TURING 图灵程序设计丛书 Web开发系列

Apress®

Pro PHP

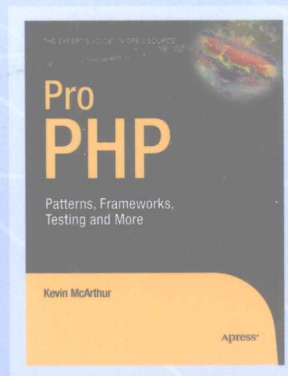
Patterns, Frameworks, Testing and More

PHP高级程序设计

模式、框架与测试

[加] Kevin McArthur 著
汪泳 等译

- 享有盛誉的PHP高级教程
- Zend Framework核心开发人员力作
- 深入设计模式、PHP标准库和JSON



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 Web开发系列

Pro PHP
Patterns, Frameworks, Testing and More

PHP高级程序设计
模式、框架与测试

[加] Kevin McArthur 著
汪泳 等译

人民邮电出版社
北 京

人民邮电出版社
样 书
专用章

图书在版编目（CIP）数据

PHP 高级程序设计：模式、框架与测试 /（加）麦克阿瑟（McArthur, K.）著；汪泳等译．—北京：人民邮电出版社，2009.7
（图灵程序设计丛书）
书名原文：Pro PHP：Patterns，Frameworks，Testing and More
ISBN 978-7-115-19317-9

Ⅰ. P… Ⅱ. ①麦…②汪… Ⅲ. PHP 语言－程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第191155号

内 容 提 要

本书采用循序渐进的方式介绍了用 PHP 进行 Web 开发的相关知识。书中首先从 OOP 采用的机制——抽象类、接口、契约式编程开始讲起，然后介绍了静态方法、单例模式、工厂模式和 PHP 6 的新特性等内容，接着介绍了测试和文档方面的内容，还介绍了标准 PHP 库 SPL 方面的知识以及 PHP 开发人员最有可能用到的 MVC 模式，最后介绍了 Ajax、JSON、SOAP Web 服务以及 SSL 客户端验证等 Web 2.0 方面的内容。

本书适合中、高级的 PHP 程序员阅读。

图灵程序设计丛书

PHP高级程序设计：模式、框架与测试

◆ 著 [加] Kevin McArthur
译 汪 泳 等
责任编辑 傅志红
执行编辑 王军花

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京顺义振华印刷厂印刷

◆ 开本：800×1000 1/16
印张：18.75
字数：443千字 2009年 7 月第 1 版
印数：1－3 000册 2009年 7 月北京第 1 次印刷

著作权合同登记号 图字：01-2008-2034号

ISBN 978-7-115-19317-9/TP

定价：45.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223

反盗版热线：(010)67171154

版权声明

Original English language edition, entitled *Pro PHP: Patterns, Frameworks, Testing and More* by Kevin McArthur, published by Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705 USA.

Copyright © 2008 by Kevin McArthur. Simplified Chinese-language edition copyright © 2009 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由Apress授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

前　言

在过去的十年间，PHP已经从一套为Web站点开发人员提供的简单工具转化成完整的OOP（面向对象编程）语言了。在Web应用开发方面，PHP现在可与Java和C#这样的主流编程语言抗衡，越来越多的公司为了给站点提供更加强大的功能都采用了PHP。原因很清楚：PHP既是一门易学的语言，又具有强大的特性。

通过阅读本书，你将会深入理解OOP理论，并学到如何使用框架和高级的系统互操作功能，最大限度地发挥出PHP编程的威力。

读者对象

这是一本高级书。我非常慎重地选择本书应该包含的内容以及读者应该掌握的知识。读者应该对HTTP和PHP有深刻的理解，也就是说，应该了解如何创建Web页面和Web表单，并且还应该理解像HTTP请求生命周期这样的关键概念。

如果你还不了解这些内容，那么建议阅读由Larry ullman合著的*PHP for the Web: Visual QuickStart Guide*一书[1]。这是一本非常好的介绍PHP编程的书，任何希望成为PHP开发人员的人都绝对应该阅读它。

如果你自己的PHP编程水平已达到中高级，那么本书正好适合你。

本书结构

本书每一章内容都在前面章节的基础上展开，同时也考虑到读者的基础参差不齐。如果你认为已经了解了某章讲述的内容，那么建议你跳过那一章，但最好还是读一下每章末尾的“小结”部分，那是每一章内容的扼要总结。不过，古人云“温故而知新”，就算是最熟练的程序员应该也会在每章中有新的收获。

