Java服务器端编程安全必读（上） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_Java_E6_9C_8D_E5_8A_A1_c104_145506.htm 一、概述 编写安全的Internet应用并不是一件轻而易举的事情：只要看看各个专业公告板就可以找到连续不断的安全漏洞报告。你如何保证自己的Internet应用不象其他人的应用那样满是漏洞？你如何保证自己的名字不会出现在令人难堪的重大安全事故报道中？如果你使用Java Servlet 、JavaServer Pages（JSP）或者EJB，许多难以解决的问题都已经事先解决。当然，漏洞仍有可能出现。下面我们就来看看这些漏洞是什么，以及为什么Java程序员不必担心部分C和Perl程序员必须面对的问题。 C程序员对安全漏洞应该已经很熟悉，但象OpenBSD之类的工程提供了处理此类问题的安全系统。Java语言处理这类问题的经验要比C少20年，但另一方面，Java作为一种客户端编程语言诞生，客户端对安全的要求比服务器端苛刻得多。它意味着Java的发展有着一个稳固的安全性基础。Java原先的定位目标是浏览器。然而，浏览器本身所带的Java虚拟机虽然很不错，但却并不完美。Sun的《Chronology of security-related bugs and issues》总结了运行时环境的漏洞发现历史。我们知道，当Java用作服务器端编程语言时，这些漏洞不可能被用作攻击手段。但即使Java作为客户端编程语言，重大安全问题的数量也从1996年的6个（其中3个是相当严重的问题）降低到2000年的1个。不过，这种安全性的相对提高并不意味着Java作为服务器端编程语言已经绝对安全，它只意味着攻击者能够使用的攻击手段越来越受到限制。那么，究竟有哪些地方容易受到攻击，其他编

程语言又是如何面对类似问题的呢？ 二、缓存溢出 在C程序中，缓存溢出是最常见的安全隐患。缓存溢出在用户输入超过已分配内存空间（专供用户输入使用）时出现。缓存溢出可能成为导致应用被覆盖的关键因素。C程序很容易出现缓存溢出，但Java程序几乎不可能出现缓存溢出。从输入流读取输入数据的C代码通常如下所示： char buffer[1000]. int len = read(buffer). 由于缓存的大小在读入数据之前确定，系统要检查为输入保留的缓存是否足够是很困难的。缓存溢出使得用户能够覆盖程序数据结构的关键部分，从而带来了安全上的隐患。有经验的攻击者能够利用这一点直接把代码和数据插入到正在运行的程序。 在Java中，我们一般用字符串而不是字符数组保存用户输入。与前面C代码等价的Java代码如下所示： String buffer = in.readLine(). 在这里，“ 缓存 ”的大小总是和输入内容的大小完全一致。由于Java字符串在创建之后不能改变，缓存溢出也就不可能出现。退一步说，即使用字符数组替代字符串作为缓存，Java也不象C那样容易产生可被攻击者利用的安全漏洞。例如，下面的Java代码将产生溢出： char[] bad = new char[6]. bad[7] = 50. 这段代码总是抛出一个java.lang.ArrayOutOfBoundsException异常，而该异常可以由程序自行捕获： try { har[] bad = new char[6]. bad[7] = 50. } catch (ArrayOutOfBoundsException ex) { ... } 这种处理过程永远不会导致不可预料的行为。无论用什么方法溢出一个数组，我们总是得到ArrayOutOfBoundsException异常，而Java运行时底层环境却能够保护自身免受任何侵害。一般而言，用Java字符串类型处理字符串时，我们无需担心字符串的ArrayOutOfBoundsExceptions异常，因此它是一种较为理想

的选择。Java编程模式从根本上改变了用户输入的处理方法，避免了输入缓存溢出，从而使得Java程序员摆脱了最危险的编程漏洞。 三、竞争状态 竞争状态即Race Condition，它是第二类最常见的应用安全漏洞。在创建（更改）资源到修改资源以禁止对资源访问的临界时刻，如果某个进程被允许访问资源，此时就会出现竞争状态。这里的关键问题在于：如果一个任务由两个必不可少的步骤构成，不管你多么想要让这两个步骤一个紧接着另一个执行，操作系统并不保证这一点。例如，在数据库中，事务机制使得两个独立的事件“ 原子化 ”。换言之，一个进程创建文件，然后把这个文件的权限改成禁止常规访问；此同时，另外一个没有特权的进程可以处理该文件，欺骗有特权的进程错误地修改文件，或者在权限设置完毕之后仍继续对原文件进行访问。 一般地，在标准Unix和NT环境下，一些高优先级的进程能够把自己插入到任务的多个步骤之间，但这样的进程在Java服务器上是不存在的；同时，用纯Java编写的程序也不可能修改文件的许可权限。因此，大多数由文件访问导致的竞争状态在Java中不会出现，但这并不意味着Java完全地摆脱了这个问题，只不过是问题转到了虚拟机上。我们来看看其他各种开发平台如何处理这个问题。在Unix中，我们必须确保默认文件创建模式是安全的，比如在服务器启动之前执行“ umask 200 ”这个命令。有关umask的更多信息，请在Unix系统的命令行上执行“ man umask ”查看umask的man文档。在NT环境中，我们必须操作ACL（访问控制表，Access Control List）的安全标记，保护要在它下面创建文件的目录。NT的新文件一般从它的父目录继承访问许可。请参见 NT文档了解更多信息。 100Test 下载