# 主题-pipeline

# 背景

《How Tomcat Works》 chap5引入了pipeline，这是tomat核心组件之一，我们专题解读一下。

我们感兴趣的内容如下：

pipeline适合什么样的场景？

Pipeline有哪些接口？

pipeline实现原理，整体架构？

pipeline运转流程？

pipeline在tomcat中的应用场景

tomcat中有哪些常用的valve

tomcat中各个版本关于pipeline部分的功能，是如何演进的？

pipeline如何作为设计模式应用到项目中？

# pipeline适合什么样的场景？

所谓的Pipeline，就是管道啦。我们在Linux中对管道已经非常熟悉了，就是执行一串命令：

|  |
| --- |
| netstat -an|grep 1521 |

tomcat这里的pipeline架构也是类似，就是执行一串逻辑。pipeline适合那种类似filter的场景，就是执行一串任务，比如在tomcat中，对一个http sevlet请求进行一连串的处理：

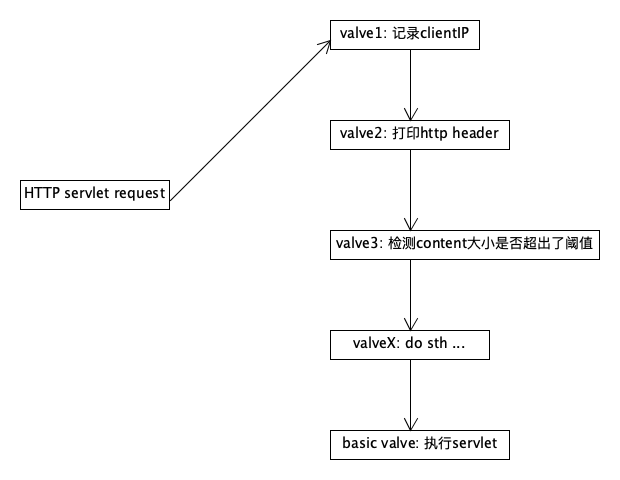


图1 pipeline适合的场景之一： http servlet request处理

# Pipeline有哪些接口？

pipeline框架有哪些接口呢？

pipeline核心接口有三个：

|  |
| --- |
| 1.Pipeline  2.Valve  3.ValveContext |

其中pipeline是流水线核心接口，valve是pipeline中执行的各个任务，ValveContext 负责执行各个valve。下面我们会具体介绍这三个核心接口。但是在介绍接口前，我们先要确认一件事情，就是下面讨论的三个接口，是基于tomcat4讨论的。后续版本可能会有所有不同(具体演进内容参考《pipeline功能演进》这章)。

## The Pipeline Interface

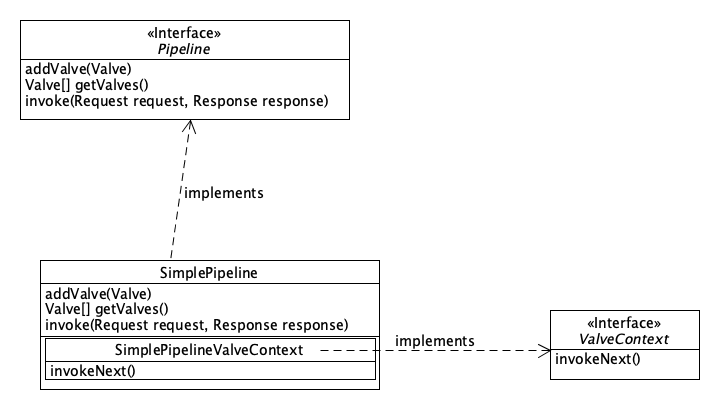


图1 pipeline接口

The Pipeline Interface作为管道的抽象接口，其包含的接口方法说明如下:

|  |
| --- |
| 1.addValve()  添加valve实例  2.getValves()  返回所有的valse实例  3.invoke()  执行pipeline下所有的valve |

The Pipeline Interface的实现类是SimplePipeline。Pipeline interface invoke()方法的

功能是执行pipeline下所有的valve那么具体是怎么执行的呢?