本书分为五个部分。

第一部分，OOP和模式。这部分是学习高级的OOP概念必备的基础知识。该部分直接进入主题，讲解了抽象类、接口、静态方法、单例和工厂之类的模式，以及异常等内容。最后介绍了PHP 6所具有的新特性。

第二部分，测试和文档编写。这部分包括了所有那些相关的“外围”概念，如测试驱动开发

① 中文版《PHP基础教程（第3版）》即将由人民邮电出版社出版。——编者注

试读结束：需要全本请在线购买： www.ertongbook.com

和自动部署等。讲述了如何编写优秀的文档，并且介绍了PHPDoc和DocBook等几种文档标准。这部分还介绍了反射API的内容，使读者了解如何从程序中获取元数据。最后还讨论了持续集成，以及如何使用Phing和Xinc这样的工具来改善开发工作流。

第三部分，SPL（标准PHP库）。SPL包含了一些最高级的PHP代码。它提供了对于像索引器和迭代器这样的高级OOP概念的语言支持，还提供了处理异常的结构，以及像观察者/发布者这样的模式。这部分信息使大家可以创建出更加优美并且结构良好的类。

第四部分，MVC模式。MVC（模型–视图–控制器）可能是PHP开发人员用到的最有用的开发模式。它可用来为应用程序创建结构，并且调配团队中最好的资源来完成工作。对这一模式的深刻理解可能是任何PHP开发人员都必须掌握的最重要的职业技能，所以本书力求完整地解释这一模式。这一部分还介绍了Zend框架，这是为众多PHP公司所接受的一套基于MVC的框架。一开始，我们介绍了如何逐步建立起一个完整的框架应用程序，并使它开始运行，然后讲解了Zend框架的核心概念和高级特性。

第五部分，Web 2.0。这部分介绍了关于Web 2.0所需要了解的所有事项。你将会学到关于Ajax、JSON、SOAP Web服务以及SSL客户端验证等方面的信息。另外，这部分还包括了大量非常有用的辅导材料，这些辅导材料是基于个人经验提供的。

联系作者

欢迎大家随时通过电子邮箱Kevin.McArthur@StormTide.ca与作者取得联系。在http://www.stormtide.ca/pro-hpp-book或者Apress出版社网站的http://www.apress.com/book/view/9781590598191页面上可以获得本书的最新信息[①]。此外，大家还可以访问#PHP EFnet通过IRC与作者聊天[②]。

致谢

本书是在数千名开发人员组成的社区十年来的技术创新基础上写成的。我感谢他们所有人，并以此书向他们的卓越成果致敬。

感谢David Fugate给了我编写这本书的机会，以及在写本书的过程中给我提供的必要的指导。

还要感谢Michael Geist，正是他提供的帮助和建议使我能够顺利地解决困难。

我还必须感谢我的朋友和家人对我的支持，没有他们，我不可能获得这些成果。

最后，我要对Apress出版社的所有人表示我最诚挚的谢意，其中有的人参与了本书的出版工作，有的人则帮助其他作者出版了高质量著作。没有他们，这本书就不会面世。

① 本书的源代码可在图灵网站（www.turingbook.com）本书主页上免费注册下载。——编者注

② 读者关于本书的各种反馈、问题均可通过图灵公司本书网页提交，或发邮件到Contact@turingbook.com。——编者注

目　录

第一部分　OOP 和模式

第 1 章　抽象类、接口和契约式编程 …… 2

1.1　抽象类 …… 2
1.2　接口 …… 4
1.3　instanceof 操作符 …… 7
1.4　契约式编程 …… 8
1.5　小结 …… 8

第 2 章　静态变量、成员和方法 …… 9

2.1　静态变量 …… 9
2.2　类中静态元素的使用 …… 10
　2.2.1　静态成员 …… 10
　2.2.2　双冒号（paamayim nekudotayim） …… 11
　2.2.3　静态方法 …… 14
2.3　“静态”特性的争论 …… 15
2.4　小结 …… 15

第 3 章　单例模式和工厂模式 …… 17

3.1　职责和单例模式 …… 17
3.2　工厂模式 …… 19
　3.2.1　图像对象工厂 …… 20
　3.2.2　可移植的数据库 …… 22
3.3　小结 …… 24

