

前言

《Java 编程思想》的原著《thinking in Java》是世界上最经典的 Java 教程，从书中读者评价中可以看到全世界的 Java 爱好者公认这是一本通俗易懂，讲解生动的经典书籍，可是翻译成中文后，却成了公认的天书了呢？原因就是到处是拗口难懂的句子，还有到处的错字，翻译的错误。它还能通俗易懂了吗？答案是否定的。我痛苦的读了本书的大半，真的很痛苦，因为我仔细推敲每一处，每一句话，与英文原著对比。除了原著的极少几处错误外，都是译著的错误，如果我说本书没有经过校对过就出版了，没有人反对吧？至少书中写的专家指导委员会是虚设的，他们如果有人看过本书就不会出现这么多错误了，我把勘误发给机械工业出版社责任人，他们百般推脱，虽然承认书的错误，可是不肯再版时更正，也不肯发勘误，理由是“太忙了”，然后又推托说他们不懂技术，不懂技术的人负责出版 Java 书籍，如何保证质量？可笑啊！

本书有这么多错误会误导多少人？浪费多少人的时间呐？简直是在犯罪。为了让更多的读者能摆脱错误的误导，我把我总结的勘误发上来，希望你们能先在书中改正然后再读，这样会轻松很多，其余的勘误部分，我一时还没有时间整理，一旦完成也发上来。

对于原版的错误我都有解释，而且我指出原版的错误都是 Bruce Eckel 网站上没有登的，所以如果你读的是英文版，本勘误对你依然有帮助，我整理的非常细致，我甚至相信我勘误完成后的部分再极难发现错误了，这些都是我工作之余抽时间整理的，熬了多少个凌晨 1、2 点。希望更多的 Java 战士们能从中受益。我白爽就值了。

欢迎志同道合的兄弟们和我共同完成全书的勘误，还原这部经典!!!

大连 白爽

qq: 67529468

Email: 67529468@qq.com

1~11 章的勘误

一、p29 倒数第二行的 48 应该改成 47。

错误：在这里，st1.i 和 st2.i 指向同一存储空间，因此它们具有相同的值 48。

改正为：在这里，st1.i 和 st2.i 指向同一存储空间，因此它们具有相同的值 47。

因为：倒数第八行 static int I=47;

二、p49 第一行 直接正确修改如下：

改正为：作为自然对数的基数使用，那么在 Java 中看到象 $1.39e-43$ 这样的表达式时，请转换您的思维，它真正的含义是 1.39×10^{-43} 。

三、p96 第一段和第二段共有 5 处错误（第三版的这里就是错误的，详见原著，很明显）：此下两段中红色的为错误。

Bowl 类使你得以看到类的创建，而 Table 类和 Cupboard 类在它们的类定义中加入了 Bowl 类型的静态成员。注意，在静态数据成员定义之前，Cupboard 类先定义了一个 Bowl 类型的非静态成员 **b3**。

由输出可见，静态初始化只有在必要时刻才会进行。如果不创建 Table 对象，也不引用 **Table.b1** 或 **Table.b2**，那么静态的 Bowl **b1** 和 **b2** 永远都不会被创建。只有在第一个 Table 对象被创建（或者第一次访问静态数据）的时候，它们才会被初始化。此后，静态对象不会再次被初始化。

更正为：

Bowl 类使你得以看到类的创建，而 Table 类和 Cupboard 类在它们的类定义中加入了 Bowl 类型的静态成员。注意，在静态数据成员定义之前，Cupboard 类先定义了一个 Bowl 类型的非静态成员 **bow13**。

由输出可见，静态初始化只有在必要时刻才会进行。如果不创建 Table 对象，也不引用 **Table.bow1** 或 **Table.bow2**，那么静态的 Bowl **bow1** 和 **bow2** 永远都不会被创建。只有在第一个 Table 对象被创建（或者第一次访问静态数据）的时候，它们才会被初始化。此后，静态对象不会再次被初始化。

