第 6 章　软件重构与交付

主要内容

通过例子学习改善代码质量的重构技术和方法，学习软件打包和交付一个完整的软件，以及自动化软件构建的技术。作为提升，通过例子学习一种新型的软件开发方式——测试驱动开发，并了解软件交付的其他方式与基本活动。

故事 7

华经理换了一台计算机，把小强给的代码复制到新机器上，却不能正确地运行，就打电话告诉了小强。小强很快就明白了是应用软件的安装问题，即要将开发的程序的各个模块文件连同数据文件、使用的外部包等组装起来，成为一个完整的、在任何机器上都可运行的软件。若是 Java 应用程序，则还需要在用户的机器上安装 Java 运行时环境等。

另一方面，小强和小雨不断地按照用户的要求扩展了程序功能，给程序添加类、方法等。代码行数在不停地增大，程序的结构越来越不清晰。由于程序不如以前那样一看就明白，定位和修改程序错误或者在合适的位置给程序添加功能就变得越发困难、费时。而且，修改代码后的回归测试有不少重复工作，手工编写和执行测试也很耗费时间。他们需要采用更好的技术和工具。

6.1　代 码 重 构

增量迭代的开发方式不断地在增加编写的代码，由于是边思考、边设计、边构造，可能会造成许多模块有重复的代码片段或者在一个模块中临时命名了变量等，使得代码欠规范，结构不清晰。例如，第 5 章例 5.1 中，为了清晰地显示菜单，每个子菜单都要打印几行“----------”，提示用户“请选择”；在案例菜单构造的“联机操练习题”中，有三种类型的算式习题，产生这三种类似习题的方法不同，用户练习习题、系统批改习题并显示批改结果的构造没有差异。这些共同的、反复使用的代码应该提取出来成为独立的函数或方法，在简化代码结构的同时，也便于在一处集中修改。

在完成构造、交付代码前改进代码质量，有助于今后的软件维护和更新。代码重构是增量迭代开发不可或缺的技术，已经成为现代软件开发的基本技术，并在很多常用的 IDE 中成为标准模块。代码重构就是在不改变软件外部行为的前提下改善它的内部结构。下面以一个较小的例子说明代码重构的基本概念和方法。

6.1.1　代码重构的案例研究

【例 6.1】 图书馆显示借阅者还书的信息。假设一个学生免费借阅图书 30 天，超出的天数按照书价和书的类型，每超出一天罚金如下：教材 0.001 元，参考书 0.005 元，新书 0.01 元，基础罚金分别是 1、1.5、3 元。提前还书都有奖励积分，三种类型书的奖分分别是 1、2、3。奖分只能用于抵消罚金，7 分抵 1 元罚金。一次还书后显示如下的信息：

```
新书"Python 进阶"借阅了 12 天.
```

```
教材"Java 导论"借阅了 45 天.
参考书"C#秘笈"借阅了 38 天.
参考书"软件构造"借阅了 28 天.
总共缴纳罚金： 4.73 元.
还书奖励： 5 点.
```

程序共有三个类：学生 Student，图书 Book 和借阅 Rental，类图如图 6.1 所示。

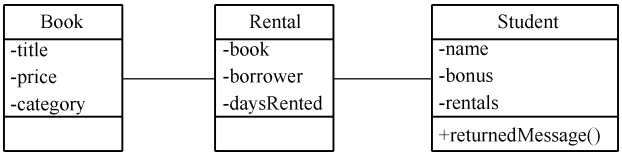


图 6.1 显示借阅者还书信息程序的类图

部分代码如下。显示借阅信息 returnedMessage()的代码在 Student 中，主要对它重构。

代码重构的例 6.1——图书馆显示借阅者还书

对象的时序图简明扼要地示意为了完成一个任务，相互沟通与交流的对象间的交互关系。方法 returnedMessage 中的交互图如图 6.2 所示，一个学生对象通过向借阅对象请求借阅的书籍，获得图书的价格和种类，再从借阅对象获得借阅时间，计算出自己所有借阅的罚金、奖励积分等，以字符串的形式返回借阅信息。

时序图是 UML 的一种动态交互图。主要成分是用方框表示的对象，在对象名称下加下画线。对象具有生存周期，用从对象图标向下延伸的一条虚线表示对象存在的时间。对象之间的交互用消息发送表示，它是一个从请求对象向服务对象标记发送消息名称的有向连线（如 getBook，它是 Rental 类实现的一个方法），对象也可以向自己发送消息。自上而下的消息连线表示消息发送的时间顺序。控制焦点是顺序图中表示时间段的符号，在这个时间段内对象将执行相应的操作，用小矩形表示。

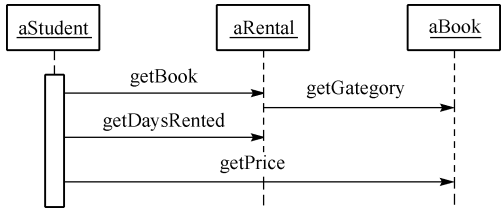


图 6.2 方法 returnedMessage 中的交互图

在下面代码重构过程中，我们将借助类图和交互图显示重构对代码结构的改变。

构造一些实验数据，代码 returnedMessage 能正确地显示期望的消息，满足了要求的功能。

```
//代码 6.1
public class Example_6_1 {
    public static void main(String[] args) {
       Book aBook;
       Student student = new Student("zhangsan");

       aBook = new Book("Python 进阶",35.5);
       student.addBook(aBook,12);
       aBook = new Book("Java 导论",37.5,1);
       student.addBook(aBook,45);
```

```
            aBook = new Book("C#秘笈",41.7,3);
            student.addBook(aBook,38);
            aBook = new Book("软件构造",29.8,3);
            student.addBook(aBook,28);

            System.out.println(student.returnedMessage());
        }
    }
```

但是，returnedMessage 的代码过于杂乱，难于调试、测试、维护和扩展，也没有办法在其他地方复用其中的某个功能。例如，如果添加一种书的类型，就需要添加一个条件和分支语句，依据新的计算规则实现罚金等计算；或者想改变某个罚金或奖励积分规则。这都需要改写代码，重新编译该方法和整个程序，然后还要调试和测试。在这样的一个程序中定位错误，对程序员是个挑战。显然，方法 returnedMessage 处理的工作超出了 Student 类的职责，也超出了一个方法应该实现的功能。下面从简单到复杂，逐步通过代码重构改进代码的质量。