第 4 章　异常 …… 26

4.1　实现异常 …… 26
　4.1.1　异常元素 …… 26
　4.1.2　扩展异常 …… 28
4.2　记录异常日志 …… 30
　4.2.1　记录自定义异常的日志 …… 30
　4.2.2　定义未捕捉的异常处理程序 …… 31
4.3　异常产生的开销 …… 31
4.4　错误代码 …… 32
4.5　类型提示和异常 …… 33
4.6　重新抛出异常 …… 33
4.7　小结 …… 34

第 5 章　PHP 6 中的新特性 …… 35

5.1　安装 PHP …… 35
5.2　PHP 6 中的 Unicode 支持 …… 37
　5.2.1　Unicode 语义 …… 37
　5.2.2　Unicode 排序规则 …… 39
5.3　命名空间 …… 40
5.4　延迟静态绑定 …… 41
5.5　具有动态特性的静态方法 …… 43
5.6　三目运算符（ifsetor） …… 43
5.7　XMLWriter 类 …… 43
5.8　小结 …… 45

第二部分　测试和文档编写

第 6 章　文档编写和编码规范 …… 48

6.1　编码规范 …… 48
6.2　PHP 注释和文法解析 …… 49
　6.2.1　注释的类型 …… 50
　6.2.2　关于文档注释的更多信息 …… 50
　6.2.3　文法解析 …… 51
　6.2.4　元数据 …… 51
6.3　PHPDoc …… 52
6.4　DocBook …… 55
　6.4.1　创建 DocBook 要用到的 XML 文件 …… 55
　6.4.2　解析 DocBook 文件 …… 56
　6.4.3　使用 DocBook 的元素 …… 59
6.5　小结 …… 62

第 7 章 反射 API……64
7.1 反射 API 介绍……64
7.1.1 获得用户声明的类……65
7.1.2 理解使用反射技术的插件架构……66
7.2 解析基于反射的文档数据……71
7.2.1 安装文档块分词器……72
7.2.2 访问文档注释数据……73
7.2.3 给文档注释数据做分词处理……73
7.2.4 解析标识符……74
7.3 扩展反射 API……76
7.3.1 解析器与反射 API 的集成……77
7.3.2 扩展反射类……78
7.3.3 更新解析器以处理行内标签……85
7.3.4 添加特性……88
7.4 小结……92
第 8 章 测试、部署和持续集成……93
8.1 用作版本控制的 Subversion……93
8.1.1 安装 Subversion……94
8.1.2 设置 Subversion……94
8.1.3 提交修改和解决冲突……95
8.1.4 激活对 Subversion 的访问功能……97
8.2 用于单元测试的 PHPUnit……98
8.2.1 安装 PHPUnit……98
8.2.2 创建第一个单元测试……98
8.2.3 理解 PHPUnit……100
8.3 用于部署的 Phing……102
8.3.1 安装 Phing……102
8.3.2 编写 Phing 部署脚本……103
8.4 Xinc，持续集成服务器……106
8.4.1 安装 Xinc……106
8.4.2 创建 Xinc 配置文件……106
8.4.3 启动 Xinc……107
8.5 用于调试的 Xdebug……107
8.5.1 安装 Xdebug……108
8.5.2 使用 Xdebug 跟踪代码执行……108
8.5.3 使用 Xdebug 执行基准测试……110
8.5.4 使用 Xdebug 检查代码覆盖……110
8.5.5 使用 Xdebug 进行远程调试……111
8.6 小结……111
第三部分 SPL（标准 PHP 库）
第 9 章 SPL 简介……114
9.1 SPL 基础……114
9.2 迭代器……115
9.2.1 Iterator 接口……115
9.2.2 迭代器辅助函数……116
9.3 数组重载……117
9.3.1 ArrayAccess 接口……117
9.3.2 计数和数组访问……117
9.4 观察者模式……118
9.5 序列化……121
9.6 SPL 自动加载……123
9.7 对象标识符……126
9.8 小结……126
第 10 章 SPL 迭代器……128
10.1 迭代器接口和迭代器……128
10.1.1 迭代器接口……128
10.1.2 迭代器……130
10.2 迭代器的实际用法……142
10.2.1 使用 SimpleXML 解析 XML 文件……142
10.2.2 使用 DBA 访问平面文件数据库……143
10.3 小结……144
第 11 章 SPL 文件和目录处理……145
11.1 文件和目录信息……145
11.2 目录的迭代访问……147
11.2.1 列出文件和目录的清单……148
11.2.2 查找文件……150
11.2.3 创建自定义文件过滤迭代器……151
11.3 SPL 文件对象操作……153
11.3.1 文件内容的迭代访问……153
11.3.2 CSV 操作……153
11.3.3 搜索文件……157
11.4 小结……158
第 12 章 SPL 数组重载……160
12.1 ArrayAccess 接口介绍……160

