目录

[前沿 1](#_Toc863198)

[第一章 物联网背景下的PHP概况 3](#_Toc863199)

[1.1 物联网系统的体系架构 3](#_Toc863200)

[1.2 感知层的技术选择 5](#_Toc863201)

[1.3 传输层的技术选择 6](#_Toc863202)

[1.4 应用层的技术选择 7](#_Toc863203)

[1.5 PHP技术的局限性 7](#_Toc863204)

[1.6 小结 7](#_Toc863205)

[第二章 Web服务器开发环境的搭建 9](#_Toc863206)

[2.1 Apache服务器的安装和使用 10](#_Toc863207)

[2.1.1 Ubuntu下Apache服务器的安装 10](#_Toc863208)

[2.1.2 Ubuntu下Apache服务器的设置 11](#_Toc863209)

[2.2 MySQL服务器的安装和使用 12](#_Toc863210)

[2.2.1 Ubuntu下MySQL服务器的安装 12](#_Toc863211)

[2.2.2 MySQL管理工具MySQL Workbench的安装和使用 13](#_Toc863212)

[2.3 PHP运行库的安装和扩展 15](#_Toc863213)

[2.3.1 Ubuntu下的PHP运行库的安装 15](#_Toc863214)

[2.3.2 常用的PHP运行库扩展程序 16](#_Toc863215)

[2.4 小结 16](#_Toc863216)

[第三章 几种常用的Web开发工具 17](#_Toc863217)

[3.1 XDebug开源的PHP调试工具 17](#_Toc863218)

[3.1.1 Linux下XDebug的安装 17](#_Toc863219)

[3.2 Vistual Code开源的跨平台程序编辑器 18](#_Toc863220)

[3.2.1 Linux下Vistual Code的安装 19](#_Toc863221)

[3.2.2 Vistual Code中PHP相关插件的安装 19](#_Toc863222)

[3.3 Eclipse for PHP Developers全能的程序编辑器 21](#_Toc863223)

[3.3.1 Linux下Oracle Java的安装 22](#_Toc863224)

[3.3.2 Linux下Eclipse for PHP Developers的安装 23](#_Toc863225)

[3.4 Chrome DevTools实用的前端调试工具 24](#_Toc863226)

[3.5 小结 25](#_Toc863227)

[第四章 PHP的面向对象分析与设计 26](#_Toc863228)

[4.1 面向对象的基本概念 26](#_Toc863229)

[4.1.1 理解类与对象 27](#_Toc863230)

[4.1.2 理解面向对象的程序设计 27](#_Toc863231)

[4.2 面向对象的构成与内存分配 28](#_Toc863232)

[4.2.1 类的声明 28](#_Toc863233)

[4.2.2 对象的实例化与内存分配 29](#_Toc863234)

[4.3 面向对象的$this和构造函数的分析 31](#_Toc863235)

[4.3.1 运行时对象属性与方法的动态访问 31](#_Toc863236)

[4.3.2 构造函数和析构函数的运行分析 33](#_Toc863237)

[4.3.3 静态函数与静态变量的运行分析 37](#_Toc863238)

[4.4 面向对象中封装性的编程思想 40](#_Toc863239)

[4.4.1 封装性的基本含义 40](#_Toc863240)

[4.4.2 封装性关键字的使用方法 41](#_Toc863241)

[4.4.3 单例模式的设计与意义 43](#_Toc863242)

[4.5 面向对象中继承性的编程思想 44](#_Toc863243)

[4.5.1 继承性的基本含义 44](#_Toc863244)

[4.5.2 继承性关键字的使用方法 45](#_Toc863245)

[4.6 面向对象中多态性的编程思想 51](#_Toc863246)

[4.6.1 父类和子类的函数重写 52](#_Toc863247)

[4.6.2 抽象类的意义与使用 55](#_Toc863248)

[4.6.3 接口类的意义与使用 56](#_Toc863249)

[4.7 小结 58](#_Toc863250)

[第五章 HTML5、CSS3、JavaScript的设计方法 59](#_Toc863251)

[5.1 HTML5的语义概念和常用标签 59](#_Toc863252)

[5.1.1 结构、样式、行为 59](#_Toc863253)

[5.1.2 HTML5中新增的文档结构标签 61](#_Toc863254)

[5.1.3 HTML5中增强的多媒体应用 64](#_Toc863255)

[5.1.4 HTML5中Form表单的新特性 65](#_Toc863256)

[5.2 CSS3中基本与高级选择器的使用 66](#_Toc863257)

[5.2.1 层叠样式表中层叠的概念 67](#_Toc863258)

[5.2.2 CSS3中的基本选择器的使用 68](#_Toc863259)

[5.2.3 CSS3中的关系选择器的使用 72](#_Toc863260)

[5.2.4 CSS3中选择器优先级的计算 73](#_Toc863261)

[5.3 面向对象编程思想在CSS3中的应用 76](#_Toc863262)

[5.3.1 CSS3中“类”概念的应用 76](#_Toc863263)

[5.3.2 “命名空间”在CSS3中的应用 77](#_Toc863264)

[5.3.3 “多用组合，少用继承”原则在CSS3中的应用 79](#_Toc863265)

[5.4 面向对象编程技术在JavaScript中的应用 81](#_Toc863266)

[5.4.1 JavaScript的运行环境 82](#_Toc863267)

[5.4.2 JavaScript中的数据类型与函数 83](#_Toc863268)

[5.4.3 JavaScript中的基本对象类型 86](#_Toc863269)

[5.4.4 JavaScript中的继承与扩展 91](#_Toc863270)

[5.5 JQuery的DOM操作和AJAX的异步通讯 97](#_Toc863271)

[5.5.1 JQuery的引入与建议 97](#_Toc863272)

[5.5.2 DOM的理解与选择器 99](#_Toc863273)

[5.5.3 Web页面中的DOM操作 101](#_Toc863274)

[5.5.4 深入理解JQuery中的事件响应 109](#_Toc863275)

[5.5.5 Ajax异步通讯的理解和使用 114](#_Toc863276)

[5.6 小结 118](#_Toc863277)

# 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。

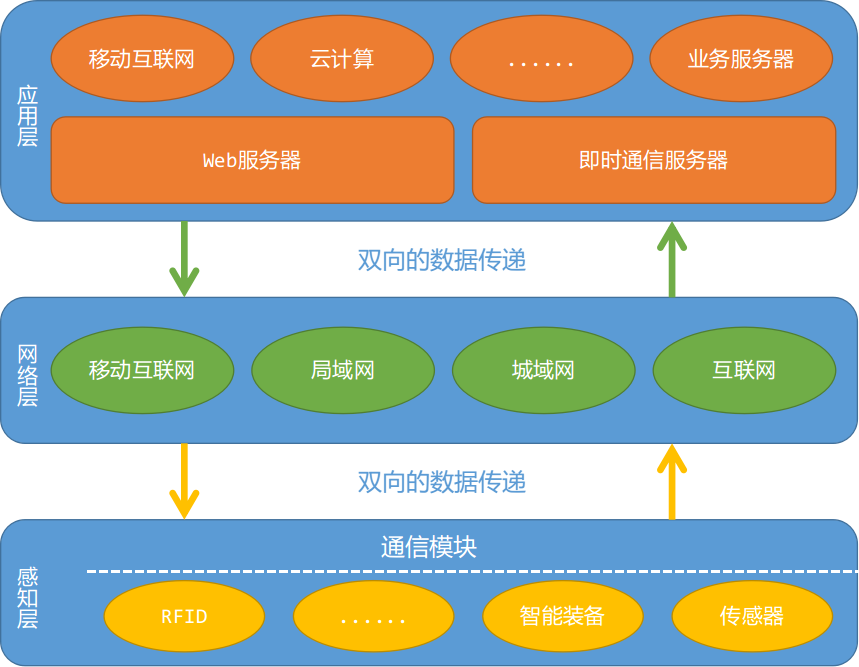


图1-1 物联网体系架构

## 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。

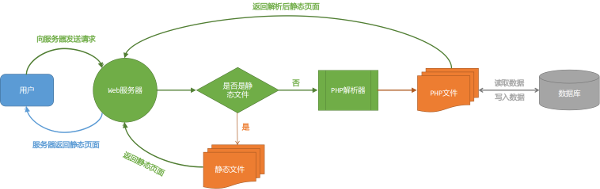


图2-1 Web服务器运行流程

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

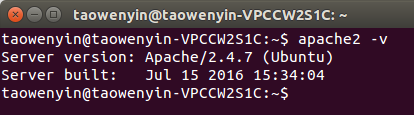


图2-3 Apache安装验证

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

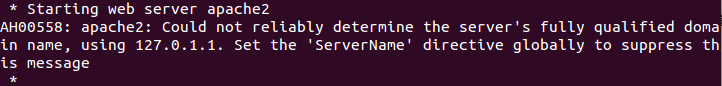


图2-4 安装Apache的错误

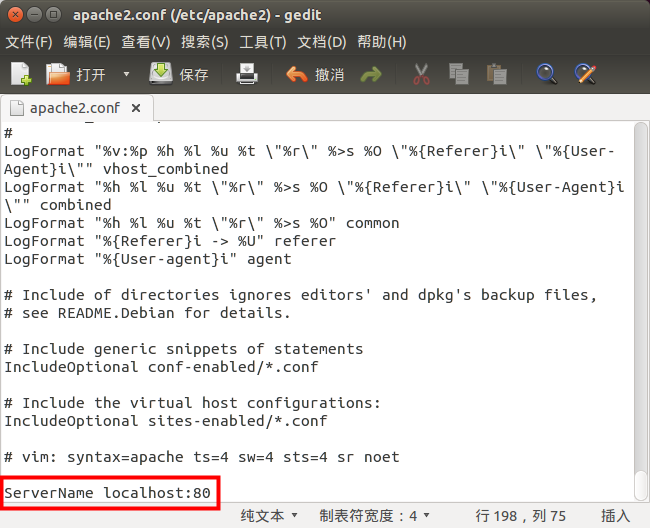


图2-5 为Apache设置ServerName

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。

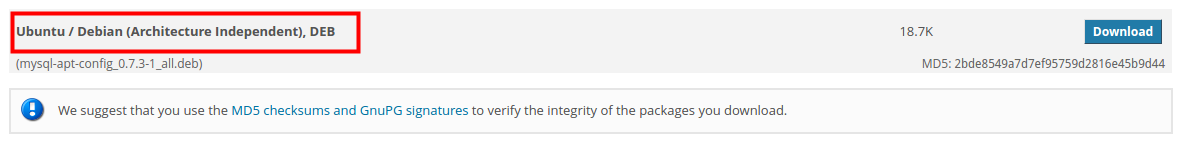


图2-8 MySQL源更新文件

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

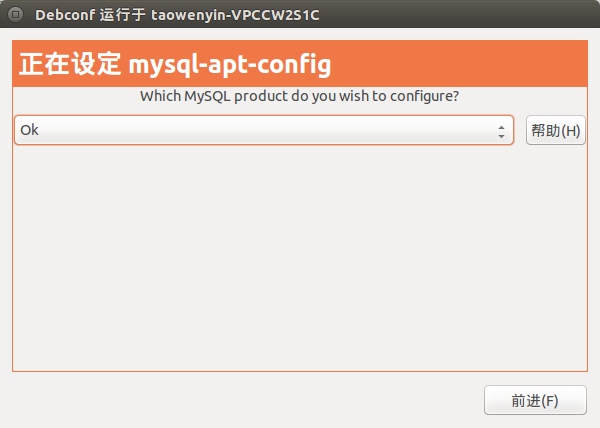


图2-9 MySQL源更新配置



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

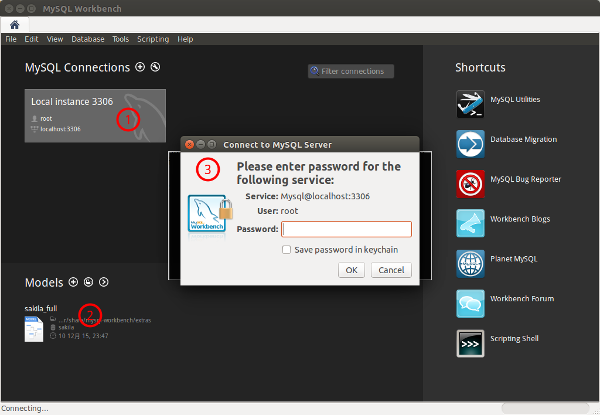


图2-12 MySQL Workbench的登陆界面

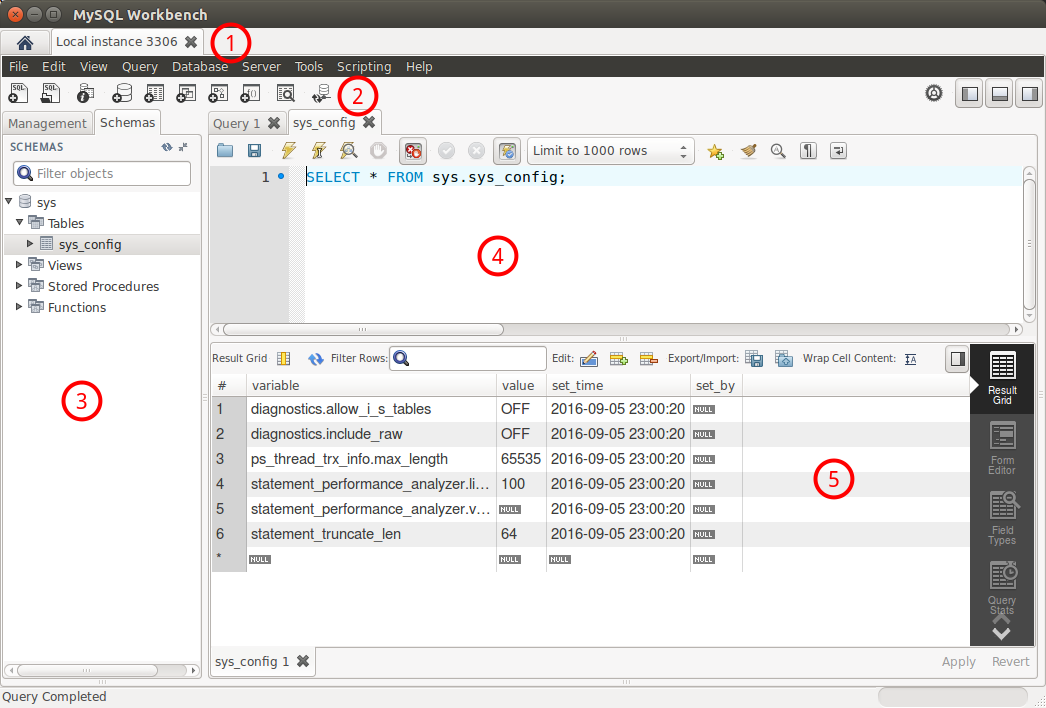


图2-13 MySQL Workbench的主界面

## 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。

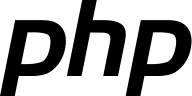


图2-14 PHP的图标

### 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

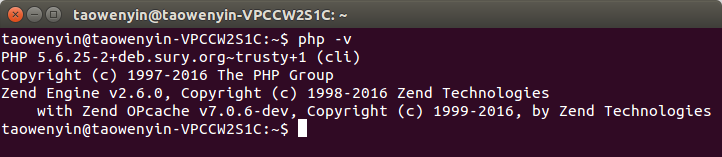


图2-14 PHP安装验证

### 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

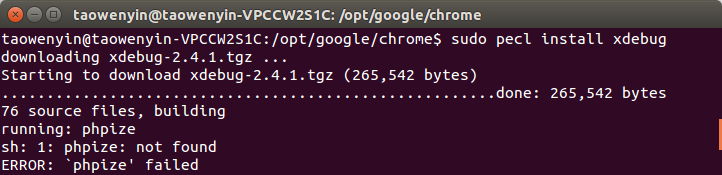


图3-2 XDebug的安装错误

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。

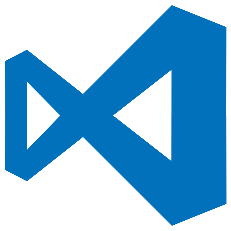


图3-1 VSCode的图标

### 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。

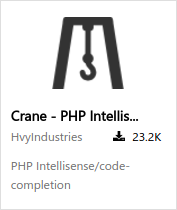
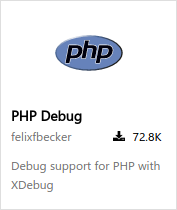


图3-3 两个PHP开发常用的插件

**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。

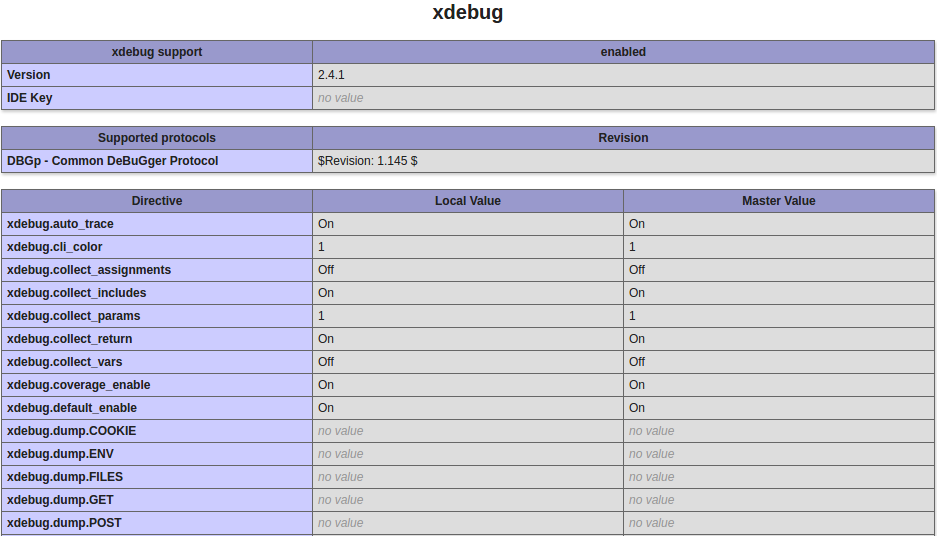


图3-4 XDebug的配置信息

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。

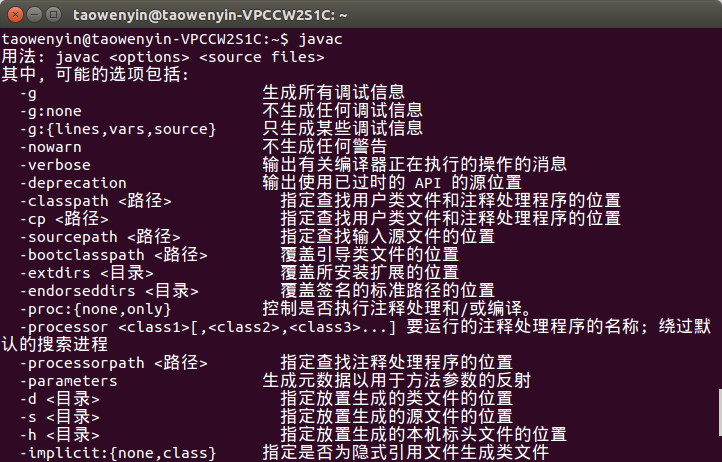
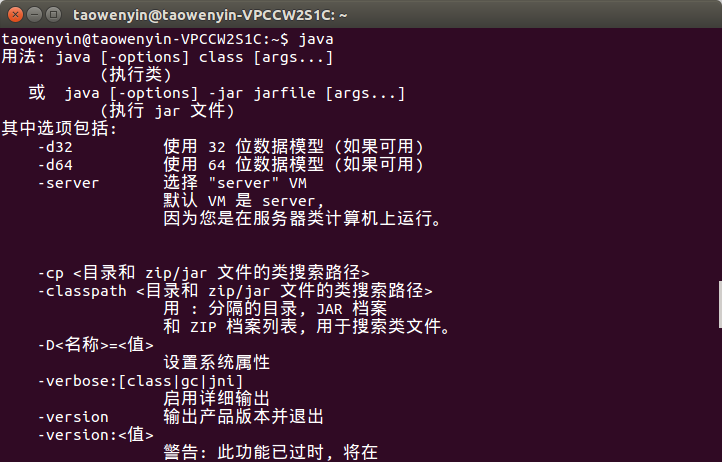