1．重构大函数

1）运用重构“提炼方法”（Extract Method），把从 switch 语句开始的计算罚金与奖励积分部分设计成一个方法 calculateFineAndBonus(Rental aRental)。

如果一个方法的代码太长或者代码需要很多注释才能理解其意图，可用一个能说明其意图的方法替换那些代码。运用“提炼方法”抽取代码时，要使用良好的编码风格设计方法名和参数，允许读者在较高的函数级别理解程序，提高程序的阅读性。“提炼方法”的步骤如下。

（1）设计一个新方法，并按照提炼代码的意图给方法命名。提炼的代码可长可短，方法名一定要能体现其含义，否则就不能提炼出方法。

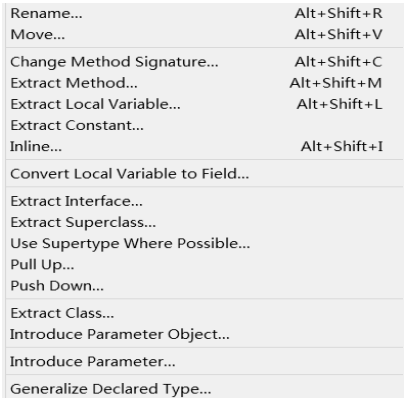
（2）把原来方法中要提炼出的代码直接复制到新的方法中（目标方法）。

（3）检查目标方法中的局部变量、引用变量和临时变量，做出相应修改，有时要运用“移除临时变量”或“分解临时变量”的策略。

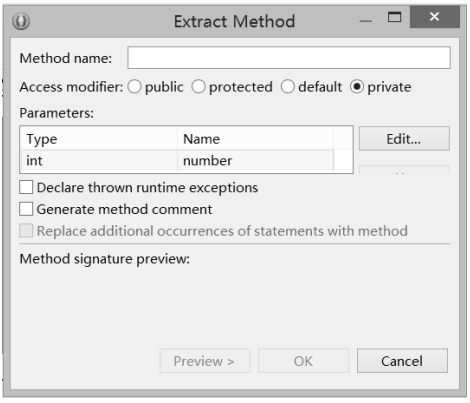
（4）在源方法中调用目标方法，取代提炼的代码。

（5）通过编译和测试。

如果使用 Eclipse，它的基本配置包含重构功能 Refactory，有些重构方法可以用工具简化。选择从语句“bonus = 0;”到“addBonus(bonus)”之间的代码，右击，选择 Refactory→Extract Method，在出现的对话框中输入一个方法名，确定后得到重构后的代码。在 Eclipse 中使用代码重构如图 6.3 所示。



(a) Eclipse 提供的重构策略



(b)“提取函数”的重构窗口

图 6.3　在 Eclipse 中使用代码重构

```
//代码 6.2: 提出方法 calculateFineAndBonus(aRental)后的部分代码
public String returnedMessage(){
        double totalAmount = 0;
        double finedAmount=0;
        int bonus = 0;  //Eclpise 提示没有使用过该变量
        String message=new String();
            for (Rental aRental:rentals){
                finedAmount = calculateFineAndBonus(aRental); //重构后的语句
                totalAmount += finedAmount;
                while (getBonus()>=7 && totalAmount>1){…}
                message += aRental+"\n";
            }
            …
            return message;
    }
    //重构提出的方法
    private double calculateFineAndBonus (Rental aRental) {
        double finedAmount;  //Eclipse 自动增加的变量声明
        int bonus;
        …
        switch(aRental.getBook().getCategory()){…}
        addBonus(bonus);
        return finedAmount;  //Eclipse 自动增加的返回语句
    }
```

2）运用“移除临时变量”，消除 returnedMessage 中的临时变量 bonus。再次运用提炼方法，将 calculateFineAndBonus 中的 else 分支语句都改成方法 addBonus(aBonus)，参数分别是原来赋值语句右边的值，即分别是 1、2、3。目的是便于今后改变奖励积分的策略。执行前面的测试，回归测试的结果不变。

3）方法 returnedMessage 中的临时变量 finedAmount 只被使用了一次，而且它的值是通过函数调用或表达式计算得到的。这时可以运用重构“以查询取代临时变量”消除，得到如下代码。

```
//代码 6.3
public String returnedMessage(){
        double totalAmount = 0;
        String message=new String();
          for (Rental aRental:rentals){
           totalAmount += calculateFineAndBonus(aRental);  //重构后改变的语句
           …
         }
         …
    }
```

使用“以查询取代临时变量”时要注意提炼的函数没有副作用，即它只是单纯的计算，没有改变任何对象、任何非局部变量的值。其次，确保代码中被删除的临时变量只赋值一次，否则就不能删除。

重构后，类图没有发生重大变化，在 Student 中新增了一个方法 calculateFineAndBonus，其类图如图 6.4 所示。

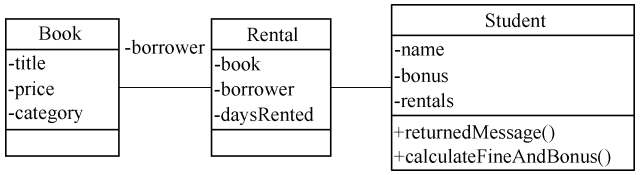


图 6.4　提炼出方法 calculateFineAndBonus 的类图

4）罚金的计算实质上是“借阅”对象的责任，计算所需要的信息都在“借阅”对象中，与借阅者 Student 无关，因而可以用“函数移动”的策略，把罚金计算方法 calculateFineAndBonus()从 Student 移动到 Rental。这时方法不再需要参数 aRental，在 returnedMessage 中变成向 aRental 发送计算罚金的消息。同时还要改变奖励积分 addBonus 的计算：积分是奖励 Student 类的，是在该类实现的。现在整个方法 calculateFineAndBonus 移到了类 Rental，积分的计算要向借阅者——即 Student 对象发送请求 getStudent().addBonus() 来实现。

```
//代码 6.4: 重构 4
  sspublic String returnedMessage(){
        double totalAmount = 0;
        String message=new String();
        for (Rental aRental:rentals){
            totalAmount += aRental.calculateFineAndBonus();  //重构的语句
//方法 calculateFineAndBonus()移到类 Rental
class Rental {
  public double calculateFineAndBonus(){
        double finedAmount=0;
        switch(getBook().getCategory()){
            case Book.TEXT_BOOK:
                if (getDaysRented()> 30){
                    //重构后改变的语句
                    finedAmount += (getDaysRented()-30)*getBook().getPrice()*0.001;
                    finedAmount += 1;
                } else {
                    getStudent().addBonus(1);    //重构后改变的语句
                }
                break;
            …
                }
    }
```

