

17 | 设计模式应用：编程框架中的设计模式

2019-12-30 李智慧

后端技术面试38讲

[进入课程 >](#)



讲述：李智慧

时长 13:01 大小 11.93M



在绝大多数情况下，我们开发应用程序的时候，并不是从头开发的。比如，我们用 Java 开发一个 Web 应用，并不需要自己写代码监听 HTTP 80 端口；不需要处理网络传输的二进制 HTTP 数据包（参考 [第 4 篇网络编程原理](#)）；不需要亲自为每个用户请求分配一个处理线程（参考 [01 篇操作系统原理](#)），而是直接编写一个 Servlet，得到一个 HttpRequest 对象进行处理就可以了。我们甚至不需要从这个 HttpRequest 对象中获取请求参数，通过 Controller 就可以直接得到一个由请求参数构造的对象。

我们写代码的时候，只需要关注自己的业务逻辑就可以了。那些通用的功能，比如监听 HTTP 端口，从 HTTP 请求中构造参数对象，是由一些通用的框架来完成的，比如 Tomcat 或者 Spring 这些。

什么是框架

框架是对某一类架构方案可复用的设计与实现。所有的 Web 应用都需要监听 HTTP 端口，也需要处理请求参数，这些功能不应该在每个 Web 应用中都被重复开发，而是应该以通用组件的形式被复用。

但并不是所有可被复用的组件都被称作框架，框架通常规定了一个软件的主体结构，可以支撑起软件的整体或者局部的架构形式。比如说，Tomcat 完成了 Web 应用请求响应的主体流程，我们只需要开发 Servlet，完成请求处理逻辑，构造响应对象就可以了，所以 Tomcat 是一个框架。

还有一类可复用的组件不控制软件的主体流程，也不支撑软件的整体架构，比如 Log4J 提供了一个可复用的日志输出功能，但是，日志输出功能不是软件的主体结构，所以我们通常不称 Log4J 为框架，而称其为工具。

一般说来，我们使用框架编程的时候，需要遵循框架的规范编写代码。比如 Tomcat、Spring、Mybatis、Junit 等，这些**框架会调用我们编写的代码，而我们编写的代码则会调用工具**完成某些特定的功能，比如输出日志，进行正则表达式匹配等。

我在这里强调框架与工具的不同，并不是在咬文嚼字。我见过一些有进取心的工程师宣称自己设计开发了一个新框架，但是这个框架并没有提供一些架构性的规范，也没有支撑软件的主体结构，仅仅只是提供了一些功能接口供开发者调用，实际上，这跟我们对框架的期待相去甚远。

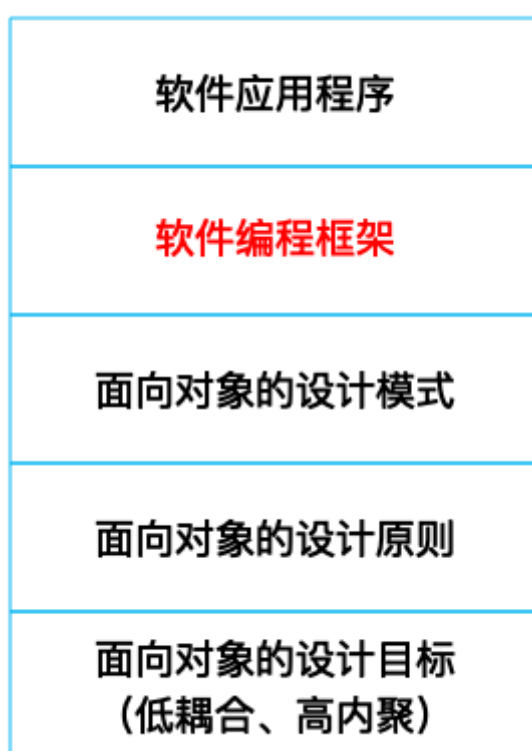
根据我们上面对框架的描述，当你设计一个框架的时候，你实际上是在设计一类软件的通用架构，并通过代码的方式实现出来。如果仅仅是提供功能接口供程序调用，是无法支撑起软件的架构的，也无法规范软件的结构。

