

扩展 Java 元模型以实现诠释的可视化建模^{*})

严 悍 张 琨 李千目 刘凤玉

(南京理工大学计算机科学与技术学院 南京 210094)

摘 要 诠释(annotation)是 Java SE5 引入的一种新的编程语言成分,目前得到广泛关注。但从可视化建模来看,已有的 Java 元模型和 UML2 元模型并不支持诠释规范,这导致诠释的语义特征在高层抽象中难以体现,也不能实现诠释的可视化、规范化建模。本文扩展已有的 Java 元模型,以支持诠释的可视化建模。文章提出了诠释的 4 个重要特征,添加了 3 个元类以扩展 Java 元模型,使其能反映诠释特征,并给出 2 个图符以支持诠释的可视化建模。新的 Java 元模型符合 MOF 规范,设计简单,方便实现以支持 MDA 及相关工具的开发。

关键词 诠释,元模型,Java,可视化建模,UML

Towards Visual Modeling of Annotation by Extending Java Metamodel

YAN Han ZHANG Kun LI Qian-Mu LIU Feng-Yu

(School of Computer Sci. & Tech., Nanjing University of Science and Technology, Nanjing 210094)

Abstract This paper proposes a new Java metamodel with annotation extended to support annotation visual modeling. Annotation is one of the new features of Java SE5. It plays significant roles on metadata and metaprogramming. But with a view to visual modeling, annotation is not supported by either existing Java metamodel or UML2 metamodel. This causes annotation semantic features can not be reflected at higher abstract level, and annotation visual modeling is impossible. This paper presents four significant features of annotation: type-based, static instance, target-associated instance and instance immutable. Three metaclasses: AnnotatableElement, Annotation and AnnotationInstance, are extended to specify the features of annotations consistently. Further, two notions are suggested for visual modeling of annotation. The new Java metamodel complies with the MOF specification. Also it is designed simple so that it's easier to realize to support MDA and related utilities development.

Keywords Annotation, Metamodel, Java, Visual modeling, UML

1 引言

诠释(annotation)是 Java SE5 引入的一种新的语言成分^[1]。诠释不同于注释(comment),它是基于类型的、对程序中元素的一种说明^[2]。目前诠释得到广泛关注,出现了多种基于诠释的编程与应用^[3~5],诠释正在扮演越来越重要的角色。但从可视化建模来看,已有 Java 平台相关的元模型未包含诠释^[6],而且 UML2 最新元模型中也未提供关于诠释的规范^[8,9],这导致诠释的应用被局限在编程技巧方面,诠释的特征难以在抽象层面上得到反映,也难以实现诠释的可视化、规范化建模。本文对已有 Java 元模型扩展诠释部分,以探讨诠释的规范化建模的有效途径,从而促进诠释相关的开发和应用。

诠释的元建模主要针对两个问题:(1)如何用 MOF 来规范诠释的概念和语义特征;(2)如何确定图形化符号来支持诠释的可视化建模。诠释的元建模不仅是可视化建模的基础,而且对于诠释的开发和应用具有重要意义。

诠释的元模型可促进诠释相关的开发应用。目前已出现一些有价值的诠释应用,如 JUnit4 定义一组诠释类型专门用于单元测试^[3];Hibernate3 则用诠释来定义实体类及其属性

如何映射到关系数据库^[4];AspectJ5 把诠释用于 AOP(Aspect-Oriented Programming),用诠释来定义 aspect, pointcut, advice 和 inter-type 成员^[5]。文[6]提出一个 Java 元模型并扩展以支持 AspectJ。诠释元模型有助于诠释建模工具、诠释类型生成、诠释处理工具的开发。

诠释元模型能促进一些新的编程范型。引入诠释的目的是对程序元素提供所谓“元数据”,以实现“说明式编程”^[2]。传统的“元编程”是基于宏或模板,目的是编写程序以生成一些程序。诠释编程可纳入元编程,这是因为诠释也能用于程序生成,如 Hibernate3^[4]和 apt^[7]。诠释也可用于“面向属性的编程”。本文探讨诠释的元模型和可视化建模,可促进元编程及相关领域的研究和应用。

