 是如何实现的：

 复制代码

```

1 @Override
2 public Session createSession(String sessionId) {
3     // 首先判断 Session 数量是不是到了最大值，最大 Session 数可以通过参数设置
4     if ((maxActiveSessions >= 0) &&
5         (getActiveSessions() >= maxActiveSessions)) {
6         rejectedSessions++;
7         throw new TooManyActiveSessionsException(
8             sm.getString("managerBase.createSession.ise"),
9             maxActiveSessions);
10    }
11
12    // 重用或者创建一个新的 Session 对象，请注意在 Tomcat 中就是 StandardSession
13    // 它是 HttpSession 的具体实现类，而 HttpSession 是 Servlet 规范中定义的接口
14    Session session = createEmptySession();
15
16
17    // 初始化新 Session 的值
18    session.setNew(true);
19    session.setValid(true);
20    session.setCreationTime(System.currentTimeMillis());
21    session.setMaxInactiveInterval(getContext().getSessionTimeout() * 60);
22    String id = sessionId;
23    if (id == null) {
24        id = generateSessionId();
25    }
26    session.setId(id); // 这里会将 Session 添加到 ConcurrentHashMap 中


```

```

27     sessionCounter++;
28
29     // 将创建时间添加到 LinkedList 中，并且把最先添加的时间移除
30     // 主要还是方便清理过期 Session
31     SessionTiming timing = new SessionTiming(session.getCreationTime(), 0);
32     synchronized (sessionCreationTiming) {
33         sessionCreationTiming.add(timing);
34         sessionCreationTiming.poll();
35     }
36     return session
37 }

```

到此我们明白了 Session 是如何创建出来的，创建出来后 Session 会被保存到一个 ConcurrentHashMap 中：

 复制代码

```

1 protected Map<String, Session> sessions = new ConcurrentHashMap<>();

```

请注意 Session 的具体实现类是 StandardSession，StandardSession 同时实现了 `java.x.servlet.http.HttpSession` 和 `org.apache.catalina.Session` 接口，并且对程序员暴露的是 StandardSessionFacade 外观类，保证了 StandardSession 的安全，避免了程序员调用其内部方法进行不当操作。StandardSession 的核心成员变量如下：

 复制代码

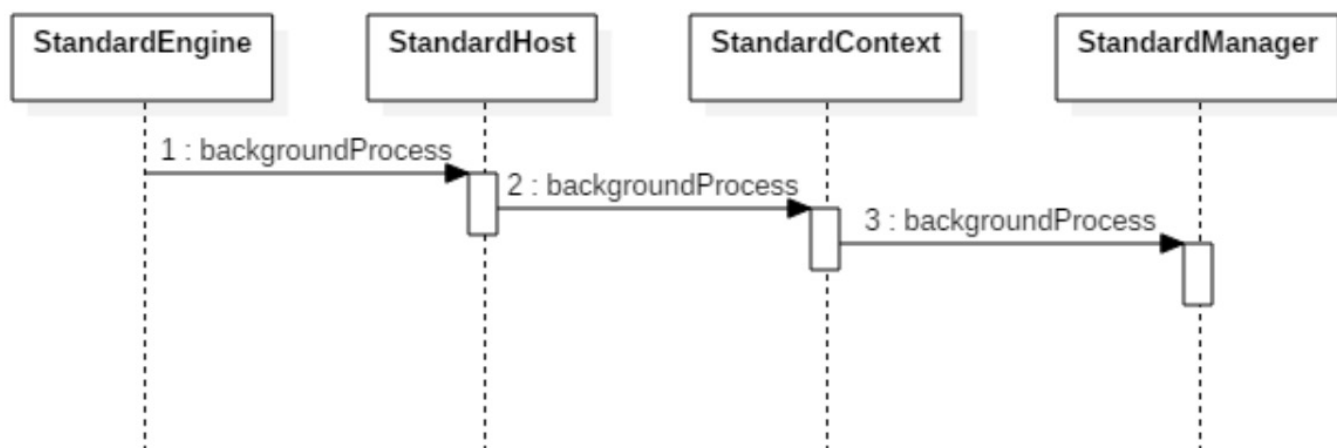
```

1 public class StandardSession implements HttpSession, Session, Serializable {
2     protected ConcurrentMap<String, Object> attributes = new ConcurrentHashMap<>();
3     protected long creationTime = 0L;
4     protected transient volatile boolean expiring = false;
5     protected transient StandardSessionFacade facade = null;
6     protected String id = null;
7     protected volatile long lastAccessedTime = creationTime;
8     protected transient ArrayList<SessionListener> listeners = new ArrayList<>();
9     protected transient Manager manager = null;
10    protected volatile int maxInactiveInterval = -1;
11    protected volatile boolean isNew = false;
12    protected volatile boolean isValid = false;
13    protected transient Map<String, Object> notes = new Hashtable<>();
14    protected transient Principal principal = null;
15 }

