目录

[前沿 1](#_Toc690703)

[第一章 物联网背景下的PHP概况 3](#_Toc690704)

[1.1 物联网系统的体系架构 3](#_Toc690705)

[1.2 感知层的技术选择 5](#_Toc690706)

[1.3 传输层的技术选择 6](#_Toc690707)

[1.4 应用层的技术选择 7](#_Toc690708)

[1.5 PHP技术的局限性 7](#_Toc690709)

[1.6 小结 7](#_Toc690710)

[第二章 Web服务器开发环境的搭建 9](#_Toc690711)

[2.1 Apache服务器的安装和使用 10](#_Toc690712)

[2.1.1 Ubuntu下Apache服务器的安装 10](#_Toc690713)

[2.1.2 Ubuntu下Apache服务器的设置 11](#_Toc690714)

[2.2 MySQL服务器的安装和使用 12](#_Toc690715)

[2.2.1 Ubuntu下MySQL服务器的安装 12](#_Toc690716)

[2.2.2 MySQL管理工具MySQL Workbench的安装和使用 13](#_Toc690717)

[2.3 PHP运行库的安装和扩展 15](#_Toc690718)

[2.3.1 Ubuntu下的PHP运行库的安装 15](#_Toc690719)

[2.3.2 常用的PHP运行库扩展程序 16](#_Toc690720)

[2.4 小结 16](#_Toc690721)

[第三章 几种常用的Web开发工具 17](#_Toc690722)

[3.1 XDebug开源的PHP调试工具 17](#_Toc690723)

[3.1.1 Linux下XDebug的安装 17](#_Toc690724)

[3.2 Vistual Code开源的跨平台程序编辑器 18](#_Toc690725)

[3.2.1 Linux下Vistual Code的安装 19](#_Toc690726)

[3.2.2 Vistual Code中PHP相关插件的安装 19](#_Toc690727)

[3.3 Eclipse for PHP Developers全能的程序编辑器 21](#_Toc690728)

[3.3.1 Linux下Oracle Java的安装 22](#_Toc690729)

[3.3.2 Linux下Eclipse for PHP Developers的安装 23](#_Toc690730)

[3.4 Chrome DevTools实用的前端调试工具 24](#_Toc690731)

[3.5 小结 25](#_Toc690732)

[第四章 PHP的面向对象分析与设计 26](#_Toc690733)

[4.1 面向对象的基本概念 26](#_Toc690734)

[4.1.1 理解类与对象 27](#_Toc690735)

[4.1.2 理解面向对象的程序设计 27](#_Toc690736)

[4.2 面向对象的构成与内存分配 28](#_Toc690737)

[4.2.1 类的声明 28](#_Toc690738)

[4.2.2 对象的实例化与内存分配 29](#_Toc690739)

[4.3 面向对象的$this和构造函数的分析 31](#_Toc690740)

[4.3.1 运行时对象属性与方法的动态访问 31](#_Toc690741)

[4.3.2 构造函数和析构函数的运行分析 33](#_Toc690742)

[4.3.3 静态函数与静态变量的运行分析 37](#_Toc690743)

[4.4 面向对象中封装性的编程思想 40](#_Toc690744)

[4.4.1 封装性的基本含义 40](#_Toc690745)

[4.4.2 封装性关键字的使用方法 41](#_Toc690746)

[4.4.3 单例模式的设计与意义 43](#_Toc690747)

[4.5 面向对象中继承性的编程思想 44](#_Toc690748)

[4.5.1 继承性的基本含义 44](#_Toc690749)

[4.5.2 继承性关键字的使用方法 45](#_Toc690750)

[4.6 面向对象中多态性的编程思想 51](#_Toc690751)

[4.6.1 父类和子类的函数重写 52](#_Toc690752)

[4.6.2 抽象类的意义与使用 55](#_Toc690753)

[4.6.3 接口类的意义与使用 56](#_Toc690754)

[4.7 小结 58](#_Toc690755)

# 前沿

狄更斯在他的双城记中第一句话就说：这是一个最好的时代，也是一个最坏的时代。

对于程序员来说确实是这样，回顾2016年可以说是前端和服务端齐头并进的时代，是新旧更替的时代，是众多新技术被广泛验证的时代，如经过2016年一整年洗礼后的React几乎成为了企业开发的标准配置，老牌的前端框架Bootstrap依然老骥伏枥，最新的Bootstrap 4已经进入了Alpha阶段，为2017年的发力做最后的准备，又如PHP 7.0在2015年下半年发布了RC版，在半年时间里收集问题、迭代更新，于2016年1月6日正式发布了7.0.2版本，使得PHP的整体性能提高了至少一倍以上，同时PHP的5.6也得到了长期的支持，最新的PHP 5.6.29在2016年12月8日正式发布，进一步巩固了PHP在服务的地位，此外阿里在2016年10月14日的杭州云栖大会上发布了基于MySQL数据库的AliSQL，在普通环境下其性能在MySQL的基础上提高了70%，而在特定环境下性能可以提高近百倍。

对于程序工程师来说，除了技术的变化，岗位的变更也在悄悄的进行，原先的前端工程师主要责任是负责Web页面的开发，但是到了2016年前端工程师熟悉的JavaScript不仅可以编写Web编写，还可以进行跨平台的应用开发，从大屏到小屏都能胜任，做到程序的一次编写就可以适用多种屏幕，甚至在性能方便不输于采用Java或者Object-C所开发的原生应用，此外原先负责后台应用的工程师也随着大数据、云计算普及也向着数据分析、数据整合、以及数据挖掘发展，特别是越来越多的中小型的创业企业通过运用类似与ThinkPHP这样基于PHP语言的快速开发框架从而验证项目的可行性，以确定项目的未来走向。

因此可以看出2016年中每一天我们的身边都在进行了翻天覆地的变化，技术的发展速度远远超过了个人学习的速度，可能几天不刷论坛，当再进入时就会冒出几个新名词，再过几天网上已经炒的沸沸扬扬，因此没有一本书或者作者能够一直站在技术的最前端，那么有没有一个全栈技术能够经久不衰，任凭外面技术如果的变化，但是它一直都能解决大部分企业的问题？答案是肯定的，采用Bootstrap+PHP+MySQL就可以。

本书从实际工程应用入手，通过实验过程和实验现象讲解系统间的内在联系，并且通过研究各系统的设计原理和设计缘起，讲解在进行系统开发时的思考方法，由浅入深、循序渐进的方式深刻剖析了如HTML 5的语义设计思想、PHP 5中的面向对象的构建理念，以及MySQL数据库中的范式的意义，本书不同于传统讲解前端或者后端的书籍，把两者之间进行割裂，读者在阅读后依然没有一个完善的体系在心中，本书想通过真实的项目把作者的实践中的所思所想、学习方法，以及相对较为完善的体系与各位读者分享，从而能够快速适应这个变化的时代。另外，本书的内容均来自作者科研、教学和项目实践，因此所涵盖的内容是作者多年来经验的总结。

本书适用读者

目前国内对于各项技术相对入门的内容都有完善的书籍，因此本书在编写时对于一些相对基础的内容加入的较少，主要是以最新的、最稳定的技术为切入点，讲述在具体项目中的开发思路，以及对许多技术中相对原理性，但是又一般在很多书籍中不涉及的内容进行项目的阐述，因此在阅读本书时需要有一定的HTML5、CSS3、PHP、MySQL基础，或者这些技术有一定的了解，主要面向的读者有一下四种：

1、对Web开发有兴趣，并想深入了解原理的读者

2、基于Web的移动互联网开发人员

3、在大中专院校进行Web相关研究的老师

4、有志从事Web开发的在校学生

本书内容与组织结构

本书是我们团队编写的关于Web开发系列丛书中的一本，目的是帮助有HTML或者PHP相关基础的工程师能够通过学习本书更加深刻的理解各系统的运作机理，并把这些应用于实际项目中，从而提高项目的可维护性和可扩展性，全书共分为8个章节。

第1章介绍了物联网环境中PHP如何定义，并且给出了物理网的体系架构和模块通信方式。

第2章介绍了解Web全栈系统的开发环境的搭建。

第3章介绍了在生产环境中Web全栈系统开发所用到的工具，以及这些工具的配置方法和使用方法，从帮助工程师提高开发效率。

第4章探讨了在面向对象程序设计思想下的PHP程序设计方法，并且讨论了在程序运行时内存的分配和回收机制，帮助工程师深入理解系统的工作原理。

第5章探讨了在HTML5中语义标签的使用方法和设计理念，然后讨论利用面向对象的概念来设计CSS3样式表，最后讲解面向对象的JavaScript的设计思路。

第6章探讨了响应式框架Bootstrap的设计原理和设计思路，并基于此讨论在前端开发中的开发规范，以及封装方法，然后在Bootstrap的基础上开发一个基于学生工作过程管理系统的登录界面。

第7章讨论了数据库设计时的几种范式的作用，并且讨论、设计出基于学生工作过程的数据库，最后探讨通过常用数据库操作指令来完成对数据库的操作。

第8章探讨了基于最新ThinkPHP 5.x框架的应用开发方法，并且在其中学习ThinkPHP的设计原理和实现过程，从而使得工程师能够更加的灵活运用该框架设计出高效的系统。

# 第一章 物联网背景下的PHP概况

毛主席在其《矛盾论：矛盾的特殊性》中说到针对问题不能“只见树木，不见森林”，头疼医头脚疼医脚。学习一项技术也是这样，需要充分了解这项技术所处的环境，以及应用场景，当前该技术最新的发展情况如何，有哪些已经是成熟的方法被广泛使用，以及这项技术的优缺点有哪些，只有这样全面的了解过后再深入的学习时就能够有的放矢、循序渐进的开展，并构建起自己完整的知识体系，从而为后续学习打下坚实的基础。不同于以往的小规模商业试用和科学研究，随着NB-IOT、LoRa等物联网技术在运营商和大型企业的推动下众多的物联网项目犹如雨后春笋般在国内落地开花，而PHP以其自身简单、易用、稳定，同时还具有成熟的生态环境等优势在物联网项目大量落地的今天也占得了一席之地。物联网项目不同于其他软件项目，它是大系统的集成，既包含了软件又包含了硬件，因此需要众多的技术相互配合才能完成。因此，本章首先想通过以构建一张完整的物联网技术架构，讲解PHP在其中是如何做到承上启下，然后以此为基础为各个功能子模块提供实现的解决思路，最后通过PHP局限性的说明给读者一个辩证的思路，从而在后续的项目开展中能够对技术的选择有清晰的认识。

本章内容

* 物联网系统的体系架构
* 感知层的技术选择
* 传输层的技术选择
* 应用层的技术选择
* PHP技术的局限性

## 1.1 物联网系统的体系架构

“物联网”这词语最初的时候是一个外来词汇，其完整的英文名叫“Internet Of Things”（简称：IOT），顾名思义就是很多物体连成的网络，所以叫做物联网。随着人工智能等技术在国家层面的大力扶持与推广，2018年下半年又出现了一个新的词汇——AIoT，即人工智能（AI）与物联网（IOT）相结合。虽然这几年各种新技术不断涌现，各个单独模块的具体实现上通过采用各种更为先进的技术进行不断改进，而物联网的基本体系架构依然没有发生根本性的改变，因此只要深入的了解物联网系统的体系架构和运作模式，读者就可以脱离纷繁复杂的技术局限，根据自身的技术特点和擅长把各类具体的实现手段进行自由组合，从而实现一个或简单或复杂的物联网系统。

要学习一个技术首先需要了解该技术最初时是用来解决什么问题，下面通过三个故事来讲述物联网的源起。

1、故事一：1991年剑桥大学特洛伊计算机实验室的科学家们为了解决去楼下倒咖啡经常空手而归的问题，在楼下咖啡壶旁安装了一个摄像头，通过图像捕捉技术实时分析咖啡是否煮好，从而帮助实验室的科学家们可以随时监控咖啡的情况，而不用经常白跑一趟。到了1993年，这套咖啡监控系统被搬到的互联网，有近240万人点击浏览了这个监控网站，甚至有更多的人希望能亲眼目睹这个已经成为网红的“特洛伊咖啡壶”。虽然现在看来这是一个很小的项目，但可以说它包含了物联网体系架构中的大部分内容，而随后的不论市场开发还是基于应用都可以找到这个咖啡壶的影子。

2、故事二：1995年微软的创始人比尔盖茨在他的新书《未来之路》中描述了物联网的基本雏形，即当时的互联网已经实现了计算机和计算机之间的联网，但是还没有实现万物互联的情况，但他相信随着技术的不断发展，未来一定可以实现。可以说比尔盖茨在他的这本预言书中所描绘的物联网情景正在被逐步实现。

3、故事三：1998年英国工程师凯文·艾什顿发现宝洁公司的不少零售店的库存信息与销售状态不符，于是在宝洁公司的一次演讲中首次提出了“物联网”的概念。随后宝洁公司与MIT成立了自动识别研究中心，主要研究RFID技术以及智能包装系统。当时，艾什顿对物联网的定义相对也更具针对性，即把各类商品通过射频识别等技术与互联网相连接，从而实现智能识别与智能管理。而MIT的自动识别研究中心提出的物联网概念就相对宽泛，即在互联网的基础上，利用RFID、无线传感器网络、数据通信等技术，构造一个覆盖世界上万事万物的“物联网”，在这个网络中，物品之间能够彼此“交流”，而无需人为干预。

从上面三个故事不难看出，不论最初的特洛伊咖啡壶还是之后的MIT的自动识别研究中心，他们所构建的物联网都有一个相同的体系架构，即三层架构，分别是感知层、网络层和应用层，如图1-1所示，具体如下：

1、感知层：主要负责获取和控制各类传感器和智能设备的数据，如小到简单的温湿度传感器，大到复杂的智能设备，如机械臂等。这部分的实现主要通过各类智能硬件来完成，其本质就是在传统的嵌入式系统上再加上一个网络模块，从而实现网络通信，并且把以往保存在本地的设备运行过程数据，通过格式化的数据传递到远端服务器。

2、网络层：主要负责把感知层的数据上传给远端应用层，以及把应用层的计算结果下发至感知层。这部分可以根据具体需求的选择是采用广域网的方式（俗称“上云”）还是采用局域网的方式，所谓城域网其本质也是局域网的一种。局域网和和广域网除了本身设备的成本之外，在使用上广域网设备还需要向运营商支付一笔网络使用费，但网络的维护由各个运营商来完成，而局域网则不需要网络使用费，但是网络的持续维护需要项目建设方来完成。

3、应用层：可以说是这个物联网系统中的计算大脑，它把从感知层、用户操作等多方面收集到的数据进行计算、分类、统计、识别等一系列操作，并把一部分结果下发至感知层，从而实现这个物联网系统从底至上再到底的闭环。应用层所包含的技术最为复杂，也是延展性最大的部分，可以说所有和计算机相关的技术都能够应用到该部分中，小到一个设备管理系统，大到包括人工智能、移动互联网、机器人等应用。

如果再要把物联网进行形象的比如，那么就可以和人进行类比，感知层就相当于人体的各个运动部件和感觉器官，而网络层则是人体内的各个传导神经和中枢神经，网络层把从感觉器官获取的信息传递到后面的应用层，而应用层就是人体的大脑，用于对收集到的数据进行计算，并反馈到运动部件进行控制。

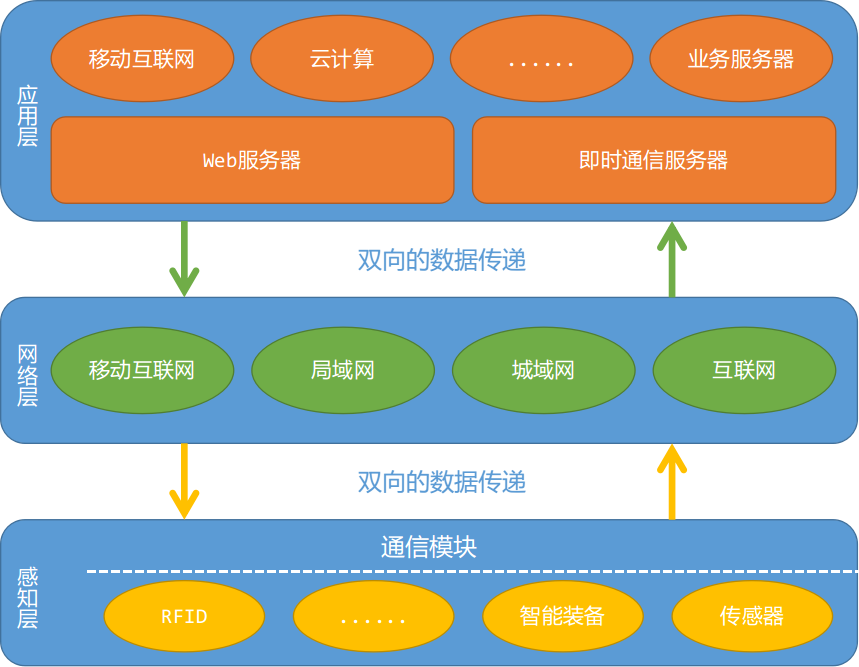


图1-1 物联网体系架构

## 1.2 感知层的技术选择

感知层的相关技术主要偏向于嵌入式系统与应用，要讲解这部分的技术可以从两个层面进行说明，分别是硬件层面和软件层面这两个方向进行，具体如下：

1、硬件层面： 根据不同的应用领域，硬件的选择大致可以分为两大类，分别是低频小数据量的信息采集与控制和高频大数据量的信息采集与控制。针对低频小数据量的信息采集与控制的这一类应用经常会采用ARM公司的Cortex-M系列相关内核的SOC芯片，其中最常用的内核为Cortex-M3和Cortex-M0。所谓SOC芯片就是片上系统，简单的理解就是在一个芯片内部集成的控制器、存储器、外围通信接口等众多内容，类似与一个完整的电脑主机，而ARM公司本身并不生产芯片，只设计相关的芯片内核，然后把这些设计的芯片内核授权给各个芯片生产商，各芯片厂商再根据所授权的IP进行例如存储器、外围通信的扩展与裁剪，最后生产出自己的芯片，所以在选择芯片时会发现即使是相同内核，如Cortex-M3，也会有众多的厂商生产应用于不同领域的各类芯片，类似于很多电脑主机，虽然都是相同的CPU，但有的内存大些，有的硬盘大些，形成中众多不同类型的电脑。目前，常见芯片厂商有意法半导体（STMicroelectronics）、恩智浦（NXP）、德州仪器（TI）等，这些公司的所生产的芯片都有各自的特点和应用领域，甚至在某些细分行业的设备研发和生产中由于一些芯片公司常年的深耕，慢慢形成了一定的行业规则，即只使其中一家或几家所生产的芯片。而针对高频大数据量的信息采集与控制就需要更具不同的应用场景，以及研发成本综合考虑是使用FPGA、GPU或Cortex-A相关芯片，然而往往为满足要求会采用多芯片的组合的解决方案来实现低成本、高性能，例如控制用Cortex-A或Cortex-M系列，而计算则使用FPGA或GPU等。此外，在研发硬件设备时还需要考虑后续的应用场景，以及是否需要出口到其他国家，因为很多场景的产品都有其国标，以及出口目标国的标准，产品必须符合这些标准才能够进行量产和售卖。

2、软件层面： 根据不同的需求，软件可以分为三大类，分别是无操作系统软件、实时操作系统（RTOS）软件和分时操作系统（OS）软件，其中无操作系统软件即裸机程序，这在Cortex-M系列芯片上使用的最为常见，它通过直接配置和读写寄存器就能很好的完成各项任务，包括传感器的读取，数据的收发等工作。而当任务要求具有较高响应或程序体量不断扩大时就需要考虑引入实时操作系统，该操作系统能够以足够快的速度处理各类请求，其处理的结果又能在规定的时间之内进行响应，同时还控制所有实时任务协调一致的运行。目前实时操作系统较为常用的有RTThread、FreeRTOS、UCOS-II/III等，其中RTThread是国人自主研发的嵌入式实时操作系统，并且经过近10年的发展，目前已经非常稳定，并且各类组件和文档都非常的丰富和完善，而FreeRTOS和UCOS-II/III则是在国际上使用非常多的操作系统，其中UCOS-II/III最早引入到国内，有专门的书籍讲解该操作系统的使用，但该操作系统只能用于学习和非盈利，如果商用就需要支付一定的费用，FreeRTOS则不同，不论商用还是学习都不收取任何的费用，但资料相对较少。有了操作系统还需要一个小屏幕进行人机交互，目前针对RTOS的UI库有三种，一种是由德国Segger开发的emWin，该库开发简单，但界面相对较老，属于Win95/98的风格，此外还有由Draupner Graphics开发的TochGFX，该UI库效果非常炫酷，但是开发难度较大，以及RTThread开发商自主研发的Persimmon（柿饼），该UI库开发简单，效果也相对现代。除了实时操作系统，另外一种就是分时操作系统，也是最常说的操作系统，嵌入式领域中针对民用的其实只有Linux一种选择，而军用和航空级的也还有一些其他选择，该操作系统通常都应用于Cortex-A系列的芯片上，用于处理如音视频或大数据量的网络通信等内容的处理。