四、p101 这个是原著画蛇添足了，见第一处代码的第十行，3 后边多一个逗号“，”，对于初始化一个数组，最后多加一个“，”意义就变了，意思成数组有四个对象了：**new Integer(1), new Integer(2), 3 和 null**，所以尽管这处代码运行正确，但是这个逗号也绝不可以加。

```
//: initialization/ArrayInit.java
```

```
// Array initialization.
```

```
import java.util.*;
```

```
public class ArrayInit {
```

```
    public static void main(String[] args) {
```

```
        Integer[] a = {
```

```
            new Integer(1),
```

```
            new Integer(2),
```

```
            3, // Autoboxing
```

```
        };
```

```
        Integer[] b = new Integer[]{
```

```
            new Integer(1),
```

```
            new Integer(2),
```

```
            3, // Autoboxing
```



这个 3 后边的逗号是多余的。

```

    };
    System.out.println(Arrays.toString(a));
    System.out.println(Arrays.toString(b));
}
} /* Output:
[1, 2, 3]
[1, 2, 3]
*///:~

```

五、p172 中间那段少一个“承”字。

错误：但是，interface 不仅仅是一个极度抽象的类，因为它允许人们通过创建一个能够被向上转型为多种基类的类型，来实现某种类似多种继变种的特性。

改正为：但是，interface 不仅仅是一个极度抽象的类，因为它允许人们通过创建一个能够被向上转型为多种基类的类型，来实现某种类似多种继承变种的特性。

六、p192 有两处错误。

第一段的最后一句话更正为：并且别的方法能以此接口为参数，来生成更加通用的代码。

第三段第二行更正为：注意方法 end()、current()和 next()都用到了 items，这是一个引用，……

七、p196 第一段最后一句话

错误：并不意味着一旦 dest()方法执行完毕，PDestination 就不可用了。

更正为：并不意味着一旦 destination()方法执行完毕，PDestination 就不可用了。

八、p219 中间一段

更正为：注意，ArrayList 已经被向上转型为 List，这与前一个事例中的处理方式正好相反。使用接口的目的在于如果你决定去修改你的实现，你所需的知识在创建处修改它，就像下面这样：

九、p228 错误严重，倒数第二行不但这句之前少翻译一句话，而且本行翻译错误：

原著：

addFirst() inserts an element at the beginning of the list.

offer() is the same as add() and addLast(). They all add an element to the tail (end) of a list.

正确译文为：

addFirst()在列表的头部插入参数指定的元素。

offer()与 as add() 和 addLast()相同，它们都在列表的尾部加入数据。

十、p229 代码下边那段话最后两句翻译错误

正确翻译为：如果你浏览一下 Queue 接口就会发现，element(), offer(), peek(), poll() 和 remove()方法也被加入到 LinkedList 中，目的是 LinkedList 可以成为一个 Queue 的实现。Queue 的完整示例将在本章稍后给出。

原著：If you look at the Queue interface, you'll see the element(), offer(), peek(), poll() and remove() methods that were added to LinkedList in order that it could be a Queue implementation. Full examples of Queues will be given later in this chapter.

//-----

下面更正一个附录中的错误，因为一般人买书先看看前言，看看附录，结果附录里有个很明显的错误，这个有些大煞风景了。

P856 附录 A 本页第七行多个“部分”二字。

错误：另外，**部分**有些部分已经被移入到电子版中。这些主题包括：

更正为：另外，有些部分已经被移入到电子版中。这些主题包括：

第十二章和十三章勘误

一、p248 12.1 概念下边第四行 翻译错误：

原句：对，错误也许会发生，但那是别人造成的，不管我的事。

正确翻译：对，错误也许会出现在别人的代码中，但我的代码中不会出现。

二、p272 倒数第二行 翻译有误

正确译文：FileNotFoundException,这么做就显得有些投机取巧了。

三、p278 **12.12.2 观点** 下面的第一行

原文：首先，Java **无谓地**发明了“被检查异常”

正确翻译：首先，**值得注意的是** Java **有效地**发明了“被检查异常”

