Java应用开发中代码生成工具的作用Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_Java_E5_BA_94_E7_94_A8_c104_644582.htm 近来，随着各种代码生成工具的不断涌现（如SpringSource的Spring Roo、Skyway Builder Community Edition 6.3及Blu Age的M2Spring等），人们又将注意力转移到了这些代码生成工具在企业级Java应用开发中所起的作用。 模型驱动开发（MDD）正获得越来越多的关注，而在一个典型的Java应用中有些内容完全是可以自动生成的。典型的Java Web应用包含数据访问对象（DAO）类、XML to Java映射文件、Spring、Log4J配置文件等，这些内容其实都是能够自动生成的。 Roo是个双向代码生成框架，可以生成Spring Web应用所需的大多数基础代码。Roo提供了一个命令行shell，同时具有tab completion、上下文感知操作以及命令行提示等特性；它还会以标准的目录格式构建Spring应用，管理构建配置文件、辅助开发者创建领域对象并能为基于REST的Web UI自动生成Web层代码。 Skyway Builder Community Edition 6.3于上个月发布，它提供了对Spring MVC的支持，可以根据新建或现有的领域模型生成基于Spring的Java CRUD应用。它集成了Spring DSL，还能生成Spring MVC与Spring Web Flow应用代码。Skyway Builder商业版Skyway Builder Enterprise Edition (EE) 6.3集成了IBM Rational Software Architect以将UML转换为Spring应用代码。此外，它还提供了对DWR（JavaScript/JSON）的支持，可以通过Spring services开发RIA应用；还能够通过JET技术定制项目级的代码生成模板。 IBM的MDD工具（叫做Rational Rhapsody）支持UML2与SysML、

需求跟踪、应用代码生成以及针对测试进行设计（DFT）等特性。Rhapsody是个双向的模型驱动解决方案，可以通过需求图、用例图、序列图、活动图以及状态图捕获项目需求。接下来用户就可以根据模型创建对需求的跟踪链接，这种链接能自动提供跟踪、影响分析以及覆盖文档等功能。Rhapsody还支持模型驱动测试（MDT），所谓模型驱动测试，实际上是一种新的方法论，它将MDD的优点引入到了测试过程中。凭借MDT，工程师可以不断对设计进行模拟以及早定位错误，同时还可以自动化单调乏味的测试、进行基于需求的测试以验证设计是否符合需求，它还能通过IBM Rational Rhapsody Automatic Test Generation Add On根据设计自动创建覆盖率测试。 最近Blu Age凭借其产品M2Spring也加入到了代码生成工具的阵营。M2Spring联合使用了MagicDraw UML与Blu Age Agile Model Transformation进行建模及自动生成基于Spring架构的应用代码。它能够在服务层（业务规则、应用服务及Web Service）、表示层（用户界面、用户角色及安全策略）和持久层（业务对象、DAO实现及DAO finder）上生成Spring Web应用所需的类和其他代码。M2Spring支持多种模型与JEE技术，如UML 2.2、OCL 2.0、XMI 2.1、EMF UML2 2.x XMI、Struts、Spring及Hibernate.。 还有一些开发工具也支持代码生成，如Project Lombok和Spoon。Lombok具有如下一些特性：自动生成默认的getter/setter方法、自动化的资源管理（通过@Cleanup注解）及注解驱动的异常处理等。 InfoQ采访了Spring Roo的项目经理Ben Alex以深入了解Java应用开发中代码生成工具所起的作用。Ben Alex说到大多数的开发者每天都在使用代码生成，无论是通过Eclipse的“ getters/setters ”特

性还是从别的地方拷贝过来一段代码，本质都是如此。其主要动机是尽快为特定的需求实现解决方案，避免每次都花费时间研究最佳的解决办法。现代的代码生成工具的动机也是如此，只不过将应用范围扩大了，从简单的“ getters/setters ”自动生成到提高应用生产力的层次上，比如构建多层的应用。就像是“ getters/setters ”自动生成一样，现代的代码生成工具也非常容易使用，其生成的代码与我们自己动手编写的差不多，同时我们还可以轻松修改生成的代码。 在使用代码生成工具之前，开发者需要注意哪些方面呢？ 开发者可以像评估IDE工具那样来评估代码生成工具。要是我的话，我就会问这个软件会提升个人生产力么？学习和使用曲线如何？我所在的组织允许使用这个软件么？一旦使用之后，要是不想用了能行么？软件的维护力度如何？是否有相应的社区呢？软件的质量如何（未解决的缺陷、最终用户的博客以及架构性的资讯信息数量等等）？能否通过插件的方式轻松增加新的功能呢？ 除了像使用IDE时可能会遇到的问题外，你还需要问自己特定于代码生成工具的问题：从长远来看，这个代码生成工具会持续不断地维护我的代码么（比如提供了哪些双向支持）？操作代码生成工具的方式对我的知识、技能与经验有哪些要求？与我经常使用的IDE的协作能力如何？能否与新版本的IDE协作良好，抑或是安装了新版本的代码生成工具或是IDE时会不会导致崩溃的情况发生？生成的代码是否自然、干净、一致且高效呢？其使用方式是否足够灵活以便让我能按照自然的方式工作？其是否完全是自动化的，还是需要我做一些额外的工作？新版本的代码生成工具（或是插件）不会搞乱我的项目吧？ 在大多数应用中，开发者都需要手工