当一个类具有太多的行为或者一个方法的实现有多个类参与、耦合性较大时，应该把方法从一个类移到使用了更多特性的另一个类中。罚金的计算与书籍借阅的时间及书籍的信息密切相关，这些信息主要在类 Rental 中，而不在 Student 内，所以把计算的方法放到类 Rental 中更合适。“函数移动”的基本步骤如下。

（1）检查被源方法使用的、在源类中定义的所有的特性，考虑是否要移动。如果一个特性仅仅被想要移动的方法使用，可能就需要移动。如果特性被其他方法使用，也考虑移动这些方法。有时一次移到一组相关的方法更容易实现。

（2）检查源类的子类和超类是否声明了要移动的方法。如果有其他的方法申明，则不能移动，除非在目标类中能够实现多态。

（3）选择一个更合适的方法名，在目标类中定义。同时，把源方法的代码复制到目标方法中，然后适当地调整代码。如果移动的方法要使用原来的资源，要考虑如何从目标方法中引用源对象，例如，可以把源对象作为参数传递给新方法。如果源方法有异常处理，则要决定哪个类处理异常。

（4）决定如何从源代码中正确地引用目标对象。也许源代码中有现成的变量或方法可以访问目标对象，否则，看能否简单地创建一个类似的方法或者在源代码中新建一个成员变量存储目标对象。

（5）把源方法改成派遣方法。例子中就是把 calculateFineAndBonus(aRental)中的参数作为发送消息的（派遣）对象，即改为 aRental .calculateFineAndBonus()。

（6）决定是否删除源方法或者把它留作派遣方法。如果移除源方法，则要把它的所有引用都替换成引用目标方法。

（7）通过编译并测试。

重构后代码结构图如图 6.5 所示。

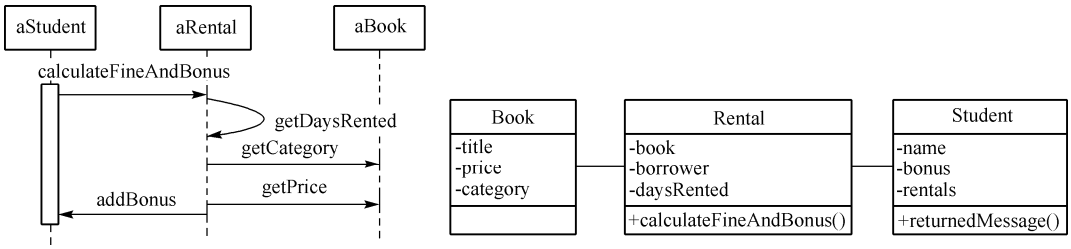


图 6.5　函数移动等重构后的类图及 calculateFineAndBonus 的交互图

2．用多态替换分支语句

改造最为复杂、庞大的代码 calculateFineAndBonus()中的分支语句。分支语句的形成是因为罚金和奖励积分的计算与书籍借阅的天数、书籍价格、特别是书籍种类等多个条件有关。重构这个方法可以使用多种技术，如“用类替换类型码”、“用多态取代类型码”、“用状态/策略模式取代类型码”或“用子类取代类型码”。

1）用类替换类型码。使用符号名称或枚举类型表示不同的书籍种类，进而实现不同的计算操作，这是面向过程式语言（如 C 语言）最基本的特性，也是分支语句的典型应用场景，它提高了代码的可读性。但是，编译程序在分析程序、检查类型到分支语句的条件时，检查的是符号名称后面隐含的数值，而不是符号名称。任何一个类型码作为参数的函数期待的是数值，没有任何机制强迫使用符号名称，所以，符号名称可能成为错误源。如果用类替换了数值，编译就检查类；程序员也可以静态地检查只能产生有效的对象，也只有这些对象可以作为参数传递给其他对象。

使用这些策略重构 calculateFineAndBonus 的步骤如下。

（1）为类型码创建一个类。这个类需要一个编码字段来匹配类型码，字段要有访问类型值的方法。对于类允许的实例，应该有一个静态变量及一个静态方法，它根据原来的编码参数返回一个恰当的实例。

（2）修改原先类的实现，使其使用这个新类。保持基于编码的接口，使用新类产生编码来更改静态字段。更改其他基于编码的方法，从新类得到编码数值。

（3）对每个使用编码的源类的方法，创建使用新类的一个新的方法。使用了以编码作为参数的每个方法都要一个以新的类的对象作为参数的新的方法。同样，所有返回编码的方法也要一个方法返回编码。

（4）逐个更改每个使用源类的代码，使它们使用新的类。

（5）删除使用编码的旧的接口代码及编码的静态声明。

（6）编译并测试所有的更改。

2）用子类取代类型码。如果类型码不影响程序的行为，则在面向对象中使用“用类替换类型码”的策略。但是，如果类型码影响程序行为，最好使用多态来处理多变的行为。程序中有分支条件语句就表明这种情形的出现。这时，首先要检测编码的值，然后根据不同的编码值执行不同的代码。重构这些条件可以采用“用多态取代类型码”，即用具有多态行为的继承结构取代类型码，为每个类型码编写一个子类。实现继承结构的最简单的途径是“用子类取代类型码”。为类型码设计一个类，为每个类型值设计一个子类。“用子类取代类型码”是实现“用多态取代类型码”的基础。如果出现了仅仅与特定类型码的对象相关的特征，也可以使用“用子类取代类型码”，同时，可以使用“下移函数”和“下移字段”来清楚地表明这些特征仅仅与特定的类相关。

“用子类取代类型码”的优点是，把变化行为的知识从类的使用者搬迁到类本身。要是增加新的变化，只需增加子类。若编程语言没有多态机制，就只能检查所有条件、改变检测的条件，所以，“用子类取代类型码”的重构特别适合代码扩展，支持变化。

3）用多态取代类型码。基本思路是把原先的方法设计成抽象方法，在子类中重载每个分支条件。如果不同对象的行为根据条件发生变化，多态就可以避免编写这些条件。在设计良好的面向对象程序中，应该很少出现根据类型码决定行为的分支语句或 if-then-else 语句。

