二

# 32 如何写出安全的Java代码？-极客时间

在上一讲中，我们已经初步接触了 Java 安全，今天我们将一起探讨更多 Java 开发中可能影响到安全的场合。很多安全问题，在特定的上下文，存在着不同的定义，尽管本质是相似或一致的，这是由于 Java 平台自身的特性所带来特有的问题。今天这一讲我将侧重于 Java 开发者的角度谈代码安全，而不是讲广义的安全风险。

今天我要问你的问题是，**如何写出安全的 Java 代码?**

## 典型回答

这个问题可能有点宽泛，我们可以用特定类型的安全风险为例，如拒绝服务（DoS）攻击，分析 Java 开发者需要重点考虑的点。

DoS 是一种常见的网络攻击，有人也称其为“洪水攻击”。最常见的表现是，利用大量机器发送请求，将目标网站的带宽或者其他资源耗尽，导致其无法响应正常用户的请求。

我认为，从 Java 语言的角度，更加需要重视的是程序级别的攻击，也就是利用 Java、JVM 或应用程序的瑕疵，进行低成本的 DoS 攻击，这也是想要写出安全的 Java 代码所必须考虑的。例如：

- 如果使用的是早期的 JDK 和 Applet 等技术，攻击者构建合法但恶劣的程序就相对容易，例如，将其线程优先级设置为最高，做一些看起来无害但空耗资源的事情。幸运的是类似技术已经逐步退出历史舞台，在 JDK 9 以后，相关模块就已经被移除。
- 上一讲中提到的哈希碰撞攻击，就是个典型的例子，对方可以轻易消耗系统有限的 CPU 和线程资源。从这个角度思考，类似加密、解密、图形处理等计算密集型任务，都要防范被恶意滥用，以免攻击者通过直接调用或者间接触发方式，消耗系统资源。
- 利用 Java 构建类似上传文件或者其他接受输入的服务，需要对消耗系统内存或存储的上限有所控制，因为我们不能将系统安全依赖于用户的合理使用。其中特别注意的是涉及解压缩功能时，就需要防范Zip bomb等特定攻击。
- 另外，Java 程序中需要明确释放的资源有很多种，比如文件描述符、数据库连接，甚至是再入锁，任何情况下都应该保证资源释放成功，否则即使平时能够正常运行，也可

能被攻击者利用而耗尽某类资源，这也算是可能的 DoS 攻击来源。

所以可以看出，实现安全的 Java 代码，需要从功能设计到实现细节，都充分考虑可能的安全影响。

## 考点分析

关于今天的问题，以典型的 DoS 攻击作为切入点，将问题聚焦在 Java 开发中，我介绍了 Java 应用设计、实现的注意事项，后面还会介绍更加全面的实践。

其实安全问题实际就是软件的缺陷，软件安全并不存在一劳永逸的秘籍，既离不开设计、架构中的风险分析，也离不开编码、测试等阶段的安全实践手段。对于面试官来说，考察安全问题，除了对特定安全领域知识的考察，更多是要看面试者的 Java 编程基本功和知识的积累。

所以，我会在后面会循序渐进探讨 Java 安全编程，这里面没有什么黑科技，只有规范的开发标准，很多安全问题其实是态度问题，取决于你是否真的认真对待它。

- 我将以一些典型的代码片段为出发点，分析一些非常容易被忽略的安全风险，并介绍安全问题频发的热点场景，如 Java 序列化和反序列化。
- 从软件生命周期的角度，探讨设计、开发、测试、部署等不同阶段，有哪些常见的安全策略或工具。

## 知识扩展

首先，我们一起来看一段不起眼的条件判断代码，这里可能有什么问题吗？

```
// a, b, c都是int类型的数值
if (a + b < c) {
// …
}
```

你可能会纳闷，这是再常见不过的一个条件判断了，能有什么安全隐患？

这里的隐患是数值类型需要防范溢出，否则这不仅仅可能会带来逻辑错误，在特定情况下可能导致严重的安全漏洞。

从语言特性来说，Java 和 JVM 提供了很多基础性的改进，相比于传统的 C、C++ 等语言，对于数组越界等处理要完善的多，原生的避免了缓冲区溢出等攻击方式，提高了软件的

安全性。但这并不代表完全杜绝了问题，Java 程序可能调用本地代码，也就是 JNI 技术，错误的数值可能导致 C/C++ 层面的数据越界等问题，这是很危险的。

所以，上面的条件判断，需要判断其数值范围，例如，写成类似下面结构。

```
if (a < c - b)
```

再来看一个例子，请看下面的一段异常处理代码：

```
try {
// 业务代码
} catch (Exception e) {
throw new RuntimeException(hostname + port + “ doesn’t response”);
}
```

这段代码将敏感信息包含在异常消息中，试想，如果是一个 Web 应用，异常也没有良好的包装起来，很有可能就把内部信息暴露给终端客户。古人曾经告诫我们“言多必失”是很有道理的，虽然其本意不是指软件安全，但尽量少暴露信息，也是保证安全的基本原则之一。即使我们并不认为某个信息有安全风险，我的建议也是如果没有必要，不要暴露出来。