图3-8 验证Java安装完成

### 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。

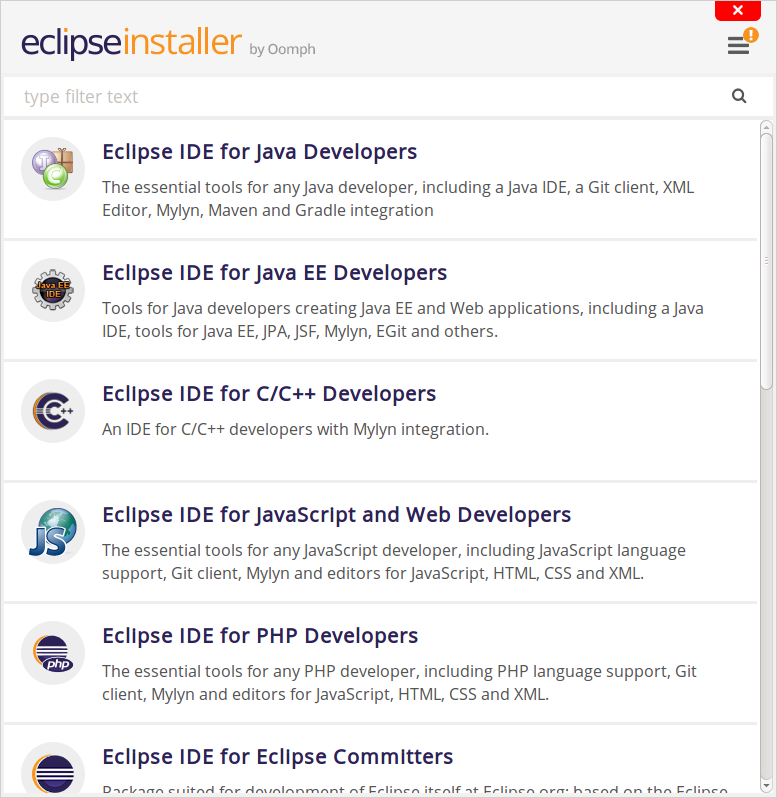


图3-9 Eclipse安装包类型选择界面

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。

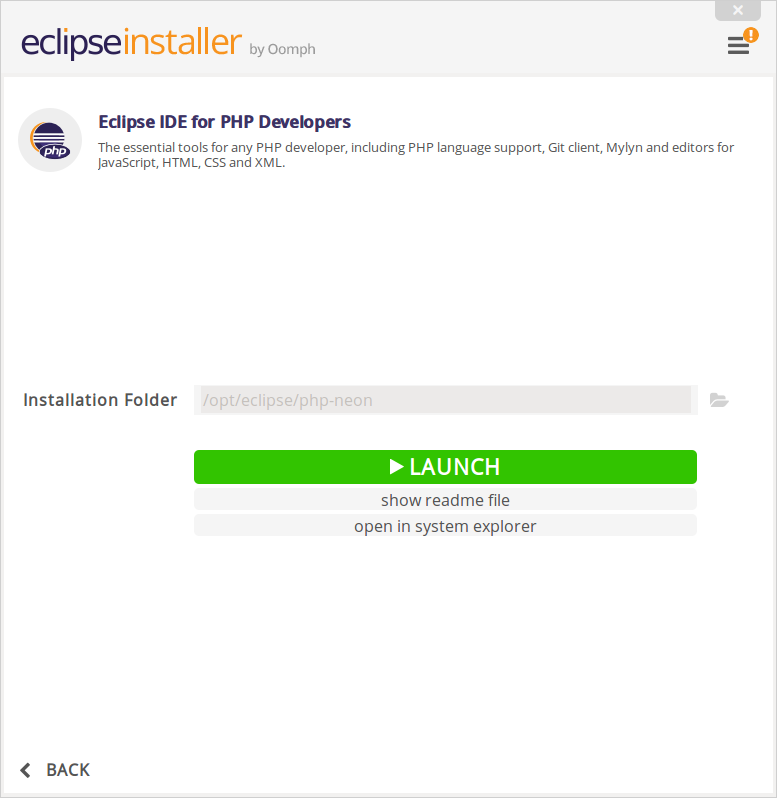


图3-11 安装完成择界面

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。

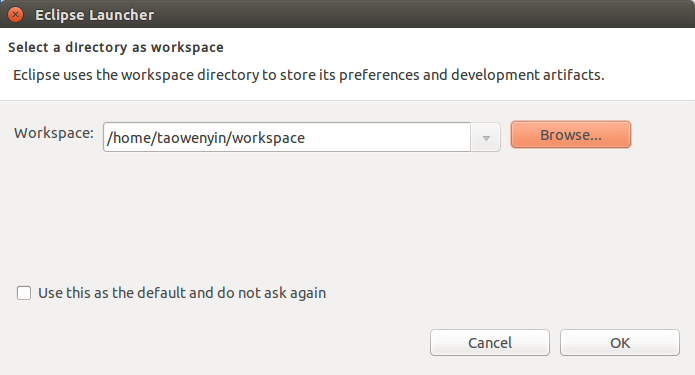


图3-12 工作目录配置

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。

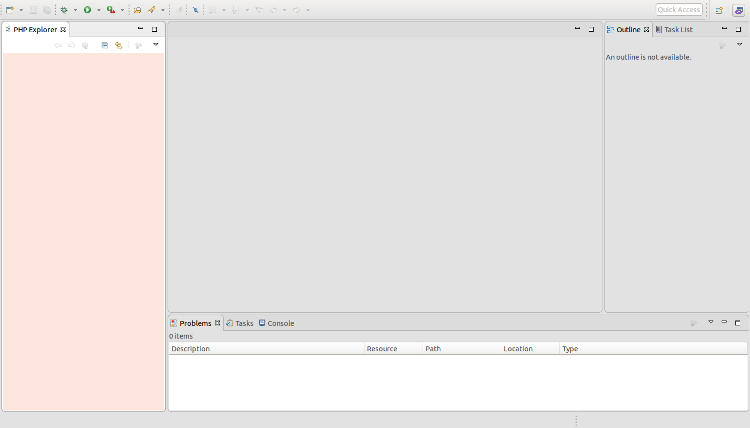


图3-13 Eclipse运行界面

## 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：

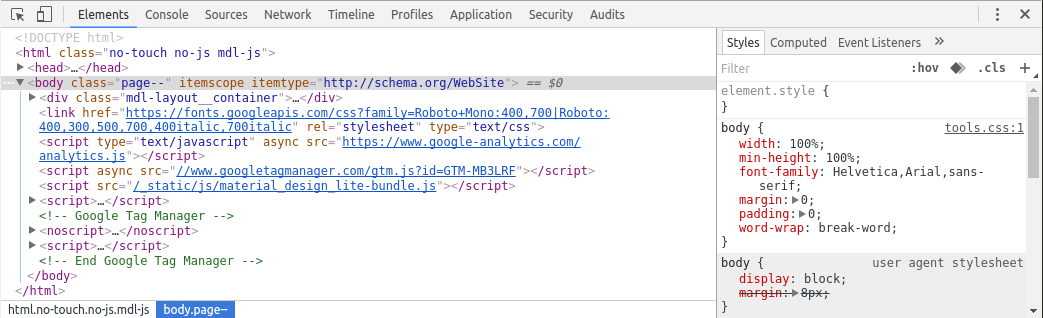


图3-14 Chrome DevTools运行界面

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。

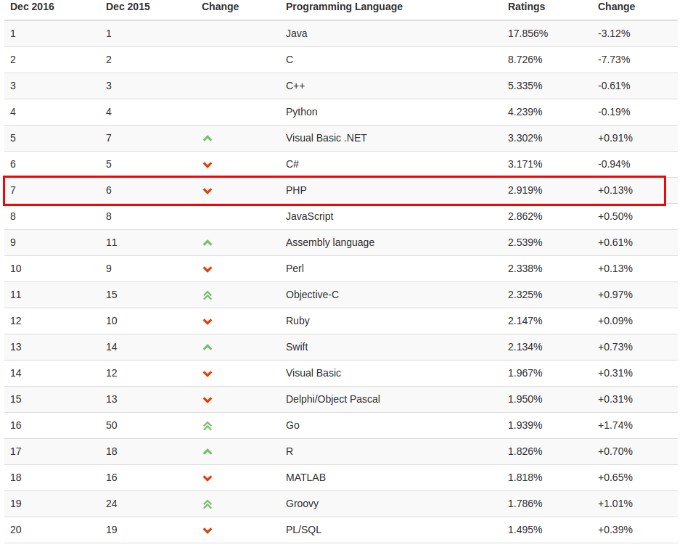


图4-1 PHP语言排行榜

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。

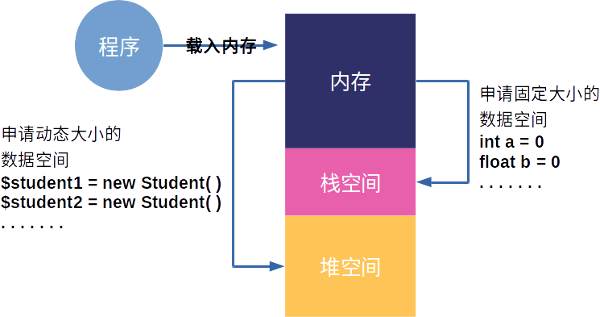


图4-2 堆栈内存的分布

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。

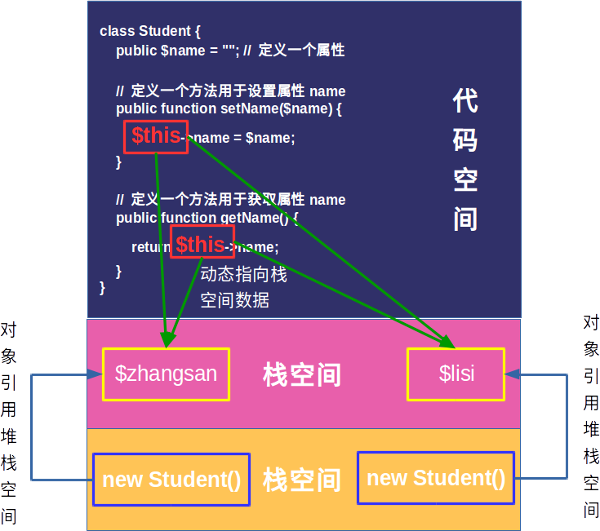


图4-3 $this指针的理解

### 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();

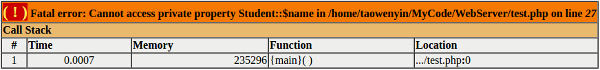
12

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');

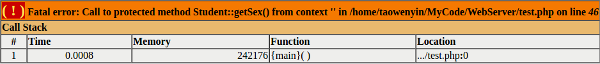
18 echo '我是'.$undergraduate->getSex().'生';

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');

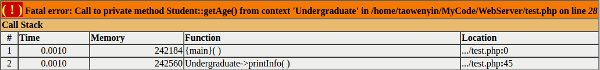
18 $undergraduate->printInfo();

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');

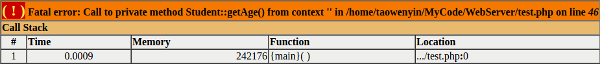
18 echo '我今年'.$undergraduate->getAge().'岁';

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。

# 第五章 HTML5、CSS3、JavaScript的设计方法

古人云：“学而不思则罔，思而不学则怠”。千百年之前的圣贤已经告诉我们在学习的过程中要不断的思考和实践，即使是已经学过的东西也要思考其中的内在联系，并建立多种解决方案的知识体系，并且比较这些方案之间的优劣。阅读本书的读者一般都具有一定前端开发的知识，本章将不对基本的知识进行讲解，而是重要的且在开发过程经常遇到的各种问题提出相对较好的解决方法，从而抛砖引玉让读者能够向技术的更深层次思考。

本章内容

* HTML5的语义概念和常用标签
* CSS3中基本与高级选择器的使用
* 面向对象编程思想在CSS3中的应用
* 面向对象编程技术在JavaScript中的应用
* JQuery的DOM操作和AJAX的异步通讯

## 5.1 HTML5的语义概念和常用标签

2014年10月，这个月对于全世界的大部分人来说没有什么特殊的意义，但是对于IT界确有着不同的思考，因为在这个月HTML5的正式版发布，这也意味着在未来浏览器所支持的标准将越来越统一，页面呈现的效果在不同浏览器之间的差异将越来越小，开发人员也能够把精力更加集中在系统的构建，而不是调整各浏览器之间的兼容性。

然而HTML5标准的最终公布却经历的8年之久，对前端有一定了解的开发者可能知道，W3C（全称：World Wide Web Consortium，万维网联盟）是Web前端领域内多项标准的制定者，但是在1999年发布HTML4.01标准之后，W3C就没有对HTML有进一步的发展，而是有了另起炉灶的想法，但是随着Web系统的体量不断增大，在各种新功能层出不穷的被推出，对于前端的要求越来越高，而此时现有标准的内容已经越来越不能满足这些复杂系统的要求，同时W3C想指定的新标准又迟迟无法公布，因此如Opera、Mozilla基金会和Apple等这样的IT界的巨头决定自己成立一个组织，即WHATWG（全称：Web超文本应用技术工作组），用于推进HTML5的标准化，随后经过多年的博弈W3C决定采用WHATWG所指定的标准，并且将其完成成了现在正式版的HTML5标准，HTML5的图标如图5-1所示。



图5-1 HTML5的图标

### 5.1.1 结构、样式、行为

虽然在HTML5标准推行的过程中出现了WHATWG，但最终的发布者还是W3C，所以和HTML5相关的标准还是秉承了W3C标准的核心理念，即将网页的结构、网页的样式和网页的行为进行分离，各自形成自己的标准，这三个部分的标准就分别对应网页结构的HTML5标准、网页样式的CSS3标准和网页行为的EMCAScript6（即JavaScript）标准，通俗的说在这个核心理念里面HTML只是用来承载整个页面的内容，即有哪些文字和段落，它不会干涉页面的外观，而CSS只是用来负责承载页面的样式，即这个页面长成什么样子，最后的JavaScript则只是承载这个页面类似按下按钮弹窗等一系列的用户交互行为，如图5-2所示。

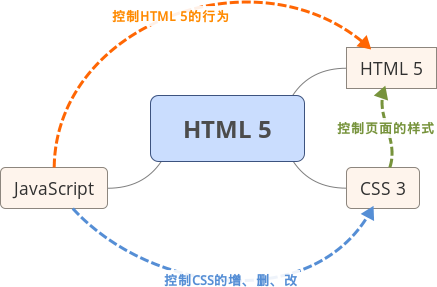


图5-2 HTML5标准间的关系

在过去传统的Web页面所采用的布局方式是以表格为基础，特别当Dreaweaver中可以实现各种控件的拖拽时，表格的布局就大行其道了，然后科技不断发展，页面要展示的内容也越来越多，复杂程度以几何数增长，同时前端开发人员也渐渐意识需要把程序和美工进行责任分离，使开发人员可以有更多的精力投入到前端代码的实现中，但是此时前端开发人员发现以表格为基础的页面布局存在布局灵活性较差，以及当页面内容复杂时表格的嵌套层次会非常多，从而严重影响后期的维护,如图5-3所示。因此为了解决这个问题，于是发明了DIV+CSS的布局方式，这种布局方式通过纯粹的CSS中浮动、位置等定位方法来实现页面布局，而HTML则专注于实现页面的内容结构，从这一刻起Web前端开发第一次把页面的结构和页面的样式进行分离，如图5-4所示。

**（缺图）**

图5-3 表格布局的案例

**（缺图）**

图5-4 DIV+CSS布局的案例

随着时间的推移，网站的重要性越发的重要，像小米在2014年花费360万美金购买域名的案例也不在少数，这些都是为了能让用户更多的了解品牌，然而除了网站域名的重要性外，能够被搜索引擎也是非常重要的，这个就是俗称的SEO（即搜索引擎优化），在网络上搜索引擎是看不见页面的样式，因此页面的好看和难看对于搜索引擎来说都是一样的，它只会搜索页面的结构，而DIV+CSS的布局方式虽然已经在SEO方面进行了很多的改进，但是有许多标签存在语义不明确，且样式和语义混淆的问题，造成无法进行有效的SEO处理，因此HTML5出现的一个很重要的目的就是删除了原先HTML4.01中和页面样式相关的标签，例如center、font、big等一系列标签，而添加了例如section、article、nav等语义更加明确的标签来帮助搜索引擎更好的了解页面的结构，使得用户在搜索时能够把最重要的内容呈现出来。

那么在实际的开发，前端开发人员要正确判断一个页面已经的使用语义标签，有一个简单的方法就是把样式去掉，即页面只以HTML的原本形式进行展示，由于HTML标签本身带有默认的样式，例如标题标签h1，它会把在该标签内的文字变大、变粗，然后通过观察网页的结构是否组织的良好，是否具有很好的可读性，例如标题是方法、重点部分是加粗，如果观察下来结构良好且具有较好的可读性，那么就可以说该页面正确使用了语义标签，如图5-5所示，就是在前端设计中非常著名的网站CSS Zen Garden（即禅意花园），该网站提供了标准的HTML文档，并且给定了一些预设的CSS类名，开发人员可以自己想象随意创造不同的前端，从图中可以看出所有完全不同的页面都具有相同的HTML文档内容，并且结构清晰、层次分明，而一旦加上了CSS就会变得千变万化，但依然突出重点，这就说明该页面的语义标签使用的较好，很好的表达了页面的思想。



图5-5 CSS Zen Garden语义标签的使用

### 5.1.2 HTML5中新增的文档结构标签

上节讲到HTML5出现的一个重要目的就是解决在HTML4.01时代的标签语义不明确造成的文档结构相对混乱的问题，下面的代码就是标准的HTML4.01的文档结构。

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

2 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

3 <head>

4 <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">

5 <title>响应式前端框架Bootstrap的使用</title>

6 </head>

7 <body>

8 <h1>HTML5的语义概念和常用标签</h1>

9 <p>014年10月......因为在这个月HTML5的正式版发布</p>

10 <h2>结构、样式、行为</h2>

11 <p>虽然在HTML5标准......还是秉承了W3C标准的核心理念</p>

12 <h2>HTML5中新增的文档结构标签</h2>

13 <h1>CSS3中常用的新特性</h1>

14 </body>

15 </html>

从上面的代码中不难看出，开发人员想表达的是一本书的两个章节，其中第一个h1标签到第二个h1标签之间的是第一章的内容，第二个h1标签开始则是第二章的内容，但在以HTML4.01为标准的时代没有办法通过明确的语义把第一章和第二章进行区分，所以造成SEO在解析时无法把两个h1标签之间内容作为第一章的内容，因此在当时开发人员为了解决这个问题，于是乎通过利用div标签外加CSS样式来表达出完整的文档结构文档的结构，具体代码如下：

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

2 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

3 <head>

4 <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">

5 <title>响应式前端框架Bootstrap的使用</title>

6 </head>

7 <body>

8 <div class="section">

9 <h1>HTML5的语义概念和常用标签</h1>

10 <p>014年10月......因为在这个月HTML5的正式版发布</p>

11 <div class="subsection">

12 <h2>结构、样式、行为</h2>

13 <p>虽然在HTML5标准......还是秉承了W3C标准的核心理念</p>

14 </div>

15 <div class="subsection">

16 <h2>HTML5中新增的文档结构标签</h2>

17 </div>

18 </div>

19 <div class="section">

20 <h1>CSS3中常用的新特性</h1>

21 </div>

22 </body>

23 </html>

但是即使是这样还依然无法解决文档结构语义不明却的问题，因为div标签本身没有任何语义，所以也不会因为添加了class属性就获得特殊的意义，同时div的使用在解析文档时也存在语义上的歧义，因为一个div元素可以被定义为章节或者子章节，也可以被用于仅仅为了页面的样式化。换句话说，在HTML4.01标准中章节的定义和章节的范围都不精确。因为在HTML4.01标准中有这类问题的存在，所以在HTML5添加了一系列新的结构标签用于表示具有完整语义的文档结构，如表5-1所示。

表5-1 HTML5中的结构标签

|  |  |
| --- | --- |
| 标签名 | 描述 |
| address | address标签表示作者的联系方式，如果该标签的父标签是body，那么就表示该页面作者的联系方式，如果该标签的父标签是article，则表示该文章作者的联系方式。 |
| article | article标签表示文档、页面、应用或网站中的独立结构，其意图在于分离可独立、可复用的结构，如论坛帖子、杂志或新闻文章、用户提交的评论、交互式组件，或者其他独立的内容。 |
| aside | aside标签表示页面中可以独立于内容的部分，并且被单独拆分时不会使整体受到影响。一般情况下用于表现为侧边栏或者被独立插入的内容，例如相关的广告、Web应用程序等。 |
| footer | footer标签表示章节内容或者body元素的页脚。一个页脚通常包含该章节作者、版权数据或者与文档相关的链接等信息，这里要注意的是footer标签内的联系信息因在address标签中。 |
| header | header标签表示一组引导性的帮助，可以包含标题元素，也可以包含其他元素，如logo、搜索表单等。 |
| h1~h6 | h1~h6标签表示标题元素有六个不同的级别，h1标签是最高级的，而h6标签则是最低的。 |
| nav | 页面的导航栏，用于表示一个含有多个超链接的区域，这个区域包含转到其他页面，或者页面内部其他部分的链接列表。 |
| section | section标签表示文档中的一个区域（或节），比如一个书的某个章节，通常该标签会有包含一个标题。 |

根据表4-1中内容可以把上方原先语义不明确的代码改为语义更加明确的代码，从而对整个页面进行优化。从下方修改后的代码可以看出该页面一共有两个大章节，分别对应body标签下面两个根section标签，而在第一个section标签中通过h1标签定义了章节的名称，以及通过p标签定义该章节的简介，然后再次利用section定了子章节，最后通过aside标签来定义该章节中的广告页，通过这样的代码调整使得页面的结构更加接近真实世界的文档结构，也符合了HTML5的设计里面。此外HTML5标准中协议申明和head部分都做了相应的简化，不需要像HTML4.01一样书写很长的一段代码，由于具体代码如下：

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>响应式前端框架Bootstrap的使用</title>

6 </head>

7 <body>

8 <section>

9 <h1>HTML5的语义概念和常用标签</h1>

10 <p>014年10月......因为在这个月HTML5的正式版发布</p>

11

12 <section>

13 <h1>结构、样式、行为</h1>

14 <p>虽然在HTML5标准......还是秉承了W3C标准的核心理念</p>

15 </section>

16

17 <section>

18 <h1>HTML5中新增的文档结构标签</h1>

19 <p>上节讲到HTML5出现的......的HTML4.01的文档结构。</p>

20 </section>

21

22 <aside>

23 <p>页面广告</p>

24 </aside>

25 </section>

26

27 <section>

28 <h1>CSS3中常用的新特性</h1>

29 </section>

30

31 <footer>

32 <p>(c) 2016 XXXX公司</p>

33 </footer>

34 </body>

35 </html>

### 5.1.3 HTML5中增强的多媒体应用

在发布HTML5标准之前，如果想在网页中嵌入视频或者音频，就必须要通过转码以及第三方软件来实现，因此过去在HTML功能欠缺的情况下诸如Flash等软件就非常的火爆，但也存在系统安全性较低、容易被入侵等问题，但是今天在HTML5标准中提供了功能强大的视频和音频标签就解决这一问题，同时原生的标签也大大提高了系统运行速度和系统的稳定性，以及系统的安全性。在HTML5中关于多媒体部分增加了两个标签audio和video，分别对应音频和视频。同时这两个标签在目前几乎所有的最新浏览器上都可以得到支持，而且这两个标签的使用非常简单，方法和使用image标签非常相似，只需要在src属性中添加对应资源的地址就可以，具体代码如下：