多态的优点：避免在程序中多处出现一组相同的条件，容易增添新的类。这时，可以设计新的子类来提供合适的方法，而类的使用者无须知道子类的存在，从而减少了程序的耦合性，便于更新和扩展。下面就运用“用多态取代类型码”改造方法 calculateFineAndBonus，消除其中的分支语句。

（1）把类 Book 改造为抽象类，新增三个子类 NewBook、Reference、TextBook。

（2）删除类 Book 中的三个静态符号常量及类型变量 int category。

（3）完成新增类的构造方法、成员变量访问方法是 toString。重构后的程序结构如图 6.6 所示。

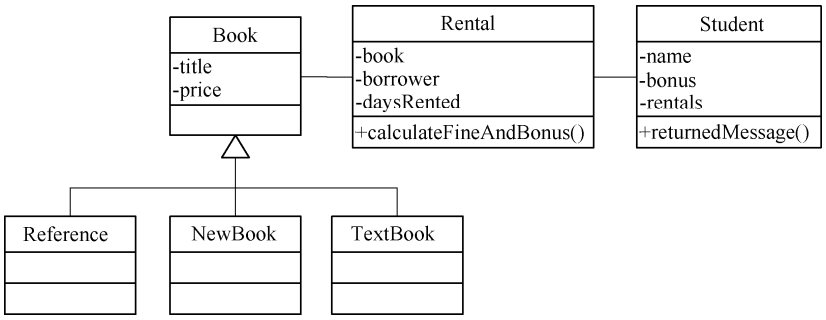


图 6.6　增加了 Book 子类后的类的结构图

（4）观察 Rental 中 calculateFineAndBonus 的任何一个分支：计算与书籍类型有关，语句 finedAmount += (getDaysRented()-30)*getBook().getPrice()*0.001;中的罚金系数与书籍的价格和类型有关。可以运用“提炼函数”的方式，用 getFine()替换表达式 getPrice()*0.001。而且，在基类 Book 中将其声明为抽象方法，三个子类分别实现它，采用不同的罚金系数 0.001、0.005 和 0.01。

（5）同样处理计算基本罚金的语句 finedAmount += 1，不同类型书籍的基本罚金不同。重构时增加一个方法 int baseFine()。

重构后的部分代码如下：

```
//代码 6.5: 重构 calculateFineAndBonus
//Book 改变为抽象类，增加子类增加
public abstract class Book{
    private String title;
    private double price;
    …
    public String toString(){
        return getCategory()+"《"+getTitle()+"》";
    }
    abstract double getFine();      //新加的方法
    abstract double baseFine();     //新加的方法
    abstract String getCategory();
    abstract int getBonus();
}
class TextBook extends Book{
    public TextBook(String name, double aPrice){
        super(name, aPrice);
    }
    public double getFine(){
        return getPrice()*0.001;
    }
    public String getCategory(){
        return "教材";
    }
    public double baseFine(){
        return 1;
    }
}
…
}
```

简单测试改动的 Book，如为一个学生创建 4 个不同的 Book 对象，打印借阅清单，同时要修改 Rental 中的字符串转换方法。

```
//代码 6.6: 测试驱动代码
public class BookTester {
    public static void main(String[] args) {
       Student student = new Student("zhangsan");
       test_case_one(student);
    }
    private static void test_case_one(Student student) {
        Book aBook;
        aBook = new TextBook("Python 进阶",35.5);
        student.addBook(aBook,12);
        …   //同前面的测试代码
```

```
            //简单的循环测试
            for (Rental aRental:student.getRentals()){
                System.out.println(aRental);
          }
        }
    }
    class Rental(){
        …
    public String toString(){
            return book.toString()+"借阅了"+daysRented+"天.";
        }
    }
```

运行结果如下：

```
教材《Python 进阶》借阅了 12 天.
新书《Java 导论》借阅了 45 天.
参考书《C#秘笈》借阅了 38 天.
参考书《软件构造》借阅了 28 天.
```

（6）方法 calculateFineAndBonus 从名称上也可以看出是完成了两个功能：计算罚金、计算奖励积分。首先是“函数更名”。名称是反映变量、函数、类、文件等含义的基本手段。复杂的函数名已经暴露出了函数代码松散的内聚性，即函数代码实现了过多的功能。把这个方法分裂成两部分：计算罚金和计算奖励积分。其中，计算罚金 getFine()的方法在 Rental，而奖励积分的计算依旧在 Student 中，因为积分属于学生而不是借阅。同时在 Book 中增加一个奖励积分的方法 int getBonus()，在其子类中实现该方法。

重构 getFine()的基本步骤：把 calculateFineAndBonus()中计算罚金的代码复制到 getFine()，其他部分不变。所有调用 calculateFineAndBonus()的代码都替换成 getFine()后即可删除。同时修改 else 分支的语句。重构后有关代码如下。

```
//代码 6.7: 在 Rental 用 getFine 替换 calculateFineAndBonus()
public double getFine(){
        double finedAmount=0;
        if (getDaysRented() > 30){
            finedAmount += (getDaysRented()-30)*getBook().getFine();
                    //重构的地方
            finedAmount += getBook().baseFine();
        } else {  //重构的地方,要在 Book 增加 getBonus(),分别返回 1、2、3
            borrower.addBonus(getBook().getBonus());
        }
        return finedAmount;
    }
//类 NewBook
    public double getBonus () {
        return 3;
    }
```

（7）考虑方法 Student 中的 returnedMessage，它通过 for 循环语句对每个借阅对象计算罚金和奖励积分。可以运用函数抽象把循环封装到一个方法中。由于 for 循环语句中还包含不确定的循环语句，抽象方法时要特别注意。一个基本原则是首先分离与循环变量无关、与其他变量无关的计算，然后再抽取与嵌套循环有关的、非独立的变量的计算。

首先构造相对独立的方法 getRentalList()，它产生显示退还的借阅清单，其中要对 Book 及其子类、Rental 分别实现 toString()。其次，构造方法 getTotalFine()和 getTotalBonus()。通过奖励积分抵消罚金的代码封装到方法 discountedBonus()中。最后，回归测试确保新增和重构的代码都要通过编译、产生正确的结果。重构后的部分代码如下。

```
//代码 6.8: 用函数抽象把循环封装到一个方法中
  //构造还书的清单,refactor 6.1
  public String getRentalList(){
        String result = "";
        for (Rental aRental:rentals) {
            result += aRental+"\n";
        }
        return result;
}
//构造所有罚金的方法,refactor 6.2
public double getTotalFine(){
    double totalAmount = 0;
        for (Rental aRental:rentals){
        totalAmount += aRental.getFine();
    }
    return totalAmount;
}
//奖励积分抵消罚金的代码,refactor 6.3
public double discountedBonus(double totalAmount){
    while (getBonus()>=7 && totalAmount >1){
        addBonus(-7);
        totalAmount --;
    }
    return totalAmount;
}
public String returnedMessage(){
    String message = getRentalList();
    double totalAmount = getTotalFine();
    totalAmount = discountedBonus(totalAmount);
    message += String.format("缴纳罚金:%.2f 元.\n",totalAmount);
    message += "还书奖励:"+getBonus()+"点.\n";
    return message;
}
```