我们看SimplePipeline.invoke()的逻辑，其实是调用SimplePipeline的内部类:

|  |
| --- |
| SimplePipelineValveContext.invokeNext() |

具体逻辑会在ValveContext Interface中详细说明。

## The Valve Interface

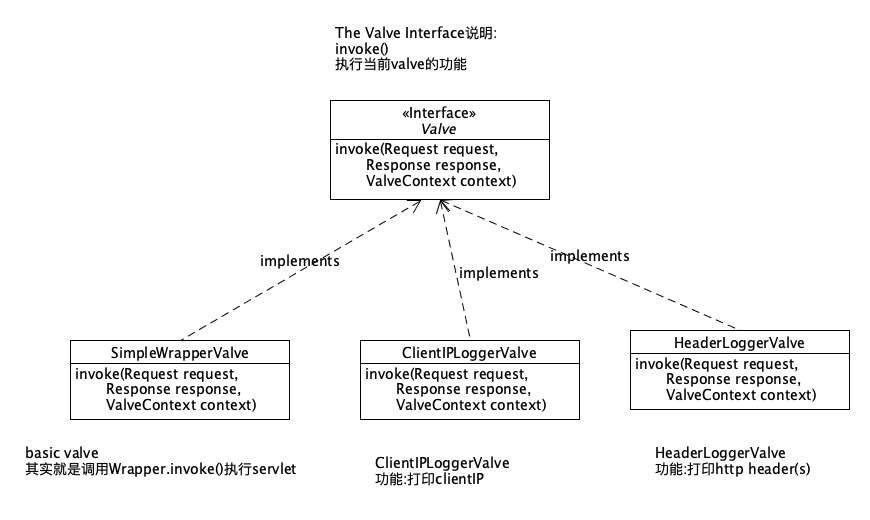


图2 valve接口

The Valve Interface作为管道中的各个任务，接口方法只有一个：

|  |
| --- |
| invoke()  执行当前valve的功能 |

Valve接口的实现类有三个，分别是两个普通valve:ClientIPLoggerValve、HeaderLoggerValve，以及一个基础valve: SimpleWrapperValve。他们的功能如上图2所示。

## The ValveContext Interface

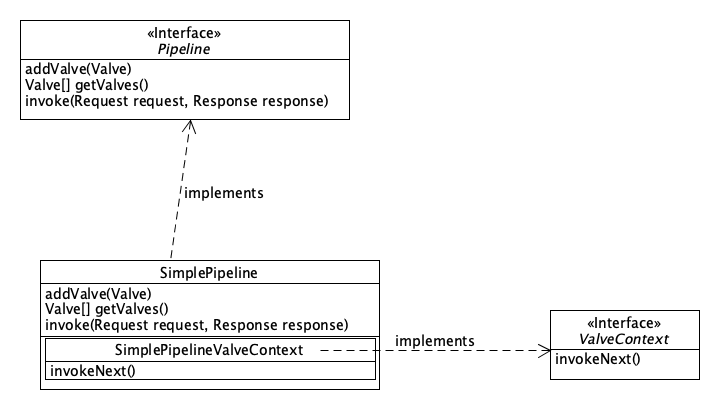


图3 ValveContext接口

ValveContext Interface作为pipeline执行各个valve的容器，只有一个接口方法:

invokeNext()。

从字面意思来看,invokeNext()是执行下一个任务(valve)。具体是什么意思呢?我们看，ValveContext接口的实现类:SimplePipelineValveContext包含了具体实现invokeNext()的逻辑。

那么,SimplePipelineValveContext.invokeNext()具体是怎么执行各个valve的呢?是按照for循环逐个执行吗?不是的.

