Atitit.继承的实现原理 Mix-in功能

在基于Java等语言中常见的类的程序设计中，通过多个类的协作实现某一种功能时，类就无法进行再利用。因此应该有一个与类不同的、可以再利用的编程单位”。

**继承装配**

|  |
| --- |
|  |
| 图7●Java只允许继承一个装配。因此就必须完整地复制代码（点击放大） |

    不过，基于接口的伪多层继承存在致命的缺点。这就是指接口无法继承装配。Ruby语言的**开发**者--日本的松本幸弘指出：“使用接口取代多层继承是Java的高明之处。不过放弃装配的继承则令人非常痛心”。

    继承2个类的功能时，Java将会复制某一个类的源代码。结果，相同的代码就会分散在各种不同的位置，可维护性就会显著降低（图7）。将相同的处理过程集中到名为类的单位中来提高面向对象的优点没有得到充分的利用。“从市场销售上来讲，Java是一种取得了巨大成功的语言。但是从编程语言的装配角度来看，我认为它是一个失败的作品”（松本）。

**Ruby语言利用Mix-in实现装配的继承**

    实际上松本开发的Ruby语言也和Java一样只允许单继承。不过，Ruby则利用相当于Java接口的方法实现了装配的继承。这就是被称为Mix-in的方法。

    Mix-in是指仅将程序中具有再利用性的功能部分集中到被称为“模块”的单位中，以供其他的类来使用。模块与类一样，其本身无法利用new运算符来生成。只有“include”即嵌入到其他的类中才能使用（图）。

    LIST 6就是使用了Mix-in方法的简单Ruby代码。定义了一个可以输出“test”字符串的、名为TestModule的模块。TestClass类嵌入了该模块。这样一来，TestClass类就可以像调用自身的方法一样调用TestModule模块中的方法。

    不过，Ruby的Mix-in所解决的只是由多层继承所造成的继承关系的复杂性。在多个模块存在同名的方法时，就会产生与多层继承相同的问题。Ruby优先处理由后面嵌入的方法（注3）。

|  |
| --- |
|  |
| 图8●Ruby中的Mix-in功能。将集中了各种功能的模块嵌入到类中来使用（点击放大） |

|  |
| --- |
|  |
| LIST 6●使用Ruby中Mix-in功能的代码。TestClass类可以像使用自己的方法一样使用TestModule模块所装配的方法（点击放大） |

**提出了新的分割单位的、面向侧面编程（Aspect-Oriented programming）**

    像MixJuice语言那样，利用横跨多个对象的单位来把握系统的观点称为“关注分隔（Separation of Concerns，SoC）”。类的相互作用也属于关注的一种。这种观点并不是仅仅单纯以对象为单位，还要由其他侧面来分割系统。

    最近这种观点已经开始受到越来越广泛的关注。基于该观点的代表性编程范式（paradigm）就是面向侧面编程。它就是以“关注”为分隔单位的。如果存在多个与类相同的关注，就通过将这些关注组合起来，实现一种功能。这种功能就是侧面（Aspect）。

    图11就是具体的例子。把移动图形来刷新画面的处理过程定义为一个侧面。图形是由点和线组成的。每一个点和线中均存在移动的方法和用于设置位置的方法。这些动作相互协作，就可实现称为画面刷新的侧面。

    LIST 8就是使用基于Java的面向侧面语言“AspectJ”，来描述侧面的源代码。在一个被称为移动图形的程序段中定义了多个类的多个方法。图形的移动结束后，最后执行重新刷新画面的处理过程。

[TR][TD][TR][TD]图11●利用“关注”分割“类”的图示。点和线以及他们各自所具有的定位和移动的方法被划分成了相同的关注。该图摘自面向侧面编程的倡导者Gregor Kiczales在面向侧面编程技术研讨会上发表的演讲资料（点击放大）

|  |
| --- |
|  |

[TR][TD][TR][TD]LIST 8●利用AspectJ描述侧面的源代码。将用于实现图像移动的各种类中的方法归纳为“move”。“move”结束以后，执行由“after”描述的画面刷新。摘自Gregor Kiczales在面向侧面编程技术研讨会上发表的演讲资料（点击放大）

|  |
| --- |
|  |

**与面向对象并不矛盾**

    面向侧面编程在对象以外导入了分割系统的单位，提高了程序的再利用性和扩展性。如果只有对象单位有分割的轴，也许确实就会存在一定的限制吧。

    不过，面向对象本身就是公认难度很高的编程范式。尽管如此面向对象语言能够普及到如此程序恐怕是因为“只有对象单位有分割的轴”吧。正因为轴只有一个好歹才能够理解。如果还要加上被称为侧面的轴，也许有很多**程序员**就会产生混乱。

    另外，面向侧面则是自由度很高的编程范式。正是因为自由度高，程序员才会感到苦恼。因为他们不知道应该如何分割或组织程序。也就是说，在某种程度上缩小自由度，并给程序员提供编程指南也可以说是编程语言的任务。

    面向侧面是一种扩展了面向对象的编程范式。如果认为面向对象的自由度合适，就可以继续使用它。就连面向侧面的倡导者加拿大英属哥伦比亚大学的Gregor Kiczales本人也表示：“没有必要非要去勉强地使用侧面来编程”。程序员要根据自己的开发风格，在追求一种易用性和扩展性适当平衡的同时，使用这种编程范式。（记者：大森 敏行、八木 玲子）

Java中鲜为人知的缺点（下）\_Java - 百科教程网\_经验分享平台[上学吧经验教程频道].htm