原文：First, it's worth noting that Java effectively invented the checked exception

注：it's worth noting that 的正确翻译是：值得注意的是……

翻译成“无谓地”恰巧是给翻译反了，尽管看起来是 worth nothing 似乎是“没有意义”的意思，但是正宗的英文意思却不是这样，这里译者确定翻译错了，表达的意思弄反了，这里是译者望文生义了，Bruce 原意是赞扬受检查异常机制是有效的。

四、p284 这个原文的陈述句被翻译成疑问句了，但是后边却没有带问号“？” 位置为本页中间处

原句：想看看以上代码到底是如何工作的吗，可以用……


正确翻译：**如果**想看看以上代码到底是如何工作的，可以用……

或者加上问号，但是原著这里不是疑问句：想看看以上代码到底是如何工作的吗？可以用……

五、p289 倒数第四行的行末，多一个“以”字

原句：%f 表示 y 是**以**一个浮点数……

正确：%f 表示 y 是一个浮点数……

六、 p291 这个我认为 Bruce 遗漏了一个参数的说明，恰巧这个参数十分的重要，也

就是忘记解释[argument_index\$]了，直接从第二个参数[flags]开始说明的，所以我编写了一段文字和一段代码来补上这个参数的说明。另外译者最好把下边的表达式（黄色）

翻译过来，这样中国的读者理解起来容易多了。如下：

`%[argument_index$][flags][width][.precision] conversion`

最最最最最理想的翻译如下（两行*之间的部分为我添加的，代码经过验证了的）：

`%[待插入项的序号$][对齐方式][最小宽度][.最大尺寸/小数点后保留位数] 类型转换字符`

注：第一个参数是数字，并且后边带个“\$”，第四个参数前带个点“.”

我们通常会遇到这种情况：在表达式中需要多次用到某个插入值，比如我们要格式化输出的字符串是：Tom has a goldfish,Tom loves it very much but her brother has a cat
需要插入项为“Tom”,“Tom”,“brother” 见如下代码：

```
import java.util.Formatter;
public class Testf{
    public static void main(String [] args){
        Formatter f=new Formatter(System.out);
        f.format("%s has a goldfish,%s loves it very much but her %s has a cat .\n","Tom","Tom","brother");
        f.format("%s has a goldfish,%1$s loves it very much but her %s has a cat .\n","Tom","brother");
        f.format("%s has a goldfish,%<s loves it very much but her %s has a cat .\n","Tom","brother");
    }
} /*Output:
Tom has a goldfish,Tom loves it very much but her brother has a cat .
Tom has a goldfish,Tom loves it very much but her brother has a cat .
Tom has a goldfish,Tom loves it very much but her brother has a cat .
*/
```

第一种表达方式中我们使用了三个插入项，即两个“Tom”一个“brother”，对于重复的插入项我们可以按第二种和第三种表达方式，指定插入项的序号+\$，或者用“<”指定使用上一个插入项，然后接下来的插入项（本例中为“Brother”）的插入位置会前移一位。

`f.format("%s has a goldfish,%1$s loves it very much but her %s has a cat .\n","Tom","brother");`

`f.format("%s has a goldfish,%<s loves it very much but her %s has a cat .\n","Tom","brother");`

最常见的应用是控制一个域的最小尺寸，

白爽认为：译者有权力对原著中的纰漏进行修改，甚至有些表达可以转换一种更适合中国人的方式，但是意思和原著相同，这样读译著才有种比原著更好的感觉，当然我说的修改都是极少的而且是小小的修改。

其实白爽对这里一直耿耿于怀，认为全世界的读者读到这里都会很纳闷，`[argument_index$]`到底是什么东西，为什么直接讲的是`[flags]`？谁看完都会有这个疑问，可能90%的人找不到答案或者带着疑问越过去了，我这样添加上之后这里可以说完美了，虽然看似简单，但是我考虑的非常全面，读者能在带箭头的图中找到任何疑问的答案，只要他善于思考。

七、p298 第二个表中第二行的“包含”二字不对，应该是“代表”
[abc] 代表 a、b 和 c 中任意一个字符（与 a|b|c 的作用相同）

八、p304 中的表中有乱码-也许是乱码吧，反正那些字符莫名其妙，原著中不是那个。
第一栏，效果那列
原句：表达式 `a\u030A` 就会匹配字符串？。
正确：表达式 `'\u003F'` 就会匹配字符串'?'。
原著中的位置是 p538