1 <video src="demo.mp4" controls>

2 <p>你的浏览器不支持video标签</p>

3 </video>

4

5 <audio src="demo.mp3" controls autoplay loop>

6 <p>你的浏览器不支持audio标签</p>

7 </audio>

此外audio和video这两个标签还共同拥有五个属性，用于控制音频或者视频是否自动播放、是否添加控制条、是否循环播放等，如表5-2所示。而video标签则再多两个width和height属性，用于设置video标签的视频大小。

表5-2 audio和video标签的共用属性

|  |  |
| --- | --- |
| 属性名 | 描述 |
| controls | 为网页中的音频或者视频显示标准的HTML5控制器。 |
| autoplay | 使网页中的音频或者视频自动播放。 |
| loop | 使网页中的音频或者视频自动重复播放。 |
| preload | 用来缓冲音频或者视频，该属性有三个可以选择的值，分别为："none"表示不缓冲文件，"auto"表示缓冲音频或者视频文件，而"metadata"表示仅仅缓冲音频或者视频文件的基本数据，如时间长度等，此外如果设置了autoplay，那么该属性就会被忽略。 |

目前能够被播放器识别的音频和视频的格式有许多种，但是常用就只有mp3、mp4等，而HTML5对于音频和视频各支持三种格式，但是不同的浏览器版本也存在着一定的差异，如表5-3、表5-4所示。这里要特别注意的是Ogg格式指的是带有Theora视频编码和Vorbis音频编码的Ogg文件，MPEG4指的是带有H.264视频编码和AAC音频编码的MPEG 4文件，而WebM则指的是带有VP8视频编码和Vorbis音频编码的 WebM文件。

表5-3 audio标签支持的多媒体格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 音频格式 | IE9 | Firefox 3.5 | Opera 10.5 | Chrome 3.0 | Safari 3.0 |
| Ogg Vorbis |  | √ | √ | √ |  |
| MP3 | √ |  |  | √ | √ |
| Wav |  | √ | √ |  | √ |

表5-4 video标签支持的多媒体格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 视频格式 | IE | Firefox | Opera | Chrome | Safari |
| Ogg | No | 3.5+ | 10.5+ | 5.0+ | No |
| MPEG 4 | 9.0+ | No | No | 5.0+ | 3.0+ |
| WebM | No | 4.0+ | 10.6+ | 6.0+ | No |

由此可以看出在不同浏览器或者相同浏览器但版本不同时，它们之间也存在一定的差异，因此HTML5中还提供了一个source标签，用于帮助audio或者video标签提供更多的音频或者视频种类以适应不同浏览器之间差异。该标签有两个属性，分别为src属性用于设置audio或者video标签的资源文件路径和type属性用于设置这些资源文件的MIME字符串，来帮助浏览器更好的解析这些资源，具体代码如下：

1 <video>

2 <source src="demo.ogg" type="video/ogg">

3 <source src="demo.mp4" type="video/mp4">

4 </video>

5

6 <audio>

7 <source src="demo.wav" type="video/x-wav">

8 <source src="demo.mp3" type="video/mpeg">

9 </audio>

### 5.1.4 HTML5中Form表单的新特性

在Web前端开发中表单通常用于登记用户的信息，所以被广泛的应用在各种论坛或门户网站中。在HTML4.01中表单通常只用于填写一些信息，然后把这些数据提交到服务器中，并且在填写这些信息时，主要以文本信息为主。而在HTML5中，表单的功能被大大的增强，使得表单在不同网站上表现的更加一致，同时还能及时的响应用户输入的数据，此外这些增加的特性对于禁止了浏览器脚本的用户也可以起到相同的作用。

在表单中用户输入的最重要的标签是input标签，该标签的属性type用于设置用户输入的类型，在HTML4.01中该属性有十个值，分别为button、checkbox、file、hidden、image、password、radio、reset、submit、text，这十个值分别表示input标签为按钮控件、多选按钮控件、上传文件控件、隐藏控件、图片控件、密码输入框、单选按钮控件、重置表单按钮控件、提交表单按钮控件和文本控件，这些控件基本能够满足日常的需求，但是在HTML5还需要考虑更多的语意和移动端的特性，因此HTML5在HTML4.01的基础上为type属性增加了如下四个值：

1、search：该值使input标签表现为一个搜索框控件，同时把输入中的换行功能去掉，此外就和input的text值效果无异，该属性值的设计主要用于更加清楚的语意表达。

2、tel：该值使input标签表现为一个输入电话号码的控件，同时把输入中的换行功能去掉，并且当程序运行在移动端时，用户的输入法会表现为电话输入状态，此外就和input的text值效果无异。

3、url：该值使input标签表现为一个Web地址输入控件，并且在输入时换行符与首尾的空格都会被自动的删除，此外当用户提交的内容格式不符合标准的URL格式时浏览器是不会允许将表单的值提交给服务器。

4、email：该值使input标签表现为一个E-Mail地址输入控件，同样原本text值时的换行功能会被自动删除，同时当程序运行在移动端时，用户的输入法会表现为邮件地址输入状态，当用户提交的内容格式不符合标准的E-Mail格式时浏览器是不会允许将表单的值提交给服务器。

HTML5除了增加input标签所表达的控件种类外，还为input标签添加了四项关于数据操作的属性，分别如下：

1、list：该属性的使用通常要和HTML5中增加的。

2、pattern：该属性通常需要和type属性值为text、tel、search、url和email的input元素相结合，在该属性值中填写相应的正则表达式，以此来规范input标签的用户输入内容。

3、form：在HTML4.01标准中表单中的所有内容都必须写在form标签内，但在实际开发中可能这些表单内容无法全部写在form标签内，因此在HTML5为input标题添加了一个form属性，该属性的值需要填写一个表单的ID号以实现input标签和表单的绑定，当绑定完成后该input标签可以不写在form标签内，但却属于绑定的表单，当该表单提交数据时依然会把该input标签的数据一起提交给服务器。

4、placeholder：该属性主要用于input与textarea标签中，以此来提示用户在此需要输入什么内容，需要注意的是placeholder属性的内容不包含回车与换行，如图5-7所示。

5、autofocus：该属性可以是表单内的某个一个控件自动获得焦点，需要注意的时是在一个HTML文件内容只有一个表单中的某一项能够拥有autofocu属性，此外当autofocus属性用于type值为hidden的input标签时，autofocus不会产生任何效果。

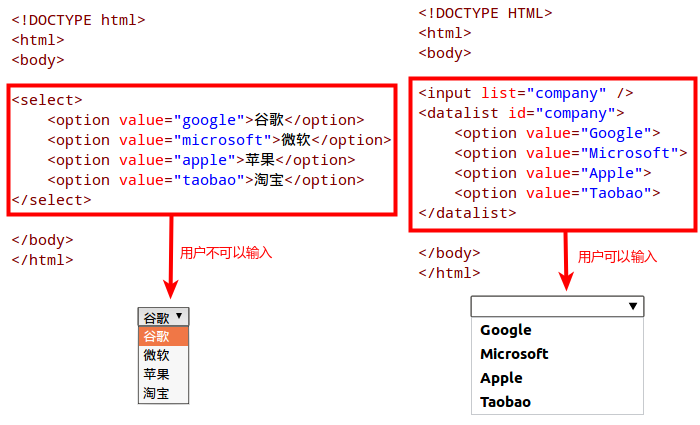


图5-6 select和datalist的区别

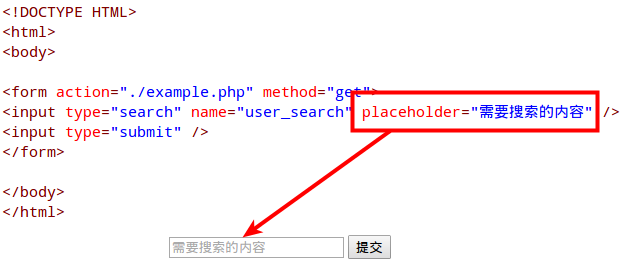


图5-7 placeholder属性的使用

## 5.2 CSS3中基本与高级选择器的使用

Cascading Style Sheets是CSS的英文全称，翻译成中文就是“层叠样式表”的意思，是在前端设计领域内专门用于开发页面样式的一种语言，正如5.1节所说的通过CSS可以帮助Web页面实现内容和样式的分离，此外CSS的使用还能够避免样式代码的重复，使得后期在维护是更加的容易，同时当Web被用于不同场合时，可以通过CSS来实现内容相同的情况下却呈现不同样式。

在过去由于Web设计标准存在滞后性与缺陷，因此造成了当时的Web开发者使用了错误的语法，然而各个浏览器厂商为了使自己的浏览器能够向后保持兼容，让这些存在缺陷的Web页面能够正常的解析，于是就发明了两种解析Web页面的模式，即标准模式（strict mode）和怪异模式（quirks mode）。其中标准模式指的是严格按照W3C制定的标准进行解析，而怪异模式指的是采用浏览器厂商自己的标准进行解析，这就造成采用这两种模式解析的网页相差甚远，其中最为显著的一个区别就是关于CSS中盒子模型宽度的界定，在标准模式下盒子模型的宽度由内容大小（width）、内边距大小（padding）和边框大小（border）三部分之和组成，而在怪异模式下盒子模型的内容大小就是已经包括了内边距大小（padding）和边框大小（border），没有把三个值进行分离，虽然看上去时很小的区别，但是在页面布局时却差别非常大。

因此在过去开发Web时都需要指定DTD（英文全程：Document Type Definition），即文档类型定义，通过指定DTD可以告诉浏览器采用那种模式进行Web解析，而到了今天HTML5的发布改变了这些，在HTML5中不再需要指定DTD，因为每个浏览器的厂商都采用相同的标准进行解析，使得Web代码在各个浏览器上能够基本保持一致性，CSS3的图标如图5-8所示。



图5-8 CSS3的图标

### 5.2.1 层叠样式表中层叠的概念

在本节的开头讲过CSS的中文名叫做层叠样式表，那么其中就有两个概念，第一个是样式表，第二个就是层叠。第一个概念非常好理解，所谓样式表就是定义用于定义页面的外观，也就是这个原因HTML5才摒弃了在HTML4.01时代很多和样式相关的标签，而把更多的和样式设置放到CSS中。而另外一个概念层叠就不是很容易理解。

在一个Web程序中，页面中的每一个元素的样式通常都是由许多个样式表共同来完成，如图5-9所示，可以看到一个div元素有多个CSS样式进行修饰。CSS正是通过各种不同方式，以及相对复杂的嵌套关系来为某个一个元素定义其样式，也就是这样复杂的关系才使得CSS具有如今强大的功能。通常来说层叠的来源主要有以下三种：

1、浏览器对于HTML元素的默认样式。

2、用户自定义的样式，例如Firefox中，可以通过“首选项”对话框中可以自定义样式，也可以建立一个单独的userContent.css的样式文件并放到“用户配置”的文件夹中。

3、开发者定义的样式，这种样式也有三种形式，即定义在外部文件的样式、定义在页面头部的样式，以及定义在特定元素本身的样式。

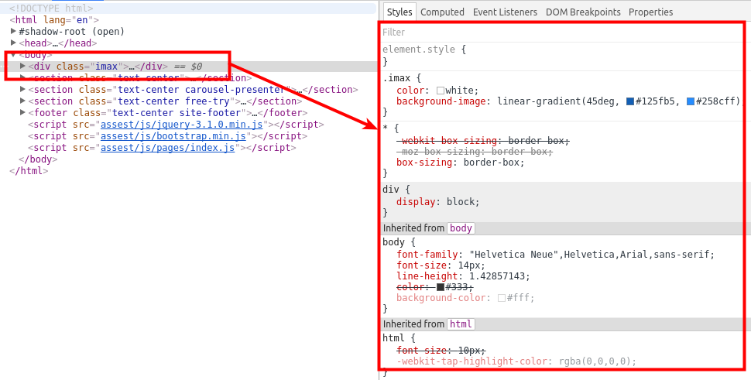


图5-9 标签的CSS修饰

当在HTML中使用strong标签时会发现strong标签内所呈现的字体会相对正常字体粗一些，这个就是上面所说的浏览器默认的样式，而当通过CSS代码把strong标签内容的文字变成红色时，那次开发人员定义的样式就覆盖浏览器的默认样式，此外如果在strong标签外添加一个父标签p，那么此时的strong就会继承p标签中的样式，同时p标签也会继承body标签标签中的许多样式，如图5-10所示。由此可以看出在CSS中样式的优先级就是开发者定义的样式优先级最高，然后时用户自定义的样式，最后才是浏览器的默认样式。

因此知道CSS的层叠概念就是标签和标签之间的继承继承关系，子标签覆盖父标签中的一些样式，同时子标签还继承父标签中的其他样式。



图5-10 CSS的层叠概念

### 5.2.2 CSS3中的基本选择器的使用

在CSS中要唯一的标识一个HTML标签、一类HTML标签以及一群不同类型的HTML标签，并改变他们的样式，那么就需要使用CSS的选择器，CSS只有通过使用选择器才能够精确的控制HTML中的任意元素。在图5-10中的代码中已经初步使用了选择，通过选择器的使用来控制body、p、strong标签的样式。CSS3中选择器的类型一共分为六种选择器，分别如下：

1、标签选择器：即以HTML标签作为标识，来改变指定标签的样式，并且只要是HTML标签都可以作为标签选择器的对象，CSS不做任何的限制，所以图5-11就是一个典型的标签选择器。此外，当在HTML文档中出现多个相同的标签，那么这些标签都会根据CSS设定的样式进行改变，如图5-11所示。



图5-11 CSS的标签选择器

2、类型选择器：在前端开发中经常需要使用Button组件，那么如果每用一个Button就为这个Button设计一个样式就显得非常繁琐，因此CSS为HTML标签提供一个class属性，该属性可以帮助开发人员进行HTML标签的分类，并且class的属性值可以任意的设置，但还是建议使用有意义的名称。此外CSS并不限制HTML标签的中class属性值的个数，所以可以使用多个属性值来共同为某个HTML标签添加样式，这也是在前端设计中所推荐的方法，但这里要注意的是在使用多个class属性值时每个属性值都需要用空格作为间隔，并且如果值的名称由多个单词组成，那么单词与单词之间需要用"-"符号作为间隔（例：class = "big-font red-font"）。最后在定义class时则需要在这些类型名字前面增加“.”符号（例：.big-font {font-size : 40px;}），此外如果要为某个元素指定定义class时就需要在“.”符号前添加元素标签（例：p.red-font {font-size : 40px; color : #F00;}），如图5-12所示。



图5-12 CSS的类型选择器

3、ID选择器：在网页开发中有个不成文的规定就是当在一个页面中需要使用ID时，那么在该页面中的ID就不能够重复，必须具有唯一性，而Class因为标识一类标签因此可以重复。虽然在页面中如果出现两个相同的ID不会导致页面无法渲染，但是这与ID所表示的语意产生了冲突。因此如果要在页面中唯一修改某个位置上的某一个标签，那么就需要使用ID选择器，ID选择的使用和类型选择器很相似，但是ID选择器的前缀不是“.”而是“#”（例：#black-board {border-style: solid; border-width: 5px; border-color: #F00;}），如果要明确的标记某一个ID元素的样式，那么就要在“#”符号前面添加元素标签（例：strong#big-font {font-size : 40px;}），如图5-13所示。



图5-13 CSS的ID选择器

4、属性选择器：当在了解网页开发相关的基础后，开发网页就不再是从头开始也代码，而是使用某一个前端框架作为基础，那么此时该框架为了实现多种不同的效果，就会为网页添加许多默认的Class和ID，此时的开发人员在进行基于此框架的二次开发时就需要根据不同属性内容添加不同的属性，此时就需要使用属性选择器，属性选择器依据使用场景的不同可以分为五种，具体格式如下：

（1）元素标签[属性名]：对具有该属性名的元素标签进行样式设定。

（3）元素标签[属性名=属性值]：对具有该属性名，且值等于该属性值的元素标签进行样式设定。

（3）元素标签[属性名^=字符串]：对具有该属性名，且值中以该字符串为开头的元素标签进行样式设定。

（4）元素标签[属性名$=字符串]：对具有该属性名，且值中以该字符串为结尾的元素标签进行样式设定。

（5）元素标签[属性名\*=字符串]：对具有该属性名，且值中以包含字符串的元素标签进行样式设定。

这里需要注意的是上面五种属性选择其中的最后三种为CSS3标准中新添加的，因此如果使用相对较老的浏览器可能会不支持这种方式，具体使用方法具体5-14所示。



图5-14 CSS的属性选择器

5、伪类选择器：在前端开发经常会碰到这样类似的需求，即需要设置按钮的普通状态、按钮的点击状态和按钮的鼠标触碰状态时的样式，另外还有需要给一群具有相同样式标签中的第一个或最后一个标签单独添加样式，这些需求都无法使用上面的选择器来完成，那么为了解决这个问题CSS3就为开发人员提供了伪类选择器。伪类选择器的使用并不会在原本的基础上添加新的Class或者ID，而是为已经存在的元素或者选择进行更加精确的选择或者为元素不同状态添加不同的样式，伪类选择器共分为两大类，即状态选择器和查询选择器，两大类总共十五个常用选择器，具体如下：

（1）状态选择器：在该类中共有八个伪类选择器，用于为标签或者选择器选中的标签设置各个状态样式。

* :link：该选择器表示选择未被访问的链接并添加样式，应用格式为在:link之前添加标签或者选择器，例：a:link {color: #FF0000}或者a.red:link {color: #FF0000;}。
* :visited：该选择器表示选择已被访问的链接添加样式，应用格式为在:visited之前添加标签或者选择器，例：a:visited {color: #00FF00}或者a.red:visited {color: #FF0000;}。
* :active：该选择器表示选择被激活的元素添加样式，应用格式为在:active之前添加标签或者选择器，例：a:active {color: #0000FF}或者a.red:active {color: #FF0000;}。
* :hover：该选择器表示选择当鼠标悬浮在元素上方时，向元素添加样式，应用格式为在:hover之前添加标签或者选择器，例：a:hover {color: #FF00FF}或者a.red:hover {color: #FF0000;}。
* :focus：该选择器表示向拥有输入焦点的元素添加样式，应用格式为在:focus之前添加标签或者选择器，例：a:focus {color: #00FF00}或者a.red:focus {color: #FF0000;}。
* :checked：该选择器表示向单选按钮、多选按钮以及下拉选项中被选中的项添加样式，应用格式为在:checked之前添加input、option标签及相关选择器，例：input[type="radio"]:checked {margin-left: 25px;}或者option:checked {color: red;}。
* :enabled：该选择器表示向任何可用的元素添加样式，即一个能够被激活（如选择、点击或接受文本输入）或获取焦点的元素，应用格式为在:enabled之前添加标签或者选择器，例：input:enabled {color: #22AA22;}。
* :disabled：该选择器表示向任何被禁用的元素添加样式，即一个不能够被激活（如选择、点击或接受文本输入）或获取焦点的元素，应用格式为在:disabled之前添加标签或者选择器，例：input[type="text"]:disabled {background: #ccc;}。

这里需要特别说明的是在状态选择器中:link、:visited、:active、:hover这四个经常组合起来应用于描述一个按钮，但是在设置的时候需要按照一定的顺序，如果顺序错误会导致样式添加失败，其正确的声明顺序为:link -> :visited -> :hover -> :active，具体使用方法如图5-15所示。



图5-15 CSS的伪类选择器中的状态选择器

（2）查询选择器：在该类中共有七个伪类选择器，用于帮助标签或者选择器选中的某个位置上的元素。

* :first-child：该选择器表示向某个元素的第一个子元素添加样式，应用格式为在:first-child之前添加标签或者选择器，例：p:first-child {font-weight:bold;}。
* :nth-child(an+b)：该选择器表示向具有相同父标签，满足在父标签中从前向后数第an+b个子标签元素，并且该子标签元素与在该选择器前添加的元素相同时所添加的样式，其中n的取值范围为大于或等于0，b是定位的偏移量，应用格式为在:nth-child(an+b)之前添加标签或者选择器，例：tr:nth-child(2n+1) {background: #ccc;}，表示设置表格的奇数行添加背景样式。
* :nth-last-child(an+b)：该选择器表示向具有相同父标签，满足在父标签中从后向前数第an+b个子标签元素，并且该子标签元素与在该选择器前添加的元素相同时所添加的样式，其中n的取值范围为大于或等于0，b是定位的偏移量，应用格式为在:nth-last-child(an+b)之前添加标签或者选择器，例：tr:nth-last-child(-n+4){font-weight:bold;}，表示设置表格的最后四行的字体为加粗。
* :nth-of-type(an+b)：该选择器表示向具有相同父标签，且满足父标签中从前向后第an+b个匹配的子标签元素添加样式，其中n的取值范围为大于或等于0，b是定位的偏移量，应用格式为在:nth-of-type(an+b)之前添加标签或者选择器，例：p:nth-of-type(2n+1) {text-align: left;}，表示设置p标签在奇数位的样式为字体左对齐。
* :first-of-type：该选择器表示匹配父元素的所有该子元素类型中第一个出现的元素。应用格式为在:first-of-type之前添加标签或者选择器，例：div:first-of-type {background-color: #F00;}，表示设置第一个父标签下的第一个div标签的背景颜色为红色。
* :last-of-type：该选择器表示匹配父元素的所有该子元素类型中最后一个出现的元素。应用格式为在:last-of-type之前添加标签或者选择器，例：div:flast-of-type {background-color: #F00;}，表示设置第一个父标签下的最后一个div标签的背景颜色为红色。
* :empty：该选择器表示匹配没有子元素的元素。应用格式为在:empty之前添加标签或者选择器，例：div:empty {background-color: #F00;}，表示设置没有子元素的div标签的背景颜色为红色。