UML2 元模型未提供诠释相关的元类^[8],但本文并非扩展 UML 元模型,而是主张扩展已有的 Java 元模型,建立一个新的 Java 元模型,理由是:(1)目前诠释仅出现在 Java 中,带诠释的 Java 元模型能直接支持诠释的建模与编程。(2)基于元模型的转换是 MDA(Model-Driven Architecture)的核心机制^[10],而 Java 元模型作为一种平台相关模型(PSM, Platform Specialize Model)规范,可为 MDA 奠定基础。(3)根据 UML 的 4 层语言架构^[9],Java 元模型位于 M2 元模型层,符合 M3

^{*})基金项目:国家自然科学基金资助项目(60273035),国家人事部出国留学人员择优资助项目。严 悍 副教授,博士,主要研究方向为软件工程与信息安全;张 琨 博士,主要研究方向为信息安全;李千目 博士,主要研究方向为网络性能与信息安全;刘凤玉 教授,博导,主要研究方向为软件方法与信息安全。

层元-元模型 MOF(Meta Object Facility)规范^[9],这使得 Java 元模型能与 UML 元模型(同属于 M2 层)能保持相容性。

总之, 诠释的元模型基于(1)诠释语义语法特征; (2)已有 Java 元模型; (3)MOF 规范; (4)UML 元模型; (5)Java 镜像 API^[11]与反射 API^[12]。诠释的元模型的意义是: (1)在抽象层面上反映诠释特性; (2)促进工具开发; (3)促进元编程和新型编程范型; (4)支持诠释的建模和编程, 支持 MDA 元模型转换。第 2 节先分析诠释的特征, 第 3 节介绍诠释的元模型, 第 4 节探讨诠释的可视化建模, 第 5 节讨论比较相关工作, 最后是结论与进一步工作。

2 诠释特征分析

新的元模型应能正确反映诠释的语义特征。经分析比较,诠释具有以下4个特征:(1)基于类型。诠释并非简单的文本注释,而是基于类型的,而且可定义一组元素值作为其成员。(2)静态实例化。某诠释类型的一个实例是用该类型对一个被诠释元素的一次诠释说明。诠释实例不同于普通对象,它是在模型或程序构建时刻创建的,而不是在运行时刻。(3)关联目标的实例化。每个诠释实例都须指明所诠释的目标,即每个诠释实例都必须关联唯一的被诠释元素,而诠释类型自身不能简单实例化。(4)实例不变性。诠释实例及其元素值在其生命周期中不可改变。

诠释的开发应用有两个特点：(1)被诠释元素自身的功能不受诠释说明的影响，但在特定处理工具支持下能扩展新的功能，如生成新代码、完成特定转换、进行特殊控制。(2)一个诠释类型的语义规范与相应的处理过程是严格分离的。诠释类型中仅说明其语义，而不说明如何处理。编程人员对程序中的元素选择合适的诠释，而对程序中的诠释如何处理则需要特定工具。诠释处理可在编译时刻进行，或在运行时刻，或两者兼有。