那么如何设计、开发一个编程框架？

我在前面讲过开闭原则。框架应该满足**开闭原则**，即面对不同应用场景，框架本身是不需要修改的，需要对修改关闭。但是各种应用功能却是可以扩展的，即对扩展开放，应用程序可以在框架的基础上扩展出各种业务功能。

同时框架还应该满足**依赖倒置原则**，即框架不应该依赖应用程序，因为开发框架的时候，应用程序还没有呢。应用程序也不应该依赖框架，这样应用程序可以灵活更换框架。框架和应用程序应该都依赖抽象，比如 Tomcat 提供的编程接口就是 Servlet，应用程序只需要实现 Servlet 接口，就可以在 Tomcat 框架下运行，不需要依赖 Tomcat，可以随时切换到 Jetty 等其他 Web 容器。

要知道，虽然设计原则可以指导框架开发，但是并没有给出具体的设计方法。事实上，框架正是利用各种设计模式开发出来的。编程框架与应用程序、设计模式、设计原则之间的关系如下图所示。



面向对象的设计目标是低耦合、高内聚。为了实现这个目标，人们提出了一些设计原则，主要有开闭原则、依赖倒置原则、里氏替换原则、单一职责原则、接口隔离原则。在这些原则之上，人们总结了若干设计模式，最著名的就是 GoF23 种设计模式，还有 Web 开发同学非常熟悉的 MVC 模式等等。依照这些设计模式，人们开发了各种编程框架。使用这些编程框架，开发者可以简单、快速地开发各种应用程序。


Web 容器中的设计模式

前面我们一再提到 Tomcat 是一个框架，那么 Tomcat 与其他同类的 Web 容器是用什么设计模式实现的？代码如何被 Web 容器执行？程序中的请求对象 HttpServletRequest 是

从哪里来的?

Web 容器主要使用了**策略模式**，多个策略实现同一个策略接口。编程的时候 Tomcat 依赖策略接口，而在运行期根据不同上下文，Tomcat 装载不同的策略实现。

这里的策略接口就是 Servlet 接口，而我们开发的代码就是实现这个 Servlet 接口，处理 HTTP 请求。J2EE 规范定义了 Servlet 接口，接口中主要有三个方法：

 复制代码

```
1 public interface Servlet {  
2     public void init(ServletConfig config) throws ServletException;  
3     public void service(ServletRequest req, ServletResponse res)  
4         throws ServletException, IOException;  
5     public void destroy();  
6 }  
7
```

Web 容器 Container 在装载我们开发的 Servlet 具体类的时候，会调用这个类的 init 方法进行初始化。当有 HTTP 请求到达容器的时候，容器会对 HTTP 请求中的二进制编码进行反序列化，封装成 ServletRequest 对象，然后调用 Servlet 的 service 方法进行处理。当容器关闭的时候，会调用 destroy 方法做善后处理。

当我们开发 Web 应用的时候，只需要实现这个 Servlet 接口，开发自己的 Servlet 就可以了，容器会监听 HTTP 端口，并将收到的 HTTP 数据包封装成 ServletRequest 对象，调用我们的 Servlet 代码。代码只需要从 ServletRequest 中获取请求数据进行处理计算就可以了，处理结果可以通过 ServletResponse 返回给客户端。

这里 Tomcat 就是策略模式中的 Client 程序，Servlet 接口是策略接口。我们自己开发的具体 Servlet 类就是策略的实现。通过使用策略模式，Tomcat 这样的 Web 容器可以调用各种 Servlet 应用程序代码，而各种 Servlet 应用程序代码也可以运行在 Jetty 等其他的 Web 容器里，只要这些 Web 容器都支持 Servlet 接口就可以了。

Web 容器完成了 HTTP 请求处理的主要流程，指定了 Servlet 接口规范，实现了 Web 开发的主要架构，开发者只要在这个架构下开发具体 Servlet 就可以了。因此我们可以称 Tomcat、Jetty 这类 Web 容器为框架。