这里需要特别说明的是在查询选择器中经常会用于实现表格的斑马纹，此外还有很多方面的应用，这里以表格的斑马纹为例示范查询选择器的使用，具体方法如图5-16所示。

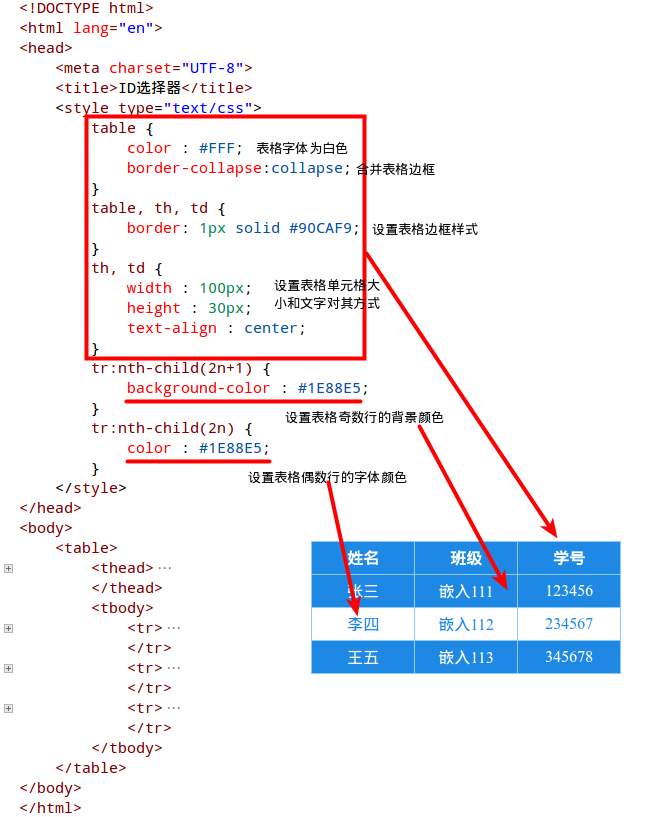


图5-16 CSS的伪类选择器中的查询选择器

### 5.2.3 CSS3中的关系选择器的使用

在上一节主要讲了CSS3中几种基本选择器，这些基本选择器都能够很好的找到页面中一个或者一类选择器，但是在实际的开发中CSS的选择要复杂的多，例如在页面中三个ClassB，它们分别在IDA、IDB、IDC下，那么此时如果要修改IDB下的ClassB样式就会发现上面的基本选择器没有办法做到，因此就需要使用CSS3中的关系选择器，关系选择器和伪选择器类似，它不会向代码中添加任何一个Class或者ID，而是通过定义多个选择器之间的关系来精确的定位到某一个标签，从而精确的修改其样式。CSS3中的关系选择器一共有五种，具体代码如图5-17所示，其五中关系管择期的意义分别如下。

1、包含选择器： 即SelectionA SelectionE {...}，也就是说定位SelectionA选择器下面的SelectionE选择器，A选择器和E选择器可以是元素，也可以是某个基本选择器。

2、后代选择器： 即SelectionA > SelectionE {...}，也就是说定位SelectionA选择器下面的SelectionE选择器，但是与包含选择器选择器不同的是，包含选择的样式可以渗透到SelectionA选择器的直接子元素、子元素的子元素，甚至更深的层次，但是后代选择器只能渗透到SelectionA选择器的直接子元素。

3、相邻兄弟选择器： 即SelectionA + SelectionE {...}，也就是说定位SelectionA后紧跟着的与SelectionA同级，且SelectionA具有相同父元素的SelectionE选择器。

4、兄弟选择器： 即SelectionA ～ SelectionE {...}，也就是定位SelectionA之后与SelectionA同级，且具有相同父元素的所有SelectionE选择器。

5、组合选择器： 即SelectionA, SelectionB, SelectionC, ... {...}，也就是定位SelectionA、SelectionB、SelectionC等多个选择器，为这些选择设置共同的样式。



图5-17 CSS的关系选择器的使用

### 5.2.4 CSS3中选择器优先级的计算

在上几节讲到网页设计中修饰一个元素通常不会只用一个CSS，而是通过多种样式的叠加来达到最终的效果，在这些样式中可能有些是整个页面的全局设置，可能有些是该页面中某个区域的设置，甚至有些设置只单独针对某一个控件，这些不同的样式通过一种规则有序的组合在一起形成了多样的页面，这个规则就是CSS3中选择器优先级。

正如前文所述，修饰一个元素通常有五种方式，分别为标签选择器、类型选择器、属性选择器、ID选择器，以及行内样式，同时各个选择器的优先级又由选择器的特殊性来决定，即针对性越强的选择器其优先级就会越大，而针对性相对一般的选择器则优先级就会越低。因此根据这一原则把标签选择器、类型选择器、属性选择器、ID选择器，以及行内样式这五种样式修饰方法的优先级分别定义为1、10、10、100、1000，如图5-18所示。



图5-18 CSS中选择器的优先级

从上图中可以知道行内样式的优先级最高，因为它的针对性最强，即只针对某一个标签进行设置，而标签选择器则标识只要是该页面中的指定标签都会获得其样式，具有最广泛的普适性，因此优先级最低。而在前端开发中浏览器引擎会计算样式的优先级，计算的方法就是样式所包含的对应优先级数值进行相加，最后得到某个样式的优先级，如果该样式计算得到的优先级比另外一个样式的优先级高，那么高优先级的样式就覆盖低优先级的样式。

例如下面的CSS代码，可以得到ID选择器#header和#left的优先级为100，标签选择器ul、li、a的优先级为1，类选择器.first的优先级为10，所以最后计算得到总的优先级值为213=100+100+1+1+10+1。

1 #header #left ul li.first a {...}

除了上面五种选择器外，CSS中的选择器还有通用选择器（\*）、子选择器（>）、相邻兄弟选择器（+）、兄弟选择器（～），以及伪选择器，但是这些选择器都不在优先级的计算范围内，因为这些选择器的优先级值都为0。此外CSS样式设定中还有一个关键字可以破坏这些预先设定的规则，该关键字就是“!important”，被该关键字修饰的样式不论本身计算得到优先级是多少，该样式的优先级都会变成最高。因此，综上所述可以得到在CSS编程时关于优先级的五条基本原则，具体如下：

原则一：当选择器的优先级数值越大，则其优先级就越高。所有的选择器都有其优先级的值，即使像伪选择器这样的选择也有其优先级的值，只不过为0，因此在参与运算时不影响最终得到的值。代码如图5-19所示，在该例子中虽然使用了关系选择器，但是没有影响优先级，不过因为数字2的代码要比数字1的代码后定义，因此在优先级相同的情况下数字2的代码覆盖了数字1的代码，所以最终的结果呈现红色。



图5-19 关系选择器不影响优先级

原则二：写在后面的样式优先级要高于写在前面的优先级。在标签的样式列表中，如果这些样式的优先级相等，那么越靠近过末尾的其优先级就越高。代码如图5-20所示，在该例子中样式的优先级相同，但是level-4出现的位置要比level-3靠后，因此最终level-4的优先级要比level-3高，所以最终的结果呈现蓝色。



图5-20 靠后的样式优先级高

原则三：每个浏览器引擎都会有自己的样式，但是开发人员所设置的样式优先级要高于浏览器引擎所设置的样式优先级。

原则四：因为CSS是层叠样式，因此子标签会继承父标签的样式，但是给子标签重新设置的样式优先级要会高于从父标签继承的优先级。代码如图5-21所示，在该例子中虽然第二个样式的优先级没有第一个样式的优先级高，同时这两个样式都作用于span这个标签，但是span获取第一个样式的设置是通过继承的方式，而获取第二个样式的设置则时是通过重新设置，因此重新设置的优先级要比继承高，所以最终的结果呈现红色。



图5-21 继承的样式优先级较低

原则五：“!important”会破坏原有的优先级，并且直接把样式的优先级变为最高。代码如图5-22所示，在该例子中虽然两个color的优先级相同，但是因为第一个color有一个“!important”的修饰，因此其优先级变为最高，所以最终的结果呈现蓝色。



图5-22 !important获取最高优先级

## 5.3 面向对象编程思想在CSS3中的应用

在上节中介绍了许多种CSS的样式选择器，如标签选择器、类别选择器、ID选择器等，同时还讲述了以及各种选择器在使用时优先级的计算。在掌握这些内容后，开发人员又会面临新的问题，即在实际开发中可能一个问题的解决方法有许多种，那么该如何选择一个比较好的解决方案就是本节要解决的问题。

目前常用的编程语言中除了C语言之外，几乎所有的语言都会提到一个概念，即面向对象，甚至在某购物网站还有关于运用面向对象的思想来编写C语言的书籍，因此可以说面向对象的概念被广泛并深刻的应用在各个领域，并由此产生了一系列的编程方法和编程思想。面向对象概念的流行是因为这种思维模式符合人们在现实生活中对客观事物的理解，能够更加准确的描述问题。

### 5.3.1 CSS3中“类”概念的应用

在前端设计中要运用现有的CSS知识去实现某一个单独的页面其实并不是很复杂，但要实现一个可维护、可扩展的，并且风格统一的CSS系统就实属不易，好在2008年由Nicole Sullivan首次提出了OOCSS的概念改变这一切，今天的我们就可以站在这些巨人的肩膀上来避免重复造轮子的问题。OOCSS即面向对象的CSS，其核心思想主要是减少CSS对HTML结构的依赖，并且增加CSS代码的重用性，从而提高开发效率。

在面向对象的概念中“类”这个概念非常的重要，很多如封装性、继承性、依赖倒置、多用组合少用继承的概念都源于“类”，因此要借鉴面向对象的概念来实现OOCSS时，首先就需要在CSS开发中引入“类”概念。所谓类，就是把页面进行模块化，如图5-23所示。在图中可以看出整个页面按照其结构可以分为两个模块，分别是模块1所示的左边一张大图，右边两张小图构成的部分，以及模块2所示的由四张小图组成的部分，因此在设计代码时可以把模块1和模块2分别设计其代码，并且在今后的页面中进行复用。



图5-23 页面模块化

但仔细看就可以发现，其实模块1和模块2之间任然有相同的部分，即在这两个模块的上面都有一个标题栏，这个标题栏包含了Logo、文字，以及二维码，那么如果我们基于图5-23所示的结构进行代码设计时就会在每个模块中把标题栏都去重复的实现一遍，这样不仅造成代码的冗余，而且在后期如果需要修改标题样式时也需要去修改每个模块中的标题样式，无形中增加了后期代码维护的成本，因此在进行模块设计时可以把标题栏部分单独剥离出来，形成一个独立的模块，如图5-24所示。通过模块的剥离不仅减少了代码的冗余，同时在后期修改时也只需要修改该模块的代码，而不用到每个模块1、模块2等中一个个的修改，从而减少了代码维护的成本。



图5-24 剥离标题栏

通过上面的分析可以得到页面设计的第一个原则，即在页面设计时尽量要保证页面中各个模块之间不含有相同的成分，如果有相同的成分，那么就将它进行剥离，形成一个独立的模块。其实图5-24中的拆分还没有完善，因为仔细观察模块2中会发现虽然模块2的结构与模块1的结构不同，但是模块2中每个方格的样式却和模块1右边的方格样式是相同的，所以在进行模块划分时可以把模块1和模块2中的小方格在进行剥离，形成模块4，如图5-25所示。

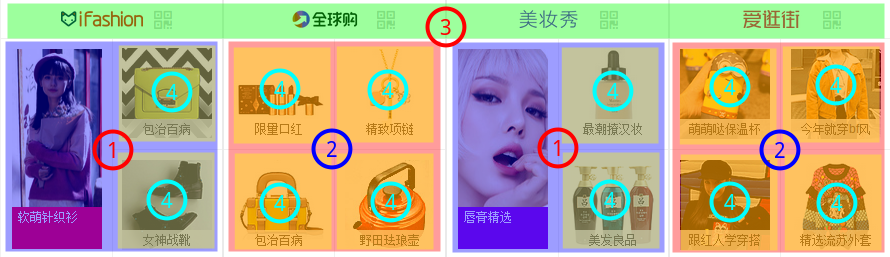


图5-25 完整的模块划分

在实际开发中模块的拆分通常会按照模块尽可能的简单，以提高模块的重用性，但是随着页面越来越复杂，拆解页面后产生的小模块就会越来越多，这样也无形中增加了另一种代码维护的难度，因此作为开发人员需要在模块的数量上和结构的单一性上做权衡，所以由此可以得带页面设计的第二个原则，就是在保证模块数量尽可能少的前提下，尽量做到模块的简单，以提高重用性。这两个原则就是面向对象程序设计中的“单一职责”。通常情况下一个类的功能越是简单，那么这个类的稳定性就越高，同时能够被重用的机会就越大，而如果一个类的功能太多，那么类的重用性就会相对较低。

### 5.3.2 “命名空间”在CSS3中的应用

在前端页面的开发中很多时候都会为一个CSS中选择器的名字而烦恼，因为在当前协作开发较为普遍的环境下很容易出现命名相同造成冲突的问题。通常来说CSS采用有意义的英文进行命名，例如页面头部的CSS通常采用head命名，页面内容的CSS通常采用main或者content进行命名，而页面尾部的CSS则采用foot进行命名，虽然这些命名都没有强制的要求，但是从目前约定俗成的规范来说这样的命名是最为合适的。

此外，一般的CSS选择器的命名上通常有两种命名方法，一种叫做划线命名法，类似于xxx-xxx这样的命名，另一种则是驼峰命名法，即命名的第一个单词全部小写，从第二个单词开始首字母大写，类似于redBorder这样的命名。采用这两种命名方法可以清楚地将单词进行分割，提高命名的可读性，但是在实际开发时如果只是单纯的运用其中的一种命名方法不仅影响代码的可读性，还不利于后期的维护，因此通用的做法是把划线命名法和驼峰命名法进行融合，同时应用于项目中，如下面的代码：

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>CSS的命名空间</title>

6 <style type="text/css">

7 .jobList {...}

8 .jobList .last {...}

9 </style>

10 </head>

11 <body>

12 <ul class="jobList">

13 <li>班主任</li>

14 <li>辅导员</li>

15 <li>专业主任</li>

16 <li class="last">院长</li>

17 </ul>

18 </body>

19 </html>

在上面的代码中通过类别选择器".jobList"来设定无序列表的样式，同时为最后一个列表项单独添加了一个类别选择器“.last”，并通过子选择器“.jobList .last”来设定最后一项的样式，采用这种方法可以完成系统的要求，但在协作开发中这种方法会因为命名过于简单，且是常用单词，造成不同开发人员之间的样式冲突，为后期的代码维护和扩展造成严重的问题，并且这一类问题在解决时也异常的困难。那么此时如果把".last"改为".jobListLast"就可以很好的降低命名的冲突，可是从CSS的结构上看".last"是隶属于".jobList"，而".jobListLast"这样的命名并没有体现出这一点。

在面向对象的概念中有一个很重要的概念——“封装性”，通过“封装性”可以把编程的对象分为“公有”和“私有”两个部分，“公有”的部分可以看做是系统对外交流的接口，而“私有”的部分则可以看做是系统内部的，并不对外公开的部分。通常来说在面向对象的程序设计中，开发人员会尽量的减少对外的借口，做到必要且充分即可，而把其他的功能都变为“私有”，采用这种设计方法可以使得系统内部不论怎么修改都不会对外界的其他部分造成影响，从而提高了整个系统的稳定性。

虽然CSS没有“封装性”这个概念，但可以把这个概念从面向对象的概念中衍生到CSS中，即要求CSS选择器的命名中能够体现“封装性”，根据这一要求可以把划线命名法和驼峰命名的特点相结合，并应用到".jobListLast"中，最终该选择器可以命名为".jobList-last"，在这个命名中“.jobList”可以看做是一个“公有”的部分，而".jobList-last"中的“last”则可以看做是在“.jobList”这个命名空间下的私有的部分。这样的命名不仅有较高的可读性，而且还能体现选择器和选择器之间的隶属关系。如果采用这种方法还会产生冲突，那么就可以为小组中的每个开发人员起一个代号，然后在选择器的命名时在最前面添加上这个代码，从而把选择器的冲突降到最低。

### 5.3.3 “多用组合，少用继承”原则在CSS3中的应用

前文讲过CSS又称“层叠样式”，在父标签和子标签之间具有继承的特性，同时在类别选择器中可以通过组合的方式为某一个标签添加多个样式，而在面向对象的程序设计中也同样有这两个概念，因此很多面向对象的程序设计中会出现的问题也会出现在CSS中。

所谓继承，就是把多个具有相同特性的东西进行抽象并形成一个父类，在父类中包含各个子类所共有的功能，而子类中除了包含父类的功能外，还包含其自身所特有的功能，通过继承可以很方便的为父类拓展新的功能，但是继承也存在一些问题，就是基于父类的任何微小变化都需要重新定义一个类，很容易引起“类爆炸”——即类数量的大幅增加，从而产生一堆不同粒度的子类，增加了系统维护的难度。而组合的概念则与继承不同，组合是将一个复杂的类拆分为稳定的部分和容易变化的部分，并把容易变化的部分单独拆分出来变为几个功能单一的独立类，而把稳定的部分变为主类，然后通过主类与不同独立类之间的组合来实现标签的不同功能。采用组合的方式可以大大减少类的数量，可以帮助程序更加的健壮，也更有弹性，下面通过两个代码的比较来进行说明，具体代码如下：

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>方式一：采用继承</title>

6 <style type="text/css">

7 .classJob, .gradeJob, .specialtyJob { border-width: 5px; }

8 .classJob li, .gradeJob li, .specialtyJob li { font-size: 20px; }

9 .gradeJob li { color: red; }

10 .specialtyJob li { color: blue; }

11 </style>

12 </head>

13 <body>

14 <ul class="classJob">

15 <li>班主任</li>

16 </ul>

17 <ul class="gradeJob">

18 <li>辅导员</li>

19 </ul>

20 <ul class="specialtyJob">

21 <li>专业主任</li>

22 </ul>

23 </body>

24 </html>

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>方式一：采用组合</title>

6 <style type="text/css">

7 .job { border-width: 5px; }

8 .f20 { font-size: 20px; }

9 .red { color: red; }

10 .blue { color: blue; }

11 </style>

12 </head>

13 <body>

14 <ul class="job">

15 <li class="f20">班主任</li>

16 </ul>

17 <ul class="job">

18 <li class="f20 red">辅导员</li>

19 </ul>

20 <ul class="job">

21 <li class="f20 blue">专业主任</li>

22 </ul>

23 </body>

24 </html>

上面两段代码相同的是都可以实现同样的想过，不同的是实现的方法不同，第一个代码采用的是继承的方法来实现，而第二个代码则采用组合的方法来实现。此时读者可以思考，如果这个时候我在body里面再添加一个新的ul标签会得到什么样的结果？对于使用继承方式来实现的程序需要在样式表中添加一个新的class，并为这个标签再添加不同的样式，包括该标签的子标签，而采用组合方式实现的程序只需要在新的ul标签的class属性中添加现有的选择器就可以实现，大大减少了类别选择器的数量，同时也使得程序更具有灵活性，由此可以得出在设计CSS时的第三个原则就是“多用组合，少用继承”，具体代码如下：

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>方式一：采用继承</title>

6 <style type="text/css">

7 .classJob, .gradeJob, .specialtyJob, .collegeJob { border-width: 5px; }

8 .classJob li, .gradeJob li, .specialtyJob li, .collegeJob li { font-size: 20px; }

9 .gradeJob li { color: red; }

10 .specialtyJob li { color: blue; }

11 .collegeJob li { color: blue; }

12 </style>

13 </head>

14 <body>

15 <ul class="classJob">

16 <li>班主任</li>

17 </ul>

18 <ul class="gradeJob">

19 <li>辅导员</li>

20 </ul>

21 <ul class="specialtyJob">

22 <li>专业主任</li>

23 </ul>

24 <ul class="collegeJob">

25 <li>院长</li>

26 </ul>

27 </body>

28 </html>

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <title>方式一：采用组合</title>

6 <style type="text/css">

7 .job { border-width: 5px; }

8 .f20 { font-size: 20px; }

9 .red { color: red; }

10 .blue { color: blue; }

11 </style>

12 </head>

13 <body>

14 <ul class="job">

15 <li class="f20">班主任</li>

16 </ul>

17 <ul class="job">

18 <li class="f20 red">辅导员</li>

19 </ul>

20 <ul class="job">

21 <li class="f20 blue">专业主任</li>

22 </ul>

23 <ul class="job">

24 <li class="f20 blue">院长</li>

25 </ul>

26 </body>

27 </html>

## 5.4 面向对象编程技术在JavaScript中的应用

Web发展之初，其主要的使用者都集中科学家身上，并且把Web作为科学资料的共享与传阅的一种方式，后来随着传阅的文件越来越多时就加入了超链接，从实现在一个页面上的显示更多的内容，再后来传阅资料所需要的形式越来越丰富，单纯的文字已经不能满足其要求，没过多久就在Web中加入了图片显示，以丰富其内容，但是随着Web系统的庞大和复杂，这些科学家和网站管理员就希望能够在网站上完成更多的工作，而不只是显示文字和图片，从而减少和减轻服务器交互的次数与压力，因此就诞生了后来的ECMAScript，而JavaScript则是ECMAScript在过去特定环境下所诞生的一个称呼，并且被延续至今，这里要特别说明的是Java和JavaScript虽然在称呼上类似，实则两个一点关系也没有，好在随着ECMAScript 6的发布，开发人员也开始使用ES6这样标准的称呼，而不再是JavaScript。在本书中为了兼容一些常用的第三方库，还是使用JavaScript这样称呼。