我们还是看SimplePipelineValveContext.invokeNext()代码:

1.先执行普通valves

2.最后再执行basice valve

那么普通valves是怎么执行的呢?这里并没有类似for循环的逻辑啊。我们随便找一个valve实现类看一下。比如ClientIPLoggerValve.invoke(),实现逻辑是这样的:

1.先执行下一个valve:

context.invokeNext(request,response);

备注：其中context就是SimplePipelineValveContext实例,所以这里invokeNext()其实就是调用invokeNext()。我们只要观察SimplePipelineValveContext.invokeNext()就可以知道，SimplePipelineValveContext通过stage字段维护了下一个需要执行的valve。

2.然后再执行当前valve自身的逻辑.

这样一看就明白了,其实就是valve在执行invoke()的时候,先执行下一个valve,再执行当前valve.这样就造成了valve执行顺序和加载顺序是相反的.比如加载valve的时候:

xx.addValve(valve1);

xx.addValve(valve2);

xx.addValve(valve3);

实际的执行顺序是这样的:

valve3.invoke();

valve2.invoke();

valve1.invoke();

basicValve.invoke();

# pipeline实现原理，整体架构

我们介绍了Pipeline Task三个接口：Pipeline/Valve/ValveContext，这三个接口是Pipeline的核心接口，组成了Pipeline的整体架构。那么，他们的关系是什么样的呢？就要看他们的实现类逻辑了：

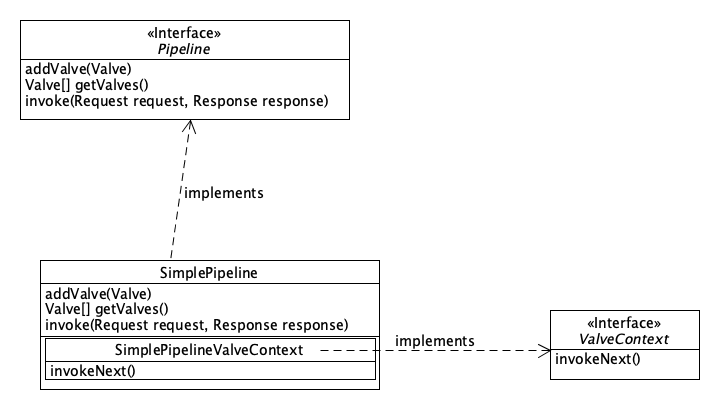


图4 pipeline整体架构

图4展示了pipeline的整体架构

1. SimplePipeline通过addValve()方法，加载各个valve实例；
2. 通过invoke()方法，执行pipeline下所有的valve
3. SimplePipeline.invoke()方法调用内部类SimplePipelineValveContext.invokeNext()，执行pipeline下所有的valve
4. SimplePipelineValveContext.invokeNext()方法先执行普通valves，最后再执行basice valve。
5. 普通valves的执行顺序，依赖各个valve自身的逻辑。

到这里，我们不仅介绍了Pipeline Task三个接口：Pipeline/Valve/ValveContext，还介绍了由这三个接口组成的Pipeline 整体架构。

# pipeline在tomcat中的应用场景

我们以TomcatWin/chap5中的代码为例，说明pipeline在tomcat中的应用场景

## 第一步:pipeline添加valves

SimpleContext实现了Pileline接口,通过调用Pileline.addValve()方法,添加各个valves。具体参考chap5/BootStrap2.java