3 诠释元模型

图1是元模型的诠释相关部分,其中扩展的关键元类是AnnotatableElement(可诠释元素)、Annotation(诠释)、An-

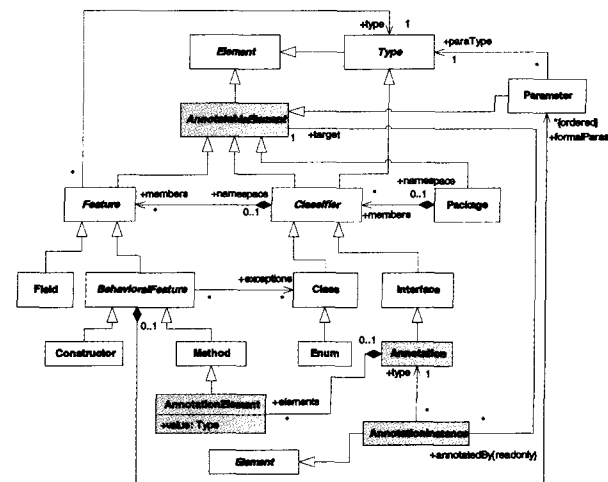


图 1 元模型的诠释相关部分

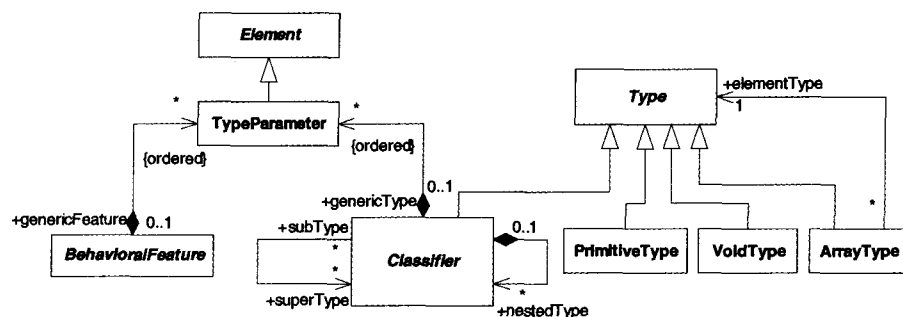


图 2 元模型的其他部分

3.2 诠释 Annotation

Annotation(诠释)表示所有诠释类型的抽象,该类的一个实例就是一种诠释类型。该类被建模为 Interface 的子类,原因有二:(1)从语法上看,诠释类型用“@interface 诠释名”或“@interface 诠释名{成员组}”来定义;(2)从实现机制来看,诠释类型在编译时将转换为 java.lang.annotation.Annotation 接口的子接口。诠释类型可以没有成员,也可定义方法、静态常量、内嵌类型(包括类、接口、诠释),但诠释类型的

notationInstance (诠释实例)。该元模型是针对文[6]中的 Java 元模型所进行的扩展。图 2 是与诠释相关的其他部分, 包括类型 Type、类属 generic、超类型 superType 与子类型 subType、内嵌类型 nestedType 等。依据 MOF 规范, 元模型主要表示各元类及其之间的泛化和关联关系。新的元模型除了要反映诠释的 4 个特征之外, 还应解决下面问题: 诠释属于什么类型, 与其他类型之间有什么关系; 可被诠释的元素有哪些; 如何进行诠释说明, 为什么诠释类型自己可被诠释等。

3.1 可诠释元素 AnnotatableElement

AnnotatableElement(可诠释元素)表示可带诠释说明的元素。“可诠释元素”包括类、接口、枚举 Enum,也包括诠释 Annotation(建模为 Classifier 的子类);域、构造器和方法 (Feature 和 BehavioralFeature 的子类);还有包 (Package) 和参量 (Parameter)。虽然程序中的局部变量也可带诠释说明,但仅限于源代码中,编译后的类文件并不保持,其作用非常有限,故此元模型中未描述。该元类建模为一个抽象类。一个可诠释元素可与多个诠释实例 (AnnotationInstance) 相关联 (annotatedBy 关联),表示被诠释元素的一组诠释说明。

特殊性有以下两方面:

(1)特殊的成员。作为成员的方法不是通常的抽象方法，而是可带缺省返回值的特殊方法。模型中用 AnnotationElement(诠释元素)类来表示此类方法，一个诠释类型中可说明多个诠释元素。这些方法的返回类型受到限制，只能是基本类型、String、类、枚举和诠释，以及这些类型的数组。该类方法不能引发异常，也不能是类属(generic)方法。

(2)受限的接口。Annotation 虽然建模为一种接口类型，

但注释类型却不能作为普通接口来用;不能被任何类实现,也不能扩展子类型(虽然图 2 中 Classifier 之间有超类型与子类

型关联,但注释不能有超注释和子注释);注释也不能是类属类型(genericType)。

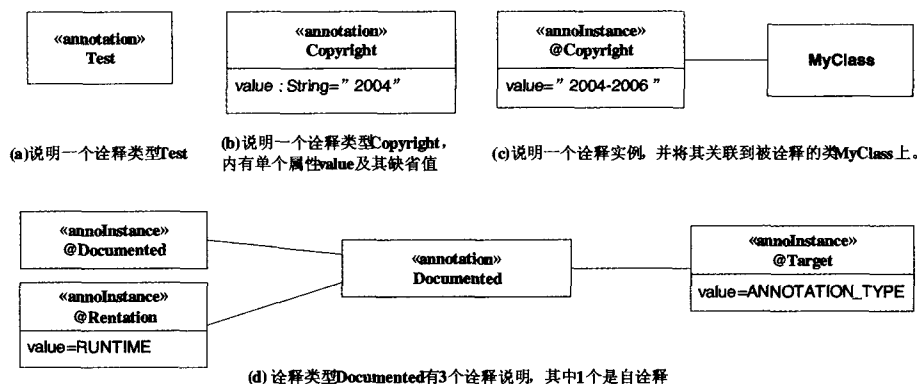


图 3 注释的可视化图符的例子(属于 M1 层的模型)