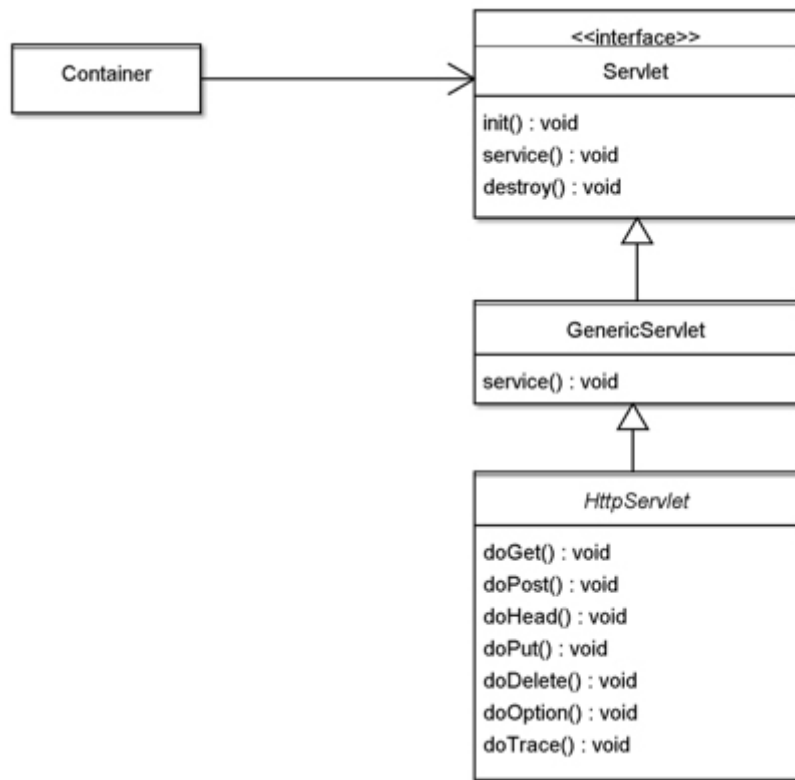
事实上，我们开发具体的 Servlet 应用程序的时候，并不会直接实现 Servlet 接口，而是继承 HttpServlet 类，HttpServlet 类实现了 Servlet 接口。HttpServlet 还用到了**模板方法模式**，所谓模板方法模式，就是在父类中用抽象方法定义计算的骨架和过程，而抽象方法的实现则留在子类中。

这里，父类是 HttpServlet，HttpServlet 通过继承 GenericServlet 实现了 Servlet 接口，并在自己的 service 方法中，针对不同的 HTTP 请求类型调用相应的方法，HttpServlet 的 service 方法就是一个模板方法。

 复制代码

```
1      protected void service(HttpServletRequest req, HttpServletResponse resp) tl
2      {
3          String method = req.getMethod();
4          if (method.equals(METHOD_GET)) {
5              doGet(req, resp);
6          } else if (method.equals(METHOD_HEAD)) {
7              long lastModified = getLastModified(req);
8              maybeSetLastModified(resp, lastModified);
9              doHead(req, resp);
10         } else if ...
```

由于 HTTP 请求有 get、post 等 7 种请求类型，为了便于编程，HttpServlet 提供了这 7 种 HTTP 请求类型对应的执行方法 doGet、doPost 等等。service 模板方法会判断 HTTP 请求类型，根据不同请求类型，执行不同方法。开发者只需要继承 HttpServlet，重写 doGet、doPost 等对应的 HTTP 请求类型方法就可以了，不需要自己判断 HTTP 请求类型。Servlet 相关的类关系如下：



JUnit 中的设计模式

JUnit 是一个 Java 单元测试框架，开发者只需要继承 JUnit 的 `TestCase`，开发自己的测试用例类，通过 JUnit 框架执行测试，就得到测试结果。

开发测试用例如下：

```
1 public class MyTest extends TestCase {
2     protected void setUp(){
3         ...
4     }
5     public void testSome(){
6         ...
7     }
8     protected void tearDown(){
9         ...
10    }
11 }
```