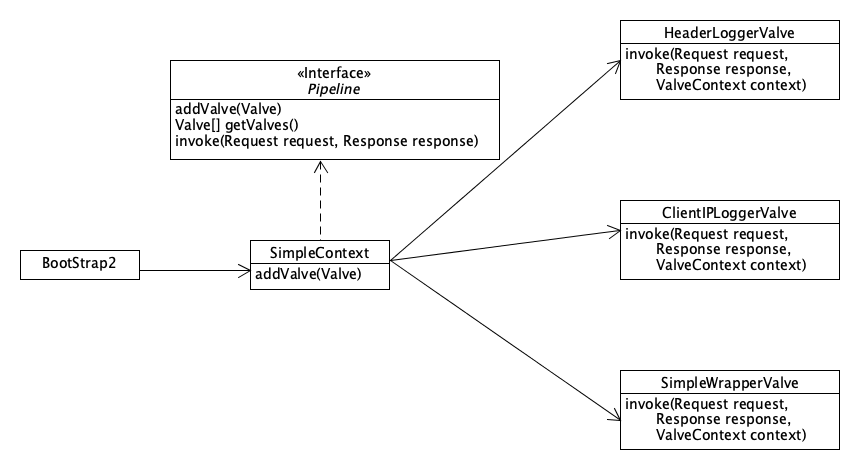


图1 pipeline添加valves

## 第二步:启动pipeline

pipeline添加了各个valve之后，如何执行各个valve呢？

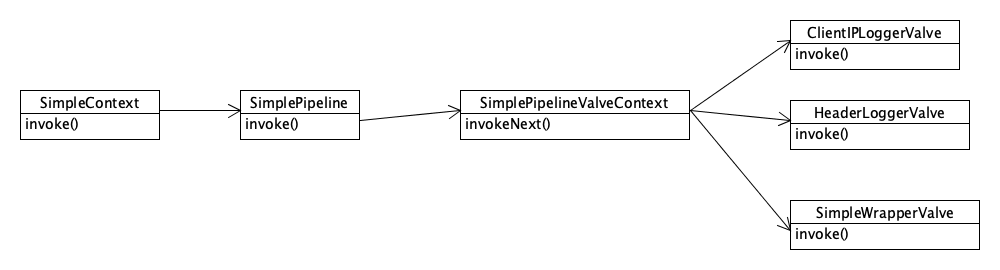


图2 pipeline执行

1.SimpleContext.invoke()

2.SimplePipeline.invoke()

3.SimplePipelineValveContext.invokeNext()

4.各个valves按顺序执行

我们以chap5 BootStrap2.java为例，介绍了pipeline的用法，这个pipeline作用于SimpleContext，意思很明确，就是每个servlet请求过来以后，都会执行ClientIPLoggerValve/HeaderLoggerValve/SimpleWrapperValve中定义的逻辑。

作为对比，chap5 BootStrap1.java把各个valve添加到某个Wrapper中，意思也非常明确，就是这个Wrapper执行servlet的时候，作用各个valve。

这个意思明白吗？就是任何对象(在tomcat中一般是Context、Wrapper)都可以利用pipeline接口框架，实现一串业务逻辑。这有点像是filter的功能。

后续我们调研一下servlet filter的实现逻辑。

# pipeline功能演进

tomcat中各个版本关于pipeline部分的功能，是如何演进的？

之前我们分析了pipeline的整体架构，但这是《How Tomcat Works》基于tomcat4的架构。在后续的版本中，pipeline功能是否有演进呢？我们解读一下。