## 1.3 传输层的技术选择

传输层的相关技术主要偏向用网络通信，要讲解这部分的技术可以从三个层面进行说明，分别是感知层的网络出口、网络通信、应用层的网络入口，具体如下：

1、感知层的网络出口： 目前感知层的网络通信按照研发方式的不同可以分为两种方式，第一种方式是购买现成的网络通信模块，通过主芯片的串行通信接口把要发送的数据发送给网络通信模块，由该模块进行数据的发送和接收，采用这种方式的优势是开发简单，周期也相对较短，当通信模块发生改变时几乎不需要修改任何程序就可以直接使用，但劣势也很明显，就是会增加设备的硬件成本。第二种方式是通过自行对协议栈软件进行研发和实现，常用的协议栈软件有Z-Stack、LwIP等，采用这种方式优点是硬件成本较低，并且可以进行深度定制，同时还可以可以充分利用网络芯片的其他硬件资源，同样劣势也很明显，即开发周期长，对研发人员有较高的要求，同时程序的稳定性也有待后续的验证。

2、通信网络： 目前通信网络的分类主要针对的是无线通信，而如果采用有线通信方式，那么通常需要使用RJ-45或者RS-485等方式连接至中心机房后再传递至云端。而无线通信则可以分为两大类，分别是广域网通信和局域网通信。其中如果采用广域网通信，那么理论上就有三种解决方案，即GSM、NB-IOT、eMTC，其中NB-IOT和eMTC都具有低功耗的特性，同时NB-IOT和eMTC可以直接在GSM网络进行建设，这也是运营商主推这些技术的原因，目前要使用该技术主要的方式是和各个本地运营商进行谈判，并购买相应的物联网卡，以及从运营商或者相关渠道购买通信模块，然后集成到自己的硬件中，然后通过AT指令就可以和服务器交互。目前各个运营商也都在实验多模通信方式，未来在一个基站上能够同时实现GSM、NB-IOT、eMTC通信，形象的比喻就是手机里的全网通。这里需要注意的是，很多基站为了保证数据通信的效率都会定期清除长时间没有数据交互的通道，因此采用GSM、NB-IOT或eMTC时需要定期发送心跳包，以保证数据链路的存在。而如果采用局域网方式那么就要根据通信范围的大小进行选择，例如通信范围为一个城市，那么可以采用LoRa通信技术来实现，通常来说低功耗一般很难覆盖远距离，而远距离一般都是功耗高，而LoRa最大特点就是在同样的功耗下传播距离是其他无线方式传的3-5倍。因为LoRa的传输距离是以公里为单位，因此通过少量的中继就可以覆盖很广的范围，而如果是通信范围是为一个区域，那么可以采用Zigbee或者蓝牙4.0+等方式来实现组网和数据通信，虽然采用这两个技术的模块单独一个只有几十米到几百米的通信距离，但是可以通过中继的方式实现无线的扩展。此外，对于局域网通信来说，不论是采用LoRa这样的远距离通信，还是Zigbee、蓝牙这样的短距离通信，要实现与服务器的交互都必须要通过一个网关来实现，这就要求开发人员对每个节点的分布和每个节点有单独的编号和统一的管理，以便在远端分辨出不同的设备。

3、应用层的网络入口： 数据一旦需要传递到远端服务器，那么就有两种方式可以选择，一种是HTTP通信，另外一种则是Socket通信。其中HTTP通信是一种短连接的通信方式，通信双方交互完数据后就会把数据链路断开，如果需要再次通信，那么就要与服务器重新链接，此外由于建立链接的过程需要经过多次握手通信，设备的功耗也相应提高，所以这种通信更适合不频繁的数据操作，例如用户交互等。要采用这种这种方式通常只需要架设Web服务器，如Apache或Nginx服务器，再配合本书后面讲的基于MVC架构的ThinkPHP软件库就可以实现。而Socket通信则是一种长连接的通信方式，通信双方的链路始终保持接通，方便了数据的实时传递，也免去了多次握手的情况，因此这种方式更适合对数据实时性要求较高或者要求低功耗的通信，其软件可以采用基于PHP的高性能异步通信解决方案，如Workerman和Swoole，其中Workerman的特点就是开发容易、能满足一般的开发需求，部署也相对简单，由于采用纯PHP进行开发，因此效率上略逊于Swoole，而Swoole采用纯C语言进行扩展，因此内存管理、通信协议、效率上略高于Workerman，但是开发相对复杂，周期也相应的有所增加。在实际的物联网项目开发中，HTTP和Socket都会同时用到，因此不论是HTTP服务器还是Socket的服务器都需要进部署才能满足各种网络通信。

## 1.4 应用层的技术选择

应用层的技术选择最为复杂，根据不同的需求可以说是千变万化，但总体上来说由三部分组成，分别是业务服务器、数据服务器，以及前端展示。其中业务服务器主要完成从Web服务器和Socket服务器传递来的数据进行计算和分析，以本书讲解的内容为基础，简单的构建就是把Socket服务器和ThinkPHP进行整合，并在其中完成相应的业务计算，以及数据反馈，而复杂构架则可以引入更多的分层服务器，从而实现更加复杂的内容。而对于数据库服务器来说，通常采用MySQL作为基础，如果项目的体量很大，以及要求响应速度较高，那么就可以再配合Hadoop等分布式的数据库，以及Redis这类内存数据库来提高相应速度。最后的前端展示，则是可以采用移动端与Web端来完成，其中移动端除了采用原生方式进行开发外，还可以采用目前最为流行的H5方式进行实现，其技术方案可以从Angular+Ionic、React Native、Weex中进行选择，而Web端的逻辑可以采用JQuery、Angular或Vue等方式来实现，而UI则有众多的选择，这里推荐Bootstrap、Amaze UI、NG-ZORRO或Element，这些都是较为现代的UI框架。

## 1.5 PHP技术的局限性

再好的东西只要寻找都能找出问题，就像PHP一样，经过20多年的发展与完善，使得它具有开发快速、配置简单、性能稳定、容易部署、生态完善等一系列优点，虽然不论何时都不断有人指出它的各种缺点，但依然阻止不了它成为Web领域中使用最为广泛的语言之一。但为了在进行技术选型时能否充分的辩证思维，也需要了解该技术的局限性，使得在项目实施时可以对对可能出现的问题和瓶颈也有一个合理的预判。PHP的诞生之初就已经决定这一门语言的局限应，即它所面向的技术领域就是Web应用，并且追求的是入门简单、快速开发，这些定位就决定的PHP具有如下三点问题：

1、页面级的资源回收：PHP属于脚本语言，采用的是解释运行机制。这种运行机制使得当PHP页面被解释并执行完成后所有的相关资源都会被回收。也就是说，PHP语言本身就不支持对象驻留等操作，因此不论变量是全局变量，还是静态成员，又或者是局部变量都会在页面执行完毕后被清空。

2、单线程、非异步的运行模式：由于历史和最初设计的原因使得该模式始终存在，因此PHP从根本上就无法应对成千上万大规模的并发请求。

3、共享的全局配置文件：这样的方式使得PHP在部署时无法进行个性化的配置和使用。

其实PHP还有很多其他问题，如函数命名不规范、安全机制相对欠缺等问题，但总体来说PHP作为一个快速验证市场或小规模应用的项目来说非常合适，同时完善的生态系统也为PHP提供许多经得起实验考验的框架，这些能否帮助开发人员快速的进行开发，减少开发中的问题，而对于大型项目来说PHP作为非核心的前端承载也非常合适。

## 1.6 小结

通过本章的学习，了解了PHP在物联网背景的应用场景和使用方式，并且深入了解了物联网中感知层、网络层以及应用层常用的技术。在本章中首先详细讲解了物联网的体系架构，以及该架构中各模块之间通信方式，其次详细了解感知层中硬件部分的基本选型方式和软件的开发模式，然后有了解了在传输层中常用的几种通信方式的特点，以及如何利用PHP实现这些数据的接收，接下来又讲解了应用层中使用的一般技术和几种常用的软件框架，最后讲了PHP技术的局限性来帮助读者能否辩证的看待问题。通过这些知识的学习读者基本可以对物联网中的各个模块的运行方式有了一个清晰的认识，由于物联网本身就是多种技术的组合，因此本章内容也只能起到引路的作用，就像提到的各种问题和各种框架，其实都可以通过其他方式进行弥补，以及替代，但所涉及的内容又会超出本章，甚至本书的内容，因此本章只能做到临渊羡鱼,不如退而结网。

# 第二章 Web服务器开发环境的搭建

古人有云：“不积跬步无以至千里，不积小流无以成江海”。软件开发也是这样，要运行一个复杂系统就必须要从一砖一瓦开始搭建起来。默认情况下计算机是无法识别PHP程序，因此想让计算机能够运行PHP程序，就需要在计算机上搭建一个PHP程序的运行环境，这个运行环境包括三个部分，分别为Web服务器、PHP解析器、数据库服务器，这三个部分有机的结合支撑了这个世界上所有的Web应用，包括网站、短链接的APP等，当然随着应用不断的复杂，这里面内容也会跟着发生巨大的变化，但概括来说整个系统运行分为以下六步，如图2-1所示，

第一步：用户通过HTTP向Web服务器发送请求，这个请求可以理解为在浏览器中输入一条网址。

第二步：Web服务器接收到请求后，就会去读取请求路径下的代码文件。

第三步：如果Web服务器发现代码为HTML这样的静态文件，那么就会直接解析，并且还给用户，如果发现代码文件不是静态文件，那么就会把该文件交给PHP解析器进行解析，同时等待PHP解析器把解析好的静态数据，并返回给Web服务器。

第四步：PHP解析器收到Web服务器提交的任务就会开始解析PHP文件，并把结果以静态数据的形式返回给Web服务器。

第五步：PHP解析器在解析的过程中很多时候会对数据库进行读写操作，从而使页面数据动态化，也就是为什么不同用户登陆到某个论坛显示的用户信息都是不同的。

第六步：当所有代码和数据都解析完成后PHP解析器会把这些数据变为静态数据返回给Web服务器，Web服务器在把数据返回给用户，从而看到的就是动态页面。

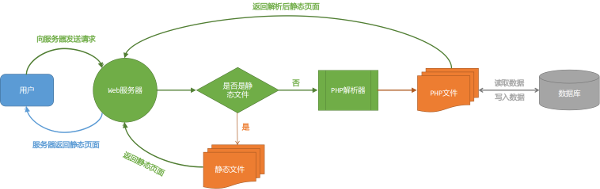


图2-1 Web服务器运行流程

目前世界上90%以上的Web服务器都是采用基于Linux操作系统，因此本书也是基于Linux的一个非常流程的发行版Ubuntu进行讲解，从而帮助读者可以快速的把书本内容应用于实际中。在Ubuntu系统中搭建完整的PHP开发环境的工具称之为LAMP，这里所说的LAMP不是一个工具，而是取四个工具名称的首字母的统称，这四个工具分别为Linux操作系统、Apache服务器、MySQL服务器、PHP，除了Linux操作系统，本章讲讲解在Ubuntu下如何安装另外三个工具。

本章内容

* Ubuntu下的Apache服务器的安装和使用
* Ubuntu下的MySQL服务器，及管理工具的安装和使用
* Ubuntu下的PHP的安装和使用

## 2.1 Apache服务器的安装和使用

目前被广泛使用的Web服务器有三种，分别是Apache、IIS、Nginx，虽然Apache已经有超过20年的历史，几乎可以说是伴随着Web发展一路走来，虽然在计算机领域里面20年可以算是老爷爷了，但是折不妨碍Apache成为Web服务器领域的领头羊，即使在今天有IIS和Nginx冲击的情况，依然不妨碍它成为全世界使用最为广泛的Web服务器，市场占有率超过50%。在国内如淘宝、优酷、新浪等知名网站都采用Apache或基于Apache的优化版作为其Web服务器，Apache的图标如图2-2所示。



图2-2 Apache的图标

### 2.1.1 Ubuntu下Apache服务器的安装

在Ubuntu下安装Apache服务器是非常简单的，因此在Ubuntu的软件源中已经包含了Apache服务器，所以只需要在终端里输入一条指令，并根据提示操作就可以完成Apache服务器的安装，当安装完服务器再在终端中执行apache2 -v，如果能返回Apache版本就说明安装成功，如图2-3所示，具体指令如下。

1 sudo apt-get install apache2

2 apache2 -v

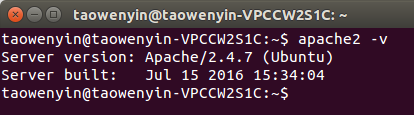


图2-3 Apache安装验证

当安装完Apache服务器时，读者可能会发现在终端里显示一个错误，如图2-4所示。这个错误的意思是没有给服务器指定域名，因此只需要在配置文件中为Apache服务器设置域名就可以，具体步骤如下。

第一步：在终端中执行下面的指令，将当前目录切换到/etc/apache2目录下。

1 cd /etc/apache2/

第二步：在终端中执行下面的指令，打开apache2.conf文件，并在最后添加ServerName localhost:80，如图2-5所示。

1 sudo gedit apache2.conf

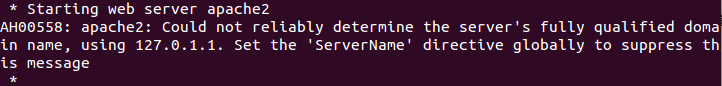


图2-4 安装Apache的错误

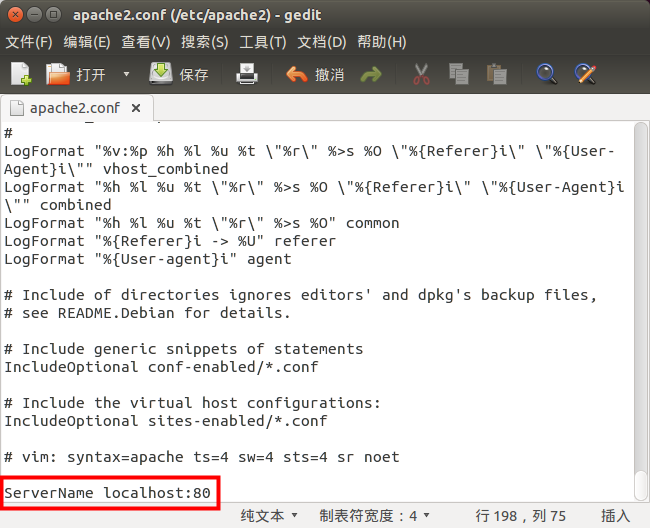


图2-5 为Apache设置ServerName

第三步：在终端中执行下面的指令来重启服务器，此时就不会看到错误产生。

1 sudo service apache2 restart

第四步：打开浏览器，在地址栏中输入localhost，如果看到Apache的主页，就表示安装完成，如图2-6所示。

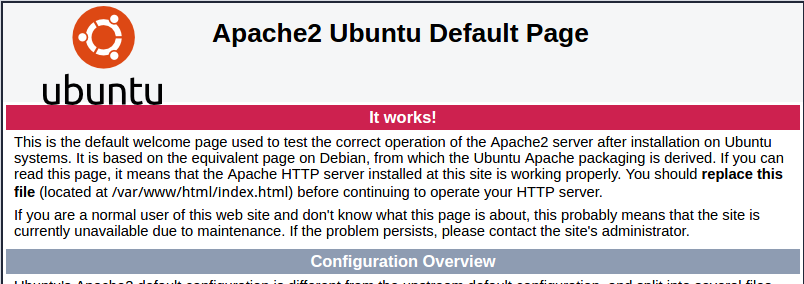


图2-6 验证Apache设置成功

### 2.1.2 Ubuntu下Apache服务器的设置

Apache服务器的主目录是经常要修改的内容，因此本节将讲述如何通过修改配置文件来实现Apache主目录位置的修改。Apache主目录位置的修改主要是修改/etc/apache2目录下面的apache2.conf文件和sites-available/000-default.conf文件。其中apache2.conf主要是修改主目录的位置，而sites-available/000-default.conf则是修改域名重定向时所进入的子目录，例如当前设置Apache主目录位置为/var/www/,而000-default.conf则设置localhost域名指向/var/www/html,那么当在浏览器中输入localhost时实际上读取的是/var/www/html下的文件，所以000-default.conf指定的目录一定是apache2.conf中主目录下的子目录。具体如下：

第一步：在终端中执行下面的命令打开，打开/etc/apache2/apache2.conf文件，并把164行中的/var/www/改成主目录的位置，然后保存并关闭。

1 sudo gedit /etc/apache2/apache2.conf

第二步：在终端中执行下面的命令打开，打开/etc/apache2/sites-available/000-default.conf文件，并把12行中的/var/www/html改成主目录下的子目录，当然也可以和主目录写的一样，那么此时就直接读取主目录下的文件，然后保存并关闭。

1 sudo gedit /etc/apache2/sites-available/000-default.conf

第三步：在终端中执行下面的命令重启Apache服务器，然后在浏览器中输入localhost就可以看到当前的目录已经切换到所指定的目录下。

1 sudo service apache2 restart

## 2.2 MySQL服务器的安装和使用

目前单机版的应用越来越少，即使是笔记本这类应用都已经从原来系统自带的笔记本变成如印象笔记、有道云笔记等类似的云笔记,而在游戏领域，原先的PlayStation和XBox上基本都是单机版的游戏，而最新的PS4和Xbox One上的游戏也由单机版向网络版发展的趋势，这些基于网络的应用都是通过有线或者无线的方式把数据存储在远端的服务器上，可以让用户随时随地的对数据进行增、删、改、查的操作，而保存这些数据的服务器就是数据库服务器，当然数据库的种类现在也非常多，有基于Key-Value数据类型的Redis数据库，有基于分布式文件存储的数据库MongoDB等，而本节讲述的是目前被广泛使用的关系型数据库MySQL，该数据库和Apache服务器一样，是开源、免费的软件，但他同样有着悠久的历史，并被一次次在大型系统中证明其高度的稳定和快速的性能，至于阿里巴巴在2016年8月9日云栖大会北京峰会上发布了基于MySQL开发了一个全新的分之AliSQL，用于进一步提升数据服务器的运营能力，MySQL的图标如图2-7所示。



图2-7 MySQL的图标

### 2.2.1 Ubuntu下MySQL服务器的安装

在Ubuntu下安装可以采用和Apache相同的方法，因为系统默认的软件源已经添加了MySQL的软件源，但该软件源中的MySQL并不是最新版的，因此本节采用从MySQL官网下载最新的软件源更新包，来更新系统自带的软件源，然后在通过命令行的方式进行安装。

进入MySQL软件源下载页面（http://dev.mysql.com/downloads/repo/apt/），在页面的最下方就是软件源的更新文件，如图2-8所示。要下载该软件源更新包需要先在MySQL官网进行注册账号，注册完成后就可以下载。

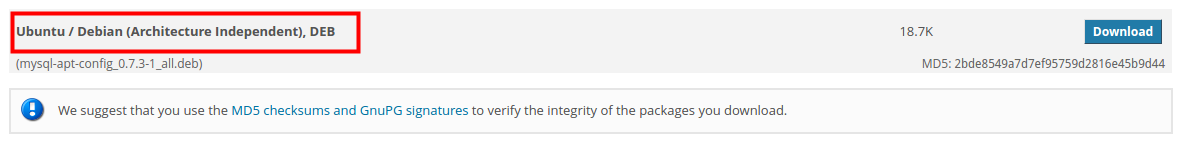


图2-8 MySQL源更新文件

下载下来后的文件是以.deb结尾的文件，该后缀名结尾的文件是Ubuntu下标准的安装包，类似于Windows下的.msi,因此只需要执行双击该文件就可以进行安装，安装的过程非常简单，只需要根据提示完成就可以，在安装过程中会弹出一个对话框用于设置MySQL的Repository，如图2-9所示，只需要在下拉列表中选择OK，然后根据提示点击前进就可以完成。当配置完成完后只需要在终端中执行更新软件源的指令，然后执行安装MySQL Server的指令就可以完成整个最新版MySQL数据库的安装，在MySQL数据库的安装过程中会要求输入MySQL的登陆密码，而用户名是默认的，为root，如图2-10所示，具体安装指令如下。