12.2 ArrayObject 介绍 161
12.3 创建一个 SPL 购物车 162
12.4 使用对象作为键值 165
12.5 小结 168

第 13 章 SPL 异常 169

13.1 逻辑异常 169
13.2 运行时异常 171
13.3 无效函数调用异常和无效方法调用异常 171
13.4 域异常 172
13.5 范围异常 172
13.6 无效参数异常 173
13.7 长度异常 174
13.8 溢出异常 175
13.9 向下溢出异常 175
13.10 小结 177

第四部分 MVC 模式

第 14 章 MVC 架构 180

14.1 为什么使用 MVC 180
14.2 MVC 应用程序布局 181
14.2.1 从 Web 服务器开始 181
14.2.2 动作和控制器 182
14.2.3 模型 182
14.2.4 视图 182
14.3 选择 MVC 框架的标准 182
14.3.1 MVC 框架的架构 182
14.3.2 MVC 框架文档 183
14.3.3 MVC 框架的社区 183
14.3.4 MVC 框架的支持 183
14.3.5 MVC 框架的灵活性 184
14.4 实现 MVC 框架 184
14.4.1 设置一个虚拟主机 184
14.4.2 创建一个 MVC 框架 185
14.5 小结 191

第 15 章 Zend 框架简介 192

15.1 设置 Zend 框架 192
15.1.1 安装 Zend 框架 192
15.1.2 创建一个虚拟主机 193
15.1.3 引导文件 194
15.2 创建控制器、视图和模型 196
15.2.1 添加索引控制器 196
15.2.2 添加视图 196
15.2.3 定义模型 197
15.3 添加功能 200
15.3.1 使用 request 和 response 对象 201
15.3.2 使用内置的操作辅助类 202
15.3.3 使用内置的视图辅助类 203
15.3.4 验证输入信息 204
15.4 小结 208

第 16 章 Zend 框架高级功能 209

16.1 管理配置文件 209
16.1.1 使用数组的方法 209
16.1.2 INI 方法 210
16.1.3 XML 方法 210
16.2 设置站点级别的视图变量 211
16.3 共享对象 211
16.4 错误处理 212
16.5 应用程序日志记录 213
16.6 缓存 214
16.6.1 缓存功能在安全性上的考虑 214
16.6.2 缓存技术 215
16.7 验证用户 217
16.8 在 PHP 语言中使用 JSON 220
16.9 自定义路由 221
16.10 管理会话 223
16.11 发送邮件 224
16.12 创建 PDF 文件 225
16.12.1 创建新的 PDF 页面 226
16.12.2 在 PDF 页面上绘图 226
16.13 与 Web 服务相集成 228
16.14 小结 229

第 17 章 应用 Zend 框架 230

17.1 模块和模型设置 230
17.1.1 常规的模块化的目录结构 230
17.1.2 模型库和 Zend_Loader 231

17.2 请求生命周期 232
17.3 创建插件 233
17.4 创建辅助类 234
17.4.1 编写操作辅助类 234
17.4.2 编写视图辅助类 234
17.5 实现访问控制功能 235
17.6 使用两步视图 238
17.6.1 创建一个主布局 238
17.6.2 使用占位符 239
17.7 小结 240

