HZ BOOKS
华章科技
PEARSON

资深专家撰写，Java之父James A. Gosling鼎力推荐，Java领域关于Java安全编码最权威、最全面、最详细的著作

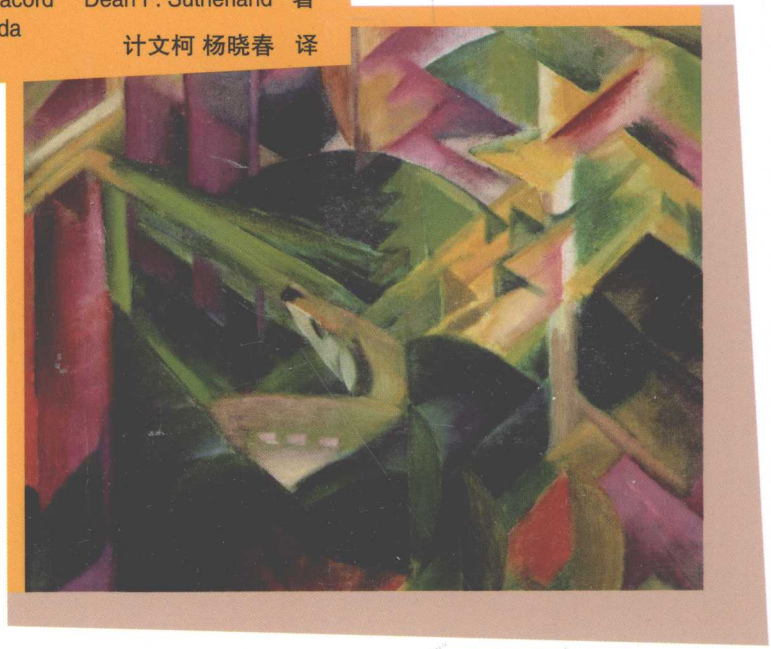
不仅从语言角度系统而详细地阐述Java安全编码的要素、标准、规范和最佳实践，而且从架构设计的角度分析Java API存在的设计缺陷和可能存在的安全风险，以及应对策略

华章程序员书库

The CERT Oracle Secure Coding Standard for Java

Java安全编码标准

Fred Long　Dhruv Mohindra
Robert C. Seacord　Dean F. Sutherland　著
David Svoboda
计文柯 杨晓春　译



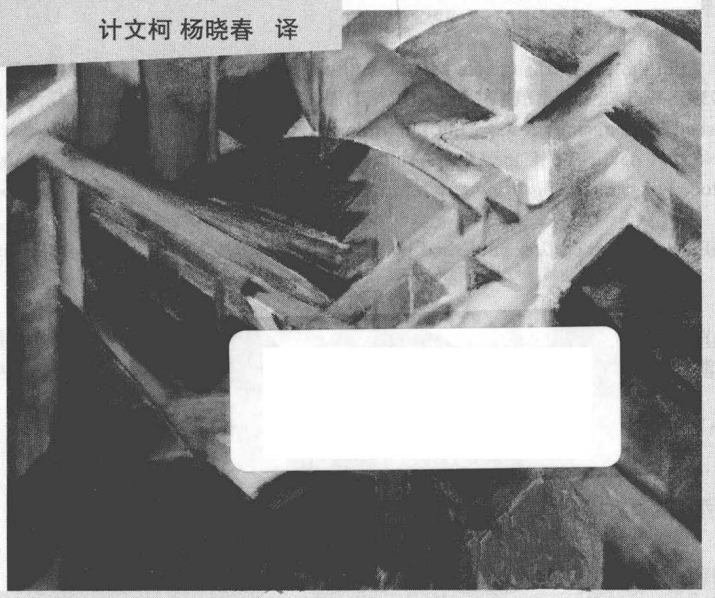
机械工业出版社
China Machine Press

华章程序员书库

The CERT Oracle Secure Coding Standard for Java

Java安全编码标准

Fred Long　Dhruv Mohindra
Robert C. Seacord　Dean F. Sutherland　著
David Svoboda
计文柯 杨晓春　译