1 sudo apt-get update

2 sudo apt-get install mysql-server

安装完成后通过下面的三条终端指令可以检查MySQL Server的状态，同时还可以停止MySQL Server和启动MySQL Server，具体指令如下。

1 # 检查MySQL Server的状态

2 sudo service mysql status

3 # 停止MySQL Server

4 sudo service mysql stop

5 # 启动MySQL Server

6 sudo service mysql start

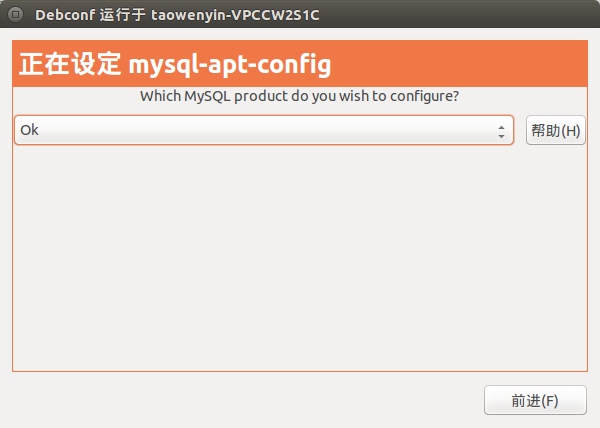


图2-9 MySQL源更新配置

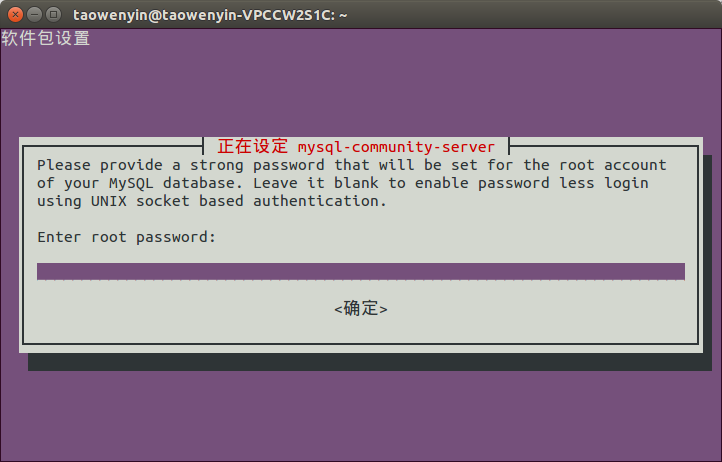


图2-10 MySQL登陆密码设置

### 2.2.2 MySQL管理工具MySQL Workbench的安装和使用

MySQL安装完成后，就可以在其中创建数据库、数据表，以及数据，当目前所有的操作都是基于命令行的方式，虽然这种方式很重要，熟练运用SQL语言是每个程序开发人员必备的素质，但是在实际的运营中，采用命令行的方式其效率就相对较低，因此需要一种GUI的方式来管理数据库，常用数据库管理软件有Navicat for MySQL和phpMyAdmin，其中Navicat for MySQL的性能和跨平台性确实能够很好的提高运营的效率，但是由于是收费软件，因此本文也不推荐，另外一款是phpMyAdmin是一款基于Web的MySQL管理工具，同时也是免费的，有兴趣的读者也可以尝试，而本文推荐的则是MySQL自己开发一款管理工具——MySQL Workbench，这款工具兼具了免费、高性能特点，可以和MySQL无缝对接。

MySQL Workbench的安装一定是在前面已经更新了软件源的基础上进行安装，虽然默认软件源中的包含了该软件，但是版本相对较老。MySQL Workbench的安装和MySQL Server的安装没有什么区别，也只需要在终端中输入下面的指令就可以完成。

1 sudo apt-get install mysql-workbench-community

安装完成，在软件列表就可以找MySQL Workbench的图标，如图2-11所示。点击该图标就可以打开该软件。首先进入MySQL Workbench的登陆界面，如图2-12所示。图中的标记1是系统自动创建的连接本地MySQL Server的快捷方式，但用户点击时就会弹出标记2的对话框，用于输入登陆的密码，当密码正确时就会进入MySQL的管理界面，此外标记3用于创建数据库的E-R图，同时还能通过E-R生成对应的SQL文件。当用户通过MySQL Workbench登陆数据库成功后，就可以看到图2-13的所示的主界面，在该界面中标记1是标签栏，用于切换不同的MySQL Server，标记2则是工具栏，可以用来打开数据库的SQL文件、创建数据库等功能，标记3用来查看该MySQL Server中包含哪些数据库、数据表，而标记4则是用于执行SQL语句，最后的标记5则显示查询结果的集合。



图2-11 MySQL Workbench的图标

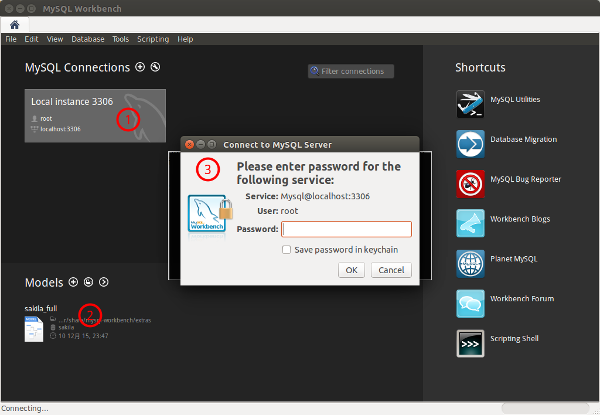


图2-12 MySQL Workbench的登陆界面

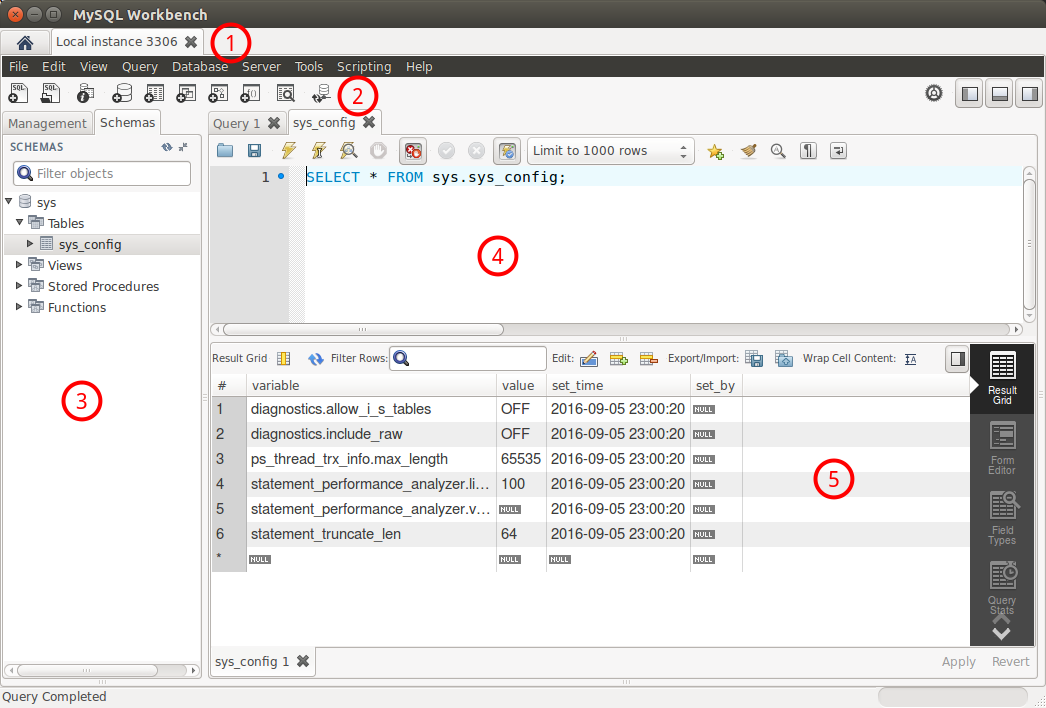


图2-13 MySQL Workbench的主界面

## 2.3 PHP运行库的安装和扩展

在讲如何安装PHP运行库之前先给各位读者讲讲为什么本书会选择以PHP为主要的编程语言，其实要实现动态网站，能用的解决方案还是非常多的，比如Python、Node.js、Ruby、Java、C#等都是非常好的选择，有的甚至在某些性能方便还超过了PHP，但是自从PHP在1994年被发明以来，经过了BBC、Facebook、百度、淘宝等大型网站的应用都充分证明的PHP的可靠性和稳定性，此外虽说PHP已经有20多年的历史，但是老而弥坚，2016年1月最新发布7.0版本性能是原先的一倍多，可见PHP团队的坚持和坚守，此外在20多年的历史中全世界又有数百万开发人员在利用PHP进行系统的开发，也因此产生了无数的第三方的标准库，供后来的开发人员使用，俗话说“大树底下好乘凉”，这句话用在PHP上面最为贴切，因为目前PHP所能解决的问题已经可以覆盖了大部分Web应用的需求，此外PHP的语法相对比较简单，且完全支持面向对象，经过这么多年其学习资料也相对丰富，使用的场景也非常广泛，因此除了可以应用于实际的项目开发中外，也是非常好的学习对象。

此外PHP其自身的特别也促使了PHP能否在20多年的发展中被世界所接受。首先，PHP对于数据库的支持非常好，能够被应用于几乎全部的数据库，所以可以很方便进行系统的部署。其次，PHP自身也是开源的软件,可以免费的获取并应用于商业应用，而不需要额外在付费,减低了开发的成本。最后，PHP是运行在Web Server上的脚本语言，只要服务器上安装的PHP脚本解析器就可以在任何平台上进行使用，如Linux、Windows等都可以运行，当然从目前的应用来说还是Linux上更多写。所以基于上面这些原因本书选择以PHP为主要开发语言，PHP的图标如图2-14所示。

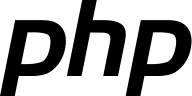


图2-14 PHP的图标

### 2.3.1 Ubuntu下的PHP运行库的安装

PHP在作者执笔时，最新的版本是5.6，此时有读者会到官网上发现有7.X的版本，此版本和5.X是完全不同的版本，相互并不兼容，但是目前从使用情况来说还是5.X使用的更为广泛，因此采用5.6的版本，而如果使用系统默认的软件源安装的话，PHP的版本只有5.5，但是本书力求最新，因此使用和MySQL类似的策略，通过添加第三方软件源的方法来安装最新的PHP，要安装最新的PHP软件源只需要在终端中执行下面两条指令就可以完成软件源的添加和更新。

1 sudo add-apt-repository ppa:ondrej/php

2 sudo apt-get update

当PHP的软件更新完成后，需要在终端里输入一条指令，并根据提示操作就可以完成PHP运行环境的安装，同时还会为Apache Server添加PHP模块，因为默认情况Apache Server是不包含PHP模块，也就无法把.php文件交给PHP解析器进行解析，当安装完服务器再在终端中执行php -v，如果能返回PHP版本，如果版本号为5.6.X就说明安装成功，如图2-15所示，具体指令如下。

1 sudo apt-get install php5.6

2 php -v

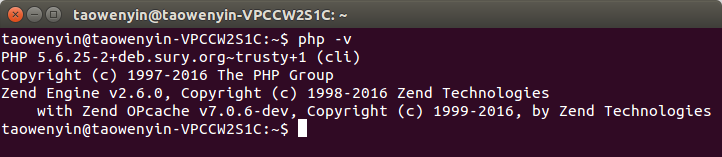


图2-14 PHP安装验证

### 2.3.2 常用的PHP运行库扩展程序

PHP的运行库扩展的目的是为了让PHP具有更加丰富的功能，在默认情况下安装的PHP只包含PHP的内核，即让服务器能够读取和解析PHP文件，但是此时的PHP连数据库都无法链接和读取，因为此时的PHP还不包括数据库的功能，因此需要为其添加扩展，而PHP功能的扩展采用插件式的方式，通过基于PHP内核的开发来完成，所以要让PHP能较为完整的服务于应用开发，那么就需要通过安装PHP扩展来完成。常见的扩展包括以下五种：

1、MySQL数据库扩展的安装。该扩展的作用顾名思义就是帮助PHP能够对MySQL数据库进行增、删、改、查的操作，具体指令如下：

1 sudo apt-get install php5.6-mysql

2、字符串多编码扩展的安装。默认情况下PHP内部所支持的文本编码方式只有ISO-8859-\*、EUC-JP、UTF-8，而采用其他编码方式的文本都没办法在PHP程序上正确显示，因此通过安装此编码让PHP更多的文本编码方式，如简体中文、繁体中文等，具体指令如下：

1 sudo apt-get install php5.6-mbstring

3、加密和解密扩展的安装。在编写代码程序时，除了要保证代码的高性能之外，还有一点是非常重要的，那就是程序的安全性。除了PHP自带的几种加密函数外，还可以通过安装PHP加密扩展库Mcrypt来获取额外20多种的加密和解密的算法，具体指令如下：

1 sudo apt-get install php5.6-mcrypt

4、图形扩展的安装。在使用论坛时很多时候需要输入验证码，单这个验证码是随机的，并不是实现安排好的图片，并且每张验证码的图片均不相同，即使内容相同，但背景或字母摆放的位置也不尽相同，此时就需要使用图形库来进行随机的绘制和显示，具体指令如下：

1 sudo apt-get install php5.6-gd

5、HTTP通信扩展的安装。在用PHP开发系统时，特别是微信系统，很多时候都需要使用HTTP和微信接口进行通信，但是默认情况下PHP是不支持HTTP通信，因此需要安装CURL扩展来帮助PHP实现HTTP通信，具体指令如下：

1 sudo apt-get install php5.6-curl

## 2.4 小结

通过本章的学习，了解了动态网站通信的基本过程，了解了常用LAMP系统的几个组成部分，同时还掌握了在Ubuntu系统下Apache服务器的安装和简单配置，及使用方法，此外掌握了如何在Ubuntu系统下安装MySQL数据库服务器，以及图形化数据库管理器MySQL Workbench的基本使用，最后还掌握了Ubuntu系统下PHP运行库的安装，以及常用扩展组件的作用及安装方法。在学完本章内容后，读者可以自行安装完整的LAMP系统，以供后续内容的使用。

# 第三章 几种常用的Web开发工具

古人有云：“工欲善其事，必先利其器”。只有装备了高等级的武器，那么上阵杀敌才能又快又准。学习PHP编程也是这样，虽然使用系统自带的记事本也可以进行编程，但是效率非常低，而且容易出错。所以本章将介绍几款常用PHP程序编辑和调试软件，熟练掌握这些软件的使用可以大幅提高编写程序的效率。

本章内容

* XDebug 开源的PHP调试工具的使用
* Vistual Code 开源的跨平台程序编辑器的使用
* Eclipse for PHP Developers 全能的程序编辑器的使用
* Chrome DevTools 实用的HTML、CSS、JavaScript调试工具的使用

## 3.1 XDebug开源的PHP调试工具

在生活中如果做错了事可能经常是没办法挽回的，就像泼出去的水一样，但是在程序的世界就不一样了，我们可以通过调试工具来反复的验证我们的程序，直到它没有问题为止，而PHP的调试工具就是本节要讲述的XDebug。XDebug是一个开放源代码的PHP程序调试工具，可以用来跟踪，调试和分析PHP程序。要安装XDebug如果采用从源码编译的方式相对较为复杂，而且对于PHP开发人员来说也没有必要了解XDebug的编译方式和源码内容，因此本节采用通过命令行的方式来安装XDebug，XDebug的图标如图3-1所示。



图3-1 XDebug的图标

### 3.1.1 Linux下XDebug的安装

XDebug的安装不同于其他应用软件的安装可以通过APT进行，因为XDebug并不是一个应用程序，而只是PHP的扩展程序，因此就需要使用安装PHP扩库社区库的方式来进行安装。PHP扩库社区库的英文全程叫The PHP Extension Community Library（简称PECL），通过PECL可以很方便的安装XDebug。要安装PECL其实非常的简单，由于系统的软件源中已经包含了PECL，因此只需要执行APT执行就可以完成，当安装完成后在执行PECL执行安装XDebug就可以完成整个XDebug的安装，具体指令如下。

1 sudo apt-get install php-pear

2 sudo pecl install xdebug

但是在通过PECL安装XDebug时可能会出现错误，如图3-2所示。该错误的意思是因为缺少phpize这个程序，而phpize默认是不会在安装PHP的时候被安装的，因此需要单独安装该程序，phpize被包含在PHP开发版里面，所以需要通过下面指令安装PHP开发版即可同步安装phpize，然后再重新通过pecl重新安装XDebug就可以正常完成整个XDebug的安装，具体指令如下。

1 sudo apt-get install php5.6-dev

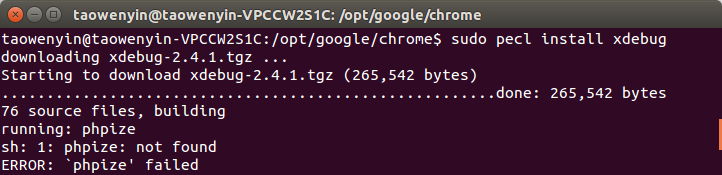


图3-2 XDebug的安装错误

当XDebug安装完成后，PECL会自动把XDebug的扩展文件xdebug.so放在/usr/lib/php/20131226目录下，但是此时PHP还无法使用XDebug，需要把扩展文件的路径配置到php.ini文件中，这里需要注意的是php.ini所在的目录是etc下，默认情况下普通用户是无法进行修改的，因此需要在用编辑工具修改php.ini文件的同时使用管理员权限打开该文件，具体执行如下：

1 cd /etc/php/5.6/apache2

2 sudo gedit php.ini

打开php.ini配置文件后，在文件的最下面添加如下XDebug的配置信息。

1 [Xdebug]

2 zend\_extension="/usr/lib/php/20131226/xdebug.so"

此外还要在该配置信息中再添加七个常用的配置信息，通过这些配置信息的添加可以增强XDebug调试PHP的性能，具体如下：

1、允许XDebug跟踪函数调用，具体配置信息如下：

1 xdebug.auto\_trace=1

2、允许XDebug跟踪函数参数，具体配置信息如下：

1 xdebug.collect\_params=1

3、允许XDebug跟踪函数返回值，具体配置信息如下：

1 xdebug.collect\_return=1

4、允许XDebug跟踪函数调用的信息输出文件目录，具体配置信息如下：

1 xdebug.trace\_output\_dir="自己指定输出目录"

5、打开XDebug的性能分析器，具体配置信息如下：

1 xdebug.profiler\_enable=1

6、XDebug的性能分析器的分析文件的存放位置

1 xdebug.profiler\_output\_dir="自己指定输出目录"

## 3.2 Vistual Code开源的跨平台程序编辑器

2016年3月中旬Vistual Code（以下简称VSCode）发布了正式版本1.0，笔者用过很多的代码编辑软件，例如VIM、Emacs、Sublime Text、Atom等，但使用一圈下来从实用性、方便性、性能等方面来说笔者还是更加喜欢VSCode。VSCode是微软发布第一款能够运行在Windows、Linux、MAC上面的跨平台的程序编辑器，在过去微软的另一款集成开发环境（以下简称IDE）Vistual Studio只能运行在Windows，而无法在其他操作系统中运行，而现在微软把代码编辑功能从Vistual Studio中剥离，进而开发了VSCode，VSCode支持几乎所有的语言，并且可以通过的使用可以很方便的实现代码提示，VSCode的图标如图3-1所示。

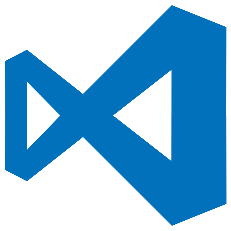


图3-1 VSCode的图标

### 3.2.1 Linux下Vistual Code的安装

进入VSCode的官网（https://code.visualstudio.com/），其会自动的识别当前系统的类型，例如笔者所使用的是Ubuntu，那么网站就会提供2中安装包，一个是基于Fedora的RPM包，还有一种是基于Ubuntu的DEB包，因此这里只需要把DEB包下载下来，并且执行下面的命令或者直接双击就就可以进行安装，如果使用命令行安装那么需要把package.deb换成下载后的文件名，如图3-2所示。当安装完成后就可以在系统的程序目录下找到Vistual Code。

1 sudo dpkg -i package.deb



图3-2 执行命令行安装VSCode

### 3.2.2 Vistual Code中PHP相关插件的安装

VSCode之所以能在极短的时间内在GitHub上获得17000多Star，同时也得到了全世界程序员的认可，其中一个很重要的原因就是VSCode可以通过插件扩展的方式来支持任何的语言，并且每天有无数的程序员在为其添加新的功能，大到微软自己有独立的团队编写插件，小到有人专门为VSCode编写文件图标的插件，甚至国内的游戏制作团队Cocos也在为其编写开发插件，以适应Coscos2D-JS的开发。

目前原生的不安装任何插件的VSCode已经为开发人员提供了如代码高亮、括号匹配、代码智能提示，以及代码片段等功能，已经可以满足基本的开发需求，此外还可以通过插件扩展更多的功能。在这里本书推荐2个插件，如图3-3所示。一个是PHP Debug，可以让VSCode实现PHP代码的调试，另外一个是Crane-PHP Intellisense，可以让VSCode实现整个项目的代码智能感知，从提供更强大的代码提示。