随着技术的发展，目前JavaScript的应用领域，已经不仅仅是在Web端，而是与HTML5、CSS3相结合，广泛的应用于服务器端、桌面应用、移动APP、游戏开发等众多领域，甚至国内外还有些厂商开发出面向物联网，并利用JavaScript实现智能控制的设备，因此可以说目前JavaScript无处不在。同时JavaScript又是一个入门非常简单的语言，只需要几行代码就可以立马看到效果，所以也特别适合初学者掌握，目前关于JavaScript基本语法的书非常多，但是很少涉及面向对象在JavaScript中的应用，而这项技术才是目前最热门，也是需求最旺盛的技术。因为面向对象的基本概念已经在第四章详细参数，因此本章讲重点讲述如何运用面向对象技术在JavaScript中的应用。

### 5.4.1 JavaScript的运行环境

要运行和编写JavaScript的程序非常简单，只需要一个浏览器就可以完成，而记事本则是编写JavaScript代码的最简单工具。当然在实践中，本书还是推荐使用VSCode作为JavaScript代码的编辑软件，这样可以为读者节约许多的时间，也提高代码编写的体验。要更好的了解和掌握JavaScript的代码，建议读者使用最新的FireFox、Chrome，以及Win10的Edge，这样可以把最新的JavaScript特性进行实践。实践JavaScript最简单的方法有三步：

1、第一步：创建一个HTML文件，并编写相应的HTML代码。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7 <title>JS运行环境</title>

8 </head>

9 <body>

10

11 </body>

12 </html>

2、第二步：在HTML文件的目录下创建文件夹JS，并创建JavaScript文件，命名为hello.js，并在其中键入如下代码。

1 console.log('Hello JavaScript');

3、第三步：回到HTML文件中在head标签的结束标签上面添加一段代码，实现把JS文件引入到HTML文件中的目的，具体代码如下。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7 <title>JS运行环境</title>

8

9 <!--引入JS文件-->

10 <script src="js/hello.js"></script>

11 </head>

12 <body>

13

14 </body>

15 </html>

4、第四步：使用浏览器打开HTML文件，并在键盘点击F12，进入开发者工具就可以在控制台（英文：Console），当看到“Hello JavaScript”就表示最简的JavaScript环境搭建完毕，以后只需要修改这个js就可以实践多种JavaScript代码，并且这个js文件夹和hello.js的名字也可以根据需要进行修改。

### 5.4.2 JavaScript中的数据类型与函数

在JavaScript中要定义一个变量，方法非常简单，只需要使用var i = 0即可，其中var是关键字，i则表示变量名，并且JavaScript是区分大写的，所以大写的i和小写的I表示的是两个不同的变量。到这里，读者也不难发现在JavaScript中，定义变量时也没有为其设定数据类型，因为它和PHP一样都是弱类型的语言，不需要显式的设定类型，而是由系统进行判断，同时它也和PHP一样采用赋值后才申请空间的方式，以提高内存使用的效率，如果读者在声明变量后没有进行赋值，那就用console.log()函数进行打印时就会在控制台打印“undefined”，因此建议读者在JavaScript中创建好变量后一定要进行初始化，即使赋值0也可以。

虽然在JavaScript中不需要为变量设置数据类型，但是其内部还是存在数据类型，主要包含两大类，分别为基本数据类型和复合数据类型，其中基本数据类型包括数值型（整型、浮点型）、字符串型、布尔型、undefined、null，而复合数据类型则包括数组和对象。JavaScript中可以通过关键字typeof来获取数据的类型，具体代码如下：

1 var i = 2.3;

2 console.log(typeof i);

3

4 var i = "Hello JavaScript";

5 console.log(typeof i);

6

7 var i = true;

8 console.log(typeof i);

执行结果：

1 number

2 string

3 boolean

这里要特别说明的是undefined和null类型，其中undefined类型是由JavaScript系统给予赋值，而null则是由开发人员进行赋值得到，虽然两者都表示空，但是在使用却有很大的区别，特别是在数值计算时，undefined和null转换的结果完全不同，具体代码如下：

1 // undefined参与数值计算

2 console.log(2.3 + undefined);

3

4 // null参与数值计算

5 console.log(2.3 + null);

6

7 // undefined参与字符串计算

8 console.log("Hello World " + undefined);

9

10 // null参与字符串计算

11 console.log("Hello World " + null);

12

13 // undefined参与布尔计算

14 console.log(!!undefined);

15

16 // null参与布尔计算

17 console.log(!!null);

执行结果：

1 NaN

2 2.3

3 Hello World undefined

4 Hello World null

5 false

6 false

从上面结果不难看出，undefined和null参与的计算分为三类，分别为数值计算、字符串运算和布尔计算。当undefined参与数值计算时会被转化为NaN，而当null参与数值计算时则会被转化为0。当undefined和null参与字符串计算时都会被直接转化为相应的字符串。当undefined和null参与布尔计算时都会被转化为false。

上面讲的都是JavaScript中的基本数据类型，而除了基本数据类型之外，JavaScript还有两个复合数据类型，即数组和对象，这里重点讲述数组，对象会在下一节中详细讲解。在JavaScript中数组的定义和基本数据类型的定义非常显示，但却有不同，数组定义是需要把一组数据用方括号围住，即var i = [1, 2, 3]。要读取数组内的数据则从通过索引来获取，而数组的索引起始值为0，而不是1，即i[0] = 1，i[1] = 2，i[2] = 3，以此类推。JavaScript因为是弱类型的语言，因此并不要求在数组的数据类型必须保持一直，所以有时会在数组中放各种类型的数据，如var i = [1, “Hello World”, null]，而更多的应用则是多维数组的应用和嵌套，特别是在网络通信中，从服务器上收到的HTTP数据，基本上都是采用数组嵌套的方式进行传输，如var i = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]，对于这类多维数组的读取和一维数组的读取方式类似，如i[1][2] = 6，其中第一个数字“1”表示读取二维数组中的第一个数组，因为数组的索引从零开始，因此实际上就是第二个数组，即[4, 5, 6]，而第二个数字“2”表示读取第二个数组中的第二个元素，也是因为数组的索引从零开始，因此实际上就是第三个元素，最后得到的就是数字6。

编写JavaScript的目的在于实现某些功能，而变量只是实现这些功能的一个重要部分，函数才是这些功能真正的载体，因此JavaScript中函数的使用必不可少。JavaScript的函数定义和PHP中函数的定义极为类似，需要四个部分组成，分别为关键字function、函数名、函数参数列表、函数体，其中函数体中除了正常的函数实现外，还有一个返回值，并且当开发人员没有返回值时，该返回值会隐式的返回undefined，而当开发人员有返回值时，则会把undefined替换为开发人员的返回值进行返回，具体代码如下：

1 function 函数名(函数参数列表) {

2 函数体

3

4 return 返回值 // 默认情况下返回undefined

5 }

因为JavaScript是一种弱类型的语言，因此它的变量可以的任意类型，甚至包括函数。很多刚接触的JavaScript的读者看源码的时候，往往会觉得JavaScript的代码非常奇怪，甚至觉得不符合规范，这是因为弱类型语言的自由性，所以要理解JavaScript，首先就要理解“函数实际上也是一种数据”这个概念，也就是说在JavaScript中可以把函数赋值给某一个变量，只有理解了这个概念才能为后续的内容打下坚实的基础。从形式上来说，函数变量有两种形式，一种是带命名函数表达式的函数标记法，还有一种则是不带命名函数表达式的函数标记法（也叫匿名函数），具体代码如下：

1 // 不带命名函数表达式的函数标记法

2 var 变量名 = function (函数参数列表) {

3 函数体

4 }

5

6 // 带命名函数表达式的函数标记法

7 var 变量名 = function 函数名(函数参数列表) {

8 函数体

9 }

这两种形式中最常用的是不带命名函数表达式的函数标记法，而另外一种则很少用，因此接下来主要讲解不带命名函数表达式的函数标记法具体应用范围和方法。不论开发人员采用哪一种方法都可以使用操作符typeof来获取变量的类型，读者实验后会发现，其返回值为“function”，也就是说该变量的类型为函数，所以可以知道所谓“函数实际上也是一种数据”就是指该变量的数据类型非常特殊，是一个函数类型，并且包含了一段可执行的完整代码，并且可以像调用函数一样的方法调用这个函数变量，具体代码如下：

1 var func = function (arg1, arg2) {

2 return arg1 + arg2;

3 }

4

5 var add = func;

6 console.log(add(1, 2));

执行结果：

1 3

从上面的程序不难看出，通过创建匿名函数得到函数变量func，并且变量func可以像普通变量一样进行赋值操作，上面例子中把变量func赋值给变量add，这是变量add也变成了函数变量，并且可以完成func的相关操作。匿名函数使用最广泛和最有意义的方式是被用于作为函数的参数传入其他函数，并且在其他函数中完成所需要的功能，下面将通过一个例子进行讲解，即实现把所有传入的参数乘以3，并且把乘3后的结果加1，具体代码如下：

1 var addOne = function (arg) {

2 return arg + 1;

3 }

4

5 function func(args, callback) {

6 var a = [];

7

8 for (var i = 0; i < args.length; i++) {

9 a[i] = callback(args[i] \* 3);

10 }

11

12 return a;

13 }

14

15 var arr = [2, 3, 4, 5];

16

17 console.log(func(arr, addOne));

执行结果：

1 Array(4)

2 0: 7

3 1: 10

4 2: 13

5 3: 16

6 length: 4

7 \_\_proto\_\_: Array(0)

从上面的代码不难看出首先定义了一个匿名函数addOne来实现参数加1并返回的功能，然后创建一个函数func，该函数中通过循环实现所有参数乘以3，并且把addOne作为参数或者回调函数的方式传入到函数func中，并把参数乘3后的结果直接传给回调函数callback，并把回调函数的值赋值给新的数组，并最终返回。虽然这个例子没有实际的意义，但是却能清楚的表达匿名函数的作用。通过匿名函数的使用可以减少变量名的创建，并且也可以实现代码各项责任的区分，实现了代码的复用，也提高了整个代码的效率。

### 5.4.3 JavaScript中的基本对象类型

在上一节中讲到在JavaScript中可以通过索引的方式来访问数组数据，然而JavaScript中的对象和数组具有相似的特性，甚至可以把数组看做是对象的一种特殊表现形式，而唯一的不同点在于数组只能通过1...N的索引来访问数据成员，而对象则通过自定义的键值进行访问，并且这些自定义的键值可以使用语义更加明确的英文单词来进行表达，比如name、age、job等，下面就通过一个实例来说明JavaScript中对象的编写和组成。

1 var person = {

2 name: "李雷",

3 age: 50,

4 job: "engineer"

5 }

6

7 // 调用成员变量

8 console.log("Person name = " + person.name);

9 console.log("Person age = " + person.age);

10 console.log("Person job = " + person.job);

执行结果：

1 Person name = 李雷

2 Person age = 50

3 Person job = engineer

在上面的例子中变量person表示的就是一个对象名称。与数组不同的是，对象使用花括号来包裹其内容，而不是使用方括号。在该例中花括号里有三组该对象的属性，或者也可以叫做成员变量，这些成员变量不需要使用关键字“var”来定义，只需要编写一个有意义的名称即可，并且成员变量之间使用逗号作为分割。这些属性的初始化也不是使用“=”好来完成，而是使用“键值对”的方式来完成，即name: "李雷"。如果要获取成员变量的值，那么只需要使用“对象名.成员变量名”的方式即可。

既然有个成员变量，那么对象就应该还能够添加成员函数，在JavaScript中添加成员函数的方法也是采用键值对的方式，只不过现在值是一个function，而不是数值，示例如下：

1 var person = {

2 name: "李雷",

3 age: 50,

4 job: "engineer",

5 sayHello: function() {

6 console.log("你们好，我的名字叫李雷");

7 }

8 }

9

10 // 调用成员函数

11 person.sayHello();

执行结果：

1 你们好，我的名字叫李雷

在创建完成类的成员函数后，经常需要调用各种成员变量，并且通过计算来获取相应的值，那么此时就特别需要注意的是，在成员函数中调用成员变量，必须在成员变量之前加上关键字this，这个this表示就是当前对象，和上一章中讲的this是相同的概念，接下来把上面例子中sayHello函数“李雷”改为使用成员变量name，示例如下：

1 var person = {

2 name: "李雷",

3 age: 50,

4 job: "engineer",

5 sayHello: function() {

6 console.log("你们好，我的名字叫" + this.name);

7 }

8 }

9

10 // 调用成员函数

11 person.sayHello();

执行结果：

1 你们好，我的名字叫李雷

从上面例子不难看出，在JavaScript中要创建一个对象实际上非常方便，但是也存着一些问题。阅读过第四章的读者可能已经有了疑问，在上面的案例中person只是单一的一个对象，那么如果要创建与person对象成员变量和成员函数相同，而成员变量的值不同的对象，那么就需要创建许多个类似的person对象，这样不仅需要编写许多的代码，而且不利于后期的维护。在PHP中对象的产生由类来完成，那么在JavaScript中有没有类的概念的，答案是有的，但是类的写法却与PHP有很大差别，这个差别导致很多刚从Java、C++、PHP等语言转换到JavaScript的工程师在最开始时会措手不及，下面就通过一个示例进行说明，示例如下：

1 function Person(name, age, job) {

2 this.name = name;

3 this.age = age;

4 this.job = job;

5 this.sayHello = function() {

6 console.log("你们好，我是一个对象，我的名字叫" + this.name);

7 }

8 }

9

10 // 调用成员函数

11 var classPerson = new Person("李雷", 50, "engineer");

12 classPerson.sayHello();

执行结果：

1 你们好，我是一个对象，我的名字叫李雷

从上面的例子可以看出，JavaScript并不使用class这个关键字，而是通过构造函数来进行对象的创建，例子中定义的function Person()就是一个类的构造函数，虽然这个构造函数的语法和普通函数的语法完全相同，但是其内部还是有很大的区别。在该例中，创建了一个带参数的构造函数，这样的做法和PHP中构造函数的写法非常类似，当然如果需要也可以使用不带参数的构造函数，随后在构造函数的函数体中创建了类的成员变量和成员函数，并且在这些成员变量和成员函数前面加上关键字this，表示这些成员变量和成员函数与被创建的对象进行绑定，随后只需要通过new关键字就可以进行对象的创建，当对象创建完成后就可以像普通的对象一个使用。这样的做法好处是可以通过构造函数，以及传入的参数来创建各种不同的对象，从而减少了代码的重复，提高了可维护性。

到这里读者应该对于JavaScript中对象的创建和使用非常清楚了，类比PHP可以发现，在PHP中实际上存在了大量的内建的方法和对象，这些方法和对象可以帮助开发人员去构建更为复杂的应用，那么在JavaScript有没有类似的内置对象呢？答案是肯定的。JavaScript中内置的对象大约可以分为三类：

* 数据对象：所谓数据对象就是Array（数组）、Boolean（布尔型）、Number（数值型）、String（字符串型）、Object（对象型），这些数据对象都提供一些常用的函数，例如数组的增、删、改、查等。
* 工具对象：所谓工具对象就是指包括Math、Date、RegExp等提供相关工具函数的类，例如通过Math获取随机数、通过Date获取当前时间、通过RegExp构建一个正则表达式等。
* 错误对象：所谓错误对象就是指当程序发生异常时抛出的错误提示，这就包括算数异常、类型异常等。

这里要特别说明的是，对于数据对象、工具对象、错误对象这三种对象来说，其内容非常的，但是要全部掌握却也是件很费力的事情，因此下来主要挑选几个代表性的对象来引导读者掌握其方法，不会把所有内容都呈现出来，如果需要查看所有对象的使用，可以到W3CSchool中去查看。

1、Object对象：前面讲了如何去创建一个类，实际上在每个JavaScript对象中还隐藏了一个父类，即Object。Object是所有JavaScript对象的父类，不论对象继承了多少个父类，但是最终的父类一定是Object，也就是说可以通过Object创建一个空对象，然后利用JavaScript对象属性可以任意修改的特性来创建一个自定义的对象，示例如下：

1 var objectPerson = new Object();

2 objectPerson.name = "李雷";

3 objectPerson.age = 50;

4 objectPerson.job = "engineer";

5 objectPerson.sayHello = function() {

6 console.log("你们好，我的名字叫Object李雷");

7 };

8

9 // 调用成员函数

10 objectPerson.sayHello();

执行结果：

1 你们好，我的名字叫Object李雷

2、Array对象：Array对象就是前面讲的数组，即var a = []和var a = new Array()都是创建数组，具有相同的效果。而在数组循环获取数组长度的length则是Array对象的一个属性，此外Array还提供一些常用的函数来方便数组的操作，如表5-4所示。

表5-4 常用的Array成员函数

|  |  |
| --- | --- |
| 函数名 | 说明 |
| push(newelement1, ...., newelementX) | 向数组的末尾添加一个或多个元素，并返回新的长度。 |
| pop() | 删除并返回数组的最后一个元素。 |
| sort(sortby) | 对数组的元素进行排序。如果调用该方法时没有参数，将按字母顺序对数组中的元素进行排序。 |
| join(separator) | 把数组中的所有元素放入一个字符串，元素通过指定的分隔符进行分隔的。 |
| splice(index, howmany, item1,....., itemX) | 从数组中添加/删除项目，然后返回被删除的项目。 |

3、String对象：该对象是对象型的字符串，在String对象内部把字符串变成一个字符数组进行存储，因此可以通过索引的方式来获取字符串中每一个字符。但是在实际使用时，开发人员一般直接对字符串进行操作，而不会显式的去创建String对象，因为在JavaScript内部，当开发人员使用String对象的成员函数来操作字符串时，会在后台创建一个String对象，因此看上去就相当于在直接操作字符串，具体实例如下：

1 var strTmp = "Hello String";

2 var strObj = new String("Hello String Obj");

3

4 // 在后台把strTmp转化为String对象后在进行操作

5 console.log(strTmp[0]);

6 // strObj中字符串已经转化为字符数组，因此可以通过索引操作

7 console.log(strObj[0]);

执行结果：

1 H

2 H

除此之外，String对象还提供了很多常用的字符串操作函数，如表5-5所示。

表5-5 常用的String成员函数

|  |  |
| --- | --- |
| 函数名 | 说明 |
| toLowerCase()/toUpperCase() | 把字符串转化为小写/大写。 |
| charAt(index) | 返回指定位置的字符。 |
| indexOf(searchvalue, fromindex) | 返回某个指定的字符串值在字符串中首次出现的位置。 |
| lastIndexOf(searchvalue, fromindex) | 返回一个指定的字符串值最后出现的位置，在一个字符串中的指定位置从后向前搜索。 |
| replace(regexp/substr, replacement) | 在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。 |
| slice(start, end) | 提取字符串的某个部分，并以新的字符串返回被提取的部分。 |
| split(separator, howmany) | 把一个字符串分割成字符串数组。 |
| substring(start,stop) | 提取字符串中介于两个指定下标之间的字符。 |

4、Math对象：该对象为开发人员提供了许多数学计算方法，因为该对象的创建与String和Array等都不同，它没有采用构造函数进行创建，因此在使用时不需要通过关键字new来进行创建，而只需要使用Math.XXX()来调用它的成员函数和成员变量。这里要特别说明的是Math中的成员变量都是不可修改的常量，例如Math.PI（圆周率中的π值）等。除此之外，Math对象提供的一些常用的数学操作函数，如表5-6所示。

表5-6 常用的Math成员函数

|  |  |
| --- | --- |
| 函数名 | 说明 |
| abs(x) | 返回数的绝对值。 |
| acos(x) | 返回一个数的反余弦。 |
| asin(x) | 返回数的反正弦值。 |
| atan(x) | 以介于 -PI/2 与 PI/2 弧度之间的数值来返回 x 的反正切值。 |
| atan2(y,x) | 返回从 x 轴到点 (x,y) 的角度（介于 -PI/2 与 PI/2 弧度之间）。 |
| cos(x) | 返回数的余弦。 |
| max(x,y) | 返回 x 和 y 中的最高值。 |
| min(x,y) | 返回 x 和 y 中的最低值。 |
| pow(x,y) | 返回 x 的 y 次幂。 |
| random() | 返回 0 ~ 1 之间的随机数。 |
| round(x) | 把数四舍五入为最接近的整数。 |
| sin(x) | 回数的正弦。 |
| sqrt(x) | 返回数的平方根。 |
| tan(x) | 返回角的正切。 |

4、Date对象：该对象为开发人员提供了时间操作的对象，并且可以通过传递四种不同类型的参数来创建该对象，这四种不同的参数分别为：无参数（表示当天时间）、字符串（获取指定时间的对象）、把时间参数分开传递（获取指定时间的对象）、时间戳（timestamp，获取指定时间对象），具体实例如下：

1 // 得到当前时间

2 var currDate = new Date();

3 console.log(currDate);

4 // 字符串解析后得到制定时间

5 var strDate = new Date("2017 08 15 13:17");

6 console.log(strDate);

7 // 传递时间参数后得到制定时间

8 var paramsDate = new Date(2017, 07, 15, 13, 17, 00, 50);

9 console.log(paramsDate);

这里要特别说明的是，当使用时间参数创建Date对象时，月份是从0开始计算，而不是1，因此1月应该传递0，而12月应该传递11，所以在上面的例子new Date(2017, 07, 15, 13, 17, 00, 50)中“07”表示的是八月。此外Date对象还提供了一些很常用的函数，比如getXXX()、setXXX()函数来获取年、月、日、小时、分钟、秒、毫秒，还有Date.parse()函数可以把时间字符串转化为时间戳，以及Date.now()函数可以获取当前时间的时间戳。