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java安全编码标准 / 朗（Long，F.）等著；计文柯，杨晓春译. —北京：机械工业出版社，2013.6
（华章程序员书库）
书名原文：The CERT Oracle Secure Coding Standard for Java

ISBN 978-7-111-42818-3

I. J… II. ① 朗… ② 计… ③ 杨… III. Java语言–程序设计 IV. TP312

中国版本图书馆CIP 数据核字（2013）第120651号

本书版权登记号：图字：01-2011-7733

本书是Java安全编码领域最权威、最全面、最详细的著作，Java之父James A. Gosling推荐。不仅从语言角度系统而详细地阐述Java安全编码的要素、标准、规范和最佳实践，而且从架构设计的角度分析Java API存在的设计缺陷和可能存在的安全风险，以及应对策略和措施。读者可以将本书作为Java安全方面的工具书，根据自己的需要，找到自己感兴趣的规则进行阅读和理解，或者在实际开发中遇到安全问题时，根据书中列出的大致分类对规则进行索引和阅读，也可以通读全书的所有规则，系统地了解Java安全规则，增强对Java安全特性、语言使用、运行环境特性的理解。本书能指导Java软件工程师设计出高质量的、安全的、可靠的、强大的、有弹性的、易用的和可维护性高的软件系统。

本书内容非常全面，包括基于Java SE 6平台的一系列应用于Java语言和类库的安全编码规则，并且对这一系列规则进行了分类，包括输入数据验证、声明和初始化、表达式、数值类型和操作、面向对象、方法使用、异常处理、可见性和原子性、锁、线程、输入输出、序列化、平台安全特性、Java运行环境等重要方面，对每一个方面所涉及的安全编码要素、规范和标准进行了详细阐释。

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：秦 健
藁城市京瑞印刷有限公司印刷
2013年6月第1版第1次印刷
186mm×240mm · 33.25印张
标准书号：ISBN 978-7-111-42818-3
定 价：99.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

译 者 序

随着软件技术的发展，特别是互联网的发展，在很多应用成为互联网应用之后，软件系统本身变成了一个开放的系统，其中也包括潜在用户数据对系统外的开放，不管是开发人员还是软件用户，除了关注软件功能实现本身，对软件安全性的关注也越来越多。

在软件开发中，由于Java语言具有开放性和可移植性，它逐渐成为构建应用的主要开发环境和语言之一，而对安全性的关注也开始成为一个热点话题。正如在本书序中，Java语言之父James Golsing说的："在Java的世界里，安全性并没有被视为一个附加功能。尽管这是一种普遍的思维方式，但如果不带着安全性去考虑问题，还是会陷入麻烦之中。"从Golsing的观点中，我们可以看出，对代码安全性的设计和考虑是需要系统学习的，因为这是一种对思维方式的反省。本书就是为了帮助开发人员加强对Java语言安全性的认识而推出的技术专著，书中结合Java应用的开发和Java语言的使用，系统地阐述了使用Java语言时的相关安全规则。本书内容非常全面，包括基于Java标准版6.0平台（Java SE 6）环境中一系列应用于Java编程语言和类库的安全编码规则，并且对这一系列规则进行了分类，包括输入数据验证、声明和初始化、表达式、数值类型和操作、面向对象、方法使用、异常处理、可见性和原子性、锁、线程、输入输出、序列化、平台安全特性、Java运行环境等各个基本的分类部分。读者可以将本书作为Java安全方面的工具书，根据自己的需要，找到感兴趣的规则进行阅读和学习，或者在开发工作和软件设计中，当遇到安全问题时，根据书中列出的大致分类对规则进行检索和阅读，另外，也可以通读全书的所有规则，系统地了解Java安全规则，增强对Java安全特性、语言使用、运行环境特性的理解。在国内的技术书籍中，像本书这样系统阐述Java安全的书籍并不多见，因此本书值得对Java安全感兴趣的读者系统阅读。

