HZ BOOKS
华章科技

基于最新OSGi R5.0规范撰写，继《深入理解Java虚拟机》后的又一实力之作

全面解读OSGi规范，深刻揭示OSGi原理，详细讲解OSGi服务，系统介绍Equinox框架的用法，并通过源代码分析其工作机制，包含大量可操作性极强的解决方案和最佳实践

华章原创精品

深入理解OSGi

Equinox原理、应用与最佳实践

Understanding the OSGi Principles, Using and Best Practices

周志明 谢小明 著

机械工业出版社
China Machine Press

华章原创精品

深入理解OSGi

Equinox原理、应用与最佳实践

Understanding the OSGi
Principles, Using and Best Practices

周志明 谢小明 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深入理解 OSGi：Equinox 原理、应用与最佳实践 / 周志明，谢小明著 . —北京：机械工业出版社，2013.1

ISBN 978-7-111-40887-1

I. 深… II. ① 周… ② 谢… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2012）第 302863 号

本书是原创 Java 技术图书领域继《深入理解 Java 虚拟机》后的又一实力之作，也是全球首本基于最新 OSGi R5.0 规范的著作。理论方面，既全面解读了 OSGi 规范，深刻揭示了 OSGi 原理，详细讲解了 OSGi 服务，又系统地介绍了 Equinox 框架的使用方法，并通过源码分析了该框架的工作机制；实践方面，不仅包含一些典型的案例，还总结了大量的最佳实践，极具实践指导意义。

全书共 14 章，分 4 个部分。第一部分（第 1 章）：走近 OSGi，主要介绍了什么是 OSGi 以及为什么要使用 OSGi。第二部分（第 2 ～ 4 章）：OSGi 规范与原理，对最新的 OSGi R5.0 中的核心规范进行了全面的解读，首先讲解了 OSGi 模块的建立、描述、依赖关系的处理，然后讲解了 Bundle 的启动原理和调度管理，最后讲解了与本地及远程服务相关的内容。第三部分：OSGi 服务与 Equinox 应用实践（第 5 ～ 11 章），不仅详细讲解了 OSGi 服务纲要规范和企业级规范中最常用的几个子规范和服务的技术细节，还通过一个基于 Equinox 的 BBS 案例演示了 Equinox 的使用方法，最重要的是还通过源码分析了 Equinox 关键功能的实现机制和原理。第四部分：最佳实践（第 12 ～ 14 章），总结了大量关于 OSGi 的最佳实践，包括从 Bundle 如何命名、模块划分、依赖关系处理到保持 OSGi 动态性、管理程序启动顺序、使用 API 基线管理模块版本等各方面的实践技巧，此外还介绍了 Spring DM 的原理以及如何在 OSGi 环节中进行程序测试。

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：孙海亮

北京市荣盛彩色印刷有限公司印刷

2013 年 2 月第 1 版第 1 次印刷

186mm × 240mm • 27 印张

标准书号：ISBN 978-7-111-40887-1

定 价：79.00 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

目　录

前言

第一部分　走近 OSGi

第 1 章　Java 模块化之路 / 2

1.1　什么是 OSGi / 2
1.1.1　OSGi 规范的演进 / 4
1.1.2　Java 模块化规范之争 / 7
1.2　为什么使用 OSGi / 11
1.2.1　OSGi 能让软件开发变得更容易吗 / 12
1.2.2　OSGi 能让系统变得更稳定吗 / 13
1.2.3　OSGi 能让系统运行得更快吗 / 14
1.2.4　OSGi 能支撑企业级开发吗 / 15
1.3　本章小结 / 16

第二部分　OSGi 规范与原理

第 2 章　模块层规范与原理 / 18

2.1　OSGi 规范概要 / 18
2.2　Bundle / 20
2.3　描述元数据 / 21
2.3.1　预定义标记 / 21

2.3.2 使用可视化工具 / 27
2.4 Bundle 的组织与依赖 / 31
2.4.1 导出和导入 Package / 31
2.4.2 约束规则与示例 / 38
2.4.3 校验 Bundle 有效性 / 44
2.5 OSGi 的类加载架构 / 45
2.5.1 父类加载器 / 46
2.5.2 Bundle 类加载器 / 47
2.5.3 其他类加载器 / 49
2.5.4 类加载顺序 / 50
2.6 定义执行环境 / 51
2.7 本地化 / 54
2.8 本章小结 / 55