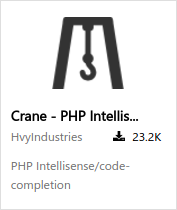
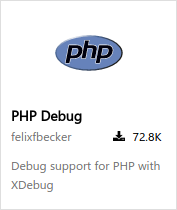


图3-3 两个PHP开发常用的插件

**1、PHP Debug插件的安装与使用**

在Vistual Code中安装PHP Debug插件是非常简单的一件事，只需要通过以下八个步骤就可以完成。

第一步：保证你的电脑处于联网状态，这个非常重要，因为插件的安装其实是从网络上下载，再有VSCode自动安装完成。

第二步：在VSCode的配置文件中添加PHP的可执行路径，因为只有这样PHP Debug才能把PHP和XDebug进行链接，代码如下：

1 // 指向可执行的 php。

2 "php.validate.executablePath": "/usr/bin/php"

第三步：安装XDebug，这个在本章的最后一节会进行详细的介绍，是真正用于PHP调试的工具，而我们的PHP Debug插件只是在VSCode和XDebug之间以建立一个沟通的桥梁。

第四步：在PHP的配置文件php.ini中添加XDebug的配置项，代码如下。其中zend\_extension后面的地址需要读者根据自己的安装目录进行修改。

1 zend\_extension=path/to/xdebug

第五步：在PHP的配置文件php.ini中打开PHP远程调试，配置代码如下所示。

1 [XDebug]

2 xdebug.remote\_enable = 1

3 xdebug.remote\_autostart = 1

第六步：重启你的Web服务器，如果你和笔者一样使用的是Linux操作系统，可以在终端中使用下面的代码进行服务器的重启。

1 service httpd restart

第七步：通过phpinfo()打印来验证你的配置是否正确，如果在打印结果的最下方出现如图3-4所示，那么就表示配置成功。

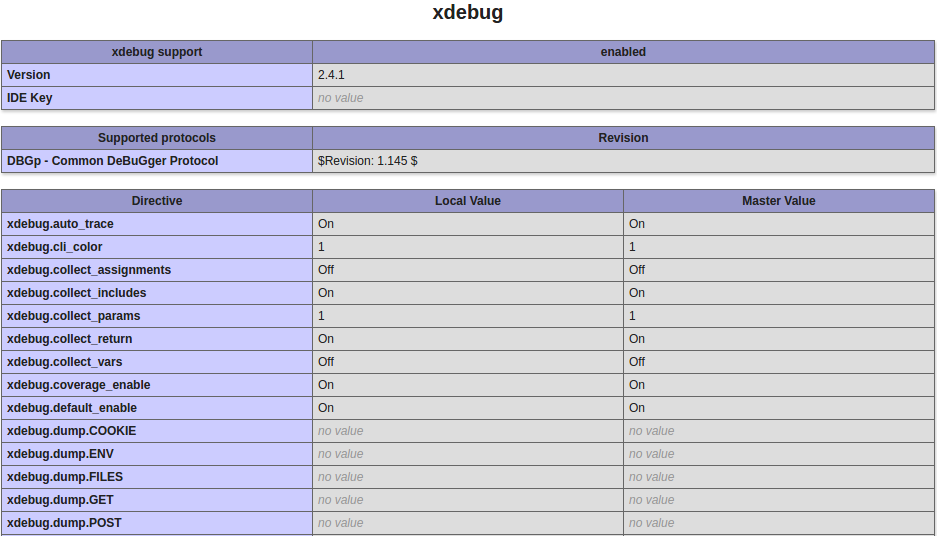


图3-4 XDebug的配置信息

第八步：这一步非常重要，用于安装VSCode插件，不过也非常的简单，只要按下键盘上的F1建，并在顶部的输入框中输入如下代码，等待几分钟，就可以完成了整个PHP Debug插件的安装，接下来就可以通过左侧导航栏中的“调试”界面进行代码的调试。

1 ext install php-debug

**2、Crane-PHP Intellisense插件的安装与使用**

和安装安装PHP Debug插件一样，Crane-PHP Intellisense插件的安装也是非常简单的一件事，只需要通过一下七个步骤就可以完成。

第一步：在键盘上按下F1建，并在顶部的输入框中输入如下代码，等待几分钟，就可以完成了整个Crane-PHP Intellisense插件的安装。

1 ext install crane

第二步：使用VSCode打开一个PHP的项目。

第三步：检查右下角的状态栏是否有Crane状态，此时可以看到Crane检索的所有PHP文件，此时当调用类中方法时就会自动弹出方法名称，如图3-5所示。

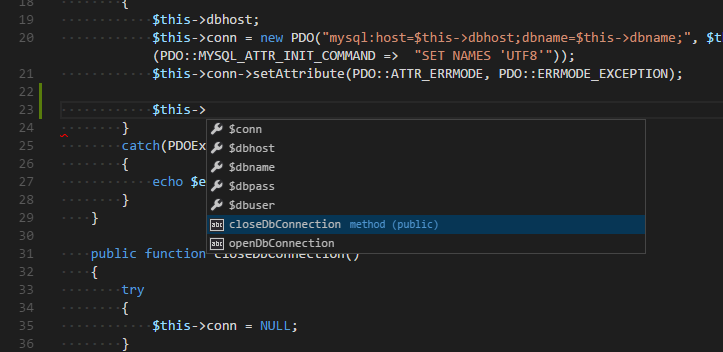


图3-5 Crane的使用

## 3.3 Eclipse for PHP Developers全能的程序编辑器

Eclipse和VSCode类似，也是一个基于插件的模式，且开放源代码程序编辑器，但由于其包含了软件开发所有需要几乎所有的功能，因此通常它为集成开发环境（简称：IDE）。Eclipse最初是在1999年由IBM发起，并和多家公司共同完成的用于开发另外一种编程语言——Java的工具，在当时市面上没有比较好的Java开发工具，相对来说只有Borland公司的JBuilder比较好用，但由于价格昂贵，不是一般程序员能够使用的，因此IBM在2001年的时候把Eclipse彻底开源，同时把该项目捐给了Eclipse基金会，用现在流行的语言说，Eclipse就是当时的爆款，而且是全球的爆款，其免费、开源、IBM做背书、由Java代码编写、平台的一致性等特点彻底俘获了所有程序员的心，目前Eclipse最新版为Eclipse Neon，其图标如图3-6所示。



图3-6 Eclipse的图标

随着Eclips的发展，已由原先开发Java发展到现在几乎可以开发任何的语言，如果说Vistual Studio是Windows平台下最好的集成开发环境，那么Eclipse就是在跨平台领域下最好的集成开发环境。不过和Vistual Studio一样，由于其功能非常的丰富，因此体积也越发的臃肿，其中许多的功能对于一个初学者来说还暂时用不到。在IBM开发Eclips时把Eclipse定义为一个程序框架，通过基于该框架的开发可以把Eclipse扩展到任意的编程语言，有的公司甚至基于Eclipse开发自己的IDE，如PHP中著名的公司Zend，该公司就是基于Eclipse开发了自己的集成开发环境Zend Studio，而我们这里所说的Eclipse for PHP Developers也是基于Eclipse开发的PHP插件，使得Eclipse可以支持PHP的开发，与Zend Studio不同的是Eclipse for PHP Developers是免费的，而且功能也能满足一般开发的需求。

### 3.3.1 Linux下Oracle Java的安装

由于Eclipse的使用需要依赖于Java，但Linux系统默认没有安装Java环境，因此在安装Eclipse for PHP Developers之前首先要把Java环境安装好。因为笔者使用的是目前非常流行的Linux发行版——Ubuntu系统，所以可以通过以下五个步骤就可以完成。

第一步：保证你的电脑处于联网状态，这个非常重要，因为软件的安装其实是从网络上下载，再由系统的软件管理器进行自动安装完成。

第二步：添加Oracle Java软件源。因为Ubuntu默认不包含Oracle Java的软件源，所以需要执行下面的命令来添加对应的软件源，结果如图3-7所示。

1 sudo add-apt-repository ppa:webupd8team/java

2 sudo apt-get update

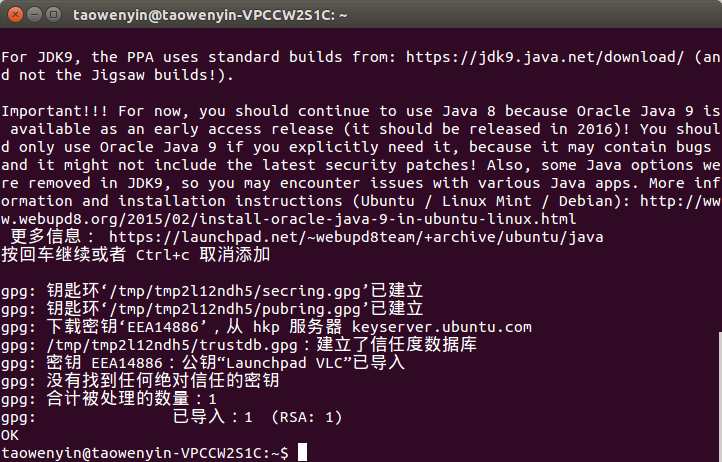


图3-7 Oracle Java软件源的添加

第三步：安装Oracle Java。添加完软件源后，安装Java是非常简单的一件事，只需要执行下面的命令就可以完成Oracle Java的下载与安装。

1 sudo apt-get install oracle-java8-installer

第四步：执行下面的命令把Java8设为默认Java环境

1 sudo apt-get install oracle-java8-set-default

第五步：在终端中输入java或javac，并且打印如图3-8所示的内容，则表示Java环境已经安装完成。

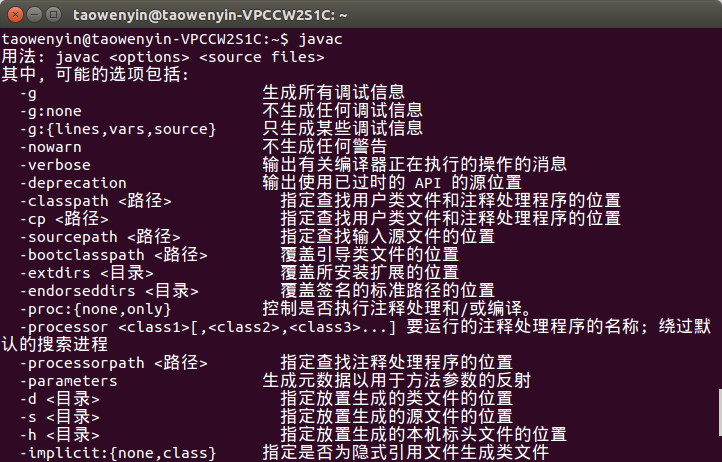
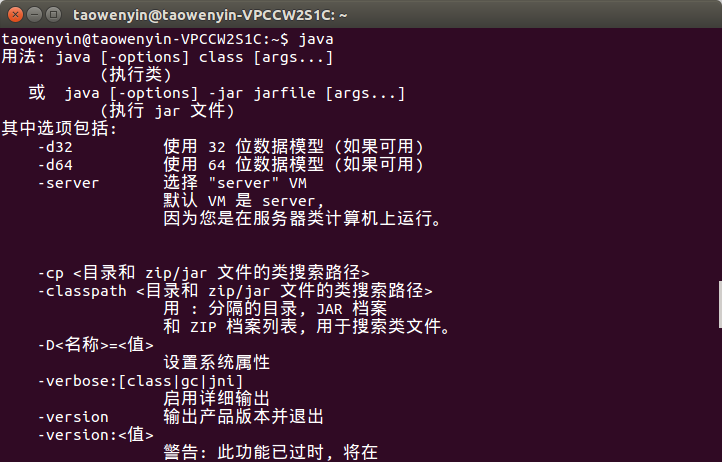


图3-8 验证Java安装完成

### 3.3.2 Linux下Eclipse for PHP Developers的安装

当进入Eclipse的官网（https://www.eclipse.org/downloads/），其会自动的识别当前系统的类型，例如笔者所使用的是Ubuntu，那么网站就会提供一个压缩包，在Neno版之前Eclipse官网都会直接提供针对不同开发语言的Eclipse程序件，但到了最新的Neno版，官网除了原先提供的内容外，还提供提供了一个集中的管理程序，本文就是通过这个管理程序来进行Eclipse for PHP Developers的安装，只需要以下几步就可以完成。

第一步：解压缩安装包eclipse-inst-linux64.tar.gz，并且双击执行eclipse-inst文件，弹出Eclipse安装包类型选择界面，如图3-9所示。

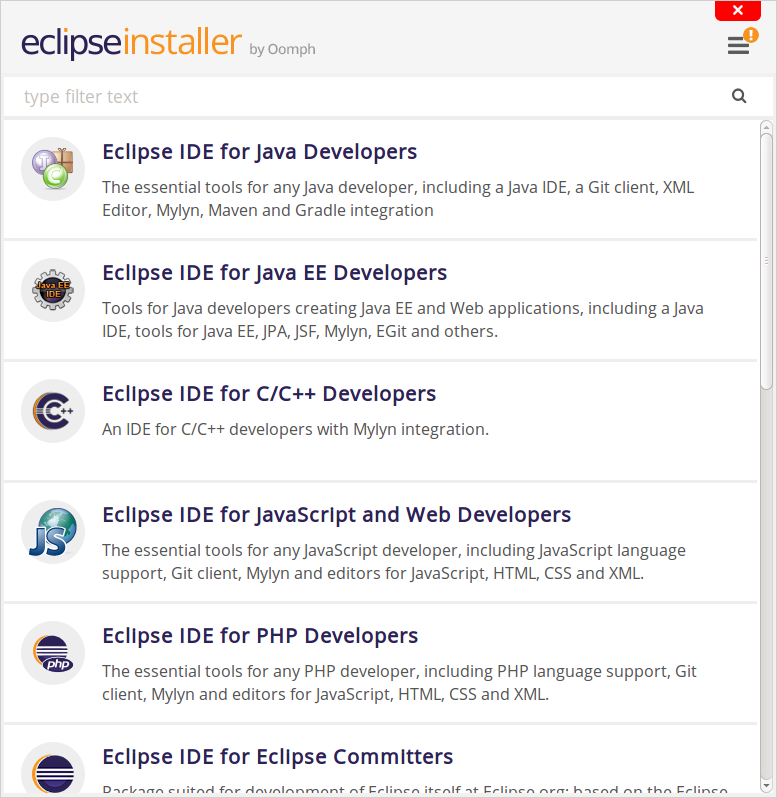


图3-9 Eclipse安装包类型选择界面

第二步：选择Eclipse for PHP Developers，设置安装路径，点击INSTALL进行安装，如图3-10所示。

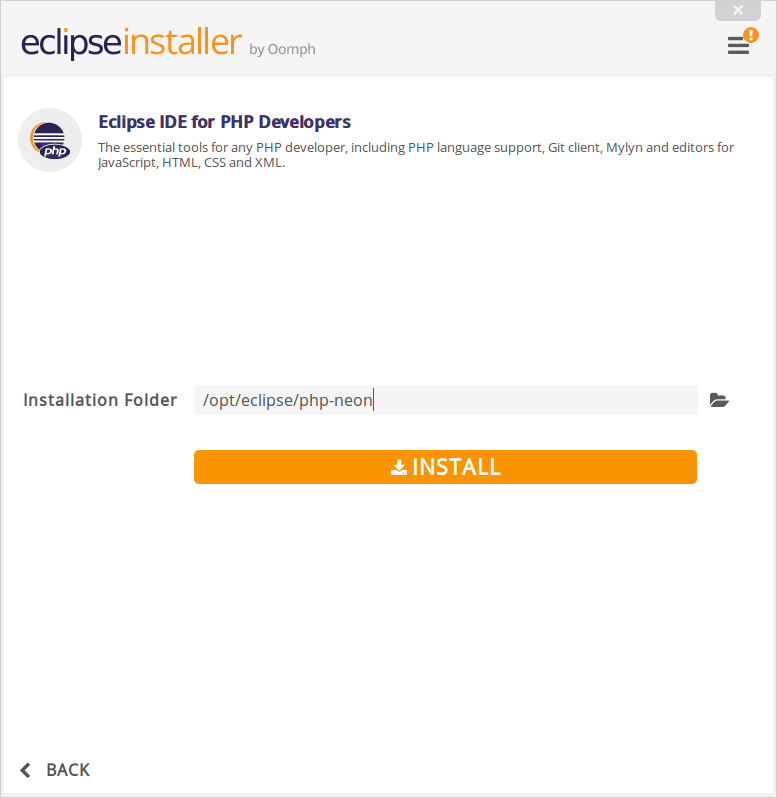


图3-10 安装路径选择界面

第三步：等待安装，当安装完成后，点击LAUNCH运行Eclipse for PHP Developers，如图3-11所示。

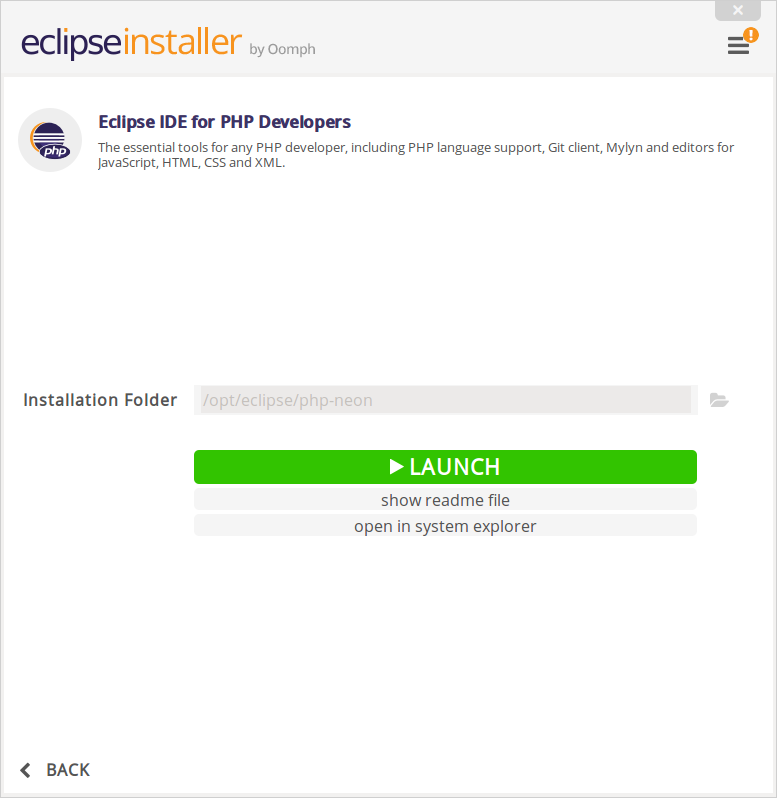


图3-11 安装完成择界面

第四步：设置Eclipse for PHP Developers的工作目录，这里需要注意的是要把Eclipse的工作目录设置在WebServer的www目录下，方便执行执行PHP程序，同时如果该工作目录以后不会再改动，那么可以把“Use this as the default and do not ask again”打勾，以后就不会弹出该界面，如图3-12所示。

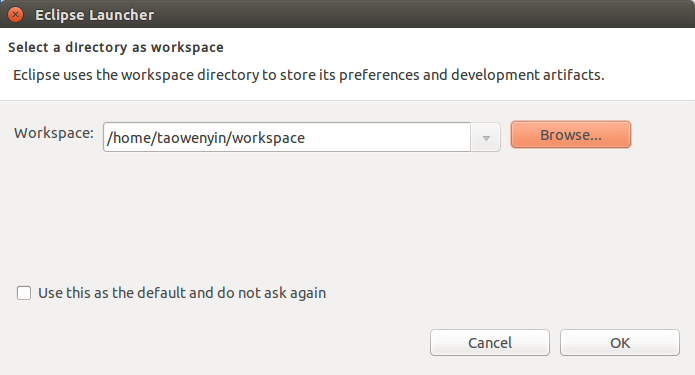


图3-12 工作目录配置

第五步：当设置完工作目录后，点击OK键就可以进入系统，此时可以通过点击菜单栏中的“File”->“New”->“PHP Project”创建PHP项目，如图3-13所示。