译者力争以通俗通畅的中文再现原著的经典知识，希望尽自己的努力，帮助读者学习这部经典著作。但由于译者水平有限，不管是在Java安全的专业领域知识，还是在对原文的理解上，甚至在中文遣词造句的表达上，在翻译的作品中，肯定会在诸多方面存在疏漏之处，还请读者不吝赐教。您的意见、建议将帮助我们改善本书的质量。欢迎将关于本书的任何问题和看法发邮件到jiwenke@gmail.com，与我们交流本书相关的信息。再次感谢！

序

James Gosling

近十年来，在计算机系统中考虑安全性已经是一个严肃的问题。过去十年的网络的爆炸性增长和我们对计算机网络互联的依赖，使安全问题提升到了一个新的层次。在 Java 的最初设计中，对安全性的处理是一个关键部分。从那时起，所有的各式各样的标准类库、框架和容器都对安全性问题有所考虑。在 Java 的世界里，安全性并没有被视为一个附加功能。尽管这是一种普遍的思维方式，但如果不带着安全性去考虑问题，还是会陷入麻烦之中。

并不仅仅因为有了基础设施的帮助，我们就可以想当然地认为安全性已经被自动考虑了。近些年来，业界发展出了一系列的安全标准和最佳实践。这本书就是一本这些实践的经验总结。它们并不是理论研究论文或者说产品市场营销的噱头。它们都是在重项目中经过严格检测的可以在企业级规模应用的产物。

前　言

在 Java 编程语言中，关键的安全编码要素是采用良好的文档和强制的编码规范。本书提供了在 Java 语言中的一系列安全编码规则。这些规则的目标是消除不安全的编码实践，因为这些不安全的因素会导致可利用的漏洞。如果应用这些安全编码标准，可以帮助设计出安全的、可靠的、健壮的、有弹性的、可用性和可维护性高的高质量系统，并且这些安全编码规范还可作为评估源代码质量特性的一个指标（不管使用的是手动的还是自动化的过程）。

对于使用 Java 编程语言开发的软件系统，这个编码规范具有广泛的影响。

本书范围

本书主要关注 Java 标准版 6.0 平台（Java SE 6）环境，其中包含一系列应用于 Java 编程语言和类库的安全编码标准。在《Java 语言规范（第 3 版）》（The Java Language Specification, 3rd edition, JLS 2005）对 Java 编程语言的行为进行了描述，该规范主要是开发本标准的主要参考资料。本标准同样涉及 Java SE 7 平台中的新特性。这些新特性主要为解决存在于 Java SE 6 和 Java SE 7 平台的安全问题提供了替代性的兼容解决方案。

C 和 C++ 语言允许一些不确定、未指定或者在实现时才确定的行为，当程序员错误假设使用的 API 和开发语言的内在行为时，它们都会导致安全漏洞。《Java 语言规范》进一步规范了标准化需求，这是因为 Java 被设计成一个“一次开发，到处运行”的语言。甚至 Java 虚拟机（JVM）或者 Java 编译器的一些行为可以由实现者自行决定。针对这些语言的特殊性，本标准也提出了安全的编码指导，以避免这些特殊性带来的问题。

仅仅关注于语言本身并不能编写出安全的软件。在 Java 应用编程接口（API）中存在的设计缺陷，在某些时候会导致这些 API 过时。在另外一些时候，这些 API 或者相关的文档还可能会被开发社区错误地解读。本标准指出了这些有问题的 API 并且强调了它们的正确使用方法，同样也包括那些广泛使用的错误设计模式（反模式）和使用习惯的例子。

Java 语言的核心和扩展 API 以及 JVM 提供的安全特性包括安全管理器、访问控制器、加密、自动内存管理、强类型检查、字节码验证等。虽然这些安全特性为大部分的应用提供了足够的安全性保证，但是对这些特性的正确使用也是非常重要的。本标准突出与设计安全架构相关的隐患和注意事项，并强调其正确的实施。坚持这个标准可以保障受信任程序的保密性、完整性和可用性（Confidentiality，Integrity，and Availability，CIA），并且有助于消除安全漏洞，这些安全漏洞会导致拒绝服务攻击、time-of-check-to-time-of-use 攻击、信息泄露、差错计算以及权限升级这些问题。

