# 主题1-如何阅读各个版本的tomcat

# 背景

我们知道，我们《How Tomcat Works》是以tomcat4为基础讲解的。那么我们的问题来了，经过tomcat多个版本的迭代，目前最新的版本，和tomcat4有什么不同？各个版本，都有哪些更新？为了解答这个问题，我们详细解读一下这个主题。

# 如何调整各个版本的tomcat源码工程？

我们从tomcat官网下载到源码之后，如何调整各个版本的tomcat以方便我们通过阅读源码，比较各个版本的实现逻辑？综合来看，还是IDEA查看源码最方便，无论是java文件查找，全文搜索，还是高亮可读性，都是非常不错的，远胜过之前我们在sublime中阅读tomcat源码工程的体验。

## 1.进入某个版本的tomcat源码目录

|  |
| --- |
| cd  /Users/zhoushuo/Documents/Tech/tomcat/Tomcat各个版本/apache-tomcat-4.1.40-src |

## 2.用idea打开这个工程

|  |
| --- |
| idea . |

## 3.设置相关配置

设置SDK/Project就不多说了，我们单独说一下如何设置Module。找到tomcat源码目录，则是为”Source”：

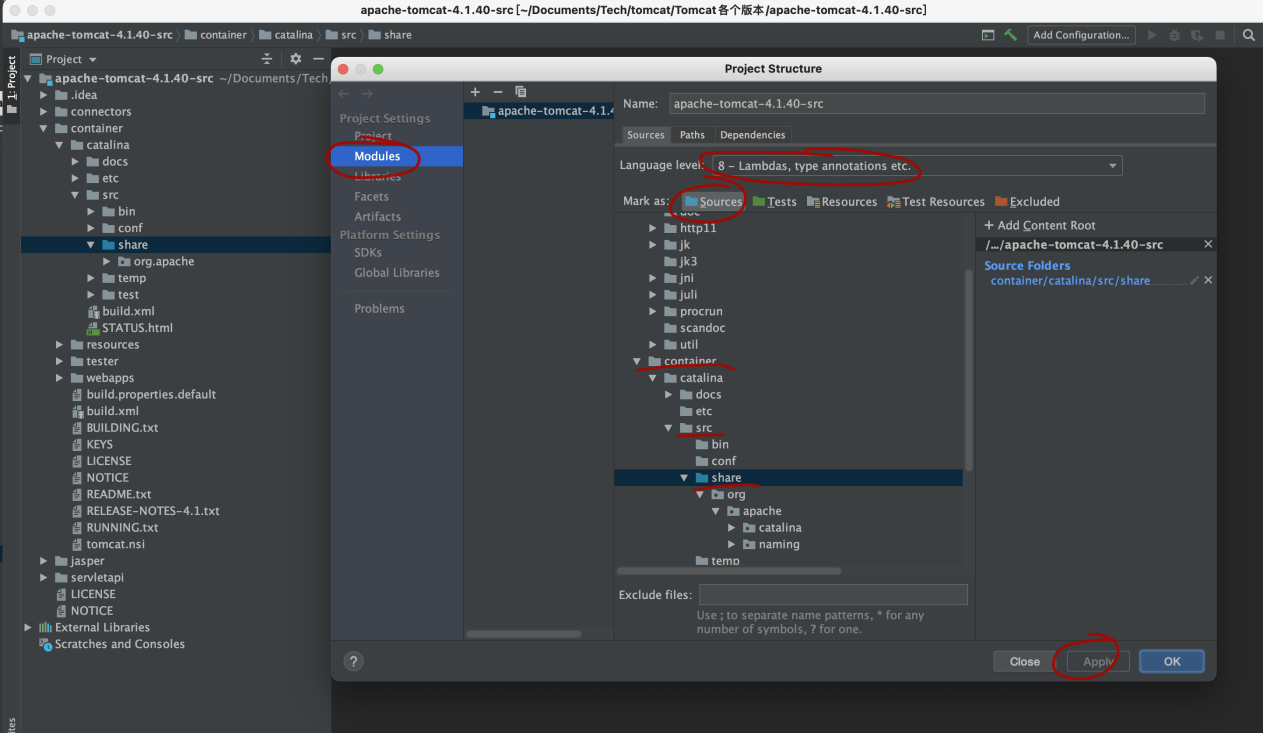


图1 idea配置tomcat源码source

至于这么找到这个目录，只要找一个典型的tomcat源码，比如StandardWrapper.java，看一下这个文件的package路径就知道了：

|  |
| --- |
| org.apache.catalina.core |

设置好Source之后，我们就能够像操作普通工程一样管理tomcat源码工程了。

# 源码解读案例1-pipeline

如果一上来就要了解整个tomcat的演进过程，那显然是不现实的。我们一般是以某个功能，或者某个主题为契入点，了解这个功能、主题的演进过程。为此，我们通过pipeline功能在不同tomcat版本的演进过程，说明tomcat源码是如何演进的