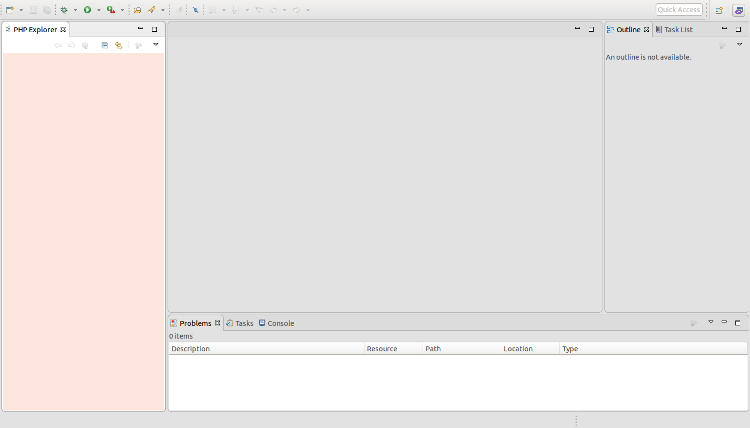


图3-13 Eclipse运行界面

## 3.4 Chrome DevTools实用的前端调试工具

以前听前辈们讲过这么一句话，世界上只有4种浏览器，分别是Trident、Webkit、Gecko、Presto，其实这些指的都不是浏览器，而是浏览器的内核，随着最新的Win10发布，在浏览器内核家族里又加入了Edge，但是除了Webkit之外其他内核都不是开源的，因此都只有个别浏览器在使用，而Webkit的代表浏览器就是Google的Chrome，同时该内核也是由Google开发的，随着Webkit内核的日渐成熟，国内的浏览器市场也是百花齐放，如360、搜狗、猎豹、QQ等知名厂商都采用了Webkit作为其浏览器的内核，因此可以说Webkit占据了国内浏览器的半壁江山，因此使用Webkit调试网页是可以说能代表最广大人民的利益，也因为这个原因本书的Web调试工具采用Webkit内置的调试工具Chrome DevTools。要使用Webkit内核的浏览器在Windows上有很多原则，如果360安全浏览器、猎豹等，但是在Linux上就只有Chrome这一个选择，因此在讲解Chrome DevTools之前首先要执行下面的指令来进行把Chrome下载，然后双击安装即可。

1 wget https://dl.google.com/linux/direct/google-chrome-stable\_current\_amd64.deb

安装好Chrome后打开Chrome DevTools有两种方式，一种是打开顶部菜单的工具，然后选择开发者工具，或者使用快捷键键Ctrl+Shift+I，打开后可以看到如图3-14的界面，一共有九大功能模块，其中最为常用的有六个，分别功能如下：

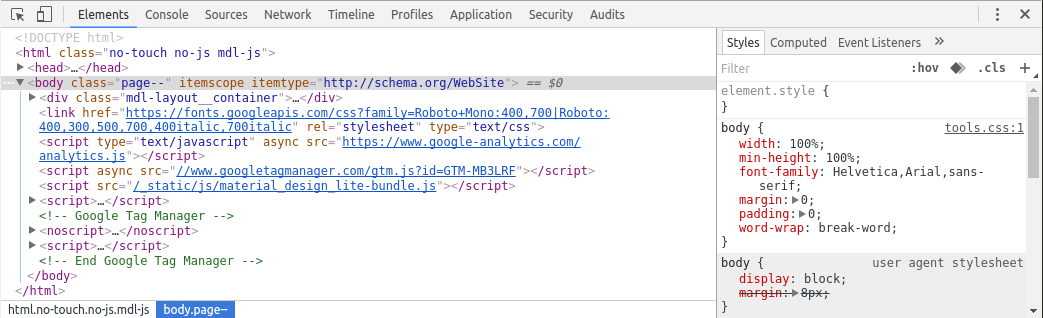


图3-14 Chrome DevTools运行界面

1、Elements：可以查看当前网页对应的HTML代码，通过在在该页面中编辑和修改页面元素的属性达到调试页面的效果，同时页面也会跟随代码的改变而改变，做到所见即所得的效果。

2、Console: 控制台工具，可以显示各种警告与错误信息，同时开发人员也可以通过该窗口进行信息打印，从而获取程序运行中的中间值。

3、Sources: 该页面主要用来对页面JavaScript的代码进行调试，该页面中可以实现对代码的暂停、单步调试、进入函数、跳出函数，以及查看临时和全局变量等。

4、Network：通过该页面可以查看当前页面从服务器端请求了哪些资源，这些资源的大小，以及所花费的时间，同时还能看到哪些资源是加载失败的，利用这些信息可以知道页面优化的方向。此外，该页面还可以查看HTTP的请求头和返回内容等。

5、Timeline：计算页面在从请求到展示整个过程中，每个行为所消耗的时间，例如计算CSS样式所需要的时间和执行JavaScript所花费的时间等。

6、Profiles：该页面是从整体上分析页面所消耗的时间，以及所占用的内存情况。

## 3.5 小结

通过本章的学习，了解Eclipse for PHP Developers开发工具的基本使用方法，同时也了解在进行Web应用开发是可以通过Chrome DevTools来进行页面的调试，此外掌握了如何在Ubuntu下进行XDebug的安装，以及基本配置方法，最后还掌握了如何在Ubuntu下安装VSCode，以及VSCode中插件的安装方法和两款用于调试和编辑PHP代码的插件，从而提高整个开发的效率。在学习完本章内容之后，读者可以自行安装XDebug、VSCode以及对应的插件，以供后续内容的使用。

# 第四章 PHP的面向对象分析与设计

古人云：”仁者见仁，智者见智“，一件事或者一个物体从不同的角度进行观察就会得出不同的结果，但这些结果又必有其相关性和区别性，就像看张艺谋的《长城》，有人觉得好看，也有人觉得不好看。在程序设计中针对一个问题的解决思路也是有多种方法，本章讲述的是目前在高级语言中最为流行，也是应用最为广泛的程序设计方法，即面向对象的程序分析与设计方法，这个方法具有很强的普适性，它可以用不同的语言以不同的语法进行表现，却有着相同的内涵，在本章我们则依托PHP编程来进行讲解。

本章内容

* 面向对象的基本概念
* 面向对象的构成与内存分配
* 面向对象的$this和构造函数的分析
* 面向对象中封装性的编程思想
* 面向对象中继承性的编程思想
* 面向对象中多态性的编程思想

## 4.1 面向对象的基本概念

到2016年距离PHP第一个版本的发布已经过去了22年，这22年中PHP由一个只是用来维护个人网页的工具发展到被Google、Facebook、Microsoft、阿里、腾讯等知名公司广泛使用的开发语言，并且在国际权威的开发语言排行榜TIOBE中始终排名在前十名，如图4-1所示的是2016年12月份的排名，目前PHP是排名在第六，可见其被应用的广泛程度。PHP目前有两个主要的分支，分别为PHP 5.x和PHP 7.x，而PHP 5.x是应用最为广泛的版本，许多第三方的框架都是基于该版本进行开发，其外在PHP 5.x中全面引入了面向对象的机制，并且保留了向下兼容的面向过程的函数，因此在本书中关于PHP的内容都是以PHP 5.x为基础上进行讲解。

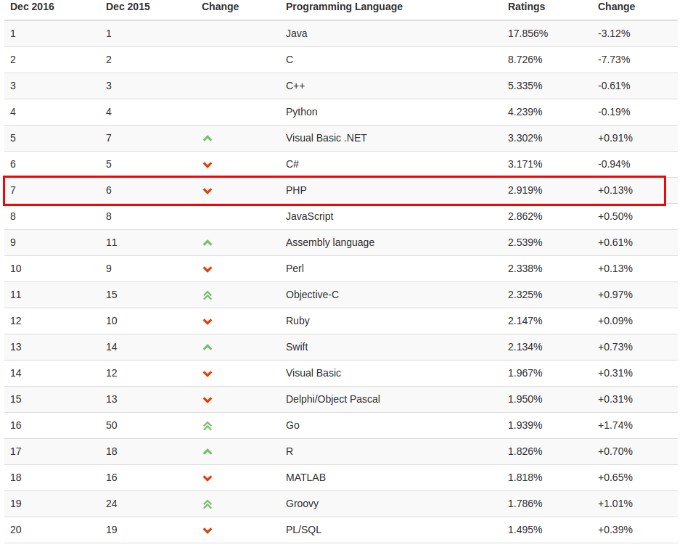


图4-1 PHP语言排行榜

完整的面向对象通常包含三个层次的概念，即面向对象的分析（Object Oriented Analysis，OOA）、面向对象的设计（Object Oriented Design，OOD），以及面向对象的编程（Object Oriented Programming，OOP），其中面向对象的分析主要目的是找出要解决的问题中各个相对独立的对象，而面向对象的设计主要目的是找出这些独立对象之间关系，最终通过面向对象的编程把这些独立的对象和关系利用计算机语言进行实现，从而形成完整的系统。在这里读者一定要明白的是，面向对象并不是一门编程语言，而是一种编程思想，在面向对象的系统中每一个对象都能够对数据进行接收、处理和发送，从而使得整个软件系统的代码更加的简洁和更容易的维护，从而满足了软件设计的基本要求，即重用性、灵活性，以及扩展性。面向对象能够成为软件领域内最为热门的话题，主要原因是它符合我们看待事物的一般规律，例如我们从桌子上拿起一瓶饮料，我们做为一个独立的对象只需要伸手去拿即可，而不需要关心我们在拿饮料的过程中肌肉是如何传动的，能量是如何被消耗的等等一系列的问题，所以通过面向对象可以很好的把这些不需要关心的行为进行封装，让其内部进行，而不对外暴露。

因此在学习PHP的面向对象程序设计时除了要学习相关的语法外，更加重要的是要学习和思考面向对象的编程思想。起初在学习其相关语法时很多读者会觉得非常的简单，但是要深入的去思考为什么要这样设计，就像在本章开头所引用的那句古语”仁者见仁，智者见智“，运用面向对象的思想编程没有统一标准的答案，只有更好，没有最好，所以读者需要多思考、多练习。

### 4.1.1 理解类与对象

为了理解类和对象之间的关系，可以举一个通俗的例子，就好比在路上看到一只小狗，我们会叫他小狗，那么小狗就是一个类，因为世界上没有什么动物叫小狗，我们只是把有四个脚，并且会汪汪叫的动物称作小狗，但在我们眼前的这只小狗就是一个实实在在的动物，它是小狗这个概念具体化的产物，所以这个具体的小狗就是对象。因此可以知道所谓的类其实是把一系列具有相同属性、相同行为的对象进行抽象后形成的类。

因此在程序设计中类通常都是一个个独立的单元，这些单元有各自的属性，以及相应的行为，同时类也是符合这些特征的无数对象的高度抽象后的结果。类不存在于客观世界，而是存在与人们对于一个问题的思考和分类，也可以说类是人们所下的一个定义，例如我们说有四个脚，并且会汪汪叫的动物叫做小狗，我也可以说有四个脚，并且会汪汪叫的动物叫做猫，这些完全都有设计者来决定。

而对象与类恰恰相反，对象是在客观世界中实实在在存在的东西，客观的世界就是由无数个各种各样的对象组合起来的，这些对象又因为某种关系把他们联系在一起，比如家里领养了只小狗，那么本身没有任何关系的两个对象家庭成员和小狗，因为领养了这只小狗就产生了关系，领养是家庭成员的一个行为，而小狗则成了这个家庭中每个人的一个属性，所以可以认为对象是构成某个系统的基本单元，并且在对象内有一组相关的属性，以及可以操作的行为。

因此通过上面的介绍相信读者对于类和对象的关系有了一定的了解，虽然已无从考证为什么“class”会被翻译成“类”，但是笔者认为这个翻译是特别贴切的，因为“类”就相当于现实生活中把客观世界的各种东西进行分类，每个类型有自己的属性和行为，然后一个类型可以产生很多各实体，每个实体都是一个对象。

### 4.1.2 理解面向对象的程序设计

在PHP 5.x之前PHP并没有引入面向对象的概念，开发人员通过一个个的函数模块来构成整个交互系统，但是到了PHP 5.x这一切就发生了改变，因为在PHP 5.x中引入了面向对象编程的所有功能，同时还保持了原来的函数模块，保证了程序的兼容性，从此开发人员可以采用更加符合对客观世界理解的方式来进行开发和解决问题。

在面向对象的程序设计中，其相关语法并不是十分困难，甚至有C语言基础的读者会觉得它非常的简单，但是面向对象的程序设计的难点在于如何使用面向对象的思想来解决问题，例如要设计一个学生管理系统，那么首先要思考的不是这个系统有多少个学生、多少个班主任、多少个辅导员，因为这些内容都是不确定的，可能这个系统用在A学校就只有50个学生，而用在B学校就有100个学生，因此正确的思考应该是把整个系统进行分类，分成学生、班主任、辅导员，并且找出学生、班主任、辅导员这几个类别中每个类别应该有的行为，然后确定学生、班主任、辅导员之间的关系，例如每个学生都属于一个班级，每个班级都有一个班主任，若干个班级组成一个年级，每个年级都有一个辅导员，这样就可以把学生、班主任、辅导员之间的关系理清楚，然后在思考所对应的专业、学院和学校，以此类推就可以得到整个系统的体系结构。

从上面的分析不难看出使用面向对象编程思想编程时通常的思考方法是自顶向下的思考，也就是说先从整个系统进行整体划分，然后像剥洋葱皮一样一层层往下进行设计，直到最后一个部分，然全部划分完成后就是要确定这个模块之间的关系，以及模块与模块之间的数据是如何传递的、方向是什么等内容，从而构成了整个面向对象的系统。

## 4.2 面向对象的构成与内存分配

在程序开发中整形、浮点型、布尔型、双精度型都被成为基本数据类型，而数组和对象都被称为复合数据类型。虽然PHP是弱类型的语言，可以要求数组中的元素类型不相同，而不会像强类型的语言那样要求数组中每个元素的类型必须相同，但数组有个明显的不足就是无法为其添加行为，并且设定每个元素的访问权限，而这些功能则是类和对象所擅长的。在上一节本书讲解的面向对象的基本概念，本节将会讲述如何通过代码的方式创建一个类和一个对象。

### 4.2.1 类的声明

要声明一个类其实方法非常的简单，主需要在前面添加一个class关键字，后面跟着类名，最后用花括号括起来就可以，这里特别要说明的是类名和class之间一定要增加一个空格，具体代码如下：

1 class 类名 {

2 // 类的内容

3 }

在创建类是有一点需要特别的强调，就是关于类名的命名规则必须要要符合国际通行的管理，即如果该类名由一个单词构成，那么该单词必须是首字母大写，如果该类名由多个单词构成，那么每个单词的首字母都要大写，其他字母小写，同时要保证这个类名具有一定的意义，切记起一些没有意义的名称，例如AABB、ABCD等。

在类的内容中主要添加一些成员方法和成员属性，这个成员方法也就是函数，它主要表明这个类具有什么功能，而成员属性则表明这个类具有什么特质，例如要定义一个人这个类，那么姓名、年龄、性别就是人这个类的属性，而能够走路、能够跑步、能够拿东西就是人这个类的功能，这些东西都需要在类中进行定义，这里需要说明的是在类中定义的变量虽然可以不赋初值，但是给一个变量赋初值是一个很好的习惯，同时在方法名前添加限定符也是很好的习惯，可以明确该函数的权限。因此可以说所有的类都是由形形色色的属性和方法构成的，其能够实现的功能比数组多出许多，并且一个PHP程序可以声明无限多个类，只要系统需要，但在设计这些类时需要思考的关键是要把各种对象中所共性的进行抽象，从而得到了真正需要的类，具体代码如下：

1 class 类名 {

2 限定符 $属性名 = 初始值;

3 ......

4

5 限定符 function 方法名(参数列表) {

6 // 函数体

7 }

8 ......

9 }

### 4.2.2 对象的实例化与内存分配

前面讲过类是许多个对象的高度抽象后的结果，因此一个类不仅仅表示一个对象，而是表示很多个对象，所以程序中不能直接对类进行操作，而是需要通过类的实例化后得到对象，然后调用对象中的各种方法，从而实现各种功能，这些方法也就是事先在类中已经定义好的方法。那么要实例化一个对象其实非常简单，只需要使用关键字new，然后后面加一个空格，空格之后跟上一个类名，最后紧跟一对小括号即可，如果此时在实例化一个对象时需要传递一些初始化参数，那么在类名之后的小括号中就需要传递一组初始化参数列表，具体代码如下：

1 $对象名 = new 类名(); // 没有参数的情况

2 $对象名 = new 类名(参数列表); // 有参数的情况

其中“$对象名”表示通过某个类创建的对象名称，可以类比为变量名，只不过它表示的内容更加的复杂，而参数列表则是对象初始化的所需要的参数，例如创建一个Student对象时，可以传入如姓名、性别、年龄等初始化信息，从而创建出不同内容的对象。当类在创建对象时PHP会调用一个特殊的函数，即构造函数。构造函数的种类有两种，一种是默认的构造函数，当创建对象时如果它的参数列表为空，那么系统就会试图去调用默认的构造函数，因为默认的构造函数不需要传递任何的参数，通过内部的代码来实现对象的初始化，采用这种方法产生的对象均有相同的内容，而另一种就是自定义的构造函数，和默认的构造函数正好相反，创建对象时的参数列表不为空，当系统创建对象时，系统通过这些参数来初始化对象，对象通过这些参数来动态的生成其内容。这里需要特别说明的是在具体的程序开发中自定义构造函数一定是由开发人员自己来实现的，而默认的构造函数可以采用系统默认的方式，也可以由开发人员自己来实现，对于这部分内容在后续的章节中会进一步说明。

当一个类产生一个对象时，其实质是通过关键字new向操作系统发送申请空间的指令，因此每使用new一个对象就会在内容中获得一块内存空间，这些空间中存放了不同的内容，虽然这些对象的类型都是相同的，但是它们之间不存在任何的关系，因为在物理空间上就是分离的，就好比世界两端的人，虽然他们都是Person，但是从姓名、生活阅历、家庭来说它们是完全不同的，可以说是完全独立的两个人。所以，要完全理解并且熟练掌握对象的使用，除了要知道对象的基本使用方法还需要了解对象是如何在内存中进行创建和分配的，这样才能更加灵活的去使用对象。

在计算机中存储器不是指通常意义上的硬盘，而是指的内存，硬盘在计算机中只是相当于外部存储器，用来持久的保存数据，而计算机程序在运行时所有的数据都需要从硬盘中读取然后放入内存中进行计算，因此当内存较大时，就减少了计算机与硬盘进行数据交换的次数，也就提高了程序的运行速度。通常来说一个程序运行时所占用的内存空间会被划分为四个部分，分别为栈数据空间、堆数据空间、全局数据空间、程序代码空间，这四部分的作用分别如下：

1、栈数据空间：该部分数据空间相对较小，并且主要用于存放一些占用空间固定的数据，例如存放一些整型、浮点型等基本数据类型，因为数据的大小固定可以很方便的分配数据空间，所以计算机在设计该部分空间时把它设计成访问速度较快的空间，以便提高程序的效率，此外该空间还被用于存放一些临时的变量，当程序超出了临时变量的作用域时，该变量就会被销毁，所以栈空间一般被用于基本数据类型的交换，以及保存临时变量保存。

2、堆数据空间：区别于栈空间，堆空间主要用于为一些具有动态内存特性的数据分配存储空间，这些数据所占用的空间大小不固定，可能随着程序的运行进行所占用的空间增加或者减少，因此堆空间通常用于存储一些数据大小不固定，且占用较大空间的数据，例如像对象、数据这样数据都是存储在堆空间中。

3、全局数据空间：该空间相对较为简单，主要用于存储一些已经初始化完成的全局的静态变量。

4、代码空间：上面三种存储空间主要用于存储数据，而代码空间则主要存放由程序员编写的程序，这些程序不是编写的源程序文件，而是通过编译器或者解释器后产生的可执行文件，因为程序在运行时不能够被修改，所以这部分的空间具有只读的特性，而不能被写入。

从上面的解释可以知道，当PHP创建一个对象后，系统就会在内存的堆空间分配一部分容量用于存放该对象的数据，但为了提高整个系统的系统运行速度和安全性，避免对内存空间的随意修改，操作系统不允许运行的程序直接访问堆空间中数据，而是需要在栈空间中创建一个整数类型的变量，该变量就是通常所说的对象名，并且在该变量中存放着堆空间中对应对象的地址，然后通过对该变量所指向的堆空间地址中的数据进行操作，从而实现对象的操作。对象数据这样特殊的存储方式是因为栈空间的存取速度非常快，几乎具有和CPU寄存器一样的速度，但是栈空间具有存放数据必须大小固定的缺点，而堆空间却没有这样的要求，不过堆空间的速度相对较慢，因此对象数据的存储正式结合这两种存储方式的优点，既满足了速度要求，又满足了存储大小的要求，如图4-2所示的就是程序运行时的内存分配。