 复制代码

每个测试用例继承 `TestCase`，在 `setUp` 方法里做一些测试初始化的工作，比如装载测试数据什么的；然后编写多个以 `test` 为前缀的方法，这些方法就是测试用例方法；还有一个 `tearDown` 方法，在测试结束后，进行一些收尾的工作，比如删除数据库中的测试数据等。

那么，我们写的这些测试用例如何被 JUnit 执行呢？如何保证测试用例中这几个方法的执行顺序呢？JUnit 在这里也使用了**模板方法模式**，测试用例的方法执行顺序被固定在 JUnit 框架的模板方法里。如下：

 复制代码

```
1 public void runBare() throws Throwable {
2     setUp();
3     try{
4         runTest();
5     }
6     finally {
7         tearDown();
8     }
9 }
```

runBare 是 TestCase 基类里的方法，测试用例执行时实际上只执行 runBare 模板方法，这个方法里，先执行 setUp 方法，然后执行各种 test 前缀的方法，最后执行 tearDown 方法。保证每个测试用例都进行初始化及必要的收尾。而我们的测试类只需要继承 TestCase 基类，实现 setUp、tearDown 以及其他测试方法就可以了。

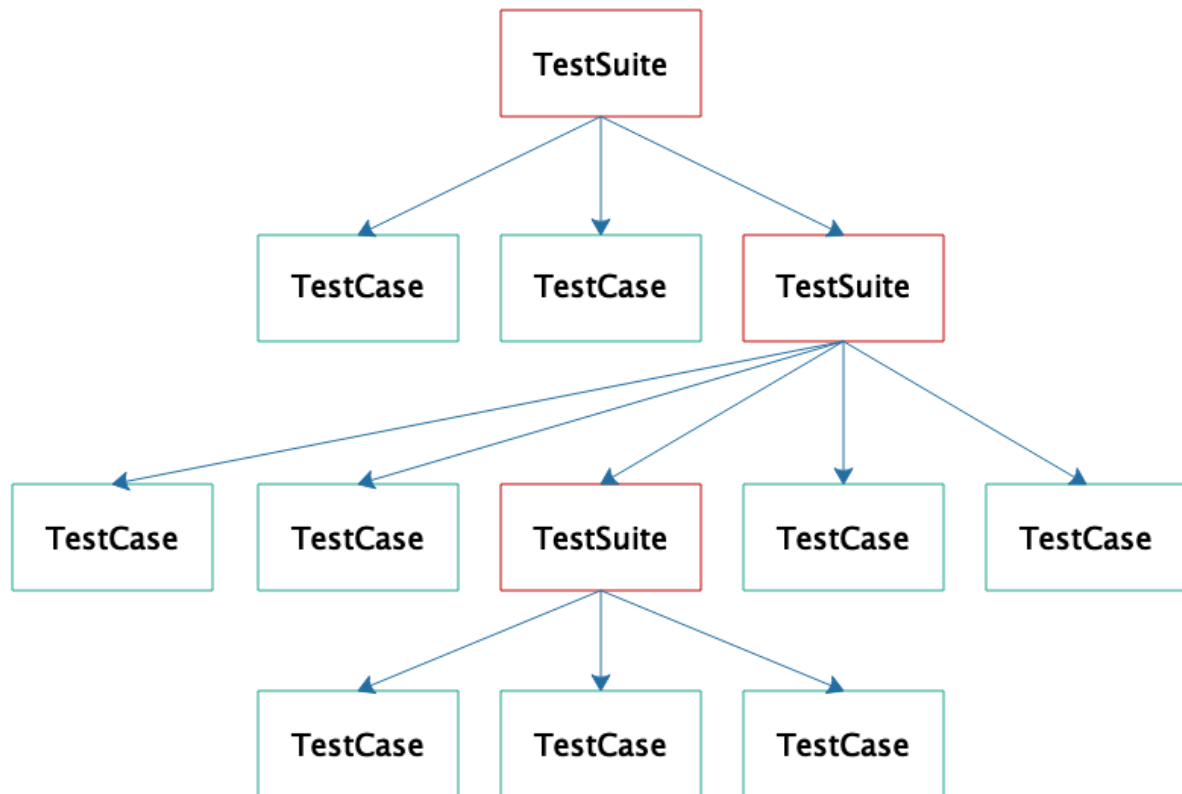
此外，一个软件的测试用例会有很多，你可能希望执行全部这些用例，也可能希望执行一部分用例，JUnit 提供了一个测试套件 TestSuite 管理、组织测试用例。