## 主题解读

要了解某个功能、某个主题的演进过程，当然先要对这个功能/主题进行深入了解。所以，在阅读源码演进过程之前，需要对该功能进行专题解读。以《主题-pipeline》为例，我们讨论了pipeline适用的场景、接口、整体架构、实现原理、在tomcat中的应用，最后我们还通过一个demo project实践了一下pipeline。

## tomcat4中pipeline的架构

我们先看tomcat4中pipeline的架构。在tomcat4中，StandardPipeline作为Pipeline架构的核心实现类，展示了pipeline的架构：

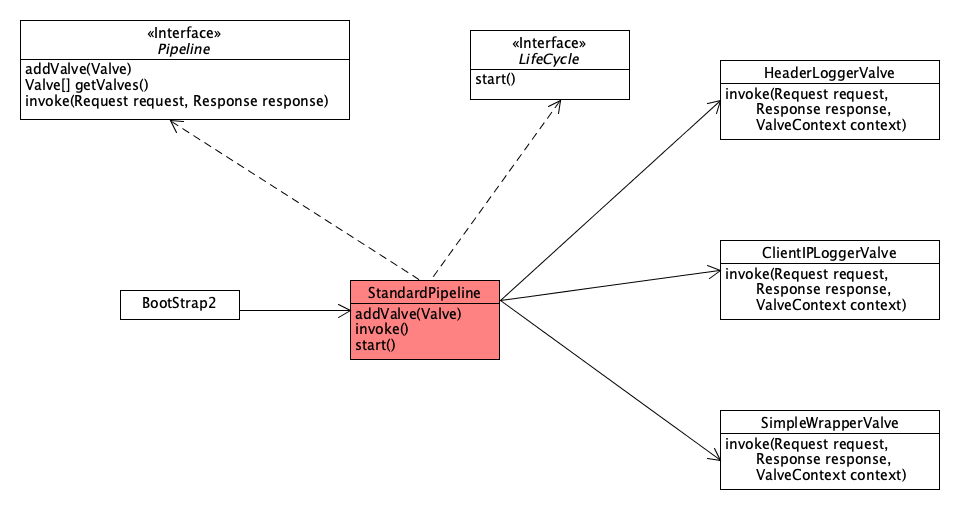


图1 tomcat4中pipeline实现类：StandardPipeline

第一步:pipeline添加valves。StandardContext实现了Pileline接口,通过调用功能addValve()方法,添加valves。

第二步:启动pileline。

启动pileline就比较有意思了，StandardContext既实现了Pileline接口，能够通过invoke()方法调用各个valve，就是我们在上面提到的，通过SimplePipelineValveContext.invokeNext()执行各个valve：

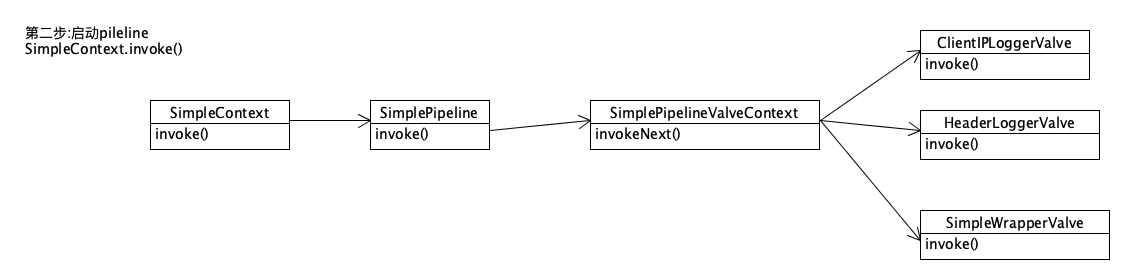


图2 StandardContext启动pileline的方式1

同时，StandardContext还实现了LifeCycle接口，通过start()方法调用各个valve。StandardPipeline.start()遍历各个valve,调用各个valve的start()方法。核心代码如下：

|  |
| --- |
| public synchronized void start() throws LifecycleException {  // Start the Valves in our pipeline (including the basic), if any  for (int i = 0; i < valves.length; i++) {  if (valves[i] instanceof Lifecycle)  ((Lifecycle) valves[i]).start();  }  ......  } |