第 3 章 生命周期层规范与原理 / 56

3.1 Bundle 标识 / 56
3.2 Bundle 状态及转换 / 57
3.2.1 安装过程 / 59
3.2.2 解析过程 / 61
3.2.3 启动过程 / 62
3.2.4 更新过程 / 63
3.2.5 停止过程 / 64
3.2.6 卸载过程 / 65
3.3 启动级别 / 65
3.3.1 设置启动级别 / 66
3.3.2 调整活动启动级别 / 67
3.4 事件监听 / 68
3.4.1 事件类型 / 69
3.4.2 事件分派 / 70
3.5 系统 Bundle / 71
3.6 Bundle 上下文 / 72
3.7 本章小结 / 73

第 4 章　服务层规范与原理 / 74

4.1　服务 / 74
4.2　OSGi 服务示例 / 75
4.3　服务属性 / 80
　4.3.1　属性过滤器 / 82
　4.3.2　预定义属性 / 83
　4.3.3　修改属性 / 84
4.4　服务工厂 / 85
4.5　服务跟踪器 / 86
4.6　引用服务 / 89
4.7　释放和注销服务 / 91
4.8　服务层事件 / 91
　4.8.1　事件类型 / 92
　4.8.2　事件分派 / 92
　4.8.3　ServiceRegistration 对象的提前请求 / 93
4.9　远程服务 / 94
　4.9.1　准备远程服务环境 / 94
　4.9.2　远程服务示例 / 96
　4.9.3　远程服务属性 / 99
　4.9.4　实现分析 / 100
4.10　服务钩子 / 101
　4.10.1　EventListenerHook / 101
　4.10.2　FindHook / 101
　4.10.3　ListenerHook / 102
　4.10.4　服务钩子示例 / 102
4.11　本章小结 / 105

第三部分　基于 Equinox 的 OSGi 应用与实践

第 5 章　Equinox 启航 / 108

5.1　建立 Equinox 开发环境 / 109
　5.1.1　建立运行环境 / 109

5.1.2 建立编译及调试环境 / 110
5.1.3 建立开发环境 / 112
5.2 Equinox 常用组件简介 / 117
5.3 Equinox 启动器 / 119
5.4 使用代码启动 Equinox / 124
5.5 本章小结 / 125

第 6 章 Equinox 基础应用与源码解析 / 126

6.1 实践项目——Neonat 论坛 / 126
6.1.1 背景与需求 / 126
6.1.2 模块划分 / 127
6.1.3 基础资料模块 / 129
6.1.4 持久化模块 / 133
6.1.5 用户交互模块 / 135
6.1.6 运行效果 / 140
6.2 Equinox 源码解析 / 142
6.2.1 OSGi 容器启动 / 142
6.2.2 Bundle 状态恢复 / 147
6.2.3 解析 Bundle 依赖关系 / 153
6.2.4 OSGi 类加载器实现 / 157
6.3 本章小结 / 162

第 7 章 服务器端 OSGi / 163

7.1 OSGi 与 Web 服务器 / 163
7.2 HTTP Service 规范简介 / 166
7.2.1 服务目标 / 166
7.2.2 服务接口 / 168
7.2.3 资源映射规则 / 169
7.2.4 请求处理过程 / 171
7.3 实践项目——Neonat 论坛的 Web 模块 / 171
7.3.1 准备依赖项 / 172
7.3.2 使用 HTTP Service / 174
7.3.3 实现 Web 交互功能 / 176

7.3.4 运行效果 / 178
7.4 HTTP Service 源码解析 / 180
7.4.1 BridgeServlet 与 OSGi 容器启动 / 180
7.4.2 BridgeServlet 与 HTTP 请求委派 / 186
7.4.3 DelegateServlet 实现原理 / 188
7.5 本章小结 / 192