5、Error对象：该对象用于程序发生错误时告诉系统发生的是什么错误，但是往往遇到这样的错误时，系统就会停止运行，但是一个好的程序应该是在遇到错误时能够自己解决，所以在JavaScript中提供try...catch...finally...的语法来捕获这些异常，并且对这些异常进行处理。其中try后面跟的是可能产生异常的代码，catch后面则跟的是如果捕获到异常时的处理内容，finally是可选内容，当写了finally时，则表示不论是否捕获异常都会在后面执行finally中的语句。异常的种类很多，这些错误都可以在浏览器的控制台中看到，并且这些异常都是Error对象的子类，Error对象中提供两个成员变量，分别为name和message，用于表示异常的名称和异常的内容，具体代码如下：

1 try {

2 // 可能产生异常的代码

3 } catch(e) {

4 // 捕获到异常时的处理内容

5 console.log(e.name);

6 console.log(e.message);

7 } finally {

8 // 不论是否捕获异常都会执行的语句

9 }

### 5.4.4 JavaScript中的继承与扩展

要深入理解JavaScript中的继承和扩展的工作原理，首先就是需要理解JavaScript对象中的一个重要属性——prototype（原型）。在讲原型之前，有一个非常重要的思想要让各位读者知道，就是在JavaScript中除了基本数据类型之外，其他任何东西都是对象，也包括函数。在上一节讲到，要创建一个对象，除了直接使用花括号来进行创建外，还可以使用构造函数来进行创建，而构造函数本身就是一个对象，prototype就是该构造函数对象的一个属性，只不过该属性一开始是一个空对象，也就是说只是一个Object，不包含任何的成员变量和成员函数。prototype的作用和构造函数类似，可以向其中添加成员变量和成员方法，以供后续对象使用，如示例代码5-1所示：

1 function Person(name, age, job) {

2 this.name = name;

3 this.age = age;

4 this.job = job;

5 this.sayHello = function() {

6 console.log("你们好，我是一个对象，我的名字叫" + this.name);

7 }

8 }

9

10 Person.prototype.home = "苏州";

11 Person.prototype.getHome = function() {

12 console.log("我的家在" + this.home);

13 };

14

15 // 调用成员函数

16 var classPerson = new Person("李雷", 50, "engineer");

17 classPerson.sayHello();

18 classPerson.getHome();

执行结果：

1 你们好，我是一个对象，我的名字叫李雷

2 我的家在苏州

例5-1 Prototype添加成员变量和方法

从上面的代码可以看到通过prototype属性可以像构造函数一样为对象添加属性和方法，但是实际上添加的home成员变量和getHome成员函数并没有添加到classPerson这个对象中，而是当classPerson在搜索getHome()函数时发现在Person类型中并没有该函数，那么JavaScript的脚本引擎就会自动到prototype中去查找，此时就会找到getHome()函数，并执行该函数，如图5-26所示，在图中可以看到getHome()和home都是在一个类型为Object，名字叫\_\_proto\_\_的对象中，该对象可以看做是prototype属性一个别名。这里还有一个非常重要的思想需要各位读者知道，就是在JavaScript中对象之间的传递都是采用址传递的方式，也就是说只传递对象的内存地址，那么prototype实际上就是保存了一个Object类型的对象地址，更加直接的说就是不论开发人员何时何地的修改Person的prototype都会影响使用Person创建出来的所有对象，如示例代码5-2所示：

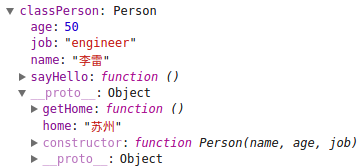


图5-26 Prototype属性的实例

1 // Person类的创建

2 ...

3

4 var classPerson = new Person("李雷", 50, "engineer");

5 Person.prototype.getAge = function() {

6 console.log("我今年" + this.age + "岁");

7 }

8

9 // 调用后加的成员函数

10 classPerson.getAge();

执行结果：

1 我今年50岁

例5-2 Prototype引用特性的使用

从上面的代码可以看出虽然getAge()函数在创建完classPerson对象后才添加，因为classPerson引用了Person的prototype对象，所以虽然是后续添加的函数，但也可以在classPerson中进行调用，因此在JavaScript的脚本引擎中成员函数和成员变量会按照从对象->原型的方式进行搜索。不过有一种情况需要注意，就是当开发人员编写的函数或者变量与原型中的函数或者变量同名，那么此时脚本引擎就需要做出决断是采用原型中的函数或变量还是使用对象本身的函数或变量，这里的答案是对象本身的优先级要高于原型中的优先级，也就是说能调用对象本身的函数或变量时就不会调用原型的函数或变量。

到这里Prototype的基本概念和使用方法也就结束了，接下来首先先来回顾下面向对象的三大特性，即封装性、继承性、多态性，在JavaScript中因为没有封装性的特性，因此人为的定义在成员变量或者成员函数之前加上下划线的就表示其具有私有性，否则就具有公有性。而多态性在上面也讲了，当对象内部的成员变量和成员函数与原型中的成员变量和成员函数发生命名相同时，使用对象内部的成员变量和成员函数，所以可以看到在JavaScript已经满足了三大特性中两个，而继承性则是本节接下来要讲的主要内容。

了解了Prototype的使用，那么JavaScript中对象的继承性正是使用这一个属性来完成。上面讲Prototype最初实际上是保存了一个指向Object对象的地址。那么既然Prototype保存的是地址，那么就先可以让它指向其他对象，而不是一个空的Object，从而实现最基本的继承，下面将通过一个例子的不断迭代来完成继承的讲解，如示例代码5-3所示：

1 // 形状的构造函数

2 function Shape() {

3 this.name = "Shape";

4 this.toString = function() {

5 return this.name;

6 };

7 }

8

9 // 多边形的构造函数

10 function Polygon(lines) {

11 this.name = "Polygon";

12 this.lines = lines;

13 this.getLines = function () {

14 return this.lines;

15 }

16 }

17

18 // 矩形的构造函数

19 function Rectangle(width, height) {

20 this.name = "Rectangle";

21 this.width = width;

22 this.height = height;

23 this.getArea = function() {

24 return this.width \* this.height;

25 }

26 }

27

28 // 使用原型实现继承

29 Polygon.prototype = new Shape();

30 // 防止修改原型造成的意外

31 Polygon.prototype.constructor = Polygon;

32

33 Rectangle.prototype = new Polygon(4);

34 Rectangle.prototype.constructor = Rectangle;

35

36 // 创建矩形的对象，并获取面积

37 var myRectangle = new Rectangle(10, 20);

38 console.log("矩形的面积 = " + myRectangle.getArea() + "，边数 = " + myRectangle.getLines());

执行结果：

1 矩形的面积 = 200，边数 = 4

例5-3 最基础的继承实例

在上面的代码中，首先创建了Shape、Polygon、Rectangle这三个构造函数，然后创建了Shape对象，并且让Polygon的构造函数原型指向这个对象，然后创建了Polygon的对象，并且让Rectangle的构造函数原型指向了这个Polygon对象，通过这些步骤从而实现了如图5-27所示的原型链关系。这里有两个注意点需要特别声明：

1、因为prototype是指向的是一个原型对象，所以只能通过new关键字来创建对象后再赋值给prototype，不能够直接继承构造函数。

2、当修改原型链时，构造函数中的constructor属性会重新指向Object，而不是对象本身的构造函数，这会造成代码的混乱。因此，在修改原型链后一定要把原型链的constructor属性重新指向构造函数。



图5-27 Prototype链的继承关系

在创建完原型链后，通过new关键字创建矩形对象myRectangle，并获取面积和边数。在这个过程中JavaScript的脚本引擎首先回到Rectangle类中寻找getArea()和getLines()这两个函数，此时只能知道getArea()，而getLines()却没有找到，于是就像我们前面说的脚本引擎会到原型链中去查找该函数，此时在Polygon的对象中正好定义了该函数，那么就执行该函数，并返回，这个就是本例中函数的执行过程。此外，还可以利用原型中的成员函数isPrototypeOf()函数来验证Shape、Polygon、Rectangle这三个类之间的关系，在上面实例中加入下面的代码：

1 // 验证原型链

2 if(Shape.prototype.isPrototypeOf(myRectangle)) {

3 console.log("Shape是myRectangle的原型");

4 } else {

5 console.log("Shape不是myRectangle的原型");

6 }

7 if(Polygon.prototype.isPrototypeOf(myRectangle)) {

8 console.log("Polygon是myRectangle的原型");

9 } else {

10 console.log("Polygon不是myRectangle的原型");

11 }

12 if(Rectangle.prototype.isPrototypeOf(myRectangle)) {

13 console.log("Rectangle是myRectangle的原型");

14 } else {

15 console.log("Rectangle不是myRectangle的原型");

16 }

17 if(Number.prototype.isPrototypeOf(myRectangle)) {

18 console.log("Number是myRectangle的原型");

19 } else {

20 console.log("Number不是myRectangle的原型");

21 }

执行结果：

1 Shape是myRectangle的原型

2 Polygon是myRectangle的原型

3 Rectangle是myRectangle的原型

4 Number不是myRectangle的原型

从上面结果不难验证其原型链的存在和关系。虽然上面的代码已经实现了基本的继承功能，但仔细观察还是会发现能够改进的地方，比如Shape、Polygon、Rectangle这三个类中都有一个成员变量name，该属性实际上在任何时候都会发生改变，但是不论是通过new关键字创建Shape，还是创建Polygon，又或是创建Rectangle时，都会在内存中创建一个name的成员变量，因为此时的name采用的是this.name = "Rectangle"的写法，这样的写法意味着name属性是绑定到this上，也就是创建的对象上，如果此时创建了1000个对象，就会有1000个完全相同的name，也就是会浪费1000-1个内存空间，那么就没有办法可以解决这个问题呢？答案当然是有的，使用原型就可以实现属性的共享，但这里要特别说明的是，这个技术只适用于不会发生改变的成员变量和成员函数，而不适用于易变的成员变量和成员函数。接下来把上面原型链创建的过程进行修改，如示例代码5-4所示：

1 // 形状的构造函数

2 function Shape() { }

3 // 添加共享数据

4 Shape.prototype.name = "Shape";

5 Shape.prototype.toString = function() {

6 return this.name;

7 }

8

9 // 多边形的构造函数

10 function Polygon(lines) {

11 this.lines = lines;

12 }

13 // 使用原型实现继承

14 Polygon.prototype = new Shape();

15 // 防止修改原型造成的意外

16 Polygon.prototype.constructor = Polygon;

17 // 添加共享数据

18 Polygon.prototype.name = "Polygon";

19 Polygon.prototype.getLines = function() {

20 return this.lines;

21 }

22

23 // 矩形的构造函数

24 function Rectangle(width, height) {

25 this.width = width;

26 this.height = height;

27 }

28 // 使用原型实现继承

29 Rectangle.prototype = new Polygon(4);

30 Rectangle.prototype.constructor = Rectangle;

31 // 添加共享数据

32 Rectangle.prototype.name = "Rectangle";

33 Rectangle.prototype.getArea = function() {

34 return this.width \* this.height;

35 }

例5-4 原型中的共享属性和方法

在上面的代码中，首先来看Shape类，在Shape类中定义了一个空的构造函数，因为Shape的原型引用了一个Object对象，同时name成员变量和toString()成员函数都属于固定不变的内容，因此把name成员变量和toString()成员函数设置到引用的Object对象中，又因为每个Shape构造函数都会引用相同的原型，因此之后每创建一个Shape对象都会包含相同的原型引用，也就包含了name成员变量和toString()成员函数，从而实现了name和toString()的共享，避免了数据的重复创建和内存资源的浪费。

接下来再看Polygon类，该类和Shape的不同点在于Polygon的构造函数中并没有把所有的成员变量和成员函数放到原型中，而是保留了成员变量lines在构造函数体内，这是因为lines并不是固定不变的，而是在构建不同的多边形时需要传入不同的边数，所以lines属性必须由Polygon对象自己所持有，而不能放在原型中进行共享，否则就会出现没创建一个Polygon对象就会覆盖并修改lines的值，从而使得不论几边形最后都具有相同的边数。

最后来看Rectangle类，该类和Polygon类相似，并没有把所有的成员变量放到原型中，并且在重新引用原型时向Polygon传入参数4，表示这是一个四边形的矩形，而如果创建的是三角形，那么此时就需要向Polygon传递参数3，由此也可见Polygon中的lines实际上是不固定的，需要根据需求不断改变。此外，还可以使用对象的hasOwnProperty()来当前属性时候属于当前对象来侧面验证上面的说法，在上面实例中加入下面的代码：

1 // 验证属性时候属于原型

2 if(myRectangle.hasOwnProperty("lines")) {

3 console.log("lines不是myRectangle原型中的属性");

4 } else {

5 console.log("lines是myRectangle原型中的属性");

6 }

7 if(myRectangle.hasOwnProperty("width")) {

8 console.log("width不是myRectangle原型中的属性");

9 } else {

10 console.log("width是myRectangle原型中的属性");

11 }

执行结果：

1 lines是myRectangle原型中的属性

2 width不是myRectangle原型中的属性

从上面的结果可以看出符合上面讨论的结果，因为lines是Polygon类的属性，因此不输于Rectangle属性，而width则是在Rectangle对象内部。

## 5.5 JQuery的DOM操作和AJAX的异步通讯

在Web开发中，除了经常要使用JavaScript来完成各种工作外，另外一个就是使用各种各样的第三方库来实现不同的功能，这些第三方库的使用可以大大提高编程的效率和开发速度，目前比较主流的JavaScript库主要有两个，一个是Google推出的基于MVC架构的AngularJS，标志着下一代Web应用的开始，而另外一个则是从2006年发布以来经过11年发展，被无数Web应用验证的JQuery。从笔者的角度来看，AngularJS更像一个是一个精装房，你需要去熟悉每个房间的作用后才能使用，即使设计者的习惯和你的编程习惯大相径庭，但你还是需要去习惯它，因为他提供功能齐备、性能优良的一整套解决方案，而JQuery更像是一个毛坯房，你可以根据自己的需要去添砖加瓦，虽然有时设计上没有精装房那么精致，但是一切都符合你的习惯。

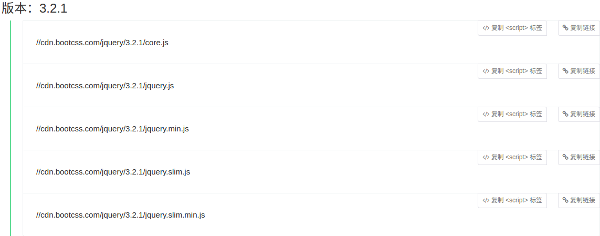
综上所述，可以知道JQuery的学习成本更低，使用也更为的方便，同时JQuery也能满足目前大部分Web应用的需求，因此本节将以JQuery为基础讲述如何实现Web DOM的操作以及如何通过Ajax与服务器进行通信。

### 5.5.1 JQuery的引入与建议

要把JQuery加入到项目中，首先就会要获取它，目前有两种引入JQuery的方法，一种是采用本地下载源文件后引用到项目中，另一种则是采用CDN的方式在线引用到项目中，下面就来介绍这两种方式的使用方法。

1、本地下载： 到笔者撰稿为止，JQuery目前最新的版本是3.2.1，读者可以到它的官方网站（http://jquery.com/）下载。进入官方网站，点击右边的“Download JQuery”进入下载页面，在该页面会看到“Download the compressed, production jQuery 3.x.x”（目前是3.2.1）和“Download the uncompressed, development jQuery 3.x.x”（目前是3.2.1）这两个超链接，读者可以任选一种一个进行下载。这两个的区别在于一个是压缩版（compressed），而另外一个则是非压缩版（uncompressed），所谓压缩版就是通过某些工具把代码中的空行、空格删除，并且把变量名改为非常简单的a、b、c，从而减少JavaScript文件的大小。这样好处是，当浏览器浏览该网页时只需要下载更少的内容就可以显示更加完整的内容，从而提高网页加载的速度。

2、CDN获取： CDN的全称是Content Delivery Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。由于国内网络环境还处于高速发展的阶段，并且一些最新的计算机相关的工具和库都存放在国外的站点上，造成通过国内网络去引用和访问国外这些站点资源时要经过一段很长的路由过程，使得速度非常慢，一旦当中有一个节点发生故障就会造成资源无法获取，因此国内一些公司利用本身资源把国外这些工具和库通过镜像的方式搬到了国内，并当用户要下载这些资源时就通过智能匹配系统为用户选择一个速度最快的节点服务器，从而提高整个网络的速度。目前国内在JavaScript方面做的比较好的两个CDN服务商有两个，一个是BootCDN（http://www.bootcdn.cn/），另一个这是Staticfile CDN（https://www.staticfile.org/）。CND的使用非常方便，进入官网后，在搜索栏搜索要查询的库，例如JQuery就会出现如图5-28所示的界面，然后把这些地址使用script标签包裹起来即可。



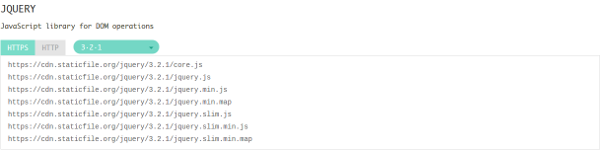


图5-28 JQuery在BootCDN和Staticfile CDN中查询的结果

当获取到JQuery的文件或者CND地址后，接下来就是把它放到页面头部。因为JavaScript的脚本引擎总是顺序执行，因此当脚本引擎读取到JQuery的引入地址后，就会从网络上或者本地读取该文件，当读取完毕后就表示JQuery准备完毕，随后开发人员就可以时候使用它进行开发。但是正如上面说的JavaScript脚本引擎是顺序执行的，此时的脚本引擎还未把当期那页面中所有的HTML代码读取完毕，如果贸然使用JQuery对文档进行操作，会可能造成要操作的HTML节点还未读取到，从而导致JQuery代码操作失败，所以要进行JQuery操作的话，一定要等到当前页面的HTML代码全部读取完毕后再执行，要实现该功能就需要把$(document).ready(function() {...})添加到head中，该函数中的document表示当前页面的Web文档，$符号是JQuery中特使的符号，可以把HTML的节点转化为JQuery可以操作的对象，而read(function() {...})函数则表示读取文档完成后执行的回调函数，开发人员的代码正应该添加在此回调函数中，才能保证读取文档中正确，具体代码如下：

1 <head>

2 <script src="JQuery存放路径/jquery-3.2.1.js"></script>

3 <script>

4 $(document).ready(function() {

5 // 添加自己的脚本代码

6 });

7 </script>

8 </head>

这里还有两点建议需要特别说明，以保证JQuery代码的正确执行，具体如下：

1、依赖后置： 如果使用某些第三方库时，需要依赖于JQuery，那么一定要把JQuery放在这些第三方库之前，以保证在读取这些库之前就已载入JQuery。

2、样式表前置： 因为在使用JQuery时经常要修改或读取页面的样式表，因此一定要把所有样式放在所有的script标签之前，以保证在使用JQuery时就已经加载完整的样式表。

### 5.5.2 DOM的理解与选择器

随着Web技术的发展，特别是HTML5技术的日渐成熟，网站的视觉效果越来越炫目，但是如果你用浏览器的开发者工具仔细观察，就会发现在这些绚丽的页面背后实际上就是无数个div、table、span等各种各样标签的增、删、改、查，并且在这些标签上面增、删、改、查各种带有颜色、大小等样式属性的的Class和ID。因此在Web开发中，JavaScript一个很重要的功能就是对这些标签、样式等进行操作，从而实现新内容的添加，老内容的修改，查询以及删除，那么这些页面中标签的集合就称为DOM。所谓DOM（Document Object Model），就是当Web浏览器在读取HTML网页时，除了对这些HTML标签进行渲染之外，还有一个主要功能就是记录这些HTML标签的层次结构，这个层次结构就是DOM，即文档对象模型。此外DOM还提供了页面的导航（就是有了这个功能，才实现了页面的前进和后退），以及增、删、改、查页面元素的工具，才是的JavaScript具有如此强大的功能，但是这里有一个概念要明确，就是DOM并不是JavaScript，而是由W3C组织所定义的一个标准，每种浏览器内核实现的方式都有所不同，但是接口却采用相同的接口。

在DOM中每个HTML元素都被称为节点，而原生的JavaScript提供两种两种方式来检索这些节点，即getElementsByTagName()和getElementsById()这两个函数，如示例代码5-5所示，结果如图5-29所示：

HTML代码：

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <title>JS运行环境</title>

9 </head>

10 <body>

11 <p>Hello HTML</p>

12 <div id="demo">Hello Div</div>

13

14 <!-- JavaScript的DOM操作 -->

15 <script src="js/5-5.js"></script>

16 </body>

17 </html>

例5-5 JavaScript节点查询

JavaScript代码：

1 console.log(document.getElementById("demo"));

2 console.log(document.getElementsByTagName("p"));

执行结果：



图5-29 JavaScript的选择器

在上面的HTML代码中为了防止文件还未加载完成就进行DOM的读取操作，因此把script标签放在body标签的最后，使得浏览器引擎能够在读取完整的HTML代码后在执行JavaScript代码。在JavaScript代码中通过原生的两个节点查询函数来获取制定的节点。从结果可以看出，通过ID查找的结果是唯一的，而通过标签名查找的节点确实一个数组，这是因为在DOM文档中可能同时出现许多个相同的节点名称，但是同一个ID却只会出现一次。

可能此时，很多读者会觉得原生的JavaScript在检索DOM时非常简单，那为什么还需要用JQuery呢？这是因为在早起，特别是HTML5还未标准化之前，没加浏览器在处理DOM时的方式都各有不同，使得如果使用原生的JavaScript会造成代码的不兼容性。虽然到了HTML5标准化之后，这种情况得到了很大的改善，但是每家浏览器多少还是有些不一样，因此JQuery的出现就会为了解决这个问题，即提供提供一致的兼容性，由JQuery来处理各个浏览器的不同，从而实现代码在各个浏览器之间呈现的效果基本相同。此外原生的JavaScript选择节点的方式比较单一，无法满足像CSS这样灵活的选择方式，而JQuery提供了类似CSS的节点选择工具，因此JQuery几乎成为了Web开发的标配沿用至今。