若软件遵循这些标准，则其使用者有能力定义细粒度的安全策略，并且在不受信的系统中执行受信的移动代码，或者在受信的系统中执行非受信的移动代码。

本书包含的类库

本书讨论的安全问题主要应用于 lang 和 util 类库，同时涉及 Collections、并发包、Logging、Management、反射、正则表达式、Zip、I/O、JMX、JNI、Math、Serialization 以及 XML JAXP 等类库。本标准不涉及已经发现的缺陷，不管这些缺陷已经得到修复还是因为设计本身就缺少安全性的考虑。本标准也包括功能缺陷，但只有这个缺陷发生频率高，造成相当大的安全问题，或它影响了基于核心平台的所有 Java 技术，它才会被考虑进来。本标准不仅涉及核心 API 的安全标准，而且包括一系列重要的安全考虑，涉及标准的扩展 API（javax 包）。

本书不会涉及的问题

本标准不会涉及以下问题：

- **设计和架构**。本标准假设产品的设计和架构是安全的，也就是说，产品有足够的设计水平，而不致产生危及其安全的漏洞。
- **内容**。本标准不关注那些在某种 Java 平台上出现的特殊问题，而关注广泛出现在所有平台上的问题。举例来说，如果一条规则只单独适用于 Java 微型版（Micro Edition, ME）或 Java 企业版（Enterprise Edition, EE），而不适用于 Java SE 版，通常不会包括在本标准内。在 Java SE 版中，那些处理 UI 或者在 Web 接口中提供的特性，比如声音、图形渲染、用户登录控制、会话管理、认证和授权等，不包含在本标准内。然而，这并不妨碍本标准讨论在网络中运行的 Java 系统由于不正确的用户输入验证而带来的风险、引入的问题和适当的缓解策略。
- **编码风格**。编码风格的问题是主观的，事实已给证明，我们不可能在制定一个适当的样式规则问题上达成共识。因而，本书只推荐了一些用户定义样式的规则，并且一致性地应用这些规则而已，并不强制使用某一种特定的编码规则。如果希望始终如一应用编码风格，最简单的方法是使用代码格式化工具。许多集成开发环境（IDE）提供这样的功能。
- **工具**。美国软件工程研究所（SEI）作为联邦政府资助的研发机构，不会推荐任何特定的厂商和工具，用来强制采纳标准。本书的用户可以自由选择工具，我们鼓励各个厂商为执行这些规则提供相应的工具。
- **有争议的规则**。一般情况下，本书会尽量避免列入那些缺乏广泛共识的有争议规则。

本书读者对象

本书主要是为 Java 语言开发人员服务的。本标准重点关注 Java 标准版 6.0 平台，同时可以为使用 Java ME 和 Java EE 或者其他 Java 语言版本的开发者提供参考。

尽管主要考虑系统的安全性，但是本标准对达到其他质量标准方面也是有参考价值的，比如可靠性、可用性、健壮性、可扩展性、可用性和可维护性等。

本标准还可以用于：

- 分析工具的开发人员，他们希望能够诊断出那些不安全和不合规则的 Java 程序。
- 软件开发经理、软件采购商或者其他希望建立专有的安全编码规范的软件开发及采购人员。
- 讲授 Java 安全编码标准的教师，可以使用本标准作为主要的或者参考的软件安全课程教材。

本标准中的一些规则可以扩展为某些组织的特定规则。但这种扩展应该与已有的规则一致，

并且和本标准兼容。

关于更好地使用这些安全编码标准，可以开设培训课程。当通过培训课程的考试之后，经过培训的人员可以认证为安全编码方面的专业人士。

内容和组织

本书包括一个介绍性的第 1 章和其他 17 章（每一章包含了特定领域的规则）。每一章的规则以一个列表的形式出现，并且包含对这些规则的风险评估摘要。最后是一个术语表和参考资源。可以按顺序阅读，先从本前言开始，然后是概述部分。对于规则部分的章节没有特定的阅读顺序，它们也可以作为参考资料进行查阅。每章的规则都是组织松散的，在一般情况下，可以以任意顺序阅读。