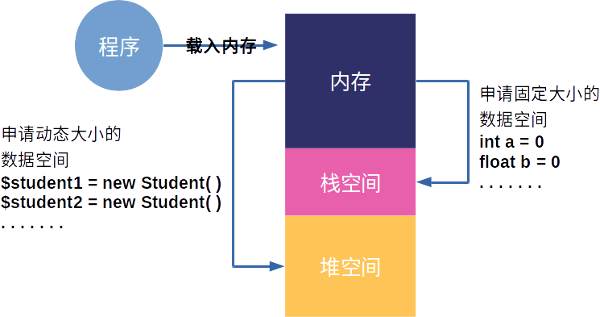


图4-2 堆栈内存的分布

在上图中把整个内存分为了三个部分，分别为内存、栈空间和堆空间。当程序运行时，操作系统会把要运行的程序全部载入到内存中，当程序遇到需要创建基本数据类型时，例如“int a = 0”和“float b = 0”时，程序就会通过操作系统向内存中的栈空间申请一块空间用来存放这些数据。而当程序遇到需要创建对象类型或者数组类型时，程序首先会通过new关键字来告诉操作系统当前程序需要在堆空间内申请一块内存空用来存放一些相对较大的数据，例如“new Student()”，此时每new一次就会在堆空间中申请一块内存空间，这些内存空间之间相互独立，当在堆中申请完空间后系统又会在栈空间申请一个内存空间，用来保存堆空间中的数据地址，例如“$student1 = new Student()”中的$student1就是用来存放这个地址的变量，这个地址的长度在32位的操作系统中是一个32位的整型，而如果在64位的操作系统中，则是一个64位的整型数据，最后操作系统把这个地址数据返回给程序，程序就可以根据这个地址对数据进行操作。

在程序中加、减、乘、除等操作实质上都是在栈空间中进行，例如“int a = 1; int b = 0; a = b”这样的操作，其本质是把在栈空间中b地址中的值赋值给a地址中的值，因为基本数据类型的数据就是存放在栈中，因此看到的现象就是数值的拷贝，但是对于类数据和数组数据这样的数据类型来说，栈空间存放的是数据在堆空间中存放的地址，因此如果直接在类对象或数组数据之间进行赋值，那么其本质是地址值的赋值，而不是数据内容的赋值，所以例如“Student student1 = new Student(); Student student2 = new Student(); student1 = student2”这样操作的结果是student2把其所指向的堆内存空间中的地址赋值给student1，使得student1和student2都指向了同一段的堆空间，此时对于这段堆内存来说就有两个变量来表示它，只要对其中一个变量进行操作，另外一个变量也可以得到操作后的结果，当然对于程序来说可以创建很多个指向相同堆空间的变量，这些变量称为别名，即虽然名字不相同，实质确实相同的东西。

## 4.3 面向对象的$this和构造函数的分析

当构建对象时传入一些参数作为对象初始化的依据，那么所得到对象的其内部会因为不同的参数而不尽相同，但是在调用对象方法时，只要对象的类型相同，其调用的函数就必定相同，此外当程序运行结束，或者对象的生命周期已经完成，这个时候就需要对对象进行销毁，本节的主要内容就是分析系统是如何实现动态对象方法的调用，以及对象在构造和销毁时的行为。

### 4.3.1 运行时对象属性与方法的动态访问

当一个类定义完成后，还可以在其中添加属性和方法，属性表示的就是类的特征，例如Student类的属性就可以包括姓名、年龄、性别等，而方法表示的就是类的行为，还以Student类来说就是读书、运动、恋爱等。当类产生对象以后就可以通过特殊运算符“->”进行访问，这些访问可以是对属性的赋值或者获取，也可以是执行某一个方法，具体代码如下：

1 $对象名 = new 类名(参数列表); // 有参数的情况

2

3 $对象名->属性名 = 值; // 属性的赋值操作

4 $属性值 = $对象名->属性名; // 属性值的获取操作

5

6 $对象名->方法名(参数列表); // 执行类的某个方法或行为

下面通过一个实例来具体说明如何通过运算符“->”来实现对属性和方法的操作，首先定义了一个Student类，并为这个类添加一个“name”属性用于表示Student类的名字，然后定义一个“setName()”方法，该方法用于设置“name”属性的值，最后再定义一个"getName()"方法，通过该方法来获取“name”属性，当完成Student类定义后，通过关键字new在内存空间中创建了Student对象，然后就使用特殊运行算符“->”来设置和读取相关属性值，具体代码如下：

1 class Student {

2 public $name = ""; // 定义一个属性

3

4 // 定义一个方法用于设置属性name

5 public function setName($name) {

6 $this->name = $name;

7 }

8

9 // 定义一个方法用于获取属性name

10 public function getName() {

11 return $this->name;

12 }

13 }

14

15 // 产生$zhangsan对象

16 $zhangsan = new Student();

17 $zhangsan->setName("张三");

18 echo $zhangsan->name."<br />";

19 echo $zhangsan->getName();

20

21 // 产生$lisi对象

22 $lisi = new Student();

23 $lisi->setName("李四");

24 echo $lisi->name."<br />";

25 echo $lisi->getName();

程序结果输出：

1 // 第一个输出的结果

2 张三

3 张三

4

5 // 第二个输出的结果

6 李四

7 李四

从上面的代码中可以了解到特殊运算“->”在实际代码编写中的使用方法。在上一节中讲到当对象在堆空间中被创建后，程序是不能够直接对其进行访问，必须要再在栈空间中创建一个变量，该变量保存了该对象在堆空间的地址，程序正是通过这个地址来完成对对象的操作，那么对于已经由类产生的对象来说要实现对属性和方法的操作只需要通过对象名的操作就可以完成，但很多时候除了外部的读取和访问外，在类的内部也需要进行先对属性进行一定的处理后再进行输出，例如女生都希望自己的一直保持年轻，所以别人在问自己年龄的时候可能会在实际的年龄上减少几岁，那么对于询问者来说所获取的年龄实际上是经过处理后的结果，而不是原本的值，因此在实际开发中类的属性经常需要被内部的其他成员函数进行访问和处理，然后再把处理后的结果进行返回。

在4.2节描述中讲到，通过new关键字产生的每个对象在堆空间中都具有相对独立的空间，但是其实现的代码却是相同的，即共用一份代码数据。从上面的例子中也不难看出，虽然都是Student类型，但是却创建了不同的对象，但是这些对象之间不存在任何的关系，那么要在对象内部正确的访问到相应属性，而不是李四的对象访问到张三的对象，或者张三的对象访问李四的对象，就需要有一个能够在运行时表示不同对象的引用，目前知道new关键字产生的对象名可以完成属性的读写，但是对象的内部是没有办法获取对象名，因此在PHP中提供了一个非常特殊的对象引用“$this指针”，这个引用代表了当前程序运行时的对象引用。在上面的例子中有两个对象，分别为$zhangsan和$lisi，那么对于对象$zhangsan来说$this就表示$zhangsan对象的引用，而对于对象$lisi来说$this就表示$lisi对象的引用，即在$zhangsan中$this指向和$zhangsan相同的内存空间，而在$lisi中$this指向和$lisi相同的内存空间，所以$this并不表示某一个特定的对象，而是表示这个对象本身的引用，如图4-3所示。

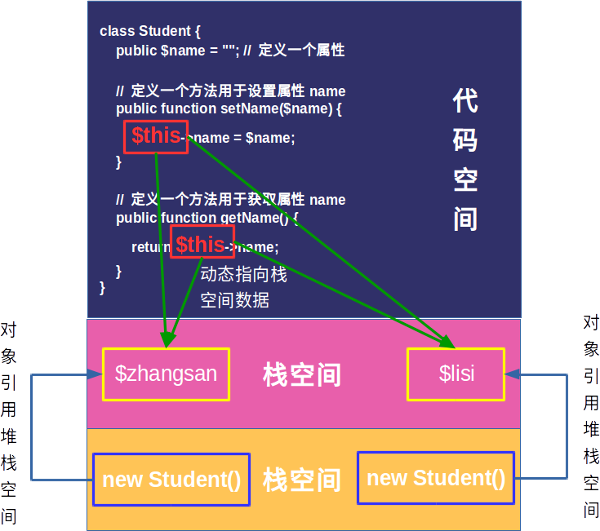


图4-3 $this指针的理解

### 4.3.2 构造函数和析构函数的运行分析

在上一节中创建了一个Student类，并在该类中添加了两个成员函数用于设置和获取Student类的名字，这一做法从语法上来说没有任何的问题，但是仔细琢磨就会发现有其不合理性。在本章的开头说过，面向对象的编程最大特点就是把现实生活中对问题的觉得方法映射到计算机中，并用计算机的语言进行表述，那么在现实生活中当学生对象创建后，他的name属性就应该存在，而不是在后续的工作中通过setName()函数进行赋予，就如一个新生进校以后他的名字是本身就存在的，而不是进校后老师给起的名字，所以name属性的值应该在创建这个对象时就赋予。同时当学生离开学校时，他在学校的学籍也会一同被注销一样，这些生活中的现象都告诉我们，对象中的有些东西应该是在对象产生时就已经有的，而有些东西则是随着对象的消失而自动消失的。因此在PHP中有两个非常特殊的函数来解决这个问题，一个叫构造函数，而另一个则叫析构函数。

这两个函数都和对象的整个生命周期息息相关，其中构造函数是是对象创建完成后第一个调用的函数，并且这个调用的过程是完全自动的。而析构函数则是在对象被系统销毁时最后一个被调用的函数，同样这个调用的过程是完全自动的。对于构造函数而言，其最主要的功能是在对象被使用前完成对象内部属性的初始化，例如上面的例子中学生对象被创建时就应该初始化内部的名字、性别等属性。而对于析构函数而言，其最主要的功能是完成对象销毁前最后的清理工作，如学生离校是把学籍从学校删除等工作。

1、构造函数的运行分析

在类中，构造函数是一个特殊的函数，当该函数没有在类中明确声明时，系统会自动为该类提供一个默认的构造函数，用于创建对象，但是这个构造函数没有任何的内容。而当开发人员在类中明确的添加了构造函数时，那么系统就不会自动为其添加一个默认的构造函数。构造函数在这个类的生命周期中只会被调用一次，即当开发人员通过new关键字创建对象时，系统就会调用开发人员所创建的构造函数。

这里要特别说明的是PHP和其他面向对象的开发语言，如Java、C++不同，在这些语言中可以为类中定义多个构造函数，并且通过函数重载的特性为不同输入参数提供不同的构造函数，但是由于PHP是弱类型的语言，从语法上来说PHP的变量可以表示任何的数据类型，因此PHP无法通过重载的形式为其通过多种构造函数，因此在PHP中构造函数只能有一个，并且名字是固定的，即\_\_construct（注意：construct前面是两个英文的下划线），具体如法如下：

1 class 类名 {

2 public function \_\_construct([参数列表]) {

3 // 构造函数的方法体，可以实现对象属性的初始化工作

4 }

5 }

那么，如果因为开发的需要必须要在实现构造函数的重载，也就可以通过默认参数的方式来间接地完成，即通过判断默认参数的方式来调用不同的函数实现，下面通过一个完善学生类来进行具体的说明，代码如下：

1 class Student {

2

3 private $name = null;

4 private $age = null;

5 private $sex = null;

6 private $className = null;

7

8 // 构造函数

9 public function \_\_construct($name, $age, $sex, $className = null) {

10 $this->name = $name;

11 $this->age = $age;

12 $this->sex = $sex;

13 if($className != null) {

14 $this->className = $className;

15 }

16 }

17

18 public function getStudentInfo() {

19 if($this->className == null) {

20 echo "学生名字：".$this->name.", 年龄：".$this->age.", 性别：".$this->sex;

21 } else {

22 echo "学生名字：".$this->name.", 年龄：".$this->age.", 性别：".$this->sex.", 班级：".$this->className;

23 }

24 }

25 }

两种构造对象的方法：

1 // 传入三个参数来构造对象

2 $baseStudent = new Student("张三", 18, "男");

3 $baseStudent->getStudentInfo();

4

5 // 传入四个参数来构造对象

6 $advStudent = new Student("李四", 25, "男", "嵌入141");

7 $advStudent->getStudentInfo();

程序结果输出：

1 // 传入三个参数的输出结果

2 学生名字：张三, 年龄：18, 性别：男

3

4 // 传入四个参数的输出结果

5 学生名字：李四, 年龄：25, 性别：男, 班级：嵌入141

从上面的例子可以看出，当构造函数中的参数没有默认值时，那么就必须要为该参数传值，而当构造函数的参数有默认值时，那么就可以不传参，PHP正是通过这个方法来实现不同构造函数的切换，同时再在构造函数的实现体中通过$this指针对象来为每一个通过new关键字创建的对象中的属性进行赋初值，从而实现不通过对象的初始化。

2、析构函数的运行分析

与构造函数相同，析构函数也是一个特殊的函数，并且是从PHP 5开始新添加的内容。主要负责对象在被销毁前执行一些特殊的操作，例如关闭数据库、释放数据等。PHP中的析构函数和构造函数一样，都由系统自动调用，开发者无法进行干预。前面讲了构造函数是在使用关键字new时系统首先调用的函数，那么对于析构函数而言系统则是通过判断该对象是否属于系统垃圾，如果是垃圾则会回收该对象，并且调用该对象的析构函数，这种方式就是PHP的垃圾回收机制。

要理解析构函数的运行就需要先理解PHP的垃圾回收机制，这种机制进过多次版本的更迭已经非常复杂，并且效率很高，在这里只做简单的讲解，想深入了解的垃圾回收机制的算法可以在网上找IBM工程师的一篇论文《Concurrent Cycle Collection in Reference Counted Systems》，其中深入讲解了一种垃圾回收的算法。PHP的垃圾回收简单来说就是“引用计数”（Reference Counting），即为每个内存中的对象分配一个计数器。当某一个对象被建立时，那么该对象的计数器就会初始化为1，因为此时一定会有一个变量来引用该对象，然后如果该对象又被新的变量引用，那么该对象的计数器就会增加1，以此类对，对象的计数器会随着引用的变量增多而增加。但如果引用该对象的变量被销毁时，那么该对象的计数器就会减1，这里要特别注意的是对象的销毁和变量的销毁是完全不一样的概念，因为变量的定义是在栈上，当该变量超过了它的作用域时就会被销毁，而对象是在堆上，是不会被销毁的。因此，当垃圾回收机制运作的时候，会检查所有对象的计数器，如果某个对象的计数器为0，那么就意味着没有任何变量在引用该对象，此时就把该对象视为系统垃圾，对其进行销毁，并回收这部分内容。

当系统要把对象销毁时，就会自动去调用对象的析构函数，如果开发人员没有添加自定义的析构函数，那么系统就会调用一个默认的析构函数，当然这个析构函数其实不做任何事情。要添加析构函数，其实和构造函数的方法非常类似，也是一个固定的名称，即\_\_destruct（注意：destruct前面也是两个英文的下划线），这里要特别说明的是，析构函数是不带任何参数的，具体如法如下：

1 class 类名 {

2 public function \_\_destruct() {

3 // 析构函数的方法体，可以实现对象销毁前的清理工作

4 }

5 }

下面再以学生类为例，添加相应的析构函数，具体代码如下：

1 class Student {

2

3 private $name = null;

4 private $age = null;

5 private $sex = null;

6 private $className = null;

7

8 // 构造函数

9 public function \_\_construct($name, $age, $sex, $className = null) {

10 $this->name = $name;

11 $this->age = $age;

12 $this->sex = $sex;

13 if($className != null) {

14 $this->className = $className;

15 }

16 }

17

18 public function getStudentInfo() {

19 if($this->className == null) {

20 echo "学生名字：".$this->name.", 年龄：".$this->age.", 性别：".$this->sex;

21 } else {

22 echo "学生名字：".$this->name.", 年龄：".$this->age.", 性别：".$this->sex.", 班级：".$this->className;

23 }

24 }

25

26 // 析构函数

27 public function \_\_destruct() {

28 echo $this->name."对象被销毁";

29 }

30 }

两种构造对象的方法：

1 // 创建好对象后，把变量赋值为null，断开与对象之间的引用

2 $baseStudent = new Student("张三", 18, "男");

3 $baseStudent = null;

4

5 // 创建好对象后，由于程序执行完毕，所以系统清空所有资源

6 $advStudent = new Student("李四", 25, "男", "嵌入141");

程序结果输出：

1 // 对象的计数器为0，系统进行回收和销毁

2 张三对象被销毁

3

4 // 系统清空资源，回收和销毁所有对象

5 李四对象被销毁

从上面的例子可以看出，不论是计数器减为0，还是资源被清空，系统都会把对象进行销毁，并且执行析构函数。在第一个例子中，当创建好“张三”对象是，该对象的计数器为1，因为有$baseStudent对其进行引用，但是后来$baseStudent被赋值为null，就会断开与原先对象之间的引用，此时就会把“张三”对象的计数器减1，此时该对象的计数器就为0，系统就会进行回收和销毁。而在第二例子中则是因为页面执行完毕，所以系统断开所有的引用，从而执行回收操作。

### 4.3.3 静态函数与静态变量的运行分析

从前面讲的内容来看，要创建一个对象的方法就是通过关键字new来实现，即通过在内存的堆中开辟一块空间，然后在内存的栈中申请一个整形的变量来存放该空间的地址，最后通过该变量来实现所有的操作。在这个看似没有问题的过程中却存在这么一个问题，即用户可以通过new关键字来创建无数个同一个类的对象，然后通过这些创建出来的对象分别访问数据，对于普通的实现来说，这一过程没有问题，但是对于一些高级的应用则不行，例如数据库访问、文件访问等具有单一数据源特性的访问中，最终的操作对象只有一个，如果有无数的对象对该单一数据源进行访问和修改就会造成数据源的不稳定和不可预见性，因此在这些应用中就不希望开发人员可以随意创建任意多个的对象，而是应该在任何情况下都只有唯一的一个对象对数据源进行操作，此时就需要用到PHP中另外一个非常重要的关键字static，该关键字应用场合不是很多，但是一旦用到却可以起到非同一般的效果。

在4.3.1中讲了计算机把内存分为堆和栈两个部分，分别用于存储大小可变的数据类型和大小固定的基本数据类型，其实在内存中还有一块区域，该区域称之为静态存储区域，该区域内的数据会在整个程序运行期间一直占用一部分的内存，并且其内存地址不会发生变化，直到程序结束后才把这部分内存释放和清空。在PHP中有两种静态的使用方法，即静态变量和静态函数。

1、静态变量的运行分析

在PHP中静态变量有三种应用场合，第一种是函数内部的静态变量，第二种是全局的静态变量，第三种则是静态的类属性。这三种应用场合除了静态全局变量外，其他两种都必须在变量的前面加上关键字static，具体代码如下：

1 // 场景一：static静态局部变量

2 function static\_local() {

3 static $local = 0 ; // 初始化为0

4 $local++;

5 echo $local . '<br>';

6 }

7

8 // 场景二：static静态全局变量

9 function static\_global() {

10 global $glo; // 初始化为0

11 $glo++;

12 echo $glo . '<br>';

13 }

14

15 // 场景三：static静态类属性

16 class StaticProperty {

17 static $count = 0;

18

19 public function \_\_construct() {

20 self::$count++;

21 }

22

23 public function getCount() {

24 return self::$count;

25 }

26 }

由于静态变量的存储的区域是在静态存储区，并且内存地址不变，直到程序运行完毕，所以使得静态变量在程序运行的过程中始终存在，所以它不会随着变量所在的程序执行完毕就销毁，同时由于是变量，因此每次再次执行该函数时都会在上次一的基础上再次修改。在上面三个场景中场景一和场景二比较好理解，在场景一中，虽然$local变量在static\_local函数内部，但是该变量是静态变量，因此每次调用static\_local函数时都会把$local加1，但是又因为$local在static\_local函数内部，因此出了static\_local的作用域就没办法再次调用该变量，只能在static\_local内部使用。在第二个场景中定义了$glo为全局变量，因为PHP中的全局变量，一定是静态变量，因此它和第一个场景中的变量有相同的性质，此外因为把$glo定义为全局变量，所以出了在static\_global内部使用外，还可以在函数的外部使用该变量，具体的实现代码如下：

两种场景函数调用的方法：

1 // 场景一：static静态局部变量

2 static\_local();

3 static\_local();

4 static\_local();

5

6 // 场景二：static静态全局变量

7 static\_global();

8 static\_global();

9 static\_global();

10 echo $glo;

程序结果输出：

1 // 场景一：static静态局部变量

2 1

3 2

4 3

5

6 // 场景二：static静态全局变量

7 1

8 2

9 3

10 3

接下来重点要讲的是第三个场景，即把类的属性定义为静态变量。在这个场景下静态的属性已经不属于某一个对象，因为这些静态属性的存储位置不在对象的存储位置处，所以在调用该属性的方法和普通属性的方法不同，需要用到关键字self，通过"self::属性名"的方式来调用该变量，如场景三中所示的样子，具体实现代码如下：

场景三的调用方法：

1 // 场景三：static静态类属性

2 $staticProperty1 = new StaticProperty();

3 $staticProperty2 = new StaticProperty();

4 $staticProperty3 = new StaticProperty();

5 echo "Count = ".$staticProperty3->getCount();

程序结果输出：

1 // 场景三：static静态类属性

2 Count = 3

从上面的分析可以得到下面的结论，即把局部变量变为静态变量后改变的是它的存储方式，从堆栈空间变换到静态存储区，也就改变了它的生存期，使得在下一次调用的时候还可以保持原来的值，而不是初始化的值。