第 8 章 用户管理服务 / 193

8.1 User Admin 服务规范简介 / 193
8.1.1 服务目标与基础概念 / 193
8.1.2 验证用户身份 / 195
8.1.3 验证用户权限 / 196
8.1.4 User Admin 事件 / 197
8.2 实践项目——Neonat 论坛用户管理模块 / 198
8.2.1 需求与依赖项分析 / 198
8.2.2 用户与用户组的实现 / 200
8.2.3 页面权限 / 201
8.2.4 用户登录与身份验证 / 202
8.3 User Admin 源码解析 / 206
8.3.1 用户管理实现 / 206
8.3.2 外部服务使用实践 / 208
8.4 本章小结 / 211

第 9 章 Preferences 服务 / 212

9.1 Preferences 服务规范简介 / 212
9.1.1 服务目标 / 212
9.1.2 数据结构 / 214
9.1.3 属性 / 215
9.1.4 并发处理 / 216
9.1.5 清理遗留数据 / 217
9.2 实践项目——Neonat 论坛持久化模块 / 217
9.2.1 编码实现 / 217
9.2.2 模块热切换 / 220

9.3 Preferences 源码解析 / 222
9.3.1 数据结构实现 / 224
9.3.2 属性存取 / 228
9.3.3 后端存储系统 / 229
9.4 本章小结 / 234

第 10 章 声明式服务 / 235

10.1 声明式服务规范简介 / 236
10.1.1 服务目标 / 236
10.1.2 定义 Component / 236
10.1.3 Component 类型 / 237
10.1.4 Component 生命周期 / 240
10.1.5 Component 属性 / 245
10.1.6 绑定与发布服务 / 245
10.1.7 激活与钝化方法 / 252
10.1.8 Component 配置总结 / 254
10.2 实践项目——使用声明式服务改造 Neonat 论坛 / 259
10.2.1 可视化编辑工具 / 259
10.2.2 DS 容器管理 / 263
10.3 DS 容器源码解析 / 264
10.3.1 容器启动 / 264
10.3.2 加载 Bundle 中的 Component / 267
10.3.3 动态依赖解析 / 272
10.4 本章小结 / 274

第 11 章 Subsystems 服务 / 276

11.1 服务目标 / 276
11.2 Subsystem 格式 / 277
11.3 Subsystem 元数据 / 278
11.3.1 SUBSYSTEM.MF 标识 / 278
11.3.2 DEPLOYMENT.MF 标识 / 281
11.4 Subsystem 类型与共享策略 / 283
11.5 组织管理 Subsystem / 285

11.6 Subsystem 部署与依赖策略 / 289
11.7 Subsystem 生命周期 / 291
11.7.1 安装 / 292
11.7.2 解析 / 293
11.7.3 启动 / 294
11.7.4 停止 / 294
11.7.5 卸载 / 295
11.8 本章小结 / 295

第四部分 最佳实践

第 12 章 OSGi 最佳实践 / 298

12.1 Bundle 相关名称命名 / 298
12.2 Bundle 划分原则 / 300
12.2.1 恰如其分地分配 Bundle 粒度 / 300
12.2.2 分离 OSGi 代码 / 300
12.2.3 分离接口和实现 / 300
12.3 依赖关系实践 / 301
12.3.1 依赖分析工具 / 301
12.3.2 避免 Require-Bundle / 303
12.3.3 最小化依赖 / 304
12.3.4 避免循环依赖 / 304
12.3.5 Equinox x-* 依赖 / 305
12.4 Equinox 专有类加载机制 / 306
12.4.1 Buddy Loading 类加载机制 / 306
12.4.2 ClassLoaderDelegateHook 类加载机制 / 307
12.5 Bundle 生命周期实践 / 309
12.5.1 启动 / 309
12.5.2 停止 / 309
12.6 服务工厂的特殊性 / 309
12.7 处理非 OSGi 的 JAR 包 / 311
12.8 启动顺序实践 / 313
12.8.1 避免启动顺序依赖 / 313

12.8.2 Start Level 的使用 / 313
12.9 Fragment Bundle 实践 / 314
12.10 保持 OSGi 动态性 / 315
12.11 API Tools 实践 / 317
12.11.1 API Baselines / 317
12.11.2 API Tools 注解 / 319
12.11.3 API Version 版本管理 / 322
12.11.4 二进制文件不兼容 / 322
12.12 本章小结 / 322