 复制代码

```
1 public static Test suite() {
2     TestSuite suite = new TestSuite("all");
3     suite.addTest(MyTest.class); // 加入一个 TestCase
4     suite.addTest(otherTestSuite); // 加入一个 TestSuite
5     return suite;
6 }
```

TestSuite 可以通过 addTest 方法将多个 TestCase 类加入一个测试套件 suite，还可以将另一个 TestSuite 加入这个测试套件。当执行这个 TestSuite 的时候，加入的测试类 TestCase 会被执行，加入的其他测试套件 TestSuite 里面的测试类也会被执行，如果其他的测试套件里包含了另外一些测试套件，也都会被执行。

这也就意味着 TestSuite 是可以递归的，事实上，TestSuite 是一个树状的结构，如下：



当我们从树的根节点遍历树，就可以执行所有这些测试用例。传统上进行树的遍历需要递归编程的，而使用**组合模式**，无需递归也可以遍历树。

首先，TestSuite 和 TestCase 都实现了接口 Test：

```
1 public interface Test {
2     public abstract void run(TestResult result);
3 }
```

[复制代码](#)

当我们调用 TestSuite 的 addTest 方法时，TestSuite 会将输入的对象放入一个数组：

```
1 private Vector<Test> fTests= new Vector<Test>(10);
2
3
4 public void addTest(Test test) {
5     fTests.add(test);
6 }
```

[复制代码](#)

由于 TestCase 和 TestSuite 都实现了 Test 接口，所以 addTest 的时候，既可以传入 TestCase，也可以传入 TestSuite。执行 TestSuite 的 run 方法时，会取出这个数组的每个对象，分别执行他们的 run 方法：

 复制代码

```
1 public void run(TestResult result) {  
2     for (Test each : fTests) {  
3         test.run(result);  
4     }  
5 }
```

如果这个 test 对象是 TestCase，就执行测试；如果这个 test 对象是一个 TestSuite，那么就会继续调用这个 TestSuite 对象的 run 方法，遍历执行数组的每个 Test 的 run 方法，从而实现了树的递归遍历。

小结

人们对架构师的工作有一种常见的误解，认为架构师做架构设计就可以了，架构师不需要写代码。事实上，架构师如果只是画画架构图，写写设计文档，那么如何保证自己的架构设计能被整个开发团队遵守、落到实处？

架构师应该通过代码落实自己的架构设计，也就是通过开发编程框架，约定软件开发的规范。开发团队依照框架的接口开发程序，最终被框架调用执行。架构师不需要拿着架构图一遍一遍讲软件架构是什么，只需要基于框架写个 Demo，大家就都清楚架构是什么了，自己应该如何做了。

所以每个想成为架构师的程序员都应该学习如何开发框架。

思考题

在 Tomcat 和 JUnit 中，还使用了其他一些设计模式，在哪些地方使用什么设计模式，解决什么问题？你了解吗？

欢迎你在评论区写下你的思考，我会和你一起交流，也欢迎你把这篇文章分享给你的朋友或者同事，一起交流一下。

点击参加 21 天打卡计划 

搞定后端技术基础



扫一扫参与小程序话题



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 16 | 设计模式基础：不会灵活应用设计模式，你就没有掌握面向对象编程

下一篇 加餐 | 软件设计文档示例模板

精选留言 (3)

 写留言



唐二毛

2019-12-30

我有个疑惑，面试给出这样简单的答案，恐怕是过不了关吧？希望老师爆点真正的干货出来！

 1

 3



Zend

2019-12-30

什么是框架

框架式对某一类架构方案可复用设计与实现

Tomcat是框架，它完成了Web应用请求响应的主体流程。

框架应用满足依赖倒置原则

...

展开 



Keep-Moving

2019-12-30

不写代码的架构师不是好司机

展开 ▾