编写业务逻辑与验证规则代码，而这些代码都不太容易自动生成，那代码生成工具是如何解决这个问题的呢？ Hunt与Thomas在The Pragmatic Programmers一书中提到了两类代码生成方式：被动的与主动的。 所谓被动生成方式就是只运行一次代码生成工具来生成结果，然后手工维护生成的代码。很多开发者都在IDE的帮助下使用过被动生成方式，如“ getters/setters ”生成器。被动生成方式的输出结果都是由开发者决定的：他们可以通过命令来生成想要的东西。被动生成方式的缺憾在于开发者必须要手工维护生成的代码，因为生成工具不会再对生成好的代码进行更新了。在你修改了生成代码所依赖的某些内容后这个问题就会变得尤为明显，因为此时生成的代码将会出错，而用户通常都会删掉代码并重新运行生成工具。 所谓主动生成方式就是每次运行时都会产生相应的结果，它需要一种方式来存储控制信息，这样就能运行并生成所需的结果。控制信息技术的差别非常大，但通常都是自动获取的元数据（比如来自于源代码解析与绑定）与通过Java注解、 JavaDoc标记、XML文件、特殊的GUI配置等信息构成的用户首选项的组合。有了这些高质量的主动生成工具（比如Roo），开发者可以像往常一样在标准的源文件中编写自己的逻辑，同时生成工具会检测到源文件的改变并只自动更新受影响的这些文件。因此我们无需告诉系统（比如Roo）想要编写客户化代码了，因为这一切已经成为系统的一部分了。 如何将优秀的架构与设计实践应用到既有自动化的代码生成又有手工编写代码的应用开发过程中呢？ 很多现代的自动生成工具都用来生成最初的应用骨架。这意味着所生成的应用反映了嵌入在生成工具中的最佳实践。接下

来，通过正常的对生成器的操作又会不断鼓励用户使用这些最佳实践，因为后面生成工具所生成的代码也反映出了同样的架构模式。这种一致性对于那些团队成员水平参差不齐或是项目需要长期维护的组织来说是个巨大的优势。 自然，用户肯定要问的一个问题就是嵌入在生成工具中的那些最佳实践真的是最佳的么？对于该问题的答案就是：被众多用户与项目所用的生成工具通常都是最佳的，因为大量的设计反馈都会被集成到工具中。用户还可能会审视实际的架构、仔细审阅其可测试性、可维护性、设计完整性、可复制性、避免独占等等。此外，大多数高质量的代码生成工具都会让一些知名的组织来对其生成的架构进行认证（比如SpringSource就将Roo架构作为最佳实践）。 相对于传统的代码生成框架，其他手段比如Grails（大多数代码都是在执行期生成的）效果如何呢？ Grails、Rails及Django等框架提供了一种被动的代码生成方式，它们会生成初始的应用骨架和特定的代码如Web控制器。接下来会在运行期（通过动态语言或反射技术）执行主动代码生成工具在开发期所执行的动作。 静态语言编译期技术相对于动态语言运行期技术相比具有很多优点，同时一些缺点也是显而易见的。它们是两种完全不同的开发方法，没有谁比谁强的说法，也没有哪一种方法能适合于所有的项目。SpringSource拥有Roo和Grails，因此用户可以选择感兴趣的进行开发，无论选择哪一个都能享受到生产力提高所带来的乐趣，同时也都能利用Spring的巨大优势。 您是如何看待常规的模型驱动软件开发（MDSD）与特定的代码生成技术呢？ 我深信寻求更高层次编程抽象的目标将会继续下去，然而当前的趋势（根据不断涌现的新框架的数量及采取这些