要使用JQuery，首先就要了解和学会选择器的使用，而JQuery的选择器完全继承了CSS的做法，也就是说只要是能通过CSS选择到的节点，那么就可以使用相同的选择器语法，并通过JQuery选择器选择到，接下来在示例代码5-5的基础上添加相应代码，结果如图5-30所示，具体代码如下：

1 console.log($("#demo"));

2 console.log($("p"));

执行结果：

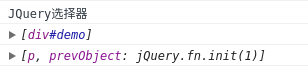


图5-30 JQuery基础选择器

从上面的结果不难看出，要使用JQuery检索节点非常方便，只需要通过$("选择器")就可以完成相关的检索，因为在CSS一节中已经讲了很多关于选择器的内容，这里就不再累述，需要查看看完整的选择器可以到W3CSchool的CSS目录下查看，里面有详细的选择器说明。细心的读者会发现，JQuery检索出来的全部是带[]的数组，而使用JavaScript则不同，这是因为通过JQuery检索得到的是一个特殊的JQuery对象集合，这个集合中包含了所有检索到的DOM节点对象，但是各位读者要记住的重要概念是，这些对象的集合虽然都是DOM节点，当并不是DOM对象，而是把DOM对象进行封装的JQuery对象。有个这个对象集合后，JQuery还帮开发人员做件非常重要的事，首先是节点的自动遍历，第二件事是实现代码的串联。

先说第一件事，自动遍历的功能可以大大提高开发人员的编程效率，例如通过$(".red")检索得到一个JQuery对象集合，那么开发人员就可以直接通过JQuery提供的增、删、改、查的函数来实现该节点内容的修改，而使用原生的JavaScript则需要在通过循环遍历的方式进行修改。看到这里读者可能会问，因为“.red”是一个类别选择器，所以可能检索出来多个，但我只想修改其中的某一个，如果直接对集合进行操作，岂不是就会造成不需要修改的也被修改了？这里有一个先决条件就是，JQuery默认通过选择器选择的节点就是开发人员要操作的节点，如果这个节点是很多个，那么就代表要对这些很多个节点进行操作，如果这个节点只有一个，那么就是对一个节点进行操作，具体的情况完全取决于前面的选择器，因此开发人员在使用选择器时应该正确的添加选择器，以便帮助JQuery找到所需要的那个节点或一群节点。

第二件事，实现代码的串联可以大大减少开发代码的数量，因为每个JQuery函数都会返回当前的JQuery对象，从而实现了代码的串联，但是要实现该功能，那么函数就必须使用JQuery自带的函数，而不能使用JavaScript或者开发人员自定义的函数。

### 5.5.3 Web页面中的DOM操作

通常来说对于DOM的操作主要分为三个方面，即HTML节点的操作、HTML节点属性的操作以及节点样式的操作。其中HTML节点的操作主要是实现节点的增、删、改、查，HTML节点属性的操作主要是实现节点的文本内容、事件对象等节点固有属性的操作，最后一个顾名思义就是对接点样式的修改。而JQuery则在此基础上增加了遍历的功能，具体如下：

1、选择器： 该类主要通过各种选择器的设置查找DOM中的制定节点，上一节中已经叙说，这里就不再累述。

2、DOM操作： 该类函数主要对DOM节点进行增、删、改，并且可以实现节点的嵌套操作。

在DOM操作中，除了查找节点之外，很多时候需要添加一个节点，例如通过HTTP通信的方式获取数据，并把这些数据添加节点中，这样的操作就需要创建新的节点，如示例代码5-6所示，运行结果如图5-31所示。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <style type="text/css">

9 .red {

10 background: red;

11 }

12 </style>

13

14 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

15 <script>

16 $(document).ready(function() {

17 // 创建一个新的li节点

18 var $new\_li = $("<li class='red'>小米</li>");

19 // 把新的li节点插入到ul下面

20 $("ul").append($new\_li);

21 });

22 </script>

23

24 <title>JQuery的新节点创建</title>

25 </head>

26 <body>

27 <div>手机品牌</div>

28 <ul>

29 <li>诺基亚</li>

30 <li>华为</li>

31 </ul>

32 </body>

33 </html>

例5-6 JQuery的节点创建

执行结果：

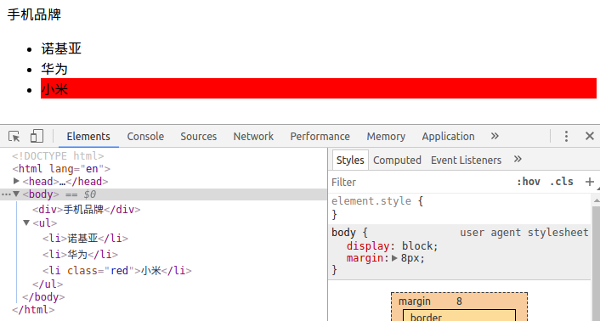


图5-30 创建新的节点

首先先看HTML中body代码，在改代码中创建了无序列表ui，并添加了两个列表项，然后在head中添加了一个用于把背景设置为红色自定义Class样式，并命名为red，此时的样式没有作用于任何的HTML代码，随后通过script中载入JQuery，并添加文档准备完毕的事件处理函数$(document).ready()。在ready()函数中你会发现要通过JQuery创建一个节点是如此的简单，只需要使用$()符号把正确的HTML填入即可，$()会自动把填入的HTML代码转化为JQuery能够识别的对象，并且通过JQuery的选择器查找到ul节点，再通过节点函数append把新增加的节点插入到ul内部，从浏览器的开发者工具可以看到，在ul标签内确实添加了一个新的li节点。

看完节点的创建，接下来看节点的插入函数，该类函数比较多，虽然使用起来比较简单，但很多初学者都容易混淆带To和不带To的区别。在JQuery的插入函数中，有类似A.append(B)和A.appendTo(B)这样的函数，其实这两个函数是相同的意思，只是表达方式不同，在A.append(B)中表示把B插入到A中，而A.appendTo(B)则是把A插入到B中，具体如表5-7所示。

表5-7 JQuery中常用的插入函数

|  |  |  |
| --- | --- | --- |
| 方法 | 描述 | 示例代码 |
| append() | 在节点内部的最后追加元素。 | $(“选择器”).append(“JQuery新节点”) |
| appendTo() | 将节点追加到指定节点内部。 | $(“JQuery新节点”).appendTo(“选择器”) |
| prepend() | 在节点内部的头部追加元素。 | $(“选择器”).prepend(“JQuery新节点”) |
| prependTo() | 将节点追加到指定节点内部的头部。 | $(“JQuery新节点”).prependTo(“选择器”) |
| after() | 在指定节点之后添加新节点（注意：不是内部） | $(“选择器”).after(“JQuery新节点”) |
| insertAfter() | 和To的意思相似，将节点插入到指定节点之后 | $(“JQuery新节点”).insertAfter(“选择器”) |
| before() | 在指定节点之前添加新节点（注意：不是内部） | $(“选择器”).before(“JQuery新节点”) |
| insertBefore() | 和To的意思相似，将节点插入到指定节点之前 | $(“JQuery新节点”).insertBefore(“选择器”) |

（1）remove()函数：该函数有两种表达方式，分别是$(”要删除节点的选择器“).remove()，以及$(”要删除节点的选择器“).remove(”对要删除节点的选择器再进行筛选的选择器“)，具体来说第一种表示查询出相关的节点后直接删除，而第二种这是在查询结果中再进行筛选，并找出最终的节点进行删除。在执行remove()函数时，除了会删除节点之外，还会返回被删除的节点，如示例代码5-7所示。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <style type="text/css">

9 .red {

10 background: red;

11 }

12 </style>

13

14 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

15 <script>

16 $(document).ready(function() {

17 // 创建一个新的li节点

18 var $new\_li = $("<li class='red'>小米</li>");

19 // 把新的li节点插入到ul下面

20 $("ul").append($new\_li);

21

22 // 删除节点，并把删除的节点打印

23 var $delete\_li = $("ul li").remove("li[class='red']");

24 console.log($delete\_li);

25 });

26 </script>

27

28 <title>JQuery节点删除</title>

29 </head>

30 <body>

31 <div>手机品牌</div>

32 <ul>

33 <li>诺基亚</li>

34 <li>华为</li>

35 </ul>

36 </body>

37 </html>

例5-7 JQuery的节点删除

执行结果：



图5-30 删除节点函数的使用

上面的代码非常容易理解，就不再阐述，这里要特别说明的是remove()函数的返回值，因为在上图中打印的返回值不仅包含了要删除的节点，还包含了没有删除的节点，这是因为remove()返回值实质上是返回要删除节点的选择器值，而如果还需要对该选择器值进行二次过滤时，那么不会对返回值造成影响，因此remove()的返回值取决于函数左边的选择器。

（2）detach()函数：该函数和remove()的功能相同，但区别在于使用remove()函数删除的节点，除了节点本身被删除之外，附加在该节点上的事件、数据等都会一并被删除，但是detach()则不同，detach()删除的节点不会移除事件和数据，当下次把删除的节点再移回来时，会和被移除之前保持一致，因此detach()函数特别适合那些因为某些原因需要被临时移除，但是又会马上恢复的节点。

（3）empty()函数：该函数用于清空某个元素的内部所有的内容和节点，也包括子节点，如表5-8所示。

表5-8 JQuery中常用的删除函数

|  |  |  |
| --- | --- | --- |
| 方法 | 描述 | 示例代码 |
| remove() | 删除指定节点。 | $(“要删除节点的选择器”).remove()或$(“要删除节点的选择器”).remove(“对要删除节点的选择器再进行筛选的选择器”) |
| detach() | 删除指定节点。 | $(“要删除节点的选择器”).detach()或$(“要删除节点的选择器”).detach(“对要删除节点的选择器再进行筛选的选择器”) |
| empty() | 清空指定节点内部的所有内容。 | $(“选择器”).empty() |

上面讲了DOM的增加、删除，并且在上一节也讲了查找，那么接下来就是DOM的修改，DOM的修改主要涉及三类函数，分别是节点的克隆、节点的替换和节点的包裹，如表5-9所示。

表5-9 JQuery中常用的修改函数

|  |  |  |
| --- | --- | --- |
| 方法 | 描述 | 示例代码 |
| clone() | 克隆一个节点，当函数参数为true时，则表示连绑定的事件也一同复制。 | $(“选择器”).clone()或$(“选择器”).clone(true) |
| replaceWith() | 将选择器A的替换为B。 | $(“A选择器”).replaceWith(“B选择器”) |
| replaceAll() | 将选择器B的替换为A。 | $(“A选择器”).replaceAll(“B选择器”) |
| wrap() | 将选择器B的包裹在每个的A选择器结果之外，如示例代码5-8所示。 | $(“A选择器”).wrap(“B选择器”) |
| wrapAll() | 将选择器B的包裹在所有的A选择器结果之外，如示例代码5-8所示。 | $(“A选择器”).wrapAll(“B选择器”) |
| wrapInner() | 将选择器A的包裹在每个的B选择器结果之外，如示例代码5-8所示。 | $(“A选择器”).wrapInner(“B选择器”) |

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

9 <script>

10 $(document).ready(function() {

11 $("li").wrap("<strong></strong>");

12

13 $("li").wrapAll("<strong></strong>");

14

15 $("li").wrapInner("<strong>手机</strong>");

16 });

17 </script>

18

19 <title>JQuery节点的包裹</title>

20 </head>

21 <body>

22 <div>手机品牌</div>

23 <ul>

24 <li>诺基亚</li>

25 <li>华为</li>

26 </ul>

27 </body>

28 </html>

例5-6 JQuery的节点包裹

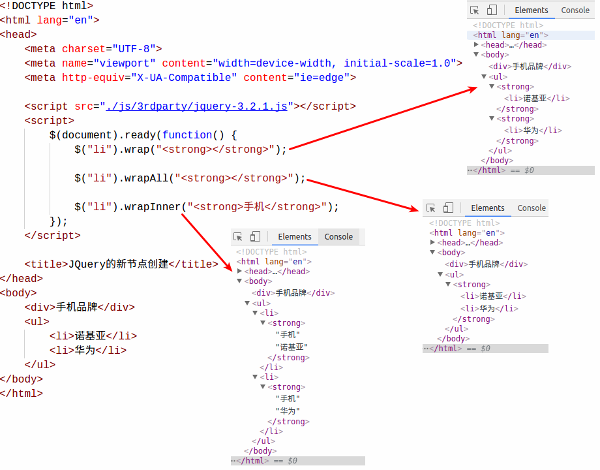


图5-31 包裹节点函数的使用

从上面的结果可以看出，wrap()函数会包括每一个通过选择器选择出来的节点，而wrapAll()函数则是把这些选择器选择出来的节点作为一个整体，在外层进行包括，这里值得注意的是，如果查询出来的节点中掺杂了其他类型的节点，那么也会一并被包裹在内，最后是wrapInner()，该函数会在所有选择器选择出来的每一个节点内容的外面添加一层包裹。

2、属性和CSS： 该类函数主要用于增、删、改、查相关属性，如高度、大小等，以及修改相关的CSS样式。

在JQuery中要增、删、改、查某个节点属性非常的方便，只需要用到两个函数，分别是attr()函数和removeAttr()函数，其中attr()函数主要完成属性的增加、修改和获取，而removeAttr()函数则完成属性的删除。首先通过示例代码5-9来展示属性的增、删、改、查。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

9 <script>

10 $(document).ready(function() {

11 // 获取属性

12 console.log($("input").attr("value"));

13 // 修改属性

14 console.log($("input").attr("value", "我在修改属性"));

15 // 获取修改后的属性

16 console.log($("input").attr("value"));

17 // 添加一个新的属性

18 console.log($("input").attr("name", "attr-test"));

19 // 获取新添加的属性值

20 console.log($("input").attr("name"));

21 });

22 </script>

23

24 <title>JQuery节点属性的增加、修改和获取</title>

25 </head>

26 <body>

27 <form>

28 <input type="text" value="这是属性修改测试" />

29 </form>

30 </body>

31 </html>

例5-6 JQuery的属性增加、修改和获取

执行结果：

1 这是属性修改测试

2 [input, prevObject: jQuery.fn.init(1)]

3 我在修改属性

4 [input, prevObject: jQuery.fn.init(1)]

5 attr-test

从上面的结果可以看出，当attr()只有一个参数的时候表示获取某一个属性的值，而当attr()有两个参数时，则用于属性的添加和修改，并且该函数会返回修改属性后的JQuery对象。在attr()函数中，第一个参数表示要查询的属性，第二个参数就表示要修改的值，如果要查询的属性不存在，那么就会把该属性作为新属性进行添加，并且值等于第二个参数，如果该属性能够查询到，那么就把该属性的值设置为第二个参数，从而实现了属性的增加、修改和获取。如果一次性想设置多个属性，那么就可以使用传入对象的方式完成，这里要特别说明的是，批量添加属性时，属性名称不需要加引号，但是属性值需要加引号，具体代码如下：

1 $("选择器").attr({

2 属性: "属性值",

3 属性: "属性值",

4 ...

5 属性: "属性值"

6 })

了解了属性的增加、修改和获取后，接下来就是属性的删除函数removeAttr()，该函数的使用非常简单，只需要把要删除的属性名称作为参数填入removeAttr(“要删除的属性名”)即可。在了解了属性操作之后，接下来就是样式操作，样式操作主要包含样式的设置和获取、样式的追加、样式的移除，以及样式的判断，这些功能分别对应attr()、addClass()、removeClass()，以及hasClass()这四个函数，这几个函数中重点要说明的是attr()和addClass()函数的区别，因为其他函数都只要传入相应的样式名称即可完成操作，非常简单，而hasClass()则返回boolean类型来判断样式时候存在。

所谓样式的值，其实也是节点class属性的值，那么既然是属性就可以通过attr()函数进行操作，但是attr()同时是对属性值的进行修改，但是样式属性值有一个特点就是可以填入多个样式值，并以空格作为间隔，所以attr无法方便的对样式值里面的某一个样式进行操作，因此JQuery提供了addClass()函数，该函数可以添加一个或多个样式，而不需要再通过复杂的attr()操作来完成。

接下来重点来说下JQuery节点中很容易混淆的两个属性函数，分别是text()和val()，text()是用于设置设置节点内容，而val()则是用于设置节点value属性的值，一般来说text()函数用于普通的节点，例如<p></p>、<b></b>等，都可以设置这些节点的显示内容，而val()则更多的用于表单控件中，因为表单控件的值很多是放在value属性中，如text、button等，因此text()和val()这两个函数的应用场合不同，需要注意区分。

3、遍历： 该类函数主要对于检索出来的DOM集合进行遍历，例如在这个集合你查找第一个、添加一个新对象等。

在JQuery中遍历主要用于检索出想要的元素，因为很多时候并不是所有节点都会添加上唯一的标示用于检索，因此就需要找到一个父节点，然后一级级的向下或向上或向下搜索，这个时候遍历就起到非常大的作用。JQuery中遍历函数按照方向分类可以分为三类，第一类是向下搜索函数children()，第二类是向上搜索函数，分别为parent()、parents()、，最后是平行搜索函数，分别为next()、nextAll()、prev()、prevAll()、siblings()。下面通过表5-10进行说明。

表5-10 JQuery中遍历函数的使用

|  |  |  |
| --- | --- | --- |
| 方法 | 描述 | 示例代码 |
| children() | 查询节点的所有子节点，当参数中传入选择器字符串时，则在子节点中进行二次过滤。 | $(“选择器”).children()或$(“选择器”).children(“选择器”) |
| parent() | 查询节点的上一级父节点，并返回，这里要注意的是parent()只查询一级。当参数中传入选择器字符串时，则在父节点中进行二次过滤。 | $(“选择器”).parent()或$(“选择器”).parent(“选择器”) |
| parents() | 查询节点的所有父节点，并返回，这里要注意和parent()进行区分，因为parents()会把所有的父节点，直到html节点为止都进行返回。当参数中传入选择器字符串时，则在父节点中进行二次过滤。 | $(“选择器”).parents()或$(“选择器”).parents(“选择器”) |
| next() | 查询节点后面的一个元素，当参数中传入选择器字符串时，则在结果中进行二次过滤，如果后面的节点不满足选择器，则函数就不返回对象。 | $(“选择器”).next()或$(“选择器”).next(“选择器”) |
| nextAll() | 查询节点后面的所有元素，当参数中传入选择器字符串时，则在结果中进行二次过滤。 | $(“选择器”).nextAll(“选择器”) |
| prev() | 查询节点前面的一个元素，当参数中传入选择器字符串时，则在结果中进行二次过滤，如果前面的节点不满足选择器，则函数就不返回对象。 | $(“选择器”).prev()或$(“选择器”).prev(“选择器”) |
| prevAll() | 查询节点前面的所有元素，当参数中传入选择器字符串时，则在结果中进行二次过滤。 | $(”选择器“).prevAll(”选择器“) |
| siblings() | 查询节点同级的所有元素，也就是nextAll()和prevAll()函数的结果集合，当参数中传入选择器字符串时，则在结果中进行二次过滤。 | $(”选择器“).siblings(”选择器“) |

### 5.5.4 深入理解JQuery中的事件响应

在上一小节中我们讲了如何通过JQuery进行DOM操作，但是读者肯定发现这些界面都不够灵活，也没有交互，这是因为在这些代码中没有添加相关的事件处理，因此都是让程序直接运行，没有任何的人为干预，而如果要让Web页面实现交互，那么就需要使用本节讲阐述的事件响应和事件处理。虽然传统的JavaScript中已经提供了一些事件处理函数，例如表单中的点击事件和提交事件，但是JQuery在此基础上进行了扩展，并且让这些事件处理变的更为简单和方便。

要了解JQuery中的事件处理函数，首先从我们非常熟悉的函数$(document).ready()开始，该函数是页面加载完毕后的处理函数，但这里要要注意的是页面加载完毕指的是DOM加载完毕，而不是所有的资源加载完毕，理解这点非常重要。例如在页面中使用标签加载了一张存放在网络上的图片，那么当脚本引擎执行到ready()的处理函数时表示页面已经加载了标签，并且可以通过JQuery对该标签进行操作，但是此时脚本引擎可能还未把该网络上的图片加载完毕，所以如果此时要对该图片进行操作就会有很大可能操作失败，因为图片的很多属性还未完整获得，那么有没有办法解决这个问题呢？那肯定是有的。在JQuery中还提供了一个页面加载完毕的处理函数，即$(window).load(function(){...})，该函数和ready()的区别就在于load()函数是要等所有资源加载完毕后再执行回调函数，那么此时有读者可能就会问，既然load()函数比ready()函数加载的内容更为完整，为什么还要使用ready()，而不是load()？这是因为ready()速度快，因为对于大部分的Web应用来说，只要DOM加载完毕就可以开始工作，而一些展示性的图片可以让它们在后台慢慢加载，所以有的网站在网络不好时，可以看到页面中的图片一张张被加载，而正常的页面跳转等操作却可以正常执行，而对于有些网页，例如在线的图片编辑软件，则需要在图片加载完毕后再进行操作，因为很多操作需要依赖完整的资源属性，所以此时就需要用load()函数来完成。

当页面加载完毕后就可以在回调函数中进行事件的绑定和操作。在JQuery中，事件的绑定有两个方法，第一种是使用专门的绑定函数进行事件绑定，第二种则是直接为节点事件添加处理函数，从笔者的角度来说更加喜欢第一种方式，因为这种方式更加的灵活，也能提供统一的事件绑定方式。因此这里先说第一种事件绑定方式，随后再说第二种方式。所谓专门的绑定函数就是使用JQuery中的on()，具体如下函数。其实事件绑定函数还有一个，即bind()，但是该函数在JQuery 3.0以上就已经被弃用了，所谓弃用，并不是代表不能用，而是不再为维护，并且在未来的某一天会从源码中直接移除，而on()函数则是在JQuery 1.7中增加的新函数，并且得到持续的维护，所以JQuery官方推荐使用on()函数来进行事件绑定，而不是使用bind()函数。