虽然 Annotation 元类中未定义自己的属性,但 Java SE5 内置 4 个“元注释”等效于其属性: @Inherited、@Retention、@Target、@Documented(参见文[1,2])。

3.3 注释实例 AnnotationInstance

AnnotationInstance(注释实例)表示注释说明,该类的一个实例表示把某种注释类型(type 关联)对特定元素(target 关联)的一次注释说明,即把一个注释类型与一个被注释元素建立关联。在编程中,一个注释实例用“@注释名”或“@注释名(属性组)”来修饰被注释元素,其中每个属性是(方法名=值)。

可注释元素 AnnotatableElement 与注释实例 AnnotationInstance 之间是多对一的关联,表示一个可注释元素可对应多个注释实例,而一个注释实例只用于说明一个元素。因注释 Annotation 自身属于可注释元素,所以一个注释类型也可被注释说明,也可自注释。例如,Java SE5 中的注释类型 @java.lang.annotation.Documented 定义如下:

```
@Documented
@Retention(value=RetentionPolicy.RUNTIME)
@Target(value=ElementType.ANNOTATION_
TYPE)
public @interface Documented{}
```

在编程中,注释说明有多种方式。Java 语言的注释说明是“以可注释元素为中心”的,即对某个元素施加多种注释实例,如上例。而 AspectJ5 则“以注释实例为中心”,把某个注释实例说明给一组元素,例如:

```
declare @method: * C.set*(.): @Test;
```

对类 C 中所有 set 开头的方法都说明一个注释实例 @Test。该元模型能同时兼容这两种方式。

3.4 一致性验证

注释的元建模采用了概念验证和实例验证来保证元模型的一致性。

- 概念验证。通过对各种建模元素的语法、语义的分析,并依据 MOF 规范建立。

- 实例验证。通过对大量程序的分析 and 建模,确保元模型、模型、程序之间的一致性。

4 注释可视化建模

为了达到可视化建模的目的,需要确定新元类的可视化图符(notion)。我们引入构造型«annotation»来表示 Annota-

tion 的实例(如图 3(a)与(b))。引入«annoInstance»来表示 AnnotationInstance 的实例(如图 3(c)),规定用“@注释名”来表示注释实例名字,且关联到被注释的元素。其它元类的图符可参考 UML2.0 中同名或相似的元类规范[8]。

5 比较与讨论

元模型与相关 API 和 UML 之间的联系与区别。

5.1 镜像 API

Java SE5 提供了镜像(mirror)API,对静态的、构建时刻的程序语义结构进行建模[11]。它对 Java 元建模具有重要参照作用。元模型与镜像 API 的主要区别在于,(1)元模型更简单。例如,镜像 API 中提供了 AnnotationDeclaration 来表示可带类属的注释类型的定义,又提供了 AnnotationType 表示参量化的注释类型。而实际上注释不能作为类属类型。(2)建模形式不同。镜像 API 主要以接口形式表示,接口中提供方法来间接地表示其特征;而 Java 元模型依据 MOF 规范,以元类的形式来进行建模,这样可直接描述元类之间的语义关联。

5.2 反射 API

Java2 SE 的反射(reflection)API 提供了动态的、运行时刻的类和对象的信息[12]。运行时刻的注释处理主要依赖反射 API,所以新的元模型也能反映运行时刻的动态特征,但具有更高的抽象级别。它与反射 API 的主要区别在于:

(1)类型结构。在反射 API 中,所有类型在运行时刻都是 Class 的对象,通过一组方法以确定运行类型究竟是类、枚举、接口还是注释。这不能反映 Java 类型的语义结构。元模型中建立了 Classifier 作为 Class 和 Interface 的抽象,枚举 Enum 作为 Class 的子类,注释 Annotation 作为 Interface 的子类,使类型结构保持一致性。

(2)抽象级别不同。注释类型在编译时转换为 java.lang.annotation.Annotation 接口的子接口,而枚举类型将转换为 java.lang.Enum 类的子类。元模型中 Enum 类作为枚举类型的元类,Annotation 作为注释类型的元类。虽然两者采用相同名称,但抽象级别不同。反射 API 中的 Enum 类、Annotation 接口都属于 M1 层(模型层)中的元素,而元模型中的 Enum 和 Annotation 属于 M2 层(元模型)的元素。

5.3 UML 元模型

Java 元模型作为一种平台相关模型,尽可能地接近

(下转第 267 页)

- lae. In CAV, 2000. 248~263
- 5 Gastin P, Oddoux D. Fast ltl to automata translation. In: CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification, London, UK, 2001. 53~65
- 6 Dill D L, Hu A J, Wong-Toi H. Checking for language inclusion using simulation preorders. In: CAV '91 Proceedings of the 3rd International Workshop on Computer Aided Verification, London, UK, 1992. 255~265
- 7 Etesami K, Wilke T, Schuller R A. Fair simulation relations, parity games, and state space reduction for Büchi automata. In: ICALP '01 Proceedings of the 28th International Colloquium on Automata, Languages and Programming, London, UK, 2001. 694~707
- 8 Henzinger T A, Kupferman O, Rajamani S K. Fair simulation. Inf. Comput., 2002, 173(1): 64~81
- 9 Andersen H R. Model checking and boolean graphs. Theor. Comput. Sci., 1994, 126(1): 3~30
- 10 Henzinger M R, Henzinger T A, Kopke P W. Computing simulations on finite and infinite graphs. In: FOCS '95 Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95), Washington, DC, USA, 1995. 453~462
- 11 Klarlund N, Kozen D. Rabin measures and their applications to fairness and automata theory. In: LICS, IEEE Computer Society, 1991. 256~265
- 12 Jurdzinski M. Small progress measures for solving parity games. In: Horst Reichel and Sophie Tison, eds. STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science. Proceedings, volume 1770 of Lecture Notes in Computer Science, Lille, France, February 2000. 290~301
- 13 Bustan D, Grumberg O. Simulation-based minimization. ACM Trans. Comput. Logic, 2003, 4(2): 181~206
- 14 Bustan D, Grumberg O. Applicability of fair simulation. Inf. Comput., 2004, 194(1): 1~18
- 15 Gurumurthy S, Bloem R, Somenzi F. Fair simulation minimization. In: CAV '02 Proceedings of the 14th International Conference on Computer Aided Verification, London, UK, 2002. 610~624
- 16 Juvekar S, Piterman N. Minimizing generalized Büchi automata. In: Thomas Ball, Robert B. Jones, eds. CAV, volume 4144 of Lecture Notes in Computer Science, Springer, 2006. 45~58

(上接第 259 页)

UML 元模型,目的是方便建立互转换机制,以实现 MDA。它与 UML 之间有以下区别:

(1)裁减。取消了许多 Java 中不需要的 UML 元类,而保留了 Java 必需的元类:Element,Classifier,Type,Class,Interface,Feature,BehavioralFeature,Parameter 等。

(2)改变与扩展。把 UML 中类似元类改变为 Java 名称,如 Field,Method,Constructor;根据 Java SE5 的新语言成分,扩展了枚举 Enum,注释 Annotation、可注释元素 AnnotatableElement、注释实例 AnnotationInstance 等,图 2 中也扩展了类属。注意本文仅对注释部分进行了详细描述。

本文提出的三个注释元类及两个图符对于 UML 扩展注释具有指导意义。UML 允许自定义构造型来扩展已有元类,以支持新的建模元素。关于如何扩展 UML 来支持注释建模,作为另一种技术途径另文再述。