九、p305 表下边第二行，翻译乱套了
原文：COMMENTS（对声明或文档有用）特别有用。
正确翻译：COMMENTS 对 and/or 式文档的声明特别有用。

十、p310 倒数第十行的后半句 少了一个字“为”。
原文：用以判断下一个输入分词是否所需的类型。
正确：用以判断下一个输入分词是否~~为~~所需的类型。

前三十三章勘误完成

第十四章类型信息

在勘误这章之前，我先给出 thinking in Java 原版中的两处错误，译者在翻译时明显没有发现，而且 Bruce eckle 在自己的勘误网站上并没有勘误这两处，但是我敢用生命来担保，是他错了。

第一处： p325 倒数第三段和倒数第一段中的两处 `getTypes()` 全是 `types()`。

可以看看上边的源代码的第九行，指的就是那个抽象方法。

第二处： p369 第十四行代码中的 `superset` 与 `subset` 反了。

这个可以从 p246 的输出结果中可以看出，但是你需要很大的耐心把这么长的代码和结果仔细看，最快得到答案的方式是看第十五和十六行来看。

```
System.out.print(superset.getSimpleName()+"extends"+subset.getSimpleName+", adds: ");
```

你想想 `superset` 可以 `extends subset` 吗？名字是不是反了？父类 `extends` 子类？

所以我不知道为什么没有人给提出这两处错误给 Bruce。

在勘误前我先说两句，对于一个翻译，如果你读不懂它说什么，也就是他写的是中国字，句子却不是中国话，那我就认为书中有错误。所以我希望你先看看书中说的是不是不知所云，再看看我说的是什么，如果你看完我的翻译后再看书中的句子却看懂了，这也算是译者翻译错了。下面有大错的，也有翻译非常不好的地方，我都更正如下，希望编辑同志认真思考。我以页为单位，尽管很多都是一页有几处错误。

第一处：p314 第一段有处不知所云和第五段有个词翻译错误，共两处错误。

第一段：

如果某个对象出现在字符串表达式中（涉及 “+” 和字符串对象的表达式），toString() 方法……

如果某个对象出现在 “对象+字符串” 形式的表达式中”， toString()方法……

第五段：

而是只与对象家族中的一个通用表示打交道……

而是只与对象家族中的一个通用代表打交道……

第二处：p315 第四段有两处错误。

动态加载使能的行为，

动态加载激活（使用）的行为， 我感觉从上下文来看翻译成 “后再使用” 更恰当。

或者根本不可能复制的。

或者根本不可能模仿的。

第三处：p316 最后两行代码是多余的，这个有些不应该啊!!!

```
public static void main(String [] args){
```

```
    Class c = null ;
```