2、静态函数的运行分析

上面详细讲解了静态变量的一些特性和使用方法，接下来要讲的是静态函数的使用。静态函数和普通函数在创建时没有太大的区别，只是需要在函数的最前面加上关键字static即可，并且静态函数只能放在类中使用，而不能把普通函数设置为静态函数具体代码如下：

1 class StaticProperty {

2 static $count = 0;

3

4 public function \_\_construct() {

5 self::$count++;

6 }

7

8 public static function getCount() {

9 return self::$count;

10 }

11 }

在实际的开发中，静态函数通常不会单独的存在，而是与普通的函数和变量共存，那么此时就特别要注意的是静态函数只能调用静态函数或者静态变量，而普通函数却能够调用静态函数和静态变量，这是因为当在类中定义了静态函数或静态变量后，系统运行时就会在内存的静态存储区内创建这些静态的函数或变量，而不用等到类的对象被创建。具体来说就是当普通的成员函数需要调用静态函数时，那么此时类的对象就已经被创建，当然也就能够调用早已在存储在静态区内的静态函数和静态变量，但是当静态函数要调用普通函数或者变量时，此时可能对象还没有在内存中创建，那么此时也就无法调用普通的函数和变量，所以为了避免这些不确定的操作，在PHP中就禁止静态函数调用普通的成员函数和变量。

除了在类的内部会调用静态函数和静态变量外，在具体开发时还经常会在类的外部调用静态函数或静态变量，此时要注意的是调用类的静态函数或静态变量不是用对象来调用，而是用类名进行调用，即“类名::静态函数名/静态变量”，具体代码如下：

1 class StaticProperty {

2 static $count = 0;

3

4 public function \_\_construct() {

5 self::$count++;

6 }

7

8 // 静态函数

9 public static function getCount() {

10 return self::$count;

11 }

12 }

13

14 // 类外部调用类的静态函数

15 echo "Count = ".StaticProperty::getCount();

程序结果输出：

1 // 静态函数的结果

2 Count = 3

## 4.4 面向对象中封装性的编程思想

基于面向对象理念的编程语言种类几乎占据了目前整个可用编程语言数量的80%以上，能取得这么大的成功主要归功于这一类的编程语言能够极大的提高编程的生产效率，特变是Java、PHP、Python等，开发人员几乎不需要了解底层的运行方式，只需要了解哪些函数是用来干什么的，哪些函数和哪些函数组合可以完成什么功能即可，可以说是“拿来主义”在编程领域的最好体现。具有这一特性的其中一个原因就是面向对象的封装性，因为有了封装性，开发人员可以把自己开发好的程序变成一个个独立的小程序，随后的开发人员只需要了解这些小程序对外的接口是什么，而不需要了解内部的构造，这样就能节省开发人员大量的时间，并且屏蔽了开发中的许多细节问题，因此本节将详细参数在面向对象编程中的封装性的应用和编程理念。

### 4.4.1 封装性的基本含义

要能灵活的运用封装性进行编程，那么首先需要了解封装性的含义。所谓封装性就好比是一台手机，我们只要通过简单的触摸或者点击就可以思想打电话、发短信能操作，那么这里的封装性就有两个层面的意思，首先是从整体上来看，通过封装性可以把一个完整的实物联系到一起，变成一个不可分割的整体，其次再从内部细节来看，通过封装性可以把系统内部的各个模块进行划分，只保留对外的接口，而不展示内部所有的细节，使得整个系统都以各自的接口为基础进行交互，减少了各模块之间的耦合性，也便于后续进行单独模块的维护和升级。

其实利用C语言中的结构体也可以编写出具有类特性的对象，这一点在嵌入式系统中非常常用，但是由于C语言并不是面向对象的语言，因此不具有面向对象语言的封装性，程序编写时通常利用变量或者函数的命名规则来进行区分，但是这些造成了很多不可预见的安全隐患，例如对于用户的性别、年龄这些基本的属性是不可以随意更改的，如果更改了就可能在后续的数据检索中无法找到需要的信息。此外对于没有封装特性的对象来说除了随意调用属性会造成问题外，所以的调用方法也会破坏系统的完整性，例如用户要去提交一份完整的材料，那么需要经过领取材料、填写材料、提交材料这三个步骤，而如果用户只调用其中的任意一个单独的步骤，那么都没有实际的意义，因此通过封装性就可以把这三个步骤封装为一个步骤，从而来完成一项功能。

所以对于封装性而言，首先是防止开发人员随意的修改其内部重要的属性，其次避免开发人员在不了解情况的时候随意调用函数造成对象内部的混乱，从而是的程序能够结构化、模块化，大大降低开发难度和成本。

### 4.4.2 封装性关键字的使用方法

在PHP中和封装性相关的关键字一共有三个，分别是private、protect、public，通过这三个关键字就可以规定了某一个属性或者某一个方法它的被访问权限，具体如下：

1、private关键字：表示该属性或者方法不能在外部被调用，即外部不可见，而在该类的继承类中也是可不见的，即不能在继承类中被调用，最后就是在该类的内部是可以被调用。

2、protect关键字：表示该属性或者方法不能在外部被调用，即外部不可见，而在该类的继承类中却是可见的，即可以在继承类中被调用，最后就是在类的内部是可以被调用。

3、public关键字：表示该属性或者方法能在外部被调用，即外部可见，而在该类的继承类中也是可见的，即可以在继承类中被调用，最后就是在类的内部是可以被调用。

从上面的描述不难看出，通过这些关键字就可以很好的定义哪些是可以在外部被调用的，而哪些只能在内部被调用，而又有哪些是只能在继承类中被调用的，这些定义和我们生活中真实的现象极为类似，例如在一个家庭中，爸爸要去买礼物给孩子，孩子负责收礼物，孩子也不会关心爸爸到哪里去买礼物，只要最终的礼物能让他开心就行，这就好比是孩子调用了爸爸的public函数去买礼物，那么爸爸根据实习情况调用private函数去买礼物，因为具体在哪里买也不需要孩子知道，等孩子长大了爸爸经常会调用protect函数来和儿子讲人生道理，这个道理只会在父子之间传递，爸爸不会把这些道理去和一个不相干的人说。因此从上面的例子也再次印证了本章开始说的，面向对象实际上就是把生活中实际处理问题的方法用计算机语言来表达，下面就通过一个实际的例子来说，具体代码如下：

1 class Student {

2 private $name = null;

3 private $age = null;

4 private $sex = null;

5

6 // 构造函数

7 public function \_\_construct($name, $age, $sex) {

8 $this->name = $name;

9 $this->age = $age;

10 $this->sex = $sex;

11 }

12

13 private function modifyAge() {

14 $this->age -= 5;

15 }

16

17 public function getStudentInfo() {

18 if($this->sex == "女") {

19 $this->modifyAge();

20 }

21 echo "学生名字：".$this->name.", 年龄：".$this->age.", 性别：".$this->sex;

22 }

23 }

在这个例子中Student类的属性全部定义为private，并且把构造函数也定义为了public，此外还定义了两个成员函数，一个为public，用于获取学生的信息，另一个为private，用于修改当前的年龄，接下来来看该类的实际调用，具体代码如下：

1 // 调用私有的属性

2 $maleStudent = new Student('张三', 28, '男');

3 $maleStudent->name = '李四';

4

5 // 调用私有的方法

6 $maleStudent = new Student('张三', 28, '男');

7 $maleStudent->modifyAge();

8

9 // 调用公有的方法，并且创建一个张三对象

10 $maleStudent = new Student('张三', 28, '男');

11 $maleStudent->getStudentInfo();

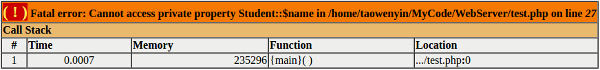
12

13 // 调用公有的方法，并且创建一个小明对象

14 $femaleStudent = new Student('小明', 28, '女');

15 $femaleStudent->getStudentInfo();

执行调用私有属性和方法的程序结果：





执行调用公有方法的程序结果：

1 学生名字：张三, 年龄：28, 性别：男

2 学生名字：小明, 年龄：23, 性别：女

从上面的程序结果不难看出，当对象访问私有属性时会触发程序的异常，导致程序无法正常执行，同样的当访问私有方法时也会出现异常，因此就可以验证不论是私有的属性还是私有的方法都不能在类的外部被调用。而当访问公有函数时就不会出现上面的异常，但是从上面调用公有方法的例子中可以看出，当学生的性别为女性时会调用一个私有的函数来修改年龄，这也符合实际的情况，通常女生都不愿意公布自己的年龄，所以把这个函数定位私有的，当必须要发布自己年龄时就在当前的年龄基础上减5，所以也可以得到结论，即在类的内部不论是公有函数还是私有函数都可以随意的调用，不收任何限制。所以在面向对象的程序设计中首先要设计好类中哪些是私有的，哪些是公有的，只有正确区分才能为后续的开发提供可扩展和可维护的支持。

### 4.4.3 单例模式的设计与意义

在4.3节的最后提出了一个问题，即在有些应用中特别是应用中，如数据库访问、文件访问等具有单一数据源特性的访问中，其操作对象只能有一个，而不允许系统有多个对象对其操作时，因此就不需要不论开发人员如何操作，其对象始终都只有一个，以避免多个对象造成的不稳定和不可预见性，本节就将讲述如何通过面向对象的封装性来解决这个问题。

要让一个类只能有一个对象，其核心就是不能让开发人员无限制的通过关键字new来调用构造函数，从而创建对象，但往往开发人员习惯性的就会通过new来创建对象，因此这个时候必须巧妙的利用其封装性来解决，即把类中的构造函数的权限改成private，使得后来的开发者无法通过new关键字进行构建，具体代码如下：

1 class Singleton {

2 private static $instance = null;

3

4 private function \_\_construct() {

5 // 私有构造函数，避免多次创建

6 }

7

8 // 通过静态函数创建对象，并判断对象时候存在，如果存在就不创建，否则就创建

9 public static function getInstance() {

10 if(is\_null(self::$instance)) {

11 self::$instance = new Singleton();

12 }

13

14 return self::$instance;

15 }

16 }

通过上面的代码不难看出，单例模式的核心就是构造函数的私有化，避免对象的多次创建，但是仅仅这样是不够的，因为对于开发者来说其关键是要能够获得该类的对象，只不过这个对象始终只有一份，而不是不能创建对象，那么有什么办法能够避开构造来调用类的函数了？就需要用到上一节中讲到的静态函数和静态变量，因为静态函数和静态变量是保存在内存的静态区域，并且只要程序一运行就会始终保持在内存中，另外其调用方法也是通过类名进行调用，而不是通过对象进行调用，这个特性正好符合单例模式的要求，所以需要在类中定义一个静态函数getInstance()，该静态函数能够返回该类的对象，又因为静态函数只能调用静态函数或者静态变量，因此在类中就需要有一个静态的变量来存在该类的对象，因此在该类的成员变量中就定义了一个静态变量叫$instance，又因为$instance是静态变量，所以它不会随着对象的消失而消失，会一直保存在内存中，也就意味着无论何时、何地都可以通过self::$instance来判断对象是否存在。

具体来说，在getInstance()函数内首先通过is\_null()函数来判断$instance是否为null，如果为null，那么就说明该类的对象还没有被创建，就可以通过new关键字来创建，要特别说明的是，这里能通过new关键来创建的原因是该函数在类的内部，而不是外部，private函数只是对外不可见，在类内部则可以任意调用，所以可以通过new关键来创建该类的对象并返回，之后每次要得到对象时，只需要通过getInstance()函数即可，如果对象已经存在就直接返回，如果不存在，那么创建后再返回。单例模式在实际开发中使用的非常多，在这里读者需要特别注意和理解的是它的实现思想，因为在实际开发中，它的变种非常多。

## 4.5 面向对象中继承性的编程思想

除了封装性外，面向对象中的继承性也是其非常重要的设计，因为有了这一特性开发人员所编写出来的程序才能够被复用和扩展，从而大大提高编程编程的效率，避免开发人员不断重复造轮子的问题，同时通过继承性也可以帮助后续的开发人员在前人程序的基础上进行不断的扩展和完善，使得整个程序具有很强的扩展性，例如在目前非常流行的几款编辑软件，VSCode、VIM、Emacs，它们都是通过继承相应的接口来实现软件无限的扩展，从而具有非常丰富的功能，所以本节将详细讲解在面向对象编程中的继承性的应用和编程理念。

### 4.5.1 继承性的基本含义

所谓继承性，顾名思义就是在某一个类的基础上创建一个新的类，并且保留原来类的相关内容，例如成员函数、成员变量等，然后在其基础上进行扩展，添加新的函数和新的成员变量等，并且这个继承关系可以始终延续下去，从而形成更加丰富的继承形式。所以在开发时，通过称被继承的类为基类或者父类，而继承的类为派生类或者子类，因此继承也可以说是子类对父类的再扩展和再完善。

这里要特别说明的是，继承性在不同编程语言中表现有所差别，例如C++语言，虽然该语言是一个面向对象的语言，但是在实现继承时可以有一个或者多个基类，而像Java和PHP则不同，其实现继承时只能从一个基类继承，而不能有多个基类。在许多刚接触编程或者接触编程不久的开发人员会经常有一个疑惑，就是为什么Java和PHP这样相较于C++而言更加高级的语言会摒弃多继承，而选择单继承，因为多继承的好处显而易见，即可以让子类有更多的父类特性，但是多继承也有其很明显，并且非常致命的两个缺点，具体如下：

1、在多个父类中具有相同名字的成员变量，从而发生歧义。在实际开发中，很多时候是无法知道父类的源代码，也因此无法修改，而且即使知道也最好不要进行修改，就会带来一个不确定性，当多个父类中使用同一个成员变量名称时，子类的引用就会产生歧义，使得系统无法判断是引用的哪一个父类中的成员变量，从而引发错误，具体伪代码如下：

1 class A {

2 public $parentValue = 0;

3 }

4

5 class B {

6 public $parentValue = 1;

7 }

8

9 class C 继承 A, B {

10 public function printParent() {

11 echo 父类->$parentValue

12 }

13 }

从上面的代码不难看出，当创建类C的对象，并且调用printParent()函数来打印父类变量时，系统无法判断是A中$parentValue还是B中$parentValue，从而引发了错误。

2、在多个父类中具有相同名字的成员函数，从而发生歧义。即当多个父类中使用同一个成员函数名称时，子类的引用就会产生歧义，使得系统无法判断调用哪一个父类中的成员函数，从而引发错误，具体伪代码如下：

1 class A {

2 public function printValue() {

3 echo '1';

4 }

5 }

6

7 class B {

8 public function printValue() {

9 echo '2';

10 }

11 }

12

13 class C 继承 A, B {

14 }

从上面的代码不难看出，当创建类C的对象，并且调用父类的printValue()函数来打印变量时，系统无法判断是A中printValue()函数还是B中printValue()函数，从而引发了错误。因为这两个错误都是非常致命的，会严重影响代码的正确运行，所以在Java和PHP中就禁止了多继承的特性，以避免这些问题。

### 4.5.2 继承性关键字的使用方法

PHP中关于继承性的关键字只有一个，即extends。继承的实现也非常简单，只要创建一个新的类，然后在类名后面添加关键字extends，并在extends关键字后面添加父类的类名即可，具体代码如下：

1 class Student {

2 private $name = '张三';

3 private $sex = '男';

4 private $age = '20';

5

6 public function getName() {

7 return $this->name;

8 }

9

10 protected function getSex() {

11 return $this->sex;

12 }

13

14 private function getAge() {

15 return $this->age;

16 }

17 }

18

19 class Undergraduate extends Student {

20 private $university = null;

21

22 public function \_\_construct($university) {

23 $this->university = $university;

24 }

25

26 public function printInfo() {

27 echo '我叫'.$this->getName().' 我的大学叫'.$this->university.'<br />';

28 }

29 }

30

31 class HighSchool extends Student {

32 private $school = null;

33

34 public function \_\_construct($school) {

35 $this->school = $school;

36 }

37

38 public function printInfo() {

39 echo '我叫'.$this->getName().' 我的高中叫'.$this->school.'<br />';

40 }

41 }

42

43 $undergraduate = new Undergraduate('南京大学');

44 $undergraduate->printInfo();

45

46 $highSchool = new HighSchool('北京外国语学校');

47 $highSchool->printInfo();

执行程序的结果：

1 我叫张三 我的大学叫南京大学

2 我叫张三 我的高中叫北京外国语学校

在上面例子中，首先声明了一个学生类Student，并且在该类中定义了三个私有的成员变量，分别是$name、$sex、$age，用于表示学生的姓名、性别、年龄，然后在Student类中定义了三个成员函数，分别为公有函数getName()、保护函数getSex()、私有getAge()，用于获取学生的姓名、性别、年龄。然后再声明一个大学生类Undergraduate和高中生类HighSchool，并且这两个类分别通过关键extends继承于父类Student，并在这两个类中分别定义了函数printInfo()用于打印相关信息，从程序的程序的结果可以看出，通过继承大大简化了程序的结构，并且实现了现有程序的复用，在类Undergraduate和HighSchool中并没有定义getName()这个函数，但是因为这个两个类都继承了Student类，因此它们就包含了Student中的相关函数，就是因为这个原因，所以在printInfo()可以调用父类Student中的getName()函数。

上面所说的原理非常好理解，但又暗藏玄机，因为在封装性这一节中讲了在面向对象的编程中可以为成员变量和成员函数添加关键字private、public、protect中的一个，以限定其访问权限，那么如果一个类和另一个类有继承关系，那么这些关键字又会起到什么作用呢？下面将以Student、Undergraduate和HighSchool类为基础进行讲解。

1、公有关键字public在继承中的使用。当在基类中的成员变量或成员函数被定义为public时，那么对于子类而言，这个成员变量或者成员函数就是可见的，也就是说在子类中是可以被调用，就像在上面例子中，因为getName()函数在基类中被定义为public，所以在子类中可以获取其函数，同时在基类中定义为public的函数还可以被子类对象在外部进行调用，具体代码如下。从下面的结果不难验证上面的说法，即public定义的函数，不仅可以在子类内部被调用，还可以被子类的对象所调用。

1 $undergraduate = new Undergraduate('南京大学');

2 echo '我的名字叫：',$undergraduate->getName().'<br />';

3

4 $highSchool = new HighSchool('北京外国语学校');

5 echo '我的名字叫：'.$highSchool->getName().'<br />';

执行程序的结果：

1 我的名字叫：张三

2 我的名字叫：张三

2、保护关键字protected在继承中的使用。当在基类中的成员变量或成员函数被定义为protected时，那么对于子类而言，这个成员变量或者成员函数就是可见的，也就是说在子类中也可以被调用，但是它与public的区别在于，被protected定义的成员变量或者成员函数不能被对象调用，而只能在子类的内部使用，当public则不同，既可以在子类的内部被调用，也可以被子类的对象调用，具体代码如下。从下面程序运行的结果不难验证上面的说法，即protected定义的函数，只能在子类内部被调用，而不能被子类的对象所调用。

protected在内部调用：

1 class Undergraduate extends Student {

2 ......

3

4 public function printInfo() {

5 echo '我叫'.$this->getName().'，我是'.$this->getSex().'生，我的大学叫'.$this->university.'<br />';

6 }

7 }

8

9 class HighSchool extends Student {

10 ......

11

12 public function printInfo() {

13 echo '我叫'.$this->getName().'，我是'.$this->getSex().'生，我的高中叫'.$this->school.'<br />';

14 }

15 }

16

17 $undergraduate = new Undergraduate('南京大学');

18 $undergraduate->printInfo();

19

20 $highSchool = new HighSchool('北京外国语学校');

21 $highSchool->printInfo();

protected在内部调用的结果：

1 我叫张三，我是男生，我的大学叫南京大学

2 我叫张三，我是男生，我的高中叫北京外国语学校

protected在外部调用：

1 class Undergraduate extends Student {

2 ......

3

4 public function printInfo() {

5 echo '我叫'.$this->getName().'，我是'.$this->getSex().'生，我的大学叫'.$this->university.'<br />';

6 }

7 }

8

9 class HighSchool extends Student {

10 ......

11

12 public function printInfo() {

13 echo '我叫'.$this->getName().'，我是'.$this->getSex().'生，我的高中叫'.$this->school.'<br />';

14 }

15 }

16

17 $undergraduate = new Undergraduate('南京大学');

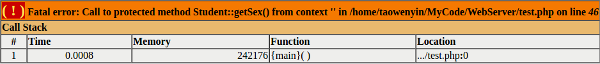
18 echo '我是'.$undergraduate->getSex().'生';

19

20 $highSchool = new HighSchool('北京外国语学校');

21 echo '我是'.$highSchool->getSex().'生';

protected在外部调用的结果：



3、私有关键字private在继承中的使用。当在基类中的成员变量或成员函数被定义为private时，那么对于子类而言，这个成员变量或者成员函数就是不可见的，也就是说在子类中不可以被调用，而只能在基类的内部使用，具体代码如下。从下面程序运行的结果可以看到不论是在子类的内部，还是通过子类对象对基类的私有函数进行调用都会触发错误，使得程序无法运行。

private在内部调用：

1 class Undergraduate extends Student {

2 ......