结论与进一步工作 注释相对于注释,其特征是基于类型、静态实例化、关联目标的实例化、实例不变性。为达到注释的可视化、规范化建模的目的,本文以 MOF 为规范,参照 UML 元模型,扩展已有 Java 元模型,提出一个新的 Java SE5 元模型,并确定了注释的图符以支持可视化建模。该元模型能反映注释的特征,可支持注释的建模和编程。该元模型具有一致性、规范性、简单性。

进一步的工作包括扩展该元模型以支持 AOP(如 AspectJ5)及 AOM(Asspect-Oriented Modeling),研究动态注释及应用,实现元模型以开发 MDA 及相关建模工具等。

(上接第 263 页)

参考文献

- 1 Maes P. Concepts and experiments in computational reflection. ACM SIGPLAN Notices, 1987, 22(12): 147~155
- 2 Bates J. The role of emotion in believable agents. Communication of the ACM, 1994, 37(7): 122~125
- 3 Maes P. Agents the reduce work and information overload. Communication of the ACM, 1994, 37(7): 31~40
- 4 Kruchten P B. The 4+1 view model of architecture. IEEE software, 1995, 12(6): 42~50
- 5 Nwana H S. Software Agents: An Overview. Knowledge Engineering Review, 1996, 11(3): 205~244
- 6 Magee J, Kramer J. Dynamic structure in software architectures. ACM SIGSOFT Software Engineering Notes, 1996, 21(6): 63~

参考文献

- 1 Gosling J, Joy B, Steele G, Bracha G. The Java Language Specification (Third Edition)[M]. Addison-Wesley Professional; 2005 ISBN 0321246780
- 2 许满武, 严悍, 张琨, 李千目. Java 程序设计[M]. 北京: 高等教育出版社, 2006, ISBN 704019645X, 普通高等教育“十五”国家级规划教材
- 3 Harold E. An Early Look at JUnit 4 [OL]. <http://www-128.ibm.com/developerworks/java/library/j-junit4.html>, 2005. 9. 13
- 4 Hibernate Annotations Reference Guide, v3. 2. 0 [OL]. http://www.hibernate.org/hib_docs/annotations/reference/en/html/, 2006. 4
- 5 Kiczales G, Mezini M. Separation of Concerns with Procedures, Annotations, Advice and Pointcuts [C]. In: proceedings of ECOOP 2005. Glasgow, UK, July 2005
- 6 Yan H, Kniessel G, Cremers A B. A Meta Model for AspectJ [R]. Technical Report IAI-TR-2004-3, October 2004, Computer Science Department III, University of Bonn, Germany, ISSN 0944-8535
- 7 Java Document. Getting Started with the Annotation Processing Tool (apt) [OL]. <http://java.sun.com/j2se/1.5.0/docs/guide/apt/GettingStarted.html>, 2005
- 8 OMG(Object Management Group), Unified Modeling Language: Specification; Superstructure [EB]. version 2. 0, 2005. 7. 4
- 9 OMG(Object Management Group), Unified Modeling Language Specification; Infrastructure [EB]. version 2. 0, 2004. 10. 16
- 10 Kleppe A, Warmer J, Bast W. MDA Explained: The Model Driven Architecture—Practice and Promise [M]. Addison-Wesley Professional; 1st edition (April 25, 2003) ISBN: 032119442X
- 11 Java Document. Mirror API [OL]. <http://java.sun.com/j2se/1.5.0/docs/guide/apt/mirror/index.html>, 2005
- 12 Java Document. Reflection API [OL]. <http://java.sun.com/j2se/1.5.0/docs/guide/reflection/index.html>, 2003

83

- 7 Richard N T. A Component and Message-Based Architectural Style for GUI Software. IEEE Transaction on Software Engineering, 1996, 22(8): 390~406
- 8 Wooldridge M. Agent-based software engineering. IEEE Proc. on Software Engineering, 1997, 144(1): 26~37
- 9 Perkowitz M, Etzioni O. Adaptive web pages: Automatically synthesizing web pages. In: Proceedings of AAAI/IAAI' 98, 1998. 727~732
- 10 Wermelinger M. Towards a chemical model for software architecture reconfiguration. IEEE Proc-Softw, 1998, 145(5): 130~136
- 11 Oreizy P, Taylor R N. On the role of software architectures in runtime system reconfiguration. IEEE Proc-Softw, 1998, 154(5): 137~144