这个重构表面上增加了代码，但是结构更加清晰，特别是主要方法 returnedMessage 清楚地列出了操作流程，每个操作动作相互独立，可分别编写或修改。

（8）最后，对通过了增式集成与重构的代码进行综合测试，包括回归测试、功能确认测试。

重构后的类图如图 6.7 所示，returnedMessage 及其中计算罚金、计算奖励积分的交互图如图 6.8 所示。

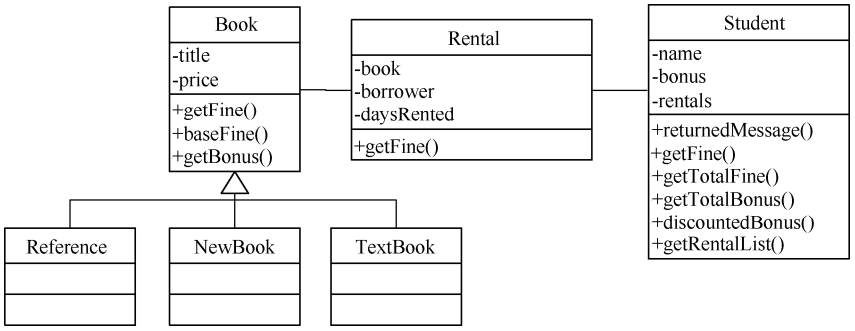


图 6.7　完成重构后的类图

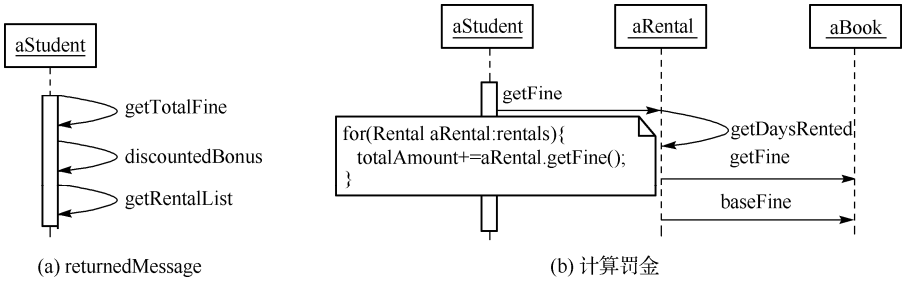


图 6.8　完成重构后的交互图

代码重构后，原先复杂的功能划分到若干相互交互的、短小的方法中。程序具有面向对象的特点：庞大复杂的功能分解到类，通过对象之间的协作完成；功能单一的代码容易复用；松散耦合的类容易扩展和维护。例如，想改变教材类书籍的罚金计算规则，只需在 TextBook 中更改 baseFine 或 getFine（分别只有一个简单表达式），重新编译和测试，而无须改动其他类。

再如，如果想根据学生类型，如本科生、研究生或博士生，制定不同的免费借书时间，不再是统一的 30 天，而分别是 30 天、50 天和 60 天。在代码重构之前，需要在不同类型书籍的条件分支中再增加三个类型的学生，计算罚金和奖励积分，这使得代码更加复杂。现在，可以把 Student 作为抽象的基类，再分别增加三个子类 UnderGraduate、PostGraduate 和 Doctorand，实现抽象方法 freeDays()，分别返回 30、50 和 60 作为免费借阅天数。扩展的相关代码如下。

```
//代码 6.9: 扩展类 Student,使其子类分别具有不同的免费借阅天数,然后修改 Rental 的方法
//类 Rental
public double getFine(){
        double finedAmount=0;
        if (getDaysRented() > 30){
            //替换: finedAmount += (getDaysRented() - 30) * getBook().getFine();
            finedAmount += (getDaysRented() - getStudent().freeDays()) *
                           getBook().getFine();
            finedAmount += getBook().baseFine();
        }
        return finedAmount;
}
```

对这个扩展的构造留做练习。

6.1.2 代码重构概述

1. 重构的基础

在极限编程或其他敏捷方法学中，重构常常是软件开发循环的一部分：开发者轮流增加新的测试和功能，重构代码来增进内部的清晰性和一致性。自动化的单元测试保证了重构不至于让代码停止工作。

重构既不修正错误，也不增加新的功能性，主要用于提高代码的可读性或改变代码内部结构与设计，使其更容易维护。重构代码可以是结构层或语意层，不同的重构手段可能是结构的调整或语意的转换，但前提是不影响代码在转换前后的行为。特别是在现有的程序结构下，给一个程序增加一个新的行为可能会非常困难，因此开发人员可能先重构这部分代码，使加入新的行为变得容易。

代码重构主要有三个时机：给程序增量地添加功能时、定位错误时或评审代码时。造成重构的主要原因是：改进软件设计、使软件更容易理解、有助于查找错误。下面列出代码重构的主要原因清单和重构的基本方法。

重构的基本方法

2. 再识重构

清理代码、提交高质量的代码是程序员的日常工作之一，是每个程序员必须做的、基本的工作。重构代码把常见的、证明切实有效的重构模式——重构的目标和重构的步骤——进行归档分类，形成了软件开发的最佳实践。重构代码时要记住以下三点。

（1）不要为了重构而重构。重构可以被当成一种能给代码变更带来帮助的措施。代码重构应该在进行代码变更前进行，这样能使程序员确信理解代码了，从而更容易、更安全地把变更引入代码。对重构动作一定要进行回归测试，然后纠正或变更，再次测试。之后可能需要对更多的代码进行重构，使代码变更的意图变得更加清晰。再次进行全面测试。

不是为了重构而重构，重构是因为想做其他的事情，而重构能帮助你完成这些事情。重构的范围应该由需要实施的代码变更或代码修正来决定——为了让代码变得更安全和更简洁。换句话说，不要对那些不打算进行变更或不会变更的代码进行重构。