```

Session 的清理

我们再来看看 Tomcat 是如何清理过期的 Session。在 Tomcat[热加载和热部署](#)的文章里，我讲到容器组件会开启一个 ContainerBackgroundProcessor 后台线程，调用自己以及子容器的 backgroundProcess 进行一些后台逻辑的处理，和 Lifecycle 一样，这个动作也是具有传递性的，也就是说子容器还会把这个动作传递给自己的子容器。你可以参考下图来理解这个过程。



其中父容器会遍历所有的子容器并调用其 backgroundProcess 方法，而 StandardContext 重写了该方法，它会调用 StandardManager 的 backgroundProcess 进而完成 Session 的清理工作，下面是 StandardManager 的 backgroundProcess 方法的代码：

复制代码

```
1 public void backgroundProcess() {
2     // processExpiresFrequency 默认值为 6，而 backgroundProcess 默认每隔 10s 调用一次，也就
3     count = (count + 1) % processExpiresFrequency;
4     if (count == 0) // 默认每隔 60s 执行一次 Session 清理
5         processExpires();
6 }
7
8 /**
9  * 单线程处理，不存在线程安全问题
10 */
11 public void processExpires() {
12
13     // 获取所有的 Session
14     Session sessions[] = findSessions();
15     int expireHere = 0 ;
16     for (int i = 0; i < sessions.length; i++) {
17         // Session 的过期是在 isValid() 方法里处理的
```

```
18         if (sessions[i]!=null && !sessions[i].isValid()) {
19             expireHere++;
20         }
21     }
22 }
```

backgroundProcess 由 Tomcat 后台线程调用，默认是每隔 10 秒调用一次，但是 Session 的清理动作不能太频繁，因为需要遍历 Session 列表，会耗费 CPU 资源，所以在 backgroundProcess 方法中做了取模处理，backgroundProcess 调用 6 次，才执行一次 Session 清理，也就是说 Session 清理每 60 秒执行一次。

Session 事件通知

按照 Servlet 规范，在 Session 的生命周期过程中，要将事件通知监听者，Servlet 规范定义了 Session 的监听器接口：

 复制代码

```
1 public interface HttpSessionListener extends EventListener {
2     //Session 创建时调用
3     public default void sessionCreated(HttpSessionEvent se) {
4     }
5
6     //Session 销毁时调用
7     public default void sessionDestroyed(HttpSessionEvent se) {
8     }
9 }
```

注意到这两个方法的参数都是 HttpSessionEvent，所以 Tomcat 需要先创建 HttpSessionEvent 对象，然后遍历 Context 内部的 LifecycleListener，并且判断是否为 HttpSessionListener 实例，如果是的话则调用 HttpSessionListener 的 sessionCreated 方法进行事件通知。这些事情都是在 Session 的 setId 方法中完成的：

 复制代码

```
1 session.setId(id);
2
3 @Override
4 public void setId(String id, boolean notify) {
5     // 如果这个 id 已经存在，先从 Manager 中删除
```



```

6      if ((this.id != null) && (manager != null))
7          manager.remove(this);
8
9      this.id = id;
10
11     // 添加新的 Session
12     if (manager != null)
13         manager.add(this);
14
15     // 这里面完成了 HttpSessionListener 事件通知
16     if (notify) {
17         tellNew();
18     }
19 }