阐述规则时采用了一致的结构。每一个规则都有一个包含在标题中的唯一标识符，在规则的标题和介绍性文字部分阐述符合规则的要求。通常，在后面会有一个或一组不符合规则的代码示例，并且会给出相应的符合规则的方案。同时给出了每个规则的风险评估和具体参考书目。有的规则也列出了相关的漏洞和准则，它们来自于：

- 《CERT C 语言安全编码标准》（The CERT C Secure Coding Standard）[Seacord 2008]
- 《CERT C++ 语言安全编码标准》（The CERT C++ Secure Coding Standard）[CERT 2011]
- ISO/IEC TR 24772。信息技术-编程语言-通过语言选择和使用以避免在程序语言中出现安全漏洞的指南 [ISO/IEC TR 24772:2010]
- MITRE CWE [MITRE 2011]
- 《Java 语言安全编码规则第 3 版》[SCG 2009]
- 《Java 编程风格要素》（The Elements of Java Style）[Rogue 2000]

标识符

每个规则都包含一个唯一标识符，这个标识符包括 3 个部分：

- 3 个字母的助记符，代表标准的一部分，用来编组类似的规则，方便找到。
- 两位数值范围为 00 ～ 99 的数字，以确保每个规则都有一个唯一标识符。
- 字母 J 表明这是 Java 语言的规则，之所以包含 J，是因为可以防止出现类似 CERT 的安全编码标准也适用其他语言的情况，从而避免产生歧义。

标识符可以被静态分析工具使用，用于诊断信息的参考规则，也可以直接使用规则标题的简称。

系统质量

在考虑选择和应用编码标准的时候，安全性（security）是一个必须考虑的系统属性。另外，我们感兴趣的属性还包括无害性（safety）、可移植性（portability）、可靠性（reliability）、可用性（availability）、可维护性（maintainability）、可读性（readability）和性能（performance）等。

许多系统属性会以有趣的方式相关联。比如，可读性是可维护性的一个属性；它们对于限制在维护阶段引入缺陷都很重要，而这些缺陷往往会导致安全问题或可靠性问题。另外，可读性会为质量保证人员的代码评审提供帮助。可靠性和可用性需要正确的资源管理，而这种资源管理对提高系统的无害性和安全性有很大的帮助作用。诸如性能和安全性这样的系统特性却是相互冲突

的，它们需要有所取舍和相互平衡。

安全编码标准的目的是提高软件的安全性。然而，基于安全和其他系统属性之间的关系，如果其他系统属性也存在对安全性的显著影响，本标准也会包括对它们进行处理的要求和建议。

优先级和层级

每一个规则都分配有优先级。优先级分配使用的度量方式基于 FMECA 模式（Failure Mode, Effects, and Criticality Analysis, FMECA）[IEC 60812]。为每个规则指定 1 ～ 3 的 3 个值：

- 严重性——如果忽略该规则，后果的严重程度：
  1 = 低（拒绝访问攻击，非正常终止）
  2 = 中（破坏数据一致性，非意愿性的数据泄露）
  3 = 高（运行非指定代码，权限升级）
- 可能性——如果违反该规则，引入缺陷而导致系统出现可被利用漏洞的可能性有多大：
  1 = 不可能
  2 = 可能
  3 = 很有可能
- 整改代价——如果对现有代码进行整改，使其满足规则所需要付出的代价：
  1 = 高（手动发现及修改）
  2 = 中（自动发现及手动修改）
  3 = 低（自动发现及修改）

对每一条规则，我们把这 3 个值相乘。该结果提供了一个度量，通过这个度量可以判断应用该规则的优先程度。这个优先程度范围为 1 ～ 27。优先级在 1 ～ 4 内的规则是 3 级规则（L3），6 ～ 9 是第 2 级（L2），12 ～ 27 是 1 级规则（L1）。因此，有可能制定一个标准，实施该标准使得在一个级别的所有规则都需要得到满足，这个级别可以是 1 级、2 级或 3 级，如图 1 所示。