## 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：

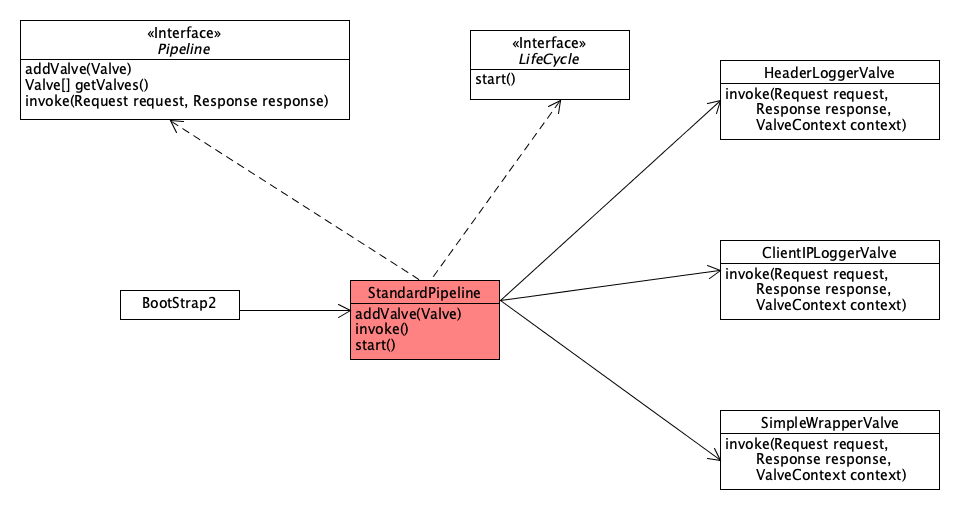


图1 tomcat4中pipeline实现类：StandardPipeline

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：

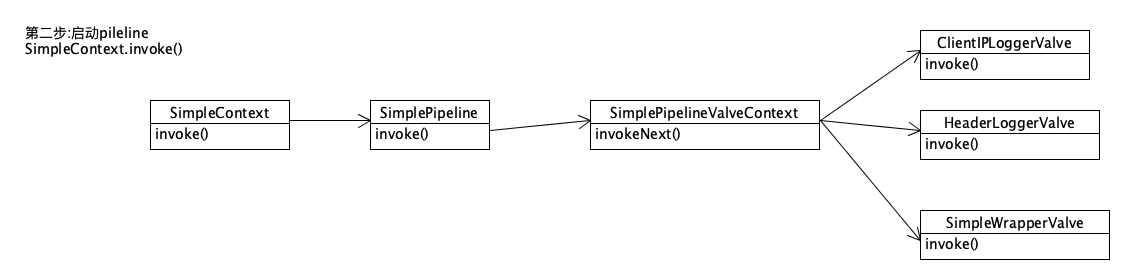


图2 StandardContext启动pileline的方式1

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

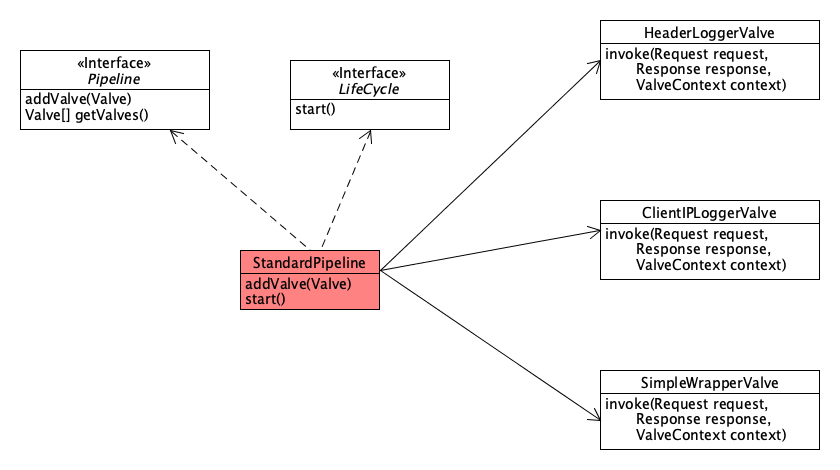


图1 tomcat5中pipeline的实现类：StandardPipeline

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：

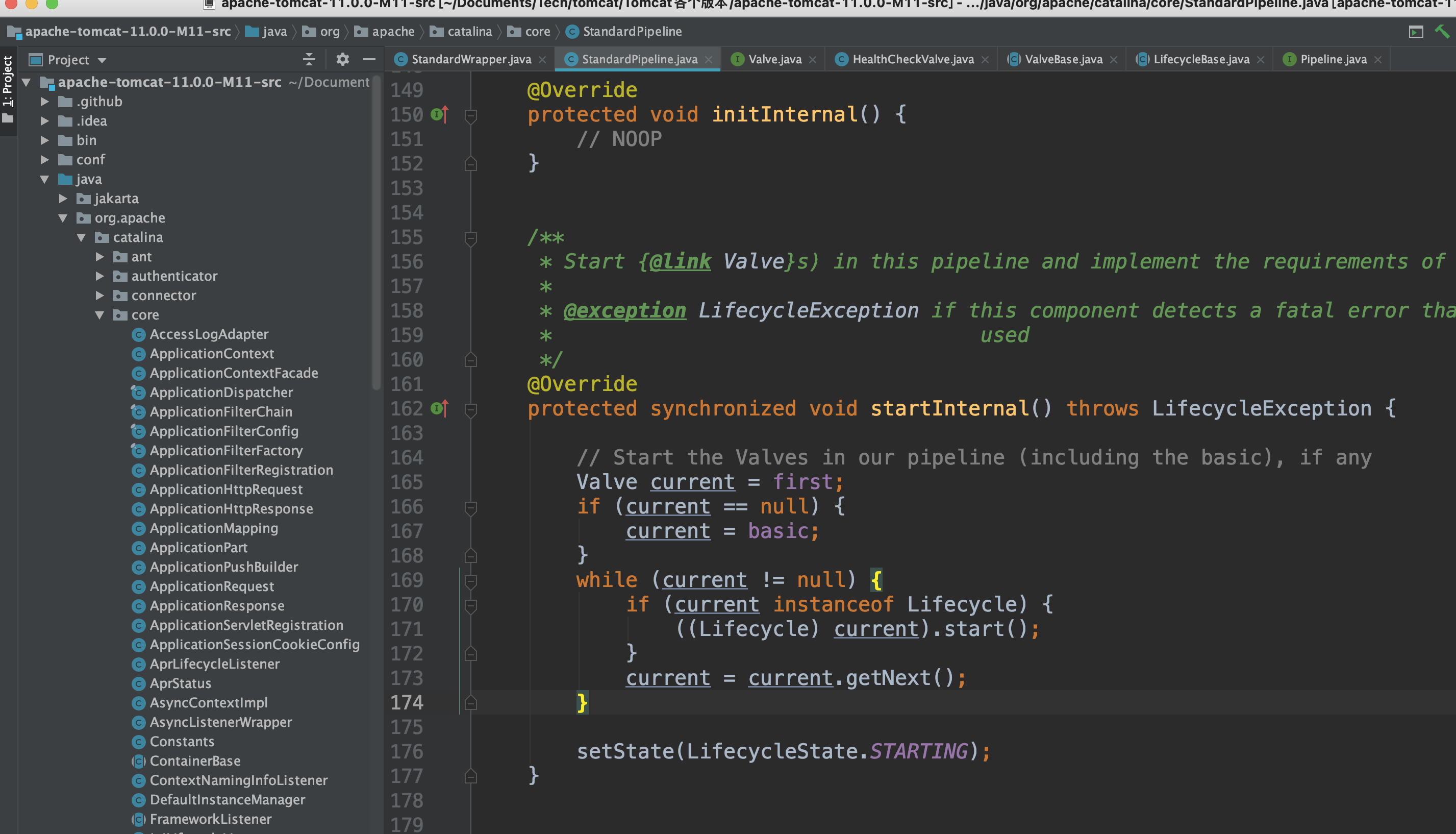


图1 Tomcat11中pipeline的架构

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

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。

在各个版本中，pipeline

# pipeline如何作为设计模式应用到项目中？

pipeline其实是一种设计模式，实现类似servlet filter这类功能。我们把pipeline从tomcat中抽象出来。单独搞一个demo。后续可以应用到其他项目中。

首先是版本选择，我们是选择书本中tomcat4为基础的实现版本，还是tomcat5及以后的版本呢？我们选择目前最新的版本：tomcat11作为参考。

具体demo工程参考：

|  |
| --- |
| PipelineDemo |

UML 参考：

|  |
| --- |
| 5-PipelineDemo.uxf |

# 后续

虽然我们对pipeline主题进行了深入的学习，但这当然是远远不够的。后续还有很多内容需要我们去挖掘。

1. tomcat有哪些重要的valve?使用场景分别在哪里？

(参考org.apache.catalina.valves这个package)

2.

# Pipeline主题总结

我们在这个主题中充分讨论了pipeline适用的场景、接口、整体架构、实现原理、在tomcat中的应用，讨论了pipeline在tomcat各个版本中功能是如何演进的。最后我们还通过一个demo project实践了一下pipeline。

后续如果碰到类似的场景，就可以把pipeline用起来。