3

4 public function printInfo() {

5 echo '我叫'.$this->getName().'，我今年'.$this->getAge().'岁，我的大学叫'.$this->university.'<br />';

6 }

7 }

8

9 class HighSchool extends Student {

10 ......

11

12 public function printInfo() {

13 echo '我叫'.$this->getName().'，我今年'.$this->getAge().'岁，我的高中叫'.$this->school.'<br />';

14 }

15 }

16

17 $undergraduate = new Undergraduate('南京大学');

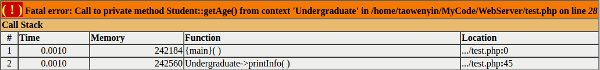
18 $undergraduate->printInfo();

19

20 $highSchool = new HighSchool('北京外国语学校');

21 $highSchool->printInfo();

private在内部调用的结果：



private在外部调用：

1 class Undergraduate extends Student {

2 ......

3

4 public function printInfo() {

5 echo '我叫'.$this->getName().'，我今年'.$this->getAge().'岁，我的大学叫'.$this->university.'<br />';

6 }

7 }

8

9 class HighSchool extends Student {

10 ......

11

12 public function printInfo() {

13 echo '我叫'.$this->getName().'，我今年'.$this->getAge().'岁，我的高中叫'.$this->school.'<br />';

14 }

15 }

16

17 $undergraduate = new Undergraduate('南京大学');

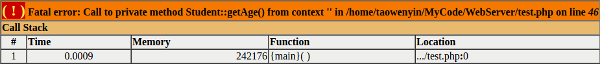
18 echo '我今年'.$undergraduate->getAge().'岁';

19

20 $highSchool = new HighSchool('北京外国语学校');

21 echo '我今年'.$highSchool->getAge().'岁';

private在外部调用的结果：



从上面的实例中可以总结三个关键字public、protected、private在基类、子类，以及外部调用上的特点，如表4-1。

表4-1 访问权限在类的成员函数和成员变量中的特点

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 | public | protected | private |
| 基类中的调用 | 可以 | 可以 | 可以 |
| 子类中的调用 | 可以 | 可以 | 不可以 |
| 对象外部的调用 | 可以 | 不可以 | 不可以 |

在PHP的类中除了普通的成员函数和成员变量外，还有一个的构造函数，不论是默认的构造函数，还是自定义的构造函数，在类中都一定会有一个构造函数，因此当类和类之间具有继承关系时就需要特别注意，这就有两种情况，分别为：

1、在基类中创建了自定义的构造函数。此时就必须在子类中首先调用基类的构造函数，虽然PHP的语法没有强制这一操作，但是如果在子类中没有调用父类构造函数，也就意味着会数据和逻辑两方面的问题。一方面是当子类在调用父类中的某些函数时会因为数据不全而无法运行，从而造成程序的错误和内部逻辑的混乱。另外一方面是无法正确的映射生活中的现象，因为从现实生活中，要有子类，首先就必须要有父类才行。因此当父类中定义了构造函数，那么在子类中就需要用到关键parent关键，该关键字用于调用父类的函数，除了普通的成员函数外，也包含构造函数，从该关键就可以调用父类的构造函数，从而初始化父类中的数据，具体代码如下：

父类构造函数的调用：

1 class Student {

2 private $name = null;

3 private $sex = null;

4 private $age = null;

5

6 public function \_\_construct($name, $sex, $age) {

7 $this->name = $name;

8 $this->sex = $sex;

9 $this->age = $age;

10 echo '父类构造函数 <br />';

11 }

12

13 ......

14 }

15

16 class Undergraduate extends Student {

17 private $university = null;

18

19 public function \_\_construct($name, $sex, $age, $university) {

20 parent::\_\_construct($name, $sex, $age);

21 $this->university = $university;

22 echo 'Undergraduate构造函数 <br />';

23 }

24

25 public function printInfo() {

26 echo '我叫'.$this->getName().'，我的大学叫'.$this->university.'<br />';

27 }

28 }

29

30 class HighSchool extends Student {

31 private $school = null;

32

33 public function \_\_construct($name, $sex, $age, $school) {

34 parent::\_\_construct($name, $sex, $age);

35 $this->school = $school;

36 echo 'HighSchool构造函数 <br />';

37 }

38

39 public function printInfo() {

40 echo '我叫'.$this->getName().'，我的高中叫'.$this->school.'<br />';

41 }

42 }

43

44 $undergraduate = new Undergraduate('张三', '男', 20, '南京大学');

45 $undergraduate->printInfo();

46

47 $highSchool = new HighSchool('小明', '女', 18, '北京外国语学校');

48 $highSchool->printInfo();

父类构造函数的调用结果：

1 父类构造函数

2 Undergraduate构造函数

3 我叫张三，我的大学叫南京大学

4 父类构造函数

5 HighSchool构造函数

6 我叫小明，我的高中叫北京外国语学校

从上面的结果不难看出，当在子类中调用了父类的构造函数，那么系统就会在子类的构造函数中先调用父类的构造函数，把父类这个对象进行创建，然后在调用子类的构造函数，把子类的对象进行创建，这样做法不仅符合程序的顺序，也符合生活中的逻辑。

2、在基类中没有创建自定义的构造函数。此时对于子类来说就没有必要通过关键parent来初始化父类，因为在父类中没有数据需要初始化，所以对于子类来说只需要使用自己的自定义构造函数，或者默认的构造函数即可。

## 4.6 面向对象中多态性的编程思想

接下来要说的就是面向对象编程三大特性中的最后一个，即多态性。多态性最重要的特点就是能让程序根据输入的不同自动选择所需要的函数进行执行，同时也可以让具有相同父类的子类具有统一的接口，却在执行过程中能够自动判断是哪一个子类，从而执行不同子类中的函数。多态性除了上面以程序的角度来思考外，也与继承性、封装性一样可以用生活中的案例来解读，例如到商场里面买东西，可以采用支付宝、微信、现金、充值卡等多种方式进行结账，但最终都会把费用支付给商家，这种通过多种方式，但产生相同结果的过程就是多态，生活中其实还有很多，例如上班时交通工具的选择等。因为在PHP的语法规定不能有重名的函数，因此PHP不像其他语言可以在一个类中实现相同函数名，不同函数参数的多态性，而是主要以父类和子类中的函数多态为主要表现形式。

### 4.6.1 父类和子类的函数重写

所谓重写（英文：override），就是当父类中有一个成员函数A，而子类在继承父类的同时把函数A也会继承过来，此时如果在子类中也定义了一个函数B，该函数与父类成员函数A具有相同函数名和相同函数参数，但是函数的实现体却不同，那么就说子类中函数B是父类成员函数A的重写。

从表面上看函数的重写只是定义了一个与父类形式相同，而内容不同的函数，似乎没有太大的意义，但是实际上重写的使用具有非常重要的实际意义。首先在父类和子类的继承过程中private是不能在子类中被继承的，而能被继承的就只有protected和public函数，因此如果要在子类中实现函数的重写，只有从protected和public这两类函数中进行挑选，其次这两类函数所表达的意义是能在子类或者外部调用的函数，通常作为个对象之间或者父类和子类之间通信的接口，在应用开发中当子类对某个接口没有特别需求时就可以使用从父类中继承的函数，从而保证对象接口的完整性，而当子类中有特别需求时，那么就可以通过重写的方式来对覆盖父类中的函数，并且通过这个覆盖的函数来实现接口相同，实现却不同。就像当我们还是婴儿的时候，我们说话的方式都是以哭声来进行表达，不论是中国人还是美国人，而当我们长大后说话的方式则各有不同，中国人采用中文来表达，而美国人则采用英语来表达，同时中国人和美国人也都可以通过哭声来表达某种情绪，下面就通过一个实例来具体的讲解。

重写函数的调用：

1 class Baby {

2 private $name = null;

3 private $sex = null;

4 private $age = null;

5

6 public function \_\_construct($name, $sex, $age) {

7 $this->name = $name;

8 $this->sex = $sex;

9 $this->age = $age;

10 }

11

12 public function getName() {

13 return $this->name;

14 }

15

16 protected function getSex() {

17 return $this->sex;

18 }

19

20 private function getAge() {

21 return $this->age;

22 }

23

24 public function talk() {

25 echo "我是婴儿，我在用哭说话<br />";

26 }

27 }

28

29 class Chinese extends Baby {

30

31 public function \_\_construct($name, $sex, $age) {

32 parent::\_\_construct($name, $sex, $age);

33 }

34

35 public function talk() {

36 echo '我叫'.$this->getName().'，我是中国人<br />';

37 }

38 }

39

40 class American extends Baby {

41

42 public function \_\_construct($name, $sex, $age) {

43 parent::\_\_construct($name, $sex, $age);

44 }

45 }

46

47 $baby = new Baby('李四', '男', 2);

48 $baby->talk();

49

50 $chinese = new Chinese('张三', '男', 20);

51 $chinese->talk();

52

53 $american = new American('Lucy', '女', 18);

54 $american->talk();

重写函数的调用结果：

1 我是婴儿，我在用哭说话

2 我叫张三，我是中国人

3 我是婴儿，我在用哭说话

在上面的案例中首先创建了一个父类Baby，在该类中通过构造函数传入姓名、年龄和性别，并可以通过三个getXXX()函数进行获取，同时在Baby类中定义了一个talk()函数，用于表示说话，因为父类表示的是婴儿，所以就在talk()函数类打印"我是婴儿，我在用哭说话"这句话。在定义完Baby类后定义它的两个子类，分别是Chinese和American，这两个类表示中国人和美国人，两者相同的地方是都在各自的构造函数中调用了父类的构造函数，不同的地方在于Chinese这个类中创建了talk()来重写的父类的函数，而在American中则没有重新。从结果可以看出，当Chinese类中重写了父类中的talk()函数后，子类对象在调用该函数时会自动调用子类中重写后的函数，而当子类中没有重写父类中的函数时，那么它就会调用父类中的函数。最后，还需要特别说明的是，当子类要重写父类中的方法时，其访问权限不能低于父类的访问权限，即当父类的某个函数访问权限为private，那么子类重写该函数时的访问权限就可以根据需求在private、protected、public中任选一个，而当父类的某个函数访问权限为protected，那么子类重写该函数时的访问权限就只能在protected、public中任选一个，以此类推。如果子类中重写函数的访问权限低于父类的访问权限，即父类中函数的访问权限是protected，而子类中重写函数的访问权限是private的话就会出现错误，具体代码如下：

子类的函数访问权限低于父类的访问权限的调用：

1 class Baby {

2 ......

3

4 protected function talk() {

5 return "我是婴儿，我在用哭说话<br />";

6 }

7 }

8

9 class Chinese extends Baby {

10 ......

11

12 private function talk() {

13 return '我叫'.$this->getName().'，我是中国人<br />';

14 }

15

16 public function printInfo() {

17 echo $this->talk();

18 }

19 }

20

21 $chinese = new Chinese('张三', '男', 20);

22 $chinese->printInfo();

子类的函数访问权限低于父类的访问权限的调用结果：



在上面案例中首先修改了父类Baby中talk函数的访问权限为protected以及函数体，然后把子类Chinese类中的talk()函数的访问权限由public改为private，从而使其访问权限比父类Baby中的访问权限要低，同时添加一个public的函数printInfo()来调用talk()函数，这样的修改从单独的某一个类来说都没有问题，但是因为Baby和Chinese存在父子关系，同时子类中的talk()函数又是重写了父类中的talk()函数，因此其语法就规定其子类的访问权限就不能低于父类的访问权限，而现在子类Chinese中的talk()函数访问权限比父类中的talk()函数的访问权限低，从而造成了整个程序的运行错误。

### 4.6.2 抽象类的意义与使用

在生活中，我们经常遇到这样一类情况，比如菜市场买半成品的熟菜，这样的熟菜回家后只要用锅烧熟，根据自己的口味稍微加些调料进行调整就可以吃了，因为熟菜在被卖掉之前已经由商家进行腌制和调味，又如买回来的手机，只需要插入SIM卡就可以使用，而不需要把手机的零部件进行拼装后再使用。类似上面的示例其实还有很多，接下来把这些案例进行抽象，就可以得出这样一条规律，即这些示例都是帮助用户完成了一套完整流程中的一部分，例如半成品的调制、手机的组装，而用户只需要完成其中的一部分需要用户自定义的部分，如回家后把这些半成品进行加热和进一步调味，以及把不同的SIM卡插入手机，最终通过用户自定义部分流程内容和预先定义流程相结合的方式完善了整个系统。

上面讲的是生活中的例子，我们说面向对象的程序设计是源于生活，读者可以思考下如果此时要用程序的语言来描述上述内容，该如何表达呢？读者可能会发现运用现有的知识无论如何都很难精确的表达出用户自定义实现完整流程中的一部分这一概念，因此在面向对象的编程语言中提出了一个叫做抽象类和抽象函数的概念。在了解抽象类之前需要先了解抽象函数，所谓的抽象函数是一个没有函数实现体，只有函数声明的函数，也就是说不需要在函数的后面加上花括号，而只需要在小括号后面直接添加分号作为结束符，抽象函数的具体定义方法如下：

1 abstract function 函数名(参数列表);

读者此时可能会疑惑，为什么要定义一个没有函数实现体的函数，因此设计抽象函数的目的就是为了在子类中能够被重写，而且是必须被重写，目的就是完成一个完整流程中需要用户自定义的部分，从而才能实现最后完整的结果，因此当父类中定义了抽象函数，那么在子类中就必须要进行重写，并完成它的该函数的实现体。了解了抽象函数后，再了解抽象类就方便很多，因为凡是有抽象函数的类，一定是抽象类，但是抽象类不一定要包含抽象函数，并且抽象类中也可以包含其他普通的函数和成员变量，抽象类的具体定义方法如下：

1 abstract class 抽象类 {

2

3 }

下面就以手机为例，讲解抽象函数和抽象类的具体使用方法，具体代码如下：

1 abstract class Mobile {

2 private function startCallFun() {

3 echo '启动电话程序...<br />';

4 }

5

6 public abstract function insertSIMCard($number);

7

8 private function parseSIM() {

9 echo '解析SIM卡程序...<br />';

10 }

11

12 public function callPhone($number) {

13 $this->startCallFun();

14 $this->insertSIMCard($number);

15 $this->parseSIM();

16 echo '拨打电话成功';

17 }

18 }

19

20 class UserMobile extends Mobile {

21 public function insertSIMCard($number) {

22 echo '插入的电话SIM卡号码为：'.$number.'<br />';

23 }

24 }

25

26 $userMobile = new UserMobile();

27 $userMobile->callPhone('1234567');

抽象函数和抽象类的调用结果：

1 启动电话程序...

2 插入的电话SIM卡号码为：1234567

3 解析SIM卡程序...

4 拨打电话成功

在上面的案例中首先定义了一个抽象类Mobile，在该类中有一个抽象函数insertSIMCard($number)，并且把电话作为参数传入到该函数中，并且还定义了一个public函数callPhone()用于实现整个完整的流程。定义完抽象类后，接下来就是定义子类UserMobile，该类继承于Mobile，因此就必须要实现insertSIMCard($number)函数，当把该函数的具体实现完毕后就可以创建该子类UserMobile的对象，并且调用callPhone()函数来完成整个打电话的过程，这里还要特别说明的是，当某一个类是抽象类的时候，它就不能产生对象，因此计算机中对象应该是一个具体的实物，而抽象类只是一个概念，因此不论从原理，还是语法上都不允许抽象类产生具体的对象。

### 4.6.3 接口类的意义与使用

在生活中，处上了节中提到的示例外，还有一类情况也是我们经常遇到，比如我们到电器城去买蓝光影碟机，就类似于过去的DVD机一样，我只需要对店员说我要买一个能播放蓝光影片的影碟机就可以，而不需要再详细的说明店员就可以帮我拿一个我所需要的影碟机，影碟机的种类很多、品牌也很多，就意味着它的解析蓝光的方式有很多种，但是只要符合能够解析蓝光光盘的影碟机，就成为蓝光影碟机。类似上面的示例其实还有很多，接下来把这些案例进行抽象，就可以得出这样一条规律，即这些示例都是为某一类产品定义了它所必须的功能，而只要某一个产品满足了这些必须的功能时，那么就把该产品称为是某一类产品，例如只要能够播放蓝光影片的都是蓝光影碟机，而具体的怎么实现播放这些蓝光光碟的，对于用户来说一概不知，而是由厂家来完成的。

上面讲的是生活中的例子，与上一节相同读者也可以思考下如果此时要用程序的语言来描述上述内容，该如何表达呢？读者可能会发现运用抽象函数可以解决，但是抽象函数的目的是为了完成整个流程中确实的一部分，而不是所有的接口，因此抽象函数从意义上来说不妥，虽然语法上可以实现，因此也出现了现有的知识无论如何都很难精确的表达清楚的问题，因此在面向对象的编程语言中提出了一个叫做接口的概念。所谓的接口其实它也是类的一种，只不过在里面所定义的函数都没有函数体，只有函数声明，并且这些声明的函数权限都是public，此外在接口中不能够定义成员变量，只能够定义成员产量，具体定义方法如下：

1 interface 接口名称 {

2 public const 常量名称 = 'Hello World';

3

4 public function 接口函数(参数列表);

5 }

看到这里读者此时依然会疑惑，为什么要定义一个没有函数实现体的函数，因此设计接口的目的就是为了规定子类中必须有哪些功能，因为只有符合这一要求才能称为是某一个类型，而子类中的其他功能相对于接口定义的函数来说都不是核心功能，因此当定义了接口后，在该接口的子类中就必须重写所有的接口函数，同时完成这些接口函数的所有具体函数体并完成它的该函数的实现体。下面就以蓝光影碟机为例，讲解接口的具体使用方法，具体代码如下：

1 interface BlurayPlay {

2 public function playBluray();

3 public function stopBluray();

4 public function nextBluray();

5 public function previousBluray();

6 }

7

8 class SonyBlurayPlay implements BlurayPlay {

9 public function playBluray() {

10 echo '播放蓝光影碟<br />';

11 }

12

13 public function stopBluray() {

14 echo '停止播放蓝光影碟<br />';

15 }

16

17 public function nextBluray() {

18 echo '播放下一个蓝光影碟<br />';

19 }

20

21 public function previousBluray() {

22 echo '播放上一个蓝光影碟<br />';

23 }

24 }

25

26 $sonyBlurayPlay = new SonyBlurayPlay();

27 $sonyBlurayPlay->playBluray();

28 $sonyBlurayPlay->stopBluray();

29 $sonyBlurayPlay->nextBluray();

30 $sonyBlurayPlay->previousBluray();

接口的调用结果：

1 播放蓝光影碟

2 停止播放蓝光影碟

3 播放下一个蓝光影碟

4 播放上一个蓝光影碟

在上面的案例中首先定义了一个接口BlurayPlay，在该类中有四个接口函数playBluray()、stopBluray()、nextBluray()、previousBluray()。定义完接口后，接下来就是定义子类SonyBlurayPlay，该类继承于BlurayPlay，因此就必须要实现上述的四个接口函数，当把这些函数的具体实现完毕后就可以创建该子类SonyBlurayPlay的对象，并且调用playBluray()、stopBluray()、nextBluray()、previousBluray()这四个函数来完成整个蓝光影碟使用的过程。这里有两点要特别说明的是，首先当某一个类是接口的时候，它就不能产生对象，因此计算机中对象应该是一个具体的实物，而接口只是一个概念，因此不论从原理，还是语法上都不允许接口产生具体的对象，其次在面向对象的编程中接口是可以多继承的，而抽象类和普通类只能单继承，具体代码如下：

1 interface 接口1 {

2

3 }

4

5 interface 接口2 {

6

7 }

8

9 class 子类 implements 接口1, 接口2, ... {

10

11 }

12

13 class 子类 extends 普通类/抽象类 implements 接口1, 接口2, ... {

14

15 }

## 4.7 小结

通过本章的学习，了解了面向对象编程中的几个基本，但非常重要的概念和意义，首先深入分析了面向对象的基本概念，树立正确的对象分析方法，其次详细分析了对象在在内存中的分配，以及堆和栈的区别，随后详细了解$this指针的意义，以及构造函数在对象创建中的作用，最后深入分析面向对象编程中的三大基本特性，即封装性、继承性、多态性，通过这些知识的学习可以帮助读者深入的了解面向对象的概念和意义，为今后的学习打下坚实的基础，本想内容需要读者更多的自我推敲，形成一个自己的理解。