（2）为理解而做简略重构。简略重构就是为理解而重构。这是用来对付那些不理解的（或不能忍受的）代码，清理它们。在计划真正动手修改前，能对代码的意图或功能有更好的理解，同时也有助于调试代码。一旦弄清楚了一个变量或方法的真正意图，给它们重命名，删除那些重复的或无用的代码，拆解复杂的条件语句，把长的程序分解成数个容易理解的小程序。

（3）不要顾虑复查或测试对代码的改动。这是为了快速推进重构——这能让代码及它们的运行原理在头脑中产生一个快速但不完备的原型。简略重构还能让程序员尝试各种不同的重构途径，学到更多的重构技巧。在这个过程中要留意那些看起来没什么用处或者特别有用的东西。这样，当真正需要修改代码时，才能把事情做正确。一点一点地完善程序，边修改边测试。

6.2　软 件 交 付

华经理遇到的问题，实际上是如何把在开发者机器上开发的应用程序安装到用户的机器上，交付给用户去运行。按照软件生命周期模型，软件开发之后的阶段是维护。如何平滑地连接这两个阶段，把开发的软件投入市场或提供给用户使用，尚缺乏充实的研究，也没有公认的标准、规范和流程。本节将概括地介绍与这个阶段相关的一些术语及活动。

6.2.1　构建与打包

把软件开发和软件维护这两个阶段之间的开发活动统称为软件交付。其作用是让最终用户使用开发的软件。软件交付的基本活动包括构建、打包、发布、安装和部署。

1．开发环境和运行环境

软件交付的核心是把程序从开发者的机器上迁移到用户的机器上。更一般地，应用程序不是在一个计算机裸机上运行的，而是在包含计算机操作系统、应用程序、配置文件等的软件环境上运行。这些环境可能还包括数据库系统、Web 服务器、应用程序需要的外部库等。按照需要和使用情况，软件环境可以分为开发环境、运行环境、测试环境等。一般而言，开发环境不同于运行环境，例如，移动应用通常还是在个人计算机上开发，在手机上使用。开发环境和运行环境上的配置也不一样：开发机上安装运行了编程工具、测试工具、代码管理工具、类库或库函数，甚至系统之外的库、试验数据、环境配置文件等；而目标机则不同，可能没有要运行应用软件的基本配置、没有安装外部库、缺少基本的数据，甚至没有运行时环境（如 Java 的运行时环境 JRE）。

2．虚拟机

C 语言程序经过编译的结果是计算机二进制指令代码（或汇编程序），可以直接在操作系统的环境下运行。因为编译产生的代码使用了计算机的指令，每个编译的 C 程序只能在特定的计算机或操作系统上运行，要在不同类型的计算机上运行，就需要重新编译。C 和 C++没有虚拟机。大多数现代编程语言能在不同的计算机（服务器、笔记本、PC 等）、工业控制机、平板设备、手机、嵌入式设备等上运行，希望尽可能地独立于计算机及其操作系统（平台或环境）。虚拟技术为每个应用程序创建一个运行的容器，把应用程序与计算平台隔离开，从而实现了应用的跨平台运行。例如，Web 程序在 Web 服务器中运行，Java 程序在 Java 虚拟机（JVM）中运行，Android 应用 apk 在 Dalvik VM 虚拟机上运行。

在 Visual Studio 环境下开发的程序，无论使用任何程序设计语言，都可以在 Windows 环境下运行。这是因为 Windows 操作系统中包含类似虚拟机的 CLR（Common Language Runtime），它先将任何.NET 程序都编译成一种中间语言，然后再转为机器指令。CLR 为.NET 应用程序提供了一个托管的代码执行环境，将原来由程序员或操作系统做的工作剥离出来交由 CLR 来完成，这些工作包括内存管理、即时编译、组件自描述、安全管理和代码验证，以及其他一些系统服务。

3．构建和打包

最简单的软件交付活动是编译（compile）和连接（link），然后把应用代码构建（build）成可

运行的程序（如 C 程序）。一般情况下，简单的 C 程序可以构建好、直接复制到目标机上运行。如果程序由多个文件组成，文件之间存在各种依赖关系（如头文件、引用库），那么就需要工具支持软件的构建和安装。

软件交付的首要工作是把构造的程序从开发环境中分离出来并打包。程序打包就是创建计算机程序的安装，即把各种编译好的文件、依赖的资源（如头文件、函数库、类库）、数据和配置文件等组装成一个可以自行解压的压缩文件，允许软件文件在多个计算机上安装运行。例如，Android 的 apk、微软的 setup.exe 或.msi 都是安装包。在目标机器上运行安装包可以将应用程序的所有文件释放到硬盘上、完成修改注册表（Windows 操作系统）、修改系统设置、创建快捷方式等。这样才能使用程序。

简单的 Java 程序打包成 jar 就能在安装了 Java 运行时环境 JRE 的任何设备上运行，无须其他安装。对于包含了多个类、引用了外部包等复杂的 Java 程序，或者想同 C 程序一样直接在 Windows 平台下运行，则需要更多的交付活动。

4．安装活动

计算机程序的安装是使程序可以运行的活动。安装可能是一个大型软件交付过程的一个部分。对不同的程序和计算机，安装过程不同。有些程序仅仅复制到计算机的文件中就可以运行了；有些程序则不能马上就运行，需要安装过程。一旦安装完毕，程序可以反复执行。软件安装过程的基本操作包括：

- 确保满足必要的系统需求；
- 检查软件的版本；
- 创建或更新程序文件和目录；
- 增加配置数据，如配置文件、Windows 注册项或环境变量；
- 使用户能访问到软件，如创建链接、快捷键或书签；
- 配置自动运行的组件，如 Windows 或 UNIX 服务；
- 激活产品；
- 更新软件版本。

本章其余部分将介绍实现构建自动化的工具，然后详细地说明 Java 的构造与交付。

6.2.2 实现构建自动化的工具

现代的 IDE 如 Eclipse、NetBeans、Visual Studio 都包含了 build 操作，利用目录结构管理开发的代码、利用各种配置指定需要资源的信息等，容易完成项目代码的编译和连接。例如，如果使用 Eclipse，编写和保存代码时会自动执行编译；单击 Run 的操作包括自动完成编译、构建和运行代码。IDE 通过可视化完成了资源配置、路径设置、外部库引入等代码的管理，简化了构建活动。此外，Visual Studio 还有专用的软件打包和安装工具 Windows Installer / InstallShield，方便了应用程序在 Windows 上的安装。