第 13 章 Spring Dynamic Modules 实践 / 324

13.1 Spring DM 入门 / 324
13.1.1 Spring DM 项目简介 / 324
13.1.2 安装 Spring DM / 325
13.1.3 简单的 Spring DM 示例 / 326
13.1.4 Bundle 和 Spring 上下文 / 331
13.1.5 <osgi:*> 命名空间 / 333
13.2 Spring DM 进阶 / 337
13.2.1 Spring DM 扩展配置 / 337
13.2.2 Web Extender / 344
13.2.3 Spring DM 服务约束 / 345
13.2.4 在 Spring 上下文中使用 BundleContext / 346
13.3 Spring DM 企业应用 / 346
13.3.1 规划 OSGi 组件 / 347
13.3.2 在 Spring DM 中使用 JPA / 348
13.3.3 事务管理 / 353
13.3.4 OSGi 企业规范中的 JPA / 358
13.4 Spring DM 和 Blueprint / 359
13.5 本章小结 / 360

第 14 章 构建可测试的 OSGi 系统 / 361

14.1 单元测试的必要性 / 362
14.2 单元测试的重要性 / 363

14.3 可测试代码的特征 / 364
14.4 OSGi 单元测试 / 365
14.4.1 如何组织测试代码 / 366
14.4.2 如何进行 OSGi 单元测试 / 367
14.5 OSGi 集成测试 / 373
14.5.1 Eclipse JUnit Plug-in Test / 374
14.5.2 Spring DM Test / 379
14.5.3 Pax Exam / 383
14.6 本章小结 / 384

附录 A Java 类加载器简介 / 385

附录 B Equinox 控制台命令 / 392

附录 C OSGi 子规范目录 / 397

附录 D OSGi 相关项目 / 399

附录 E Equinox 启动配置参数 / 401

第一部分
走近 OSGi

第 1 章　Java 模块化之路

第 1 章　Java 模块化之路

Java 可能是近 20 年来最成功的开发技术，因其具备通用性、高效性、平台移植性和安全性而成为不同硬件平台理想的开发工具。从笔记本电脑到数据中心，从游戏控制台到科学超级计算机，从手机到互联网，Java 技术无处不在。

Java 能够让程序员使用同一种语言为服务器、智能卡、移动电话和嵌入式设备开发程序，极大地提升了软件的研发效率。不过，仅靠统一的语言还不足以让软件业迅速提升至成熟的工业化阶段。不同软件系统、不同硬件设备下的程序都经常会有相同的业务需求和设备间交互通信的需求，例如很多设备都需要互联网接入的功能，如果通用于不同设备的网络标准件不存在，那就只能为每个设备都开发一个连接互联网的模块，这样效率和质量都难以保证。假如把开发中经常遇到的需求进行抽象，将它们统一规范起来作为标准件提供，任何设备都通过预定义好的协议和接口来使用这些标准件，那么构造一个大型程序的主要工作很可能就只是根据需求选择合适的模块，然后再写少量的黏合代码而已。

标准件是区别小手工作坊和大工业化最明显的标志。今天，个人计算机的硬件已经到达了工业化阶段，无论哪个公司生产的显示器、键盘、鼠标、内存和 CPU，都遵循统一规范的接口工作。要获得不同功能、性能的计算机，只要选择适当的硬件模块进行组装即可。与此相对的，大部分计算机软件都还是从零开始进行编码开发的。软件业还远不如硬件成熟，但是软件工业化是一股不可逆转的潮流，实现这个目标的第一步就是要制定不同功能模块的标准，以及模块间的黏合及交互方式。Java 业界内已经有了很多的技术规范，例如 EJB、JTA、JDBC、JMS 等，欠缺的是一个组织者或扮演黏合剂的角色，直到 Java 有了 OSGi……

1.1　什么是 OSGi

OSGi（Open Service Gateway Initiative，直译为“开放服务网关”）实际上是一个由 OSGi 联盟（OSGi Alliance，如图 1-1 所示）发起的以 Java 为技术平台的动态模块化规范。