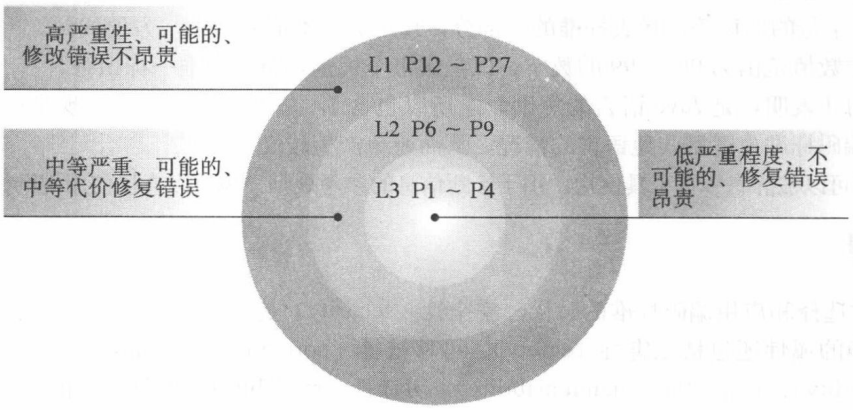


图 1　层级和优先等级

设计该度量的主要目的是表示为修正项目而付出的努力，并不表示为了实现该标准而需要付出的额外精力。

符合性测试

软件系统可以通过验证，从而符合本书中规定的规则。

规范性文字与非规范性文字

在本书中，属于编码标准的部分被视为规范性的内容，其他部分则作为建议给出。在这些规则中，规范性的描述是符合标准的要求。这些规范性描述使用强制式的语言表述，如“必须”、“应当”、“要求”。尽管对于一些规则而言，实现自动化分析并不具备可能性且并非必要，但是每一个规则的规范性部分必须是可分析的。

非规范性部分描述规则的最佳实践或有益的建议。非规范性描述并不设立符合性要求，它常常使用动词“应该”或者短语如“推荐”、“好的做法”。规则的非规范性部分可能不太适合做自动化检测，主要是因为如果在现有代码中做这种检测，可能存在过多误报。当分析新的代码时（新的代码指按照该编码规范开发的代码），自动化检测也许能够发挥作用。

本书中所有的规则都有一个非规范性部分。这个部分仅应用于以下场合：

- 遵循广为人知的最佳实践。
- 如果这条规则描述的方式被广泛采用，对以下两种情况都不产生影响：一是不会违反规则的规范性部分，二是该方法在应用到代码时，是无害的，虽然那些代码并不适用于那些规范性部分。

整个非规范性指导不包含在本编码规则内。然而，本书的作者正在计划发布这些非规范性指导。

自动化分析

为了确保源代码符合本安全编码标准，需要做必要的检查。检查的最有效方法是使用一个或多个分析工具（分析仪）。当一个规则不能由工具检查时，则必须采用人工审核方式。

本书中的许多规则会提示是否已经存在分析工具。这些分析工具可以诊断是否违反规则，甚至可以提示该规则是否可以做自动化检测。这部分的信息可能会有些变化，因为现有的检测工具正在发展中，同时也正开发出新的检测工具。

当选择一个源代码分析工具的时候，理想的状态是，这个工具能够适用于本规范尽可能多的规则。并不是所有的规则都需要自动化检测工具，还是有一些规则需要通过手动检测才能完成的。

完整性和可信性

对于规则来说，一个分析工具在最大程度上应该是完整的，并且是可信的。如果在检测中不会给出假阴性结果，也就说，它能够在一个程序中检测出违反该规则的所有问题，那么称其为可信的。如果不会给出假阳性结果或者误报，则认为该工具的检测是完整的。对于特定的规则，它可能产生的结果如下所示：