但是，无论是构建，还是创建安装包，使用 IDE 仍然需要开发者一步一步地手工操作，选择正确的应用代码、各种库及数据、图标等资源，然后才能构建或生成交付的软件。一旦其中的一个资源发生变化，还需要从头到尾手工操作一遍。

使用 Ant、make 等独立的构建工具可以得到与操作系统或 IDE 无关的代码。更有意义的是，通过编写脚本或批处理文件能使程序的构建工作自动化。

1．经典构造工具 make

经典构造工具 make

2．Java 的构建工具 Ant

C 语言有 make 工具来帮助构建任务的批量完成。make 本质上是通过在 Makefile 文件中编写命令脚本，计算各种操作的依赖关系，然后执行命令。这就意味着开发者可以容易地通过使用操作系统特有的命令或编写新的命令程序扩展该工具。当然，这也意味着 make 将自己限制在了特定的操作系统或类型上（如 UNIX）。

由于 Java 应用是与平台无关的，当然不会用平台相关的 make 脚本来完成批处理任务。Ant（Another neat tool）是一个跨平台的 Java 库和命令行工具，与 make 类似，用编写的脚本执行编译、汇编、测试和运行等构建任务。除了 Java 之外，Ant 也能构建 C、C++等应用程序。Ant 具有如下特点：（1）Ant 是用 Java 语言编写的，而且它的构建规则用扩展性标识语言 XML（Extensible Markup Language）描述，具有很好的跨平台性；（2）Ant 由一系列任务组成，这些任务是用 XML 文件描述的脚本（类似 Makefile 文件的构建文件 build.xml），结构清晰，容易书写和维护；（3）由于 Ant 的跨平台性和操作简单的特点，它很容易集成到一些开发环境和技术中，包括持续构建技术、Android 开发环境。

下面概要介绍 Ant 的使用，借此学习代码构建的基本原理和过程。想深入学习 Ant 及构建工具的读者请参阅相关文献。

下载、安装 Ant 后，还要设置环境变量。基本上和设置 JDK 相似，首先设置 ANT_HOME，再加入 Path。在 Windows 命令行输入 ant –version，如果安装正确，就会显示当前 Ant 的版本信息，如图 6.9 所示。

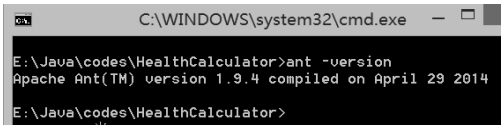


图 6.9　运行命令 ant 可以确认是否安装成功

Ant 将项目的构建任务分解，它分为工程、目标和任务这三个层次。工程用来描述处于项目层次的内容。目标由用户来编写，不同的目标对应于用户使用的一个操作任务单元。操作包括编译、测试、打包及操作系统的命令，如创建子目录等。操作由 Ant 在任务的层次上完成，所有任务都可以被目标调用，在目标内被组合装配起来完成用户自定义的一个过程，实现自动化工作的需要。Ant 中的所有设定要素都是遵守 XML 规范的，并存储在默认的 build.xml 文件中。

【例 6.2】 以一个程序“健康计算器”为例。创建一个 Java 工程 HealthCalculator，其中只有一个类 HealthCalculator，工程中只有两个目录，一个用于指示打包的 MF 文件，一个指挥 Ant 工作的脚本文件 build.xml。HealthCalculator 程序和 Java 项目文件如图 6.10 和图 6.11 所示。

图 6.10　HealthCalculator 程序

图 6.11　Java 项目文件

Ant 的 build 文件是一个标准的 XML 文件，它包含一个根节点 Project。每个 Project 定义了一个或多个 Target，每个 Target 又是一系列 Task（每个 Task 是一段可被执行的代码，如利用 javac 或 jar 等工具进行操作）的集合。各个元素标签之间的关系要用好 Ant 工具。如下编写的 build.xml 文件，在工程目录中创建两个目录 classes 和 dist，分别存放项目的 Java 类和打包文件。其中，命令 mkdir 用于建立目录，delete 是删除目录；srcdir 是原文件所在目录，destdir 是编译后目标文件所在目录；jar 是 Java 的打包命令，结果文件名是 jarfile，basedir 是要打包文件所在目录。manifest.mf 包含执行程序的入口类。depends 属性是 target 之间相互依赖的关系。

```
<?xml version="1.0" ?>
<project name="HealthCalculator" default="archive" >
<target name="init">
    <mkdir dir="build/classes" />
    <mkdir dir="dist" />
</target>
    <target name="compile" depends="init" >
    <javac srcdir="src" destdir="build/classes" includeantruntime="on" />
</target>
<target name="archive" depends="compile" >
    <jar destfile="dist/HealthCalculator.jar"
        basedir="build/classes"
        manifest="MANIFEST.MF" />
</target>
</project>
```

运行 Ant，显示了执行初始化、编译和构建过程的信息，如图 6.12 所示。

```
E:\Java\codes\HealthCalculator>ant
Buildfile: E:\Java\codes\HealthCalculator\build.xml

init:
    [mkdir] Created dir: E:\Java\codes\HealthCalculator\build
    [mkdir] Created dir: E:\Java\codes\HealthCalculator\dist

compile:
    [javac] Compiling 1 source file to E:\Java\codes\HealthCalculator\build

archive:
      [jar] Building jar: E:\Java\codes\HealthCalculator\dist\HealthCalculator.j
ar

BUILD SUCCESSFUL
Total time: 5 seconds
```

图 6.12　成功运行 Ant 所显示的信息

图 6.13　运行后的目录及 jar 文件

运行后工程目录中多了两个目录 build 和 dist。build\classes 包含了 Java 类，dist 包含了 jar 可运行程序，如图 6.13 所示。

第一次执行 Ant 花费了 5s。如果代码发生一些变化，再次执行 Ant 构建时，Ant 会根据依赖关系重新构建发生变化及受影响的部分，不做重复的工作（如建立子目录），从而大大提高运行速度。如果代码没做任何变动，第 2 次构建花费 1s。

如同 JUnit 是 Eclipse 的内置测试工具，Ant 作为构建工具通常也包含在 Eclipse 中。选择 Run→External Tools→Open External Configurations 进入窗口，就可以看到 Ant Build 的配置。