第四处：p317 第三段有翻译错误的地方，第五段最后那句话翻译的完完全全的超级的 super 不好!!! 世界上没有人能知道这是什么话。

第三段：

在 main（）中调用的 Class.interfaces()方法返回的是 Class 对象，它们表示在感兴趣的 Class 对象中所包含的接口。

在 main（）中调用的 Class.interfaces()方法返回的是 Class 对象数组，它们表示在本 Class 对象中所包含的接口。

第五段：

当然，在你可以发送 Object 能够接受的消息之外的任何消息之前，你必须更多地了解它，并执行某种转型。

当然，你必须先对这个对象有些了解并做某种转型，才可以对其发送 Object 类本身（不包括子类）不能接收的消息。

第五处：p320 第四段竟然说出汉语的定语从句了，英语中定语从句很好理解，汉语这么说可有些莫名其妙了，而且多了四个字。第六段的 “尽管” 两字多余。

第四段：

Class 引用总是指向某个 Class 对象，它可以制造类的实例，并包含可作用于这些实例的所有方法代码。

Class 引用总是指向某个 Class 对象，而 Class 对象可以制造类的实例，并包含可作用于这些实例的所有方法代码。

第六段：

尽管泛型类引用只能……

尽管泛型类引用只能……

这个如果你不相信我的话，还是看原著，说的两个东西，可以用尽管……但是 嘛？

第六处：p325 除了我先前指出的原著的一处错误外，还有一句话翻译的不好。最后一段。

唯一所需提供的就是你希望使用 `randomPet()` 和其他方法来创建的宠物类型的 `List`。

唯一所需提供的就是物类型的 `List`，而这些宠物你会使用 `randomPet()` 和其他方法来创建。

第七处：p331 最后一段中的第二句到最后翻译的有问题。

我直接给出正确的翻译吧：

下面的事例中，基类 `Part` 包含一个工厂对象的列表。那些类型应由 `createRandom()` 方法产生，那些类型的工厂都被添加到 `partFactories List` 中，从而被注册到了基类中：

第八处：p350 倒数第二段的第二行少一个字。

如果基类来别人的类，

如果基类来自别人的类，

第 15 章 泛型也有不少错误，但是第一页错误比较严重，我先勘误这页吧，真的好累，一边低头看书，一边打字，脖子累断了！ %>_<%

P352 第一段的发挥实在是不应该，意思和原著正好反了。

但是，考虑到除了 `final` 类不能扩展，其他任何类都可以被扩展，所以这种灵活性大多数时候也会有一些性能损耗。

当然，除了 `final` 类以外的其他任何类都可以被扩展，所以这种灵活性大多数时候会自觉地被使用。

第六段错误太多，我干脆把正确翻译写出来吧，这种意译更合理些。

然而，如果你了解其他语言（例如 C++）中的参数化类型机制的话，在你使用 `Java` 泛型机制时，你会发现不是你期望中的每件事实际它都能做到。使用别人已经构建好的泛型类会相当容易。但是如果你要自己创建一个泛型实例，就会遇到许多令你诧异的事情。在本章中，我的任务之一就是向你解释，`Java` 中泛型的特征怎么会是这个样子。

第十五章勘误

一、p373 有两处错误，一处是多字，另一处是翻译错误。

代码上边的一行：下面~~是~~的示例是对这个谜题的一个补充。

改正为：下面的示例是对这个谜题的一个补充。

倒数第三段：因此，你可以知道诸如类型参数标识符和泛型类型边界这类的信息—你却无法知道用来创建某个特定实例的实际的类型参数。

改正为：因此，你可以明白诸如类型参数标识符方面的事和泛型类型的边界的局限—也就是你无法知道用来创建某个特定实例的实际的类型参数。

二、p374 最后一段的第一句话翻译的不知所云。（我翻译的为意译）

原句：由于有了擦除，`Java` 编译器无法将 `manipulate()` 必须能够在 `obj` 上调用 `f()` 这一需求映

射到 HasF()拥有 f()这一事实上。

更正为：由于有了擦除，Java 编译器无法保证 manipulate()必须能调用 obj 上的 f()方法，而 HasF 恰好确实有 f()这个方法。

三、p375 中间一段的中间一行

原句：因此，类型参数和它们在有用的泛型代码中的应用，通常比简单的类替换要更复杂。但是，不能因此而认为<T extends HasF>形式的任何东西都是有缺陷的。

更正为：因此，类型参数和它们在泛型代码中的应用通常比简单的类替换要更复杂。但是，不能因此而认为<T extends HasF>形式的任何东西都是有缺陷的。

四、p377 这里是原著的错误，为什么错误？我相信我不需要任何解释了，因为这是个非常低级的错误，Bruce 他也只是个人，他也会犯低级错误。

本页顶端第二行代码：public void set(T arg) {arg = element; }

更正为：public void set(T arg) { element = arg; }

五、p392 这里也是原著的错误，extends 打成了 Extends，这个也是绝对不可以弄错的，因为 Java 大小写敏感。 位置为第六行中间位置

原句：add()的参数就变成了 “Extends Fruit”。

更正为：add()的参数就变成了 “extends Fruit”。

六、p409 倒数第四行的后半行有个错字“强”

原句：可以解决在这种强况下的类型检查问题，

更正为：可以解决在这种情况下的类型检查问题，