这种暴露还可能通过其他方式发生，比如某著名的编程技术网站，就被曝光过所有用户名和密码。这些信息都是明文存储，传输过程也未必进行加密，类似这种情况，暴露只是个时间早晚的问题。

对于安全标准特别高的系统，甚至可能要求敏感信息被使用后，要立即明确在内存中销毁，以免被探测；或者避免在发生 core dump 时，意外暴露。

第三，Java 提供了序列化等创新的特性，广泛使用在远程调用等方面，但也带来了复杂的安全问题。直到今天，序列化仍然是个安全问题频发的场景。

针对序列化，通常建议：

- 敏感信息不要被序列化！在编码中，建议使用 transient 关键字将其保护起来。
- 反序列化中，建议在 readObject 中实现与对象构件过程相同的安全检查和数据检查。

另外，在 JDK 9 中，Java 引入了过滤器机制，以保证反序列化过程中数据都要经过基本验证才可以使用。其原理是通过黑名单和白名单，限定安全或者不安全的类型，并且你可以进行定制，然后通过环境变量灵活进行配置， 更加具体的使用你可以参考 ObjectInputFilter。

通过前面的介绍，你可能注意到，很多安全问题都是源于非常基本的编程细节，类似 Immutable、封装等设计，都存在着安全性的考虑。从实践的角度，让每个人都了解和掌握

这些原则，有必要但并不太现实，有没有什么工程实践手段，可以帮助我们排查安全隐患呢？

### 开发和测试阶段

在实际开发中，各种功能点五花八门，未必能考虑的全面。我建议没有必要所有都需要自己去从头实现，尽量使用广泛验证过的工具、类库，不管是来自于 JDK 自身，还是 Apache 等第三方组织，都在社区的反馈下持续地完善代码安全。

开发过程中应用代码规约标准，是避免安全问题的有效手段。我特别推荐来自孤尽的《阿里巴巴 Java 开发手册》，以及其配套工具，充分总结了业界在 Java 等领域的实践经验，将规约实践系统性地引入国内的软件开发，可以有效提高代码质量。

当然，凡事都是有代价的，规约会增加一定的开发成本，可能对迭代的节奏产生一定影响，所以对于不同阶段、不同需求的团队，可以根据自己的情况对规约进行适应性的调整。

落实到实际开发流程中，以 OpenJDK 团队为例，我们应用了几个不同角度的实践：

- 在早期设计阶段，就由安全专家组对新特性进行风险评估。
- 开发过程中，尤其是 code review 阶段，应用 OpenJDK 自身定制的代码规范。
- 利用多种静态分析工具如FindBugs、Parfait等，帮助早期发现潜在安全风险，并对相应问题采取零容忍态度，强制要求解决。
- 甚至 OpenJDK 会默认将任何（编译等）警告，都当作错误对待，并体现在 CI 流程中。
- 在代码 check-in 等关键环节，利用 hook 机制去调用规则检查工具，以保证不合规代码不能进入 OpenJDK 代码库。

关于静态分析工具的选择，我们选取的原则是“足够好”。没有什么工具能够发现所有问题，所以在保证功能的前提下，影响更大的是分析效率，换句话说是代码分析的噪音高低。不管分析有多么的完备，如果太多误报，就会导致有用信息被噪音覆盖，也不利于后续其他程序化的处理，反倒不利于排查问题。

以上这些是为了保证 JDK 作为基础平台的苛刻质量要求，在实际产品中，你需要斟酌具体什么程度的要求是合理的。

### 部署阶段

JDK 自身的也是个软件，难免会存在实现瑕疵，我们平时看到 JDK 更新的安全漏洞补丁，其实就是在修补这些漏洞。我最近还注意到，某大厂后台被曝出了使用的 JDK 版本存在序列化相关的漏洞。类似这种情况，大多数都是因为使用的 JDK 是较低版本，算是可以通过

部署解决的问题。

如果是安全敏感型产品，建议关注 JDK 在加解密方面的路线图，同样的标准也应用于其他语言和平台，很多早期认为非常安全的算法，已经被攻破，及时地升级基础软件是安全的必要条件。

攻击和防守是不对称的，只要有一个严重漏洞，对于攻击者就足够了，所以，不能对黑盒形式的部署心存侥幸，这并不能保证系统的安全，攻击者可以利用对软件设计的猜测，结合一系列手段，探测出漏洞。

今天我以 DoS 等典型攻击方式为例，分析了其在 Java 平台上的特定表现，并从更多安全编码的细节帮你体会安全问题的普遍性，最后我介绍了软件开发周期中的安全实践，希望能对你的工作有所帮助。

## 一课一练

关于今天我们讨论的题目你做到心中有数了吗？你在开发中遇到过 Java 特定的安全问题吗？是怎么解决的呢？

请你在留言区写写你对这个问题的思考，我会选出经过认真思考的留言，送给你一份学习奖励礼券，欢迎你与我一起讨论。

别忘了今晚 8 点半我会做客“极客 Live”，和你一起聊聊 Java 面试那些事儿。在“极客时间”App 内点击“极客 Live”即可加入直播，今晚我们不见不散。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。

上一页 下一页