第五部分 Web 2.0

第 18 章 Ajax 和 JSON 242
18.1 JSON 和 PHP 242
18.1.1 JSON 扩展 243
18.1.2 Zend 框架中的 JSON 244
18.2 JSON 和 JavaScript 244
18.3 一些 Ajax 项目 248
18.3.1 GET 请求 248
18.3.2 POST 请求 249
18.4 小结 252
第 19 章 Web 服务和 SOAP 协议介绍 253
19.1 PHP Web 服务架构介绍 253
19.2 WSDL 介绍 254
19.2.1 WSDL 术语 254
19.2.2 WSDL 文件 254
19.3 SOAP 介绍 256
19.4 使用 PHP SOAP 扩展 257
19.4.1 SoapClient 类的方法和选项 259
19.4.2 SoapServer 类的方法和选项 261
19.5 真实的示例 261
19.6 小结 264
第 20 章 高级 Web 服务 265
20.1 复杂类型 265
20.1.1 复杂类型示例 265
20.1.2 类映射 270
20.2 身份验证 271
20.2.1 HTTP 验证 271
20.2.2 通信密钥验证 271
20.2.3 客户端证书验证 272
20.3 会话 272
20.4 对象和持久化 273
20.5 二进制数据传输 274
20.6 小结 276
第 21 章 证书验证 277
21.1 PKI 安全性 277
21.1.1 CA 277
21.1.2 Web 服务器证书 278
21.1.3 客户端证书 278
21.1.4 根 CA 证书 278
21.2 设置客户端证书验证 278
21.2.1 创建客户端的证书验证机制 279
21.2.2 创建一个自签名的 Web 服务器证书 281
21.2.3 为 SSL 配置 Apache 服务器 283
21.2.4 创建客户端证书 284
21.2.5 只允许证书验证过的客户端访问服务器 286
21.2.6 测试证书 287
21.3 PHP 验证控制 287
21.3.1 将 PHP 绑定到证书上 288
21.3.2 设置 Web 服务验证 288
21.4 小结 289

Part 1

第一部分

OOP 和模式

本部分内容

- 第 1 章　抽象类、接口和契约式编程
- 第 2 章　静态变量、成员和方法
- 第 3 章　单例模式和工厂模式
- 第 4 章　异常
- 第 5 章　PHP 6 中的新特性

试读结束：需要全本请在线购买： www.ertongbook.com

第1章

抽象类、接口和契约式编程

本章将介绍抽象类、接口和一种称为契约式编程的技术。使用这些OOP机制，所编写的代码就不限于只能计算或者输出内容了。这些机制能够在概念层次上定义类之间交互作用的规则，也为应用程序的扩展和定制提供了基础。

1.1 抽象类

抽象类（abstract class）机制中总是要定义一个公共的基类（base class），而将特定的细节留给继承者来实现。具体地说，当需要创建一个基础的对象，而创建所需的某些方法并没有完整地定义出来时，就需要用到抽象类。通过使用抽象概念，可以在开发项目中创建扩展性很好的架构。

例如，文件格式解析功能的实现就非常适合使用抽象方式。实现这一功能时，我们知道，为了与其他类交互，需要一系列方法，如`getData()`或`getCreatedDate()`。然而，我们希望将解析文件格式的方法留给为某种特定文件格式而设计的继承类来实现。通过使用抽象类，我们可以定义一个必须存在的`parse()`方法，而不需要明确这个方法是如何实现的。当然，为了实现起来更加容易，我们也可以将这一抽象的需求和完整定义的方法放在同一个类中。

由于抽象类没有为它所声明的所有方法定义实现的内容，大家可能会将抽象类看作是分部类。抽象类可以不实现所有方法，它具有定义抽象方法的特殊能力，这些抽象方法只是缺少方法体的方法原型。当抽象类被继承时，这些方法将会被实现。然而，抽象类不一定只包含抽象方法，我们也可以在其中定义具有完整实现体的方法。

说明 方法的原型（prototype）是指方法的定义中剔除了方法体之后的签名。它包括存取级别、函数关键字、函数名称和参数。它不包含花括号（`{}`）或者括号内部的任何代码。例如`public function prototypeName($protoParam);`就是一个方法的原型。

由于抽象类没有为它所声明的所有方法都定义实现，所以使用`new`操作符是不可以直接创建它的实例的。要创建实例，就必须创建另一个扩展抽象类的类，并重写所有之前声明的抽象方法原型。通过扩展类，我们就能创建特殊的对象，而且它们同样能够保证提供一套公共的功能。

要充分发挥抽象类的特点，就必须牢记以下规则。

- 某个类只要包含至少一个抽象方法就必须声明为抽象类。
- 声明为抽象的方法，在实现的时候必须包含相同的或者更低的访问级别。例如，如果某个方法在抽象类中的访问级别是受保护的，在继承类中它就必须是受保护的或者公共的，而不能是私有的。
- 不能使用new关键字创建抽象类的实例。
- 被声明为抽象的方法不能包含函数体。
- 如果将扩展的类也声明为抽象的，在扩展抽象类时，就可以不用实现所有的抽象方法。在创建具有层次结构的对象时，这种做法是很有用的。