OSGi 联盟是由 Sun Microsystems、IBM、Ericsson 等公司于 1999 年 3 月成立的一个世界性的开放标准化组织，最初的名称为 Connected Alliance，该组织成立的主要目的原本在于使服务提供商通过住宅网关为各种家庭智能设备提供服务。最初的 OSGi 规范也只是关注于嵌入式领域，前三个版本的 OSGi 规范主要满足诸如机顶盒、服务网关、手机等应用环境的模块化需求。从第四个版本开始，OSGi 将主要关注点转向了 Java SE 和 EE 领域，并且在这些领域中获得了很大的发展，成为 Java 平台事实上的模块化规范。

图 1-1　OSGi 联盟

随着 OSGi 技术的不断发展，OSGi 联盟的成员数量已经由最开始的几个增长到目前超过 100 个[⊖]，很多世界著名的 IT 企业都加入到 OSGi 的阵营之中，如 Adobe、IBM、Oracle、SAP、RedHat 和 Siemens 等。它们推出的许多产品都支持 OSGi 技术，甚至产品本身就使用了 OSGi 技术构建，例如 IBM 的 WebSphere、Lotus 和 JAZZ，Oracle 的 GlassFish 和 Weblogic，RedHat 的 JBoss，Eclipse 基金会的 Eclipse IDE、Equinox 及之下的众多子项目，Apache 基金会的 Karaf、Aries、Geronimo、Felix 及之下的众多子项目等。这些 IT 巨头的踊跃参与，也从侧面证明了 OSGi 技术有着非常广阔的市场前景。

OSGi 技术的影响同时也延伸到了 Java 社区，JSR-232 提案的通过说明 OSGi 技术已经被 Java ME 领域所认可，而 JSR-291 提案则奠定了 OSGi 技术在 Java SE 和 Java EE 领域标准模块化规范的地位（OSGi 与 Java 模块化规范的历史将在 1.1.2 节会详细介绍）。

OSGi 的诸多优秀特性，如动态性、模块化和可扩展能力逐渐被越来越多的开发者所认识和欣赏，越来越多的系统基于 OSGi 架构进行开发。在这些系统的开发过程中，又会向 OSGi 提出一个又一个新的需求，所以 OSGi 规范所包括的子规范与技术范畴也在不断发展、日益壮大，如图 1-2 所示。

今天，OSGi 的已经不再是原来 Open Service Gateway Initiative 的字面意义能涵盖的了，OSGi 联盟给出的最新 OSGi 定义是 The Dynamic Module System for Java，即面向 Java 的动态模块化系统。

2012 年 7 月，OSGi 联盟发布了最新版的 OSGi R5.0 规范，这次发布的规范包括 OSGi 核心规范 R5.0 和 OSGi 企业级规范 R5.0。在 Java SE 领域，Eclipse 和 NetBean 两款集成开发工具的成功已经完全证明了 OSGi 在桌面领域是能担当重任的。最近两三年来，OSGi 的发展方向主要集中在 Java EE 领域，在 OSGi 企业专家组（EEG）的努力下，OSGi 的企业级规范 R5.0 版相比两年前发布的 R4.2 版又增加了许多新的内容，OSGi 技术在服务端和企业级领域正迅速走向成熟。

⊖ OSGi 联盟成员：http://www.osgi.org/About/Members。

图 1-2 OSGi 及相关技术

1.1.1 OSGi 规范的演进

OSGi 在 R4 版之前都处于初级阶段，规范的主要关注点是在移动和嵌入式设备上的 Java 模块化应用。在这个初级阶段中有一些很成功的案例，比如 BMW（宝马）汽车使用 OSGi 架构实现的多媒体设备控制程序（内部是西门子 VDO 系统），要使用不同型号的电子设备，只更换对应程序模块便取得了很好的效果。但是初级阶段的 OSGi 在 Java 其他主流应用领域（企业级、互联网、服务端、桌面端等）的影响力还比较有限，因此下面简要介绍这部分的历史。OSGi 规范在初级阶段一共发布了三个版本：

- OSGi Release 1 (R1)：2000 年 5 月发布。
- OSGi Release 2 (R2)：2001 年 10 月发布。
- OSGi Release 3 (R3)：2003 年 3 月发布。