可以看到，就是简单的遍历各个valves。

因此，从源码分析来看，StandardPipeline支持两种方式执行各个valve。

我分析可能是tomcat处于转型阶段，为了兼容新老版本的pipeline，所以这么做。

## Tomcat5中pipeline的架构

我们来到tomcat5，看到pipeline的实现类也是StandardContext。StandardContext也是既实现了Pipeline接口，实现了添加valve的功能(addValve())，又实现了LifeCycle接口，通过LifeCycle.start()启动valves。

第一步:pipeline添加valves

StandardPipeline实现了Pileline接口,

通过调用功能addValve()方法,添加valves

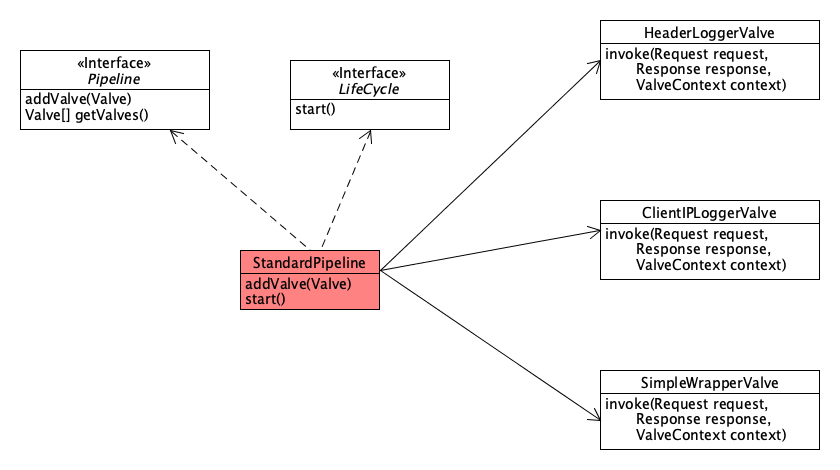


图1 tomcat5中pipeline的实现类：StandardPipeline

第二步:启动pileline。

Tomcat5启动pipeline的方式稍微有些不同。Tomcat4中是for循环简单遍历各个valves，类似array。Tomcat5是while循环遍历valves，类似链表。

|  |
| --- |
| public synchronized void start() throws LifecycleException {   Valve current = first;  if (current == null) {  current = basic;  }  while (current != null) {  if (current instanceof Lifecycle)  ((Lifecycle) current).start();  registerValve(current);  current = current.getNext();  }  ......  } |

看到这里我们就有疑问了，current valve执行完毕后，怎么知道current.getNext() valve是谁呢？

我们看一下StandardPipeline.addValve()就知道了：

|  |
| --- |
| public void addValve(Valve valve) {  if (first == null) {  first = valve;  valve.setNext(basic);  } else {  Valve current = first;  while (current != null) {  if (current.getNext() == basic) {  current.setNext(valve);  valve.setNext(basic);  break;  }  current = current.getNext();  }  }  } |

可以看到，valve在add进来的时候，就指定了next valve。

备注：各个valve执行方式和tomat4也有所不同

## Tomcat11中pipeline的架构

我们参考：

|  |
| --- |
| 工程：  apache-tomcat-11.0.0-M11-src  代码：  StandardPipeline.java  核心方法：  startInternal() |