表 1　可信性和完整性

		假阳性	
		Y	N
假阴性	N	假阳性可信	可信并完整
	Y	假阳性不可信	不可信

如果工具有太多的假阳性结果，那么它会浪费开发人员的时间，让他们失去对结果的兴趣，结果是无法得到有用的价值，找到真正的缺陷，因为这些缺陷都淹没在噪声当中了。如果工具有太多假阴性结果，那么会错过许多缺陷，而形成一种虚假的安全感。在实践中，我们需要在这两种情况之间找到平衡点。

在减少假阴性和假阳性上有许多权衡。很显然，最好能够两者都减少，这在一定程度上可以通过许多技巧和算法来帮助实现。

分析应该是一个可信的过程，这意味着我们可以依赖于分析工具的输出结果。因此，开发人员必须保证存在这种信任。在理想情况下，这可以通过工具供应商运行认可的测试来完成。但也存在这种情况，就是若没有正式的验证方案，使用验证套件测试分析工具也是可能的。

CERT 源代码分析实验室

CERT 创建的源代码分析实验室（Source Code Analysis Laboratory，SCALe），为软件系统提供了适用于 CERT 安全编码标准的一致性测试，这种测试包括用于 Java 的 CERT Oracle 安全编码标准。

SCALe 使用多种分析手段评估用户的源代码，包括静态分析工具、动态分析工具、模糊测试等。CERT 会向开发人员报告违反安全编码规范的地方。开发人员可以修复这些缺陷，然后再提交软件进行第二次评估。

在开发人员解决 SCALe 测试出的问题之后，SCALe 团队会决定这个版本的软件是否符合 CERT 标准，并且给开发人员颁发证书，并将该软件系统列入通过该标准的列表。

一个成功的一致性测试表明，SCALe 分析并不能检测出所有违反 CERT 标准中定义的规则。成功的一致性测试也无法保证不会违反这些规则，并且也不能保证这些整个软件永远是安全的。SCALe 也不能够测试那些未知的安全漏洞，不能找出概要设计和架构设计的问题，以及代码的运行环境或者可移植性的问题。通过测试的软件仍然可能是不安全的，例如，如果软件实现的是一个不安全的架构和设计。

在本书的规则讨论中，包括了规则的特例以及这些特例产生的情况。当开发人员从一条该规则中引用这些特例时，他们必须在代码中标识出相关的特例信息。至少需要通过注释的方式在注释中标出该特例，如下所示：

```
// MET12-EX0 applies here
```

作者目前正在开发一套 Java 注释，它将允许程序员以可读的和静态分析工具可以访问的方式标识这种特例。通过一致性测试，可以确定异常是否适用于其他情况，这是由 SCALe 的分析师来完成的。

第三方类库

静态分析工具，如 FindBugs 这个 Java 字节码分析工具，经常可以在第三方类库和自定义代

码中发现违反安全编码标准的行为。出现在第三方类库中违反安全规则的处理方式是和自定义代码一样的。

遗憾的是，开发人员并不是总能修改第三方类库代码或者说不能说服供应商修改代码。在这种情况下，系统就不能通过一致性测试，除非解决这些问题（将替换该类库，或者开发一个自己的类库）或者将这些问题以文档形式记录下来。对第三方类库中出现的这些问题的处理方式和自定义代码的处理方式是一样的，也就是说，开发人员必须自己能够证明虽违反这些规则中但不会导致安全漏洞。然而它们的代价是各不相同的，对于自定义代码，修改问题可能更经济，而对于第三方类库，文档化问题可能更容易。

一致性测试流程

对于每一个编码标准，源代码分为证实不符、符合和完全符合这几种情况，它们适用于本书中的每一条规则。

- **证实不符**。如果出现一条或多条违例，并且不允许出现偏差，那么称为证实不符。
- **符合**。如果没有发现一条违例，则称为符合。
- **完全符合**。如果验证满足规则的所有情况，那么认为该代码完全符合。