从 OSGi R4 版开始，OSGi 的目标就从“在移动和嵌入式设备上的 Java 模块化应用”发展为“Java 模块化应用”，去掉了“在移动和嵌入式设备上的”这个限定语，这意味着 OSGi 开始脱离 Java ME 的约束，向 Java 其他领域进军。同时也意味着 OSGi 需要考虑如何去迁移遗留的异构系统、如何去支持大规模开发等非嵌入式领域的问题了。因此，OSGi R4 版规范的复杂度相应地高出 R3 版许多。笔者可以给出两个最直观的数据：规范文档的页数从 R3 的 450 页增加到 900 页，规范中定义的外部接口（统计 API 中 public 方法）数量从 R3 的 661 个增加到 1432 个[⊖]。

⊖ 此数据来源于 World Congress 2005 的 PPT：http://www.osgi.org/wiki/uploads/Congress2005/1012_1015_lopez。pdf。

OSGi R4 版本分为两个部分，2005 年 10 月发布的核心规范（包含服务纲要规范）和 2006 年 9 月发布的移动设备规范（移动设备规范已停止发展，目前最新的 OSGi 移动设备规范依然是这个版本）。从更具体的角度来讲，OSGi R4 解决了 R3 的许多遗留问题和限制，以下列举了部分在核心规范中比较关键的改进：

- 在 OSGi R3 版本中，模块导出的 Package 是全局唯一的，不允许同一个 Package 存在多个版本。这点限制放在资源受限的嵌入式环境中一般不会有问题，但是放在整个 Java 领域就不妥了，因为引用不同版本的第三方包对于规模稍大一点的程序来说是很常见的事情。
- OSGi R3 中的模块缺乏对模块本身的扩展机制，所有的资源、代码都必须在模块中是静态存在的，无法运行时动态添加。在 OSGi R4 中，出现了 Fragment Bundle 的概念。
- OSGi R3 的 Package 导入和导出无论是版本、可见性和可选择性都很粗糙，例如在导入时指明一个 Package 的版本，语义就只能是导入不小于这个版本的 Package，而对于要明确具体版本范围（如 [2.5，3.0)）的需求就不适用；又如在导出 Package 时，一个 Package 中的所有类要么全部导出，要么全部隐藏。在 OSGi R4 中改进了 version 参数，也为导入导出加入了许多子参数来方便精确过滤范围。

除了在核心规范中对 R3 版的改进外，许多目前非常常用的、在服务纲要规范中的 OSGi 服务也是在 R4 版才开始出现的。这些服务对提升开发人员的工作效率及系统的鲁棒性[⊖]有很大帮助，例如 R4 版首次出现的声明式服务就是对 R3 版之前的程序化服务模型的重大改进。

尽管从 OSGi R3 到 OSGi R4 发生了很大的变化，R4 版规范依然保持了很好的向后兼容性，绝大部分能运行于 OSGi R3 的模块都可以不经修改地迁移到 OSGi R4 之中。

2007 年 5 月发布的 OSGi R4.1 是一个修正版性质的规范，只是核心规范发生了很小的变化，服务纲要规范和移动设备规范并没有跟随发布 R4.1 版，整个 R4.1 版没有新增任何服务。OSGi R4.1 版本的推出，最重要的任务是适配 JSR-291 提案，让 JSR-291 提案顺利通过 JCP 的投票，成为整个 Java 业界标准的一部分。

在 OSGi R4.1 版本中，值得一提的改进是处理了 Bundle 延迟初始化的问题，增加了 Bundle-ActivationPolicy 标识来指明 Bundle 的启动策略。在此之前，OSGi 实现框架只能通过自己的非规范的标识来完成类似的事情，例如 Equinox 的私有的 Eclipse-LazyStart 标识。

2009 年 9 月，OSGi R4.2 版核心规范发布；在次年 3 月，还发布了 OSGi R4.2 企业级规范。OSGi R4.2 是一个包含了许多重要改进的版本。首先，随着 OSGi 实现框架的数量逐渐增多，OSGi R4.2 开始着手解决 OSGi 框架自身的启动问题，提供了操作 OSGi 框架的统一 API。在此之前，启动 Felix、Equinox、Knopflerfish 或其他 OSGi 框架，必须使用完全不同

⊖ 鲁棒性是 Robustness 的音译，是指当系统受到不正常干扰时，是否还能保证主体功能正常运作。可参考维基百科：http://zh.wikipedia.org/zh/ 鲁棒性 _(计算机科学)。