核心代码片段如下：

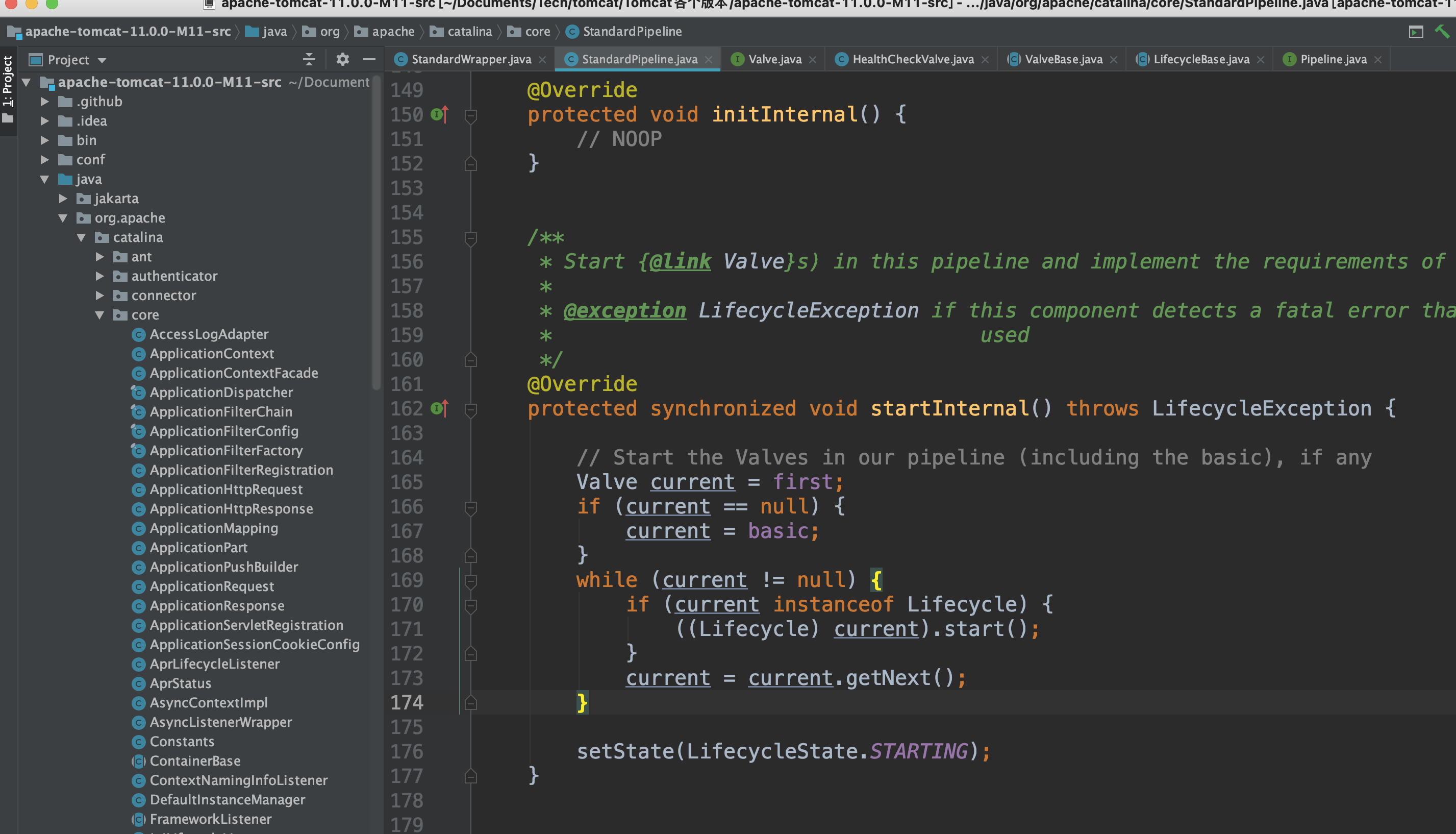


图1 Tomcat11中pipeline的架构

一看这段代码就知道，tomcat11实现pipeline的方式和tomat5是类似的，就是通过一个while循环遍历各个valve，然后执行valve。

## 总结

到此为止，我们就搞清楚了pipeline功能演进的脉络：tomat4处于转型之中，tomcat5-tomcat11就基本定型了。一直沿用一种pipeline实现方式。

另外，从pipeline这个功能的演进过程我们可以看到，《How Tomcat Works》这本主要是依托于tomcat4。

当然，基于pipeline还有很多相关源码，比如，有哪些重要的valve实现(参考org.apache.catalina.valves这个package)、

# 总结