```

从代码我们看到 setId 方法调用了 tellNew 方法，那 tellNew 又是如何实现的呢？

 复制代码

```

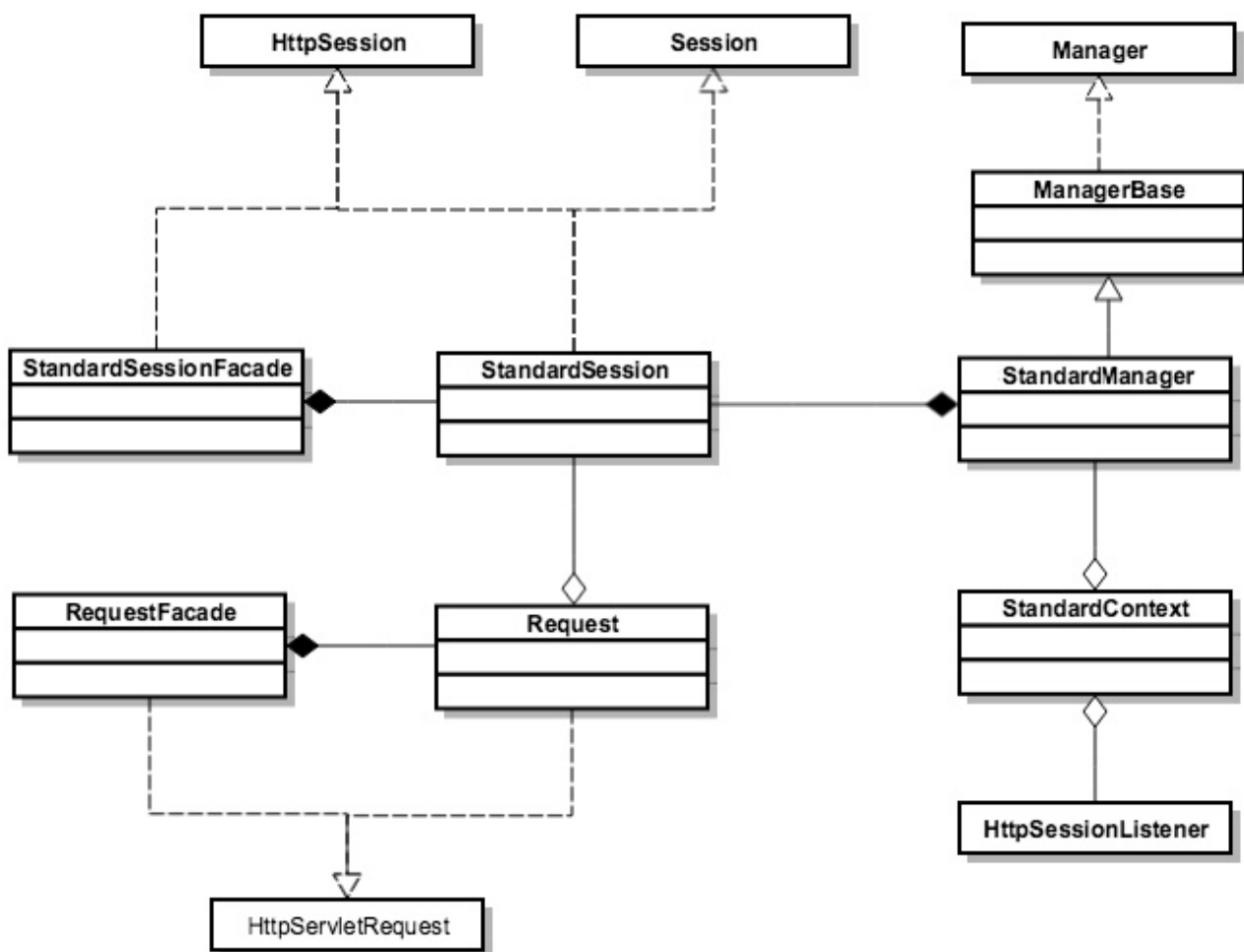
1 public void tellNew() {
2
3     // 通知 org.apache.catalina.SessionListener
4     fireSessionEvent(Session.SESSION_CREATED_EVENT, null);
5
6     // 获取 Context 内部的 LifecycleListener 并判断是否为 HttpSessionListener
7     Context context = manager.getContext();
8     Object listeners[] = context.getApplicationLifecycleListeners();
9     if (listeners != null && listeners.length > 0) {
10
11         // 创建 HttpSessionEvent
12         HttpSessionEvent event = new HttpSessionEvent(getSession());
13         for (int i = 0; i < listeners.length; i++) {
14             // 判断是否是 HttpSessionListener
15             if (!(listeners[i] instanceof HttpSessionListener))
16                 continue;
17
18             HttpSessionListener listener = (HttpSessionListener) listeners[i];
19             // 注意这是容器内部事件
20             context.fireContainerEvent("beforeSessionCreated", listener);
21             // 触发 Session Created 事件
22             listener.sessionCreated(event);
23
24             // 注意这也是容器内部事件
25             context.fireContainerEvent("afterSessionCreated", listener);
26
27         }
28     }
29 }

```


上面代码的逻辑是，先通过 StandardContext 将 HttpSessionListener 类型的 Listener 取出，然后依次调用它们的 sessionCreated 方法。

