ref: <http://wiki.jikexueyuan.com/project/java-design-pattern/visitor-discuss.html> （卡奴达摩）

访问者模式讨论篇：java的动态绑定与双分派

#### **1·java中的动态绑定**

所谓的**动态绑定**就是指在程序执行期间（而不是在编译期间）判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

**java继承体系中的覆盖就是动态绑定的**，看一下如下的代码：

|  |
| --- |
| **class** Father {  **public** **void** method() {  System.***out***.println("This is Father's method");  }  } |
| **class** Son1 **extends** Father {  **public** **void** method() {  System.***out***.println("This is Son1's method");  }  } |
| **class** Son2 **extends** Father {  **public** **void** method() {  System.***out***.println("This is Son2's method");  }  } |
| **public** **class** Test {  **public** **static** **void** main(String[] args) {  Father s1 = **new** Son1();  s1.method();  Father s2 = **new** Son2();  s2.method();  }  } |
| ***执行结果 :***  This is Son1's method  This is Son2's method |
| 通过运行结果可以看到，尽管我们引用的类型是Father类型的，但是运行时却是调用的它实际类型（也就是Son1和Son2）的方法，这就是动态绑定。 |

#### **2·java中的静态绑定**

相对于动态绑定，**静态绑定**就是指在编译期就已经确定执行哪一个方法。

在java中，方法的重载（方法名相同而参数不同）就是静态绑定的，重载时，执行哪一个方法在编译期就已经确定下来了。看一下代码：

|  |
| --- |
| **class** Father {}  **class** Son1 **extends** Father {}  **class** Son2 **extends** Father {} |
| **class** Execute {  **public** **void** method(Father father) {  System.***out***.println("method(Father father) invoked !");  }  **public** **void** method(Son1 son) {  System.***out***.println("method(Son1 son) invoked !");  }  **public** **void** method(Son2 son) {  System.***out***.println("method(Son2 son) invoked !");  }  } |
| **public** **class** Test {  **public** **static** **void** main(String[] args) {  Father father = **new** Father();  Father s1 = **new** Son1();  Father s2 = **new** Son2();  Execute exe = **new** Execute();  exe.method(father);  exe.method(s1);  exe.method(s2);  }  } |
| ***执行结果:***  method(Father father) invoked !  method(Father father) invoked !  method(Father father) invoked ! |
| 在这里，程序在编译的时候就已经确定使用method(Father father)方法了，不管我们在运行的时候传入的实际类型是什么，它永远都只会执行method(Father father)这个方法。也就是说，java的重载是静态绑定的。 |

#### **3·instanceof操作符与转型**

有时候，我们希望在使用重载的时候，程序能够根据传入参数的实际类型动态地调用相应的方法，也就是说，**我们希望java的重载是动态的，而不是静态的**。但是由于java的重载不是动态绑定，我们只能通过程序来人为的判断，我们一般会使用instanceof操作符来进行类型的判断。我们要对method(Father father)进行修改，在方法体中判断运行期间的实际类型，修改后的method(Father father)方法如下：

|  |
| --- |
| **public** **void** method(Father father) {  **if** (father **instanceof** Son1) {  method((Son1) father);  } **else** **if** (father **instanceof** Son2) {  method((Son2) father);  } **else** **if** (father **instanceof** Father) {  System.***out***.println("method(Father father) invoked !");  }  } |
| 请**注意**，我们必须把判断是否是父类的条件（也就是判断是否为Father类的条件）放到最后，否则将一律会被判断为Father类，达不到我们动态判断的目的。 |
| ***执行结果:***  method(Father father) invoked !  method(Son1 son) invoked !  method(Son2 son) invoked ! |

修改代码后，程序就可以动态地根据参数的实际类型来调用相应的方法了。但是这种实现方式有一个明显的缺点，它是伪动态的，仍然需要我们来通过程序来判断类型。*假如Father有100个子类的话，还是这样来实现显然是不合适的*。必须通过其他更好的方式实现才行，我们可以使用双分派方式来实现动态绑定。

#### **4·用双分派实现动态绑定**

首先，什么是双分派？还记得[23种设计模式（9）：访问者模式](http://wiki.jikexueyuan.com/project/java-design-pattern/visitor-pattern.html)中一开始举的例子吗？

类A中的方法method1和method2的区别就是，method2是双分派。

我们可以看一下java双分派的特点：首先要有一个访问类B，类B提供一个showA(A a) 方法，在方法中，调用类A的method1方法，然后类A的method2方法中调用类B的showA方法并将自己作为参数传给showA。双分派的核心就是这个this对象。说到这里，我们已经明白双分派是怎么回事了，但是它有什么效果呢？就是可以实现方法的动态绑定，我们可以对上面的程序进行修改，代码如下：

|  |
| --- |
| **class** Father {  **public** **void** accept(Execute exe) {  exe.method(**this**);  }  }  **class** Son1 **extends** Father {  **public** **void** accept(Execute exe) {  exe.method(**this**);  }  }  **class** Son2 **extends** Father {  **public** **void** accept(Execute exe) {  exe.method(**this**);  }  } |
| **class** Execute {  **public** **void** method(Father father) {  System.***out***.println("method(Father father) invoked !");  }  **public** **void** method(Son1 son) {  System.***out***.println("method(Son1 son) invoked !");  }  **public** **void** method(Son2 son) {  System.***out***.println("method(Son2 son) invoked !");  }  } |
| **public** **class** Test {  **public** **static** **void** main(String[] args) {  Father father = **new** Father();  Father s1 = **new** Son1();  Father s2 = **new** Son2();  Execute exe = **new** Execute();  father.accept(exe);  s1.accept(exe);  s2.accept(exe);  }  } |
| ***执行结果:***  method(Father father) invoked !  method(Son1 son) invoked !  method(Son2 son) invoked ! |

可以看到我们修改的地方，在Father，Son1，Son2中分别加入一个双分派的方法。调用的时候，原本是调用Execute的method方法，现在改为调用Father的accept方法。

运行结果符合我们的预期，与使用instanceof操作符的效果是一样的，也实现了动态绑定，但是双分派方式的可扩展性要好的多。

*双分派实现动态绑定的本质，就是在重载方法委派的前面加上了继承体系中覆盖的环节，由于覆盖是动态的，所以重载就是动态的了*。