在类的声明中使用abstract修饰符就可以将某个类声明为抽象的。代码清单1-1中的代码定义了一个抽象类，其中包含了一个具有完整实现的方法和一个将在继承类中实现的抽象方法。

代码清单1-1 定义一个抽象基类

```
abstract class Car {
  //任何基类方法
  abstract function getMaximumSpeed();

}
```

由于这个类是抽象的，不能实例化，它本身起不到什么作用。要让这个类起作用并且获得一个实例，首先必须扩展它。例如，可以创建一个从Car类继承的名为FastCar的类，并且它定义了一个getMaximumSpeed()方法，如代码清单1-2所示。

代码清单1-2 继承抽象类

```
class FastCar extends Car {
  function getMaximumSpeed() {
    return 150;
  }

}
```

现在有了一个可以实例化的类FastCar。下一步，我们可以创建另外一个类Street，它将用到抽象类提供的公共功能，如代码清单1-3所示。

代码清单1-3 使用抽象类提供的公共功能

```
class Street {
  protected $speedLimit;
  protected $cars;

  public function __construct($speedLimit = 200) {
    $this->cars = array(); //对变量进行初始化
    $this->speedLimit = $speedLimit;
```

```
  }

  protected function isStreetLegal($car) {
    if($car->getMaximumSpeed() < $this->speedLimit) {
      return true;
    } else {
      return false;
    }
  }

  public function addCar($car) {
    if($this->isStreetLegal($car)) {
      echo 'The Car was allowed on the road.';
      $this->cars[] = $car;
    } else {
      echo 'The Car is too fast and was not allowed on the road.';
    }
  }

}
```

Street类包含了一个addCar()方法，它的作用是获得一个派生的Car类的实例。现在，我们可以使用Street类，并且向addCar()方法传入一个FastCar类的实例，如代码清单1-4所示。addCar()方法调用了isStreetLegal()方法，而这个方法将会调用定义在FastCar类中的getMaximumSpeed()方法。

代码清单1-4　使用抽象类

```
$street = new Street();
$street->addCar(new FastCar());
```

使用抽象类，我们就可以确定所有继承自Car的对象都会实现getMaximumSpeed()方法，从而共享这个公共的功能。如果继承自Car的类没有定义这个方法，就会导致语法错误，程序也就不会运行。这种限制确保了在实例化层次上的兼容性，这样我们就不需要在发生错误之后调试代码以找出某个对象没有包含特定方法的原因了。

然而，抽象类也有一些限制。PHP只支持从一个基类继承，而不支持从两个或者更多的抽象类继承。从两个或更多的基类继承的能力通常被称为多重继承，PHP在设计上是禁止这种功能的。原因在于，当两个或更多的类定义了具有相同原型且具有完整实现的方法时，从多个类继承会导致不必要的复杂性。当发现需要从两个或更多抽象类继承时，可以将基类的方法拆开，然后使用接口达到相同的目的，正如1.2节所描述的那样。

1.2　接口

接口是一种类似于类的结构，可用于声明实现类所必须声明的方法。例如，接口通常用来声明API，而不用定义如何实现这个API。

虽然接口与抽象类很相似，但是接口只能包含方法原型，而不能包含任何完整定义了的方法。

这可以防止使用抽象类时可能出现的方法冲突，从而能在给定的实现类上使用多个接口。然而，既然接口不能定义具有完整实现的方法，因此如果我们希望为继承者提供默认功能，就必须单独提供一个非抽象的基类。

为了声明接口，需要使用关键字interface。

```
interface IExampleInterface {}
```

说明 大多数开发人员选择在接口名称前加上大写字母I作为前缀，以便在代码和生成的文档中将其与类区别开来。

和继承抽象类需要使用extends关键字不同的是，实现接口使用的是implements关键字。

```
class ExampleClass implements IExampleInterface {}
```

如果将某个类标记为实现了某个接口，但却没有实现这个接口的所有方法，我们将会看到与以下描述类似的错误。

```
Fatal error: Class ExampleClass contains 1 abstract method and must therefore
be declared abstract or implement the remaining methods
(IExampleInterface::exampleMethod)
```