本期精华

今天我们从 Request 谈到了 Session 的创建、销毁和事件通知，里面涉及不少相关的类，下面我画了一张图帮你理解和消化一下这些类的关系：



Servlet 规范中定义了 `HttpServletRequest` 和 `HttpSession` 接口，Tomcat 实现了这些接口，但具体实现细节并没有暴露给开发者，因此定义了两个包装类，`RequestFacade` 和 `StandardSessionFacade`。

Tomcat 是通过 Manager 来管理 Session 的，默认实现是 StandardManager。StandardContext 持有 StandardManager 的实例，并存放了 HttpSessionListener 集合，Session 在创建和销毁时，会通知监听器。

课后思考

TCP 连接的过期时间和 Session 的过期时间有什么区别？

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 31 | Logger组件：Tomcat的日志框架及实战

下一篇 33 | Cluster组件：Tomcat的集群通信原理

精选留言 (7)

 写留言



-W.LI-

2019-07-23

session是会话的生命周期，每次请求都会重置超时时间，TCP链接超时，链接就被回收了(节约资源)，如果session没失效可以从新创建TCP链接通过sessionId找到之前的会话。sessionId存在cookie里面，通过http协议的head头传过来。

老师好，感觉设计模式大多数的作用就是，解耦，复用，提高系统的简装性，灵活性，还有别的作用么?...

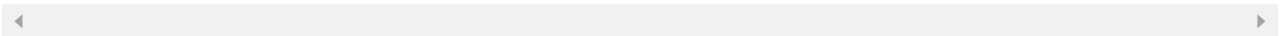
展开 ▾

作者回复: 在4线程1.6万数据的条件下，ConcurrentHashMap 存取速度是ConcurrentSkipListMap 的4倍左右。

但ConcurrentSkipListMap有几个ConcurrentHashMap 不能比拟的优点：

- 1、ConcurrentSkipListMap 的key是有序的。
- 2、ConcurrentSkipListMap 支持更高的并发。ConcurrentSkipListMap 的存取时间是 $\log(N)$ ，和线程数几乎无关。也就是说在数据量一定的情况下，并发的线程越多，ConcurrentSkipListMap越能体现出他的优势。

参见：<http://www.java-forums.org/new-java/13840-hashmap-vs-skiplistmap.html>



6



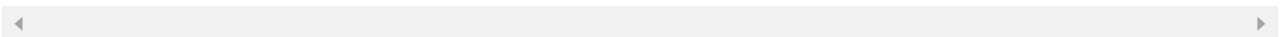
永钱

2019-07-23

Tcp是系统网络层面的，而session是应用层面的，应用层面完全由应用控制生命周期，他们之间没什么关系。不知道理解对不对，求指正

展开 ▾

作者回复: 对的



2



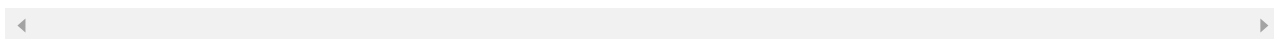
发条橙子。

2019-07-24

老师 我有个其他的疑问，一般我们用tomcat起java程序的时候都用的jvm的默认参数，那如果我想更改一些jvm参数，应该在tomcat哪里配置

展开 ▾

作者回复: 在bin目录下新建一个setenv.sh的文件, 在这个文件里加jvm参数



👍 1



magicnum

2019-07-23

一个是传输层连接的断开时间, 另一个是应用层用户会话的过期时间, 两者没啥关系, 但是超时目的其实都是为了减少服务器资源占用

展开 ▾



👍 1

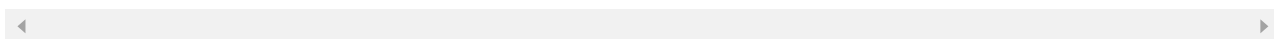


梁中华

2019-07-24

不谈下分布式session在tomcat中的实现吗? 比如多个tomcat实例如何共享session?

作者回复: 下篇会讲这个



小飞

2019-07-23

其实在StandardManager 中没有 backgroundProcess()和 processExpires()的具体实现的。他们的实现提取到了基类ManagerBase中

展开 ▾



WL

2019-07-23

老师请问一下如果是集群部署的Tomcat容器怎么实现分布式的session, 还能有standardManager管理吗? 我看有filestore和jdbcstore两个类, 但是感觉靠这两个类好像也没法实现, 请老师指点一下具体怎么集群情况下的session管理

展开 ▾

作者回复: 下篇会讲到