框架的开发者数量得到的结论）则聚焦在编程语言的革新与DSL上。这些生成工具之间的一个主要区别在于其DSL方式，一些工具以简单的命令为特色（像是“ create-controller ”）而另一些则提供了高级的shell（如tab completion、上下文感知、命令隐藏等）。这些高级的shell不仅降低了使用难度，而且还简化了日渐强大的（以及更高的抽象层上）命令的使用。 综上所述，代码生成工具依旧会在软件开发中扮演着重要角色。静态语言仍将占据着统治地位（来源：Tiobe），同时这些语言还是需要借助于代码生成技术来提升程序员的生产力。甚至是动态语言框架也会在某些地方使用代码生成技术，如初始化应用结构、关键的应用代码等等。像更高层的DSL、高级的shell、创新性的编译单元分离及高质量的双向支持等未来趋势会一直支撑着下一阶段的代码生成技术革命。 InfoQ还采访了Skyway团队的Jack Kennedy以了解软件开发中的代码生成技术。现在很多项目都使用了敏捷与精益软件开发方法学，如Scrum、XP或是Kanban，那对于这类项目，代码生成技术会起到什么作用呢？就这个问题，Jack回答到：不管采用何种开发方法，基于模式的代码生成与自动化技术始终都会影响到软件开发生命周期的各个阶段（分析/设计/构建/测试/部署）。代码生成由需求中的概念性元素开始，然后将其转换为具体的基于模式的设计，最后又将设计转换为基于模式的代码与测试单元。对于敏捷与精益软件开发方法来说，使用基于模式的代码生成与自动化技术的动因是相当重要的。尽管这些开发方法对于小项目来说还是非常成功的，但在整个软件产业当中要想更加充分地采用这些开发方法还是面临着相当大的挑战，首当其冲的便是处理更大、

分布式、复杂需求的能力。问题的中心在于对高素质开发人员的渴求以及缺少对大项目投入产出估算的能力。将基于模式的代码生成与自动化技术引入到软件开发的整个生命周期当中可以为组织提供如下好处：提升新手的能力并通过久经考验的组件来降低估算工作量。由于现代方法学一直强调持续集成与可论证的开发迭代，因此通过最小的投资和一致的质量来生成功能应用已经成为迫在眉睫的任务了。 代码生成骨架既可以节省时间、降低人力需求，又能保证代码质量，因为大多数应用代码（对于Web来说，就是Log4J与Spring配置文件等内容）都是自动生成的。这是代码生成工具的最大价值所在么？ 基本上来说，软件生成平台通过自动化开发任务且遵循着目标部署环境的实现需求而为用户创造价值。其背后的隐式价值所在就是一致性、高质量及重用性。这种价值可以通过系统输入的代价与输出的成果之间的对比来进行衡量。总的来说，目标就是将代价最小化的同时保证价值的最大化。过去，代码生成工具仅仅被限制为根据生成系统所接收的“ 概念 ”而产生相应的代码。对于数据建模来说，这通常意味着可以使用一种格式来创建数据模型，然后生成该数据模型的“ 类 ”实现。很多 UML解决方案都提供了这类生成工具（而且是专有排外的）。然而，在这种情况下，很多团队都发现投入产出比并不太好。 由于企业应用的实现技术已经简化了很多，同时软件的生成技术也已经很成熟了，与之对应，能够自动生成的功能范围得到了极大的扩展。具备生成基于CRUD应用能力的代码生成工具让很多开发者转向了现代的RAD开发环境与工具，这些工具包含了DSL与生成技术。这是生成技术的一个转折点，然而却还不能代表最终

的价值所在。随着工业界不断提供更广泛的生成选项与输入格式，我们将会看到有越来越多的开发者转向代码生成技术。 你如何看待模型驱动开发与代码生成技术的未来？ 未来一段时间的重点将放在填平传统的MDD方法/工具与新涌现的DSL之间的沟壑上，使快速应用开发成为可能。当前所用的DSL将会变得更加强大，当然复杂性也会不断攀升，它会整合安全、验证及应用流。生成技术将会跨越简单的“代码生成”阶段，转而成为构建与部署过程中的一个标准组成部分。可以从UML和DSL上获取的软件资产类型还将不断增加，同时还会跨越更广的技术栈集合以生成更多的应用代码。随着开发者不断看到可以从这些简单的语法当中自动生成大量的功能，DSL还将继续繁荣下去。这些DSL的工具选项将变得更加复杂，同时还将提供多种不同的输入方式与格式，包括文本与可视化元素。我们期望最后这些软件供应商会在生成并运行基于DSL的应用部署上展开激烈的竞争。 编辑特别推荐: 指点一下：到底该不该去考JAVA认证? Java面试题：第一锦 Java面试题：第二锦 Java面试题：第三锦 Java面试题：第四锦 Java面试题：第五锦 一个有趣的编程:程序员的爱情故事 