这个错误意味着，如果接口的任何一个方法都没有被声明，那么这个方法就会被当作是抽象的。既然包含抽象方法的类必须被声明为抽象的，那么，为了成功解析这个类，就必须将它标记为抽象的。为了解决这一错误，可以实现所有在基类里被声明为抽象的但在继承类中没有被实现的方法。我们应该实现这些方法，而不是将类标记为抽象的，否则会使这个类不能实例化，而且会将错误继续扩大而已。

为了形成一个完整的类，我们必须实现接口中的所有方法，这样其他类才能依赖于接口中定义的所有方法。只要有一个接口方法没有实现，就会破坏定义公共接口的作用，因而这是不允许的。

前面提到过，接口优于抽象类的一点是每个类可以使用多个接口。当希望在一个类中实现两个或者两个以上接口时，我们可以用逗号将它们隔开。例如，如果有一个具有数组风格的对象，希望它同时具有迭代和计数的功能，那么我们可以定义这样的一个类。

```
class MyArrayLikeObject implements Iterator, Countable {}
```

使用接口完全有可能实现与抽象类相同的操作。通常，在子类和父类之间存在有逻辑上的层次结构时，应该使用抽象类。而在希望支持差别较大的两个或者更多对象之间的特定交互行为时，使用抽象类就会显得不合理，此时应该使用接口。

例如，假设我们希望将代码清单1-1到代码清单1-3中的代码从抽象类的形式转换到接口上。首先，需要创建一个叫做ISpeedInfo的接口。

```
interface ISpeedInfo {
```

```
  function getMaximumSpeed();
}
```

这个接口定义了getMaximumSpeed{}方法，替换了Car类中的抽象方法。

下一步，将Car类中的抽象方法删除掉。然后修改FastCar的声明，让它实现ISpeedInfo接口，如下所示。

```
class Car {
  //任何基类方法
}

class FastCar extends Car implements ISpeedInfo {
  function getMaximumSpeed() {
    return 150;
  }
}
```

这段代码会实现与之前使用抽象类方法几乎一模一样的操作。唯一一个重要的区别在于，在使用抽象类的方法中，可以确定实现类有一个getMaximumSpeed()方法。而在使用接口的方法中，Car类不必知道继承者实现了ISpeedInfo接口。我们看一下代码清单1-5中的代码。

代码清单1-5 使用一个缺少了必须实现的方法的类

```
interface ISpeedInfo {
  function getMaximumSpeed();
}

class Car {
  //任何基类方法
}

class FastCar extends Car implements ISpeedInfo {
  function getMaximumSpeed() {
    return 150;
  }
}

class BadCar extends Car{}

$a = new BadCar();
echo $a->getMaximumSpeed();
```

运行代码清单1-5中的代码，会产生以下错误。

```
Fatal error: Call to undefined method BadCar::getMaximumSpeed()
```

这是因为BadCar类没有实现ISpeedInfo接口，而Car类在调用getMaximumSpeed()方法时没有检查这个接口。不过，就像1.3节所描述的那样，使用instanceof操作符可以检测到这一错

误条件。

1.3 instanceof 操作符

instanceof操作符是PHP中的一个比较操作符。它接受左右两边的参数，并返回一个Boolean类型的值。这个操作符是用来确定对象的某个实例是否为特定的类型，或者是否从某个类型继承，又或者是否实现了某个特定的接口。

说明 “类型”代表的是某个类的运行时定义。PHP在解析类或者接口定义时，会创建一个“类型”。

例如，为了避免如代码清单1-5所示的错误，我们可以使用instanceof操作符来确定Car类的继承者是否实现了ISpeedInfo接口，如代码清单1-6所示。

代码清单1-6 使用instanceof操作符

```
class Street {
  protected $speedLimit;
  protected $cars;

  public function __construct($speedLimit = 200) {
    $this->cars = array(); //初始化变量
    $this->speedLimit = $speedLimit;
  }

  function isStreetLegal($car) {
    if($car instanceof ISpeedInfo) {
      if($car->getMaximumSpeed() < $this->speedLimit) {
        return true;
      } else {
        return false;
      }
    } else {
      //扩展类必须实现ISpeedInfo才能使street合法
      return false;
    }
  }

  public function addCar($car) {
    if($this->isStreetLegal($car)) {
      echo 'The Car was allowed on the road.';
      $this->cars[] = $car;
    } else {
      echo 'The Car is too fast and was not allowed on the road.';
    }
  }

}
```

试读结束：需要全本请在线购买： www.ertongbook.com