偏差规则

严格遵守所有的规则是不可能的，因此，定义违反特定规则的偏差是必要的。偏差应用在下面这个场合，在这个场合中，一个真阳性的问题作为违例提出，但并不能就此判断代码就是不安全的。这可能是软件的设计和架构特性导致的结果，或者存在这个违例，安全编码规范并没有考虑到这种情况。在这种情况下，定义允许的偏差可以防止标准过度严格。但偏差不能用于解释系统性能、可用性和其他不安全性方面的问题。成功通过一致性测试的一套软件系统，必须不存在编码错误导致的已知漏洞。

偏差要求由主任评估师进行评估，如果开发人员可以提供足够的证据，表示该偏差不会引入安全漏洞，那么可以接纳这个偏差请求。偏差不应该经常使用，因为修复代码缺陷总比证明代码没有缺陷要容易，并且不会导致系统安全漏洞。

一旦已完成评估过程，会有报告详细说明代码是否符合或者不符合安全编码标准的相应规则，并会提供给开发人员。

CERT SCALe 认证

被 CERT 认可符合安全编码规范的开发组织可以在其网站上使用如图 2 所示的标识。该标识必须指定通过一致性测试的软件，该标识不适用于未经测试的软件、公司和组织。

图 2　CERT SCALe 标识

除了满足以下条件的软件的补丁，对软件进行任何修改之后，都需要进行一致性设计。直至重新测试该软件，并且认定满足一致性要求之后，这个打过补丁的软件才能使用 CERT SCALe 标识。

满足以下 3 个标准的软件补丁不违反一致性设计：

- 该补丁是在代码中修复安全漏洞所必需的，并且是软件维护所必需的。
- 该补丁不引入新的功能特性。
- 该补丁不引入违反软件之前通过的那些安全编码规范规则的问题。

使用CERT SCALe标识之后，可以列入软件组织列表，与卡耐基-梅隆大学签订服务协议，软件会被认定为符合CERT标准。更多信息请联系：securecoding@cert.org。

致　谢

向为本书成功付梓而付出努力的每个人表示感谢。

贡献者

Siddarth Adukia、Lokesh Agarwal、Ron Bandes、Scott Bennett、Kalpana Chatnani、Steve Christey、Jose Sandoval Chaverri、Tim Halloran、Thomas Hawtin、Fei He、Ryan Hofler、Sam Kaplan、Georgios Katsis、Lothar Kimmeringer、Bastian Marquis、Michael Kross、Masaki Kubo、Christopher Leonavicius、Bocong Liu、Efstathios Mertikas、Aniket Mokashi、David Neville、Todd Nowacki、Vishal Patel、Jonathan Paulson、Justin Pincar、Michael Rosenman、Brendan Saulsbury、Eric Schwelm、Tamir Sen、Philip Shirey、Jagadish Shrinivasavadhani、Robin Steiger、Yozo Toda、Kazuya Togashi、John Truelove、Theti Tsiampali、Tim Wilson 和 Weam Abu Zaki。

评审者

Daniel Bögner、James Baldo Jr.、Hans Boehm、Joseph Bowbeer、Mark Davis、Sven Dietrich、Will Dormann、Chad R. Dougherty、Holger Ebel、Paul Evans、Hari Gopal、Klaus Havelund、David Holmes、Bart Jacobs、Sami Koivu、Niklas Matthies、Bill Michell、Philip Miller、Nick Morrott、Attila Mravik、Tim Peierls、Kirk Sayre、Thomas Scanlon、Steve Scholnick、Alex Snaps、David Warren、Ramon Waspitz 和 Kenneth A. Williams。

编辑

Pamela Curtis、Shannon Haas、Carol Lallier、Tracey Tamules、Melanie Thompson、Paul Ruggerio 和 Pennie Walters。

Addison-Wesley 出版社

Kim Boedigheimer、John Fuller、Stephane Nakib、Peter Gordon、Chuti Prasertsith 和 Elizabeth Ryan。

特别感谢

Archie Andrews、David Biber、Kim Boedigheimer、Peter Gordon、Frances Ho、Joe Jarzombek、Jason McNatt、Stephane Nakib、Rich Pethia 和 Elizabeth Ryan。

目　录