6.2.3　Java 程序的打包与交付

开发的 Java 程序有两种基本的应用方式：供其他开发者复用的代码，独立于开发环境的可运行程序。无论哪种形式，都要将 Java 程序（类）打包成 jar 文件（Java Archive File，Java 档案文件）。jar 文件是一种压缩文件，可以用 WinRAR、WinZip 等打开。jar 文件与 ZIP 文件的区别是：jar 文件中包含一个 META-INF/MANIFEST.MF 的清单文件，这个文件的作用类似 Makefile，但它是用 XML 格式描述的。

任何计算设备只要安装了 JRE，就可以用命令“java –jar”运行 jar 程序；在 Windows 机器上双击 jar 程序可以使其运行。如果设备上没有安装 JRE，则首先需要把 jar 程序和 JRE 一起封装成 exe 文件，然后才能运行。下面说明如何在 Eclipse 中完成 Java 程序的打包。

1．复用 Java 代码的打包

仍以“健康计算器”的程序 HealthCalculator 为例，其源码放在如下包中：

```
package cbsc.cha6.delivery;
```

（1）在 Eclipse 中打开项目 HealthCalculator，右击项目，选择 Export。

（2）选择“Java/JAR file”，单击 Next 按钮，进入 JAR File Specification 页面。

（3）在 Select the resources to export 中选择想要包含的项目文件夹，而无须放进一些不必要的文件夹，以节省最后 jar 文件的空间。本例选择 HealthCalculator。下面还有几个选项。

- Export generated class files and resources，表示只导出生成的.class 文件和其他资源文件。
- Export all output folders for checked projects，表示导出选中项目的所有文件夹。
- Export java source file and resouces，表示导出的 jar 包中将包含源代码*.java，如果不想泄漏源代码，就不选此项。
- Export refactorings for checked projects，把一些重构的信息文件也包含进去。

本例选择第一个选项。

在 Select the export destination 中选择导出的 jar 路径。

单击 Next 按钮，进入 JAR Packaging Options 页面。

（4）选择是否导出那些含有警告 warning 或错误 errors 的*.class 文件，一般不做任何选择。单击 Next 按钮，进入 JAR Manifest Specification 页面。

（5）可以对项目做一些配置。

- Generate the manifest file，系统自动生成 MANIFEST.MF 文件，如果项目没有引用其他的，可以选择这一项，内容可能为空。
- Use existing mainfest from workspace，选择自定义的.MF 文件。
- Seal content，要封装整个 jar 或指定包 packages。
- Main class，选择可运行程序的入口。如果项目自成一体可以运行，选择此项，接着出现一个窗口，可能会出现若干类，选择一个作为主类（有 main 方法、开始运行的那个类），打包得到的 jar 项目是可运行的程序，运行是从主类的 main()开始的。如果项目不是可运行的程序（如只是可复用的类或 JUnit 的测试），则不必选择此项；即使选择了，可能也没有可选的类。

（6）单击 Finish 按钮就完成了打包。

这样，HealthCalculator.jar 中的类就如同其他类一样可以复用。例如，把 HealthCalculator.jar 复制到 JDK 安装目录的 jre\lib\ext 文件夹中，其他程序就可以引用其中的类，如：

```
import cbsc.cha6.delivery.java.*;
```

2．可运行 Java 程序的打包

创建可执行 jar 包的关键在于让 Java 命令知道 jar 包中哪个是主类。这需要借助清单文件 MANIFEST 明确地给出，有三种方式。

方式一：上面步骤（5）中，编写一个 MANIFEST.MF 文件，在清单文件中增加如下两行：

```
Main-Class: cha6.refactory.BinaryOperation

```

注意这个文件格式要求非常严格：

```
Main-Class:<空格>包名.类名<回车>
```

方式二：上面步骤（5）中，仍然让 MANIFEST.MF 文件空着，在 Main class 选项，单击 Browse 按钮，选择主类。

方式三：打开生成的 jar 压缩文件，打开目录 META-INF 下的 MANIFEST.MF 文件，如同方法一，在清单文件中增加两行。

这样构建的 jar 程序就可以在任何安装了 JRE 的设备上运行了。

当然，还有简单方式生成可运行的 Java 程序包，就是在上面步骤（2）中选择“Runnable JAR file”。但是，这种方式会把程序运行依赖的所有类、第三方 jar 包等资源整合到一个单独 jar 包中，造成存储空间的浪费，降低运行效率。

3．在 Windows 上直接运行 Java 程序的制作

Java 程序不能直接在 Windows 操作系统上直接运行，需要借助一些工具。工具 exe4j、launch4j 等可以把 Java 程序的 jar 包、运行环境 JRE 及图标等其他资源封装成一个后缀是 exe 的 Windows 可运行文件。这个文件能搜查某个 JRE 版本或使用捆绑的 JRE，并设置运行时环境。下面用 launch4j 说明把一个 jar 格式的应用转换成 Windows 可执行程序的过程，只填写了必需的选项。

（1）下载、安装并运行 launch4j，出现图 6.14 所示的界面。制作简单的 exe 文件只需要操作 Basic、Header 和 JRE 三个卡片式窗口。

（2）在 Basic 卡片式窗口，必须完成带红色星号的选项：在输出 output 中输入制作的运行文件，后缀一定加.exe；在 input 中选择要包装的 jar 文件；Java download URL 会自动填写。

（3）在 Header 卡片式窗口，Header Type 有两个选项：GUI 或 Console，默认选项是 GUI。如果 Java 程序是 GUI，选择无关；如果 Java 程序不是 GUI，则必须选择 Console，否则程序无法运行。

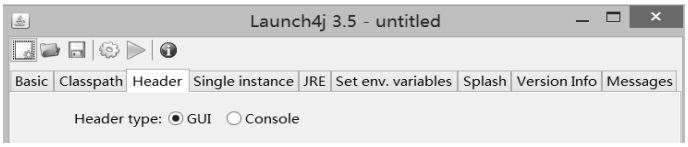


图 6.14　launch4j 中设置运行程序界面的窗口

（4）在 JRE 卡片式窗口，输入绑定的 JRE 的路径，如图 6.15 所示。

（5）单击功能菜单的齿轮图标，生成 exe 文件。下方的 Log 区域显示制作过程：编译、连接、打包，制作是否成功，如图 6.16 所示。

将 Java 程序包 jar 文件转换成 Windows 上可执行的 exe 文件，方便了 Java 程序的运行和使用，也可以提高源码及资源的安全性。但是，这同时也失去了 Java 的初衷——跨平台性。