1 JQuery对象.on("事件类型"[, "选择器"][, 数据对象], 事件处理函数)

从上面函数声明可以看出，on()函数最少可以只传递两个参数，一个是事件类型，另一个则是事件处理函数。JQuery中事件类型主要分为四大类，分别为浏览器事件（Browser Events）、表单事件（Form Events）、键盘事件（Keyboard Events）、鼠标事件（Mouse Events），具体如表5-11、5-12、5-13、5-14所示。

表5-11 JQuery的浏览器事件

|  |  |
| --- | --- |
| 事件类型 | 事件描述 |
| resize | 浏览器窗口尺寸改变时触发。 |
| scroll | 当用户在节点内执行了滚动操作时触发。 |

表5-12 JQuery的表单事件

|  |  |
| --- | --- |
| 事件类型 | 事件描述 |
| blur | 一个节点失去焦点将触发，虽然该事件开始时只在表单内使用，当现在适用于所谓的节点。 |
| change | 一个表单节点的值发生改变时将触发。此事件仅限用于input元素，textarea和select元素。对于下拉选择框，复选框和单选按钮，当用户用鼠标作出选择，该事件立即触发，但对于其他类型的input元素，该事件触发将推迟，直到元素失去焦点才会触点。 |
| focus | 当一个节点获得焦点时触发，获取焦点的方式有两种，一种是鼠标点击，还有一种是使用Tab键进行切换。 |
| select | 当用户在一个元素中进行文本选择时会触发。此事件只能用在text和textarea。 |
| submit | 当用户试图提交表单时会触发。 |

表5-13 JQuery的键盘事件

|  |  |
| --- | --- |
| 事件类型 | 事件描述 |
| keydown | 键盘按键按下时触发 |
| keypress | 键盘点击时触发 |
| keyup | 键盘按键弹起时触发 |

表5-14 JQuery的鼠标事件

|  |  |
| --- | --- |
| 事件类型 | 事件描述 |
| click | 鼠标点击时触发 |
| contextmenu | 鼠标右键时触发 |
| dblclick | 双击鼠标时触发 |
| hover | 鼠标经过节点时触发 |
| mousedown | 鼠标按下时触发 |
| mouseenter | 鼠标移入节点时触发，不会产生，冒泡事件 |
| mouseleave | 鼠标移出节点时触发，不会产生冒泡事件 |
| mousemove | 鼠标在节点内部移动时触发 |
| mouseout | 鼠标移出节点时触发 |
| mouseover | 鼠标移入节点时触发 |
| mouseup | 鼠标弹起时触发 |

当需要绑定事件时只需要把上表中的事件类型名称作为字符串传入第一个参数即可。在了解完事件类型后，接下来看事件的事件处理函数，所谓事件处理函就是在事件触发时进行处理的函数，该函数可以采用两种方式，一种是带参数的事件处理函数function(e){...}，另外一种是不带参数的事件处理函数function(){...}，这两种方式唯一的区别就是参数，该参数就是事件对象（Event Object），用于记录事件相关的信息，常用的事件信息名称有target、currentTarget、data、pageX、pageY、type、which、metaKey，要获取这些信息只需要通过“e.事件信息名称”即可，这些常用的事件信息具体如表5-15所示。

表5-14 JQuery的常用事件信息

|  |  |
| --- | --- |
| 事件类型 | 事件描述 |
| target | 触发当前事件的对象 |
| currentTarget | 事件冒泡过程中的当前DOM对象 |
| data | 事件数据 |
| pageX | 发生事件时，鼠标相对Web浏览器的X轴坐标 |
| pageY | 发生事件时，鼠标相对Web浏览器的Y轴坐标 |
| type | 触发事件类型，和表5-11至表5-14的内容一致 |
| which | 获取触发鼠标或者键盘事件的按键 |
| metaKey | 如果是MAC，那就是判断时候按下Meta键，如果是Windows，那么就是判断时候按下Win键。如果按下该值就是true，否则就是false |

这里要特别说明有三个事件信息，分别是data、target和currentTarget。先说事件数据data，上面讲到在绑定事件时最少可以传递两个参数，分别是事件类型和事件处理函数，还有两个是可选参数，其中第二个可选参数就是事件数据，该事件数据实际上是一个以键值对保存的JS对象，当绑定事件并触发事件时，就可以通过事件对象获得，例如“事件对象.data.数据Key值”这样的形式，但是采用这种方式无法实现事件数据的动态化，而需要都进行事件的重新绑定，这样显然是不合适的，因此JQuery提供了另外一个函数来模拟指定事件的触发，并且向事件内传递参数，该函数就是trigger("事件类型", 事件数据对象)，当节点绑定事件后就可以通过该函数进行触发，并且传入相应的事件数据对象，而不需要重新绑定事件。

第二个要讲的就是target和currentTarget。要了解事件触发是这两个参数的区别，有限就必须了解JavaScript中的事件冒泡效应。所谓事件冒泡，其实就是一种事件传递的方式，即当子节点触发一个事件时，父节点、父父节点，直至bod节点对于该事件的处理方式，下面通过一个例子来进行说明，如示例代码5-10所示。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

9 <script>

10 $(document).ready(function() {

11 $("body").bind("click", function(e) { // 为Body绑定点击事件

12 console.log("Body节点被触发...");

13 })

14 $("#parent").bind("click", function(e) { // 为Parent绑定点击事件

15 console.log("父节点被触发...");

16 });

17 $("#child").bind("click", function(e) { // 为Child绑定点击事件

18 console.log("子节点被触发...");

19 });

20 });

21 </script>

22

23 <title>JQuery节点属性的增加、修改和获取</title>

24 </head>

25 <body>

26 <div id="parent">

27 <div id="child">子节点</div>

28 </div>

29 </body>

30 </html>

例5-10 JQuery的事件冒泡

执行结果：

1 子节点被触发...

2 父节点被触发...

3 Body节点被触发...

从结果可以看出当子节点触发点击事件时，不仅子节点执行了处理函数，而且子节点的父节点，直到Body节点都触发了点击事件，这就是JavaScript中的事件冒泡效应，就像水底的一个小气泡，一点点的向上移动，直到浮出水面。有时这样的事件冒泡效应会带来很多烦恼，比如不想让父节点进行相应，只想在触发该事件的对象中相应，那么此时就需要调用事件对象的stopPropagation()函数来停止事件的冒泡。除此之外，JQuery有些绑定的事件也可以不产生冒泡效应，例如mouseenter和mouseleave这两个事件，因此如果不想让鼠标进入节点和离开节点时产生冒泡事件，那么就建议不要是mouseout和mouseover，而改为mouseenter和mouseleave。

那么现在再回到target和currentTarget，这两个事件信息的区别就在于target表示的是触发事件的对象，而currentTarget如果在触发节点类，那么就和target值的是同一个对象，如果是经过冒泡后的事件，那么currentTarget就是至当前触发该事件的DOM对象，即target的父节点，下面在例5-10的基础上进行修改，代码如下：

1 $("body").bind("click", function(e) { // 为Body绑定点击事件

2 console.log("Body节点被触发...");

3 if(e.target == e.currentTarget) {

4 console.log("在Body中，target和currentTarget相同");

5 }else {

6 console.log("在Body中，target和currentTarget不同");

7 }

8 })

9 $("#parent").bind("click", function(e) { // 为Parent绑定点击事件

10 console.log("父节点被触发...");

11 if(e.target == e.currentTarget) {

12 console.log("在Parent中，target和currentTarget相同");

13 }else {

14 console.log("在Parent中，target和currentTarget不同");

15 }

16 });

17 $("#child").bind("click", function(e) { // 为Child绑定点击事件

18 console.log("子节点被触发...");

19 if(e.target == e.currentTarget) {

20 console.log("在Child中，target和currentTarget相同");

21 }else {

22 console.log("在Child中，target和currentTarget不同");

23 }

24 });

执行结果：

1 子节点被触发...

2 在Child中，target和currentTarget相同

3 父节点被触发...

4 在Parent中，target和currentTarget不同

5 Body节点被触发...

6 在Body中，target和currentTarget不同

从结果可以验证上面的说法，在Child节点中target和currentTarget相同，而在后面的冒泡事件处理函数中这两个事件信息就不相同，因为currentTarget表示的是当前冒泡时间的DOM对象。

最后来讲事件绑定函数中第一个可选参数，该参数是一个选择器，如果该参数为null，则绑定函数直接忽略该参数，这类事件绑定称为直接事件，而如果绑定函数不为null，那么该事件就变为了委托事件。所谓的直接事件通俗来说就是我直接让事件处理对象来处理事件，即直接把事件绑定到某一个节点对象，而委托事件则不是直接绑定到一个目标节点上，而是绑定到目标节点的父节点上，委托父节点来管理子节点的事件处理，具体代码如下：

1 $( "#dataTable tbody" ).on( "click", "tr", function() {

2 console.log( $( this ).text() );

3 });

在上面的代码中，为表格绑定了一个点击事件，但其实是为表格中的每一个表格项绑定了事件，该事件只有在点击表格项的时候才会起作用。使用委托事件有如下两个优点：

1、自动绑定有效事件：委托事件可以防止节点存在的情况下频繁的绑定事件和解绑事件，例如在表格的处理中，只有表格项才能有事件处理，但是表格经常要删除和添加，如果不用委托事件，那么每删除一个表格项就需要解绑事件，而每添加一个表格项就需要增加事件绑定，这样的做法效率很低，因此采用委托事件就可以让表格自己来处理表格项的事件。

2、减少系统开销：委托事件的另一个好处就是，当子节点比较多时，如果一个个去绑定事件，那么系统的开销会非常大，而如果采用委托事件，那么只需要在父节点上绑定一个事件，子节点就相当于也绑定了事件。

既然有事件绑定，那么就一定有事件解绑，JQuery的事件解绑的使用非常简单，就是off()函数，具体代码如下。off()函数除了没有事件数据之外，其他方法和on()函数相同，读者可以参考使用。

1 JQuery对象.off("事件类型"[, "选择器"], 事件处理函数)

上面就是第一种事件绑定的方法，接下来就说第二种事件绑定方法，该方法非常简单，因此就举一个例子进行说明，如示例代码5-11所示。

1 <!DOCTYPE html>

2 <html lang="en">

3 <head>

4 <meta charset="UTF-8">

5 <meta name="viewport" content="width=device-width, initial-scale=1.0">

6 <meta http-equiv="X-UA-Compatible" content="ie=edge">

7

8 <script src="./js/3rdparty/jquery-3.2.1.js"></script>

9 <script>

10 $(document).ready(function() {

11 $("body").click(function(e) {

12 console.log("Body节点被触发...");

13 });

14 $("#parent").click(function(e) {

15 console.log("父节点被触发...");

16 });

17 $("#child").click(function(e) {

18 console.log("子节点被触发...");

19 });

20 });

21 </script>

22

23 <title>JQuery节点属性的增加、修改和获取</title>

24 </head>

25 <body>

26 <div id="parent">

27 <div id="child">子节点</div>

28 </div>

29 </body>

30 </html>

例5-11 JQuery的简单事件绑定

执行结果：

1 子节点被触发...

2 父节点被触发...

3 Body节点被触发...

例5-11是对例5-10的一个修改，在这个例子中通过事件函数来进行事件的绑定，绑定的函数名和事件类型完全相同，并且回调函数的使用也和on()函数相同，唯一的不同就是数据对象传入时的位置，完整的函数声明如下：

1 JQuery对象.事件类型名([数据对象,] 事件处理函数);

### 5.5.5 Ajax异步通讯的理解和使用

在了解了DOM操作和事件处理后，你会发现整个页面已经能动起来了，并且使得Web具有很好的交互性，但是此时Web中的数据始终保持不变，这是因为还没有和后台服务器进行数据交互，因此本节将会讲解如何通过Ajax进行数据交互和解析。所谓Ajax并不是一门语言，而是多种技术的统称，目前Ajax已经成为Web中HTTP通信的标准方式，几乎所有的网站前台都是通过Ajax来完成与后台的交互，并且由于浏览器的支持，Ajax的使用几乎不需要使用任何插件就可以在任何浏览器上使用，并且Ajax可以实现局部数据的刷新，也就是说当数据获取完毕后，不需要进行整个页面的刷新，而只需要进行局部的数据更改即可，同时Ajax采用数据的异步通信，也就是说当使用Ajax通信时，会在浏览器内部启动一条线程，在该线程中完成整个数据的交互，而不会在界面的主线程中运行，当数据获取完毕后再把数据推送到主线程中进行解析，这样可以使得网络通信时也不会影响界面操作，从而提高了Web应用的体验。

虽然Ajax得到了大部分浏览器的支持，使得通过原生的JavaScript就可以实现Ajax的各种通信，但是每个浏览器多少在使用细节上还是有许多不同的地方，因此JQuery就对这些不同点进行了封装，使得所有浏览器都采用统一的Ajax接口，并且使得Ajax的使用变得更为简单和方便。JQuery中关于Ajax的核心函数一共就只有六个函数，并且并且函数可以按照从底层到高层的方式进行划分，其中最底层的函数为ajax()，该函数实现了最核心Ajax的功能，第二个层次的函数就是load()、get()和post()，这三个函数根部不同的业务对ajax()函数进行了二次的封装，得到更加精简的数据通信方式，最后一个层次就是getScript()和getJSON()，这两个函数不仅在业务层进行了封装，而且把数据类型也进行了封装，接下自下而上的对六个函数进行详细的说明。

1、ajax()：该函数实际上使用并不是很多，因为上层函数已经可以完成大部分的工作，但是了解该函数的使用有助于其他高层函数的深刻理解。ajax()函数有两种形式，具体如下：

1 ajax( "发送HTTP请求的服务器地址" [, Ajax的通信参数设定对象 ] );

2 ajax( [ Ajax的参数设定对象 ] )

其中核心的就是Ajax的通信参数对象设定，既然是对象，那么就是以键值对的形式进行保存，具体如表5-15所示。

表5-15 ajax函数中常用通信参数

|  |  |  |
| --- | --- | --- |
| 参数名称 | 默认值 | 参数说明 |
| beforeSend | function(jqXHR jqXHR, PlainObject settings) | 请求发送前的回调函数，用来修改请求发送前jqXHR，即JavaScript中Ajax的请求对象XMLHttpRequest，但是JQuery把其封装为jqXHR，该函数可以用来设置自定义HTTP头信息等。如果在beforeSend函数中返回false，该HTTP请求将被取消。 |
| complete | function( jqXHR jqXHR, String textStatus ) | 请求完成后回调函数。该回调函数得到2个参数： jqXHR对象和一个描述请求状态的字符串，该字符串描述了请求的结果状态，一共有7个，分别为"success"、"notmodified"、"nocontent"、"error"、"timeout"、"abort"和"parsererror"。 |
| contentType | 'application/x-www-form-urlencoded; charset=UTF-8' | 指定发送到服务器的内容类型。 |
| data |  | 发送到服务器的数据。 |
| dataType |  | 从服务器返回期望的数据类型，如果没有指定，jQuery将尝试通过MIME类型的响应信息来智能判断是xml、json、script、html中的哪一个。 |
| error | function( jqXHR jqXHR, String textStatus, String errorThrown ) | 请求失败时调用此函数。jqXHR对象描述发生错误类型的一个字符串和捕获的异常对象。如果发生了错误，textStatus除了得到null之外，还可能是timeout、error、abort以及parsererror。 并且errorThrown接收HTTP状态的文本部分，如“Not Found”、“Internal Server Error”等。 |
| method | "GET" | HTTP请求方法，例如"POST"、"GET "、"PUT"等。 |
| mimeType |  | 用来覆盖XHR中的MIME类型。 |
| statusCode |  | 一个包含HTTP响应状态码，以及对应处理函数的对象。 |
| success | function( Anything data, String textStatus, jqXHR jqXHR ) | 请求成功后的回调函数，其中data是从服务器返回的数据，textStatus是一个描述状态的字符串。 |
| timeout |  | 设置请求超时时间，以毫秒为单位。 |
| url |  | 发送HTTP请求的服务器地址。 |
| mimeType |  | 用来覆盖XHR中的MIME类型。 |
| mimeType |  | 用来覆盖XHR中的MIME类型。 |

从上表可以看到如果直接使用ajax()函数，还是比较复杂的，因此JQuery就在此基础上进行了二次的封装，其中的很多参数在封装时已经填入了默认值，而只暴露一些常用的接口，接下来来看下进行二次封装后的三个函数load()、get()和post()。

2、load()：该函数的作用是从服务器载入数据，并且把返回的结果放入到通过选择器选择的节点中，该函数的具体形式如下：

1 $("选择器").load("服务器请求地址" [, 请求时传入的数据对象] [, 请求完成时的回调函数])

该函数有两个可选参数，分别为以键值对保存的请求时传入的数据对象，另一个这是当请求完成时的回调函数，该函数的表现形式为complete(responseText, textStatus, XMLHttpRequest)，其中responseText为服务器返回的结果，textStatus为通信状态，而XMLHttpRequest则是Ajax的通信对象。这里需要注意的是，当load()函数没有传入数据对象时采用的GET明文通信，而当load()函数有传入数据对象时，则采用POST通信方式。

3、get()：该死表示向服务器发出一个HTTP GET请求，并从服务器上获取数据，该函数的具体形式如下：

1 jQuery.get("服务器请求地址" [, 请求时传入的数据对象] [, 请求成功时的回调函数] [, 预期的数据类型] )

该函数有三个可选参数，第一个可选参数是以键值对保存的请求时传入的数据对象，第二个可选参数是当请求成功时的回调函数，该函数的表现形式为function( PlainObject data, String textStatus, jqXHR jqXHR )，其中data为服务器返回的数据对象，textStatus为通信状态，而jqXHR则是JQuery封装的Ajax通信对象，第三个可选参数则是这次请求希望从服务器返回的数据类型，可以是xml、json、script、text、html中的一个。上面说了，所有这些封装后的函数，都可以用ajax()函数来表达，那么get()函数实际上就下下面ajax()函数的封装：

1 $.ajax({

2 url: "服务器请求地址",

3 data: 请求时传入的数据对象,

4 success: 请求成功时的回调函数,

5 dataType: 预期的数据类型

6 });

细心的读者可能会发现，在上面这些参数和回调函数中没有出现请求失败函数，这是因为在JQuery中所有的Ajax函数都会返回一个jqXHR对象，而失败的回调函数就在此对象中完成，因此完整的GET操作代码应该采用如下这种方式：

1 $.get(

2 url,

3 data,

4 function(data, textStatus, jqXHR) {

5 // 请求成功的处理函数

6 },

7 dataType)

8 .done(function() {

9 // 请求完成的处理函数

10 })

11 .fail(function() {

12 // 请求失败的处理函数

13 })

14 .always(function() {

15 // 不论成功、失败都会回调的处理函数

16 });

4、post()：POST函数的请求和GET函数的请求非常的类似，只不过在函数名方面有所区别，这里就不再累述。虽然使用方法类似，都是采用HTTP进行通信，但实际上这这两个通信方式却有很多不一样的地方，首先采用GET方式，所有数据信息会放在URL地址栏中，并且采用明文方式进行传递，而采用POST方式，则是把数据信息放在HTTP的Content里面进行传递，不会在地址栏中显示，提高了数据的安全性，其次采用GET方式时，因为是放在地址栏中，而URL的大小是有限制的，通常不能大于2KB，而采用POST则不受限制，因此POST经常传递一些图片、视频等多媒体文件，最后一个就是在GET方式的话，数据会被缓存起来，可以从浏览器的历史记录中获得，所以通常情况下在传递需要数据隐秘或大数据文件时采用POST方式，而如果是小数据量，也没有隐秘可言时可以采用GET方式。同样post()方式也是ajax()的再封装，因此也可以使用ajax()函数来表达，具体如下：

1 $.ajax({

2 type: "POST",

3 url: "服务器请求地址",

4 data: 请求时传入的数据对象,

5 success: 请求成功时的回调函数,

6 dataType: 预期的数据类型

7 });

5、getScript()和getJSON()：最后来说两个最上层的封装函数，这两个函数都是采用GET方式进行提交，区别在于getScript()用于载入JavaScript文件，因此不需要传递数据对象，而getJSON()则是用于从服务器上获取以JSON形式编码的数据，因此在发送HTTP请求时，可能需要传递数据对象，两个函数的原型和采用ajax()函数的表达形式如下：

1 getScript( "服务器请求地址" [, 请求成功时的回调函数 ] )

2 $.ajax({

3 url: "服务器请求地址",

4 dataType: "script",

5 success: 请求成功时的回调函数

6 });

7

8 getJSON( "服务器请求地址" [, 请求时传入的数据对象 ] [, 请求成功时的回调函数 ] )

9 $.ajax({

10 dataType: "json",

11 url: "服务器请求地址",

12 data: 请求时传入的数据对象,

13 success: 请求成功时的回调函数

14 });

## 5.6 小结

通过本章的学习，了解了Web网页的基本组成部分，并且深入了解部分非常重要的概念，首先深入分析了HTML5中语义标签的作用和使用方法，帮助开发人员树立正确的HTML编写思路，其次详细分析深入分析了面向对象的CSS构建，通过这部分的构建让开人员可以知道多用组合等优秀的设计理念，然后使用大量篇幅讲解了面向对象JavaScript编程方法，特别是如何在JavaScript中实现面向对象三大基本要素的方法，以及JavaScript中原对象的使用，最后讲了一个JavaScript高级封装库JQuery，通过JQuery来实现DOM的操作、事件的响应，以及HTTP通信，通过这些知识的学习读者基本可以有一个基于现代编程思想的整体Web前段开发理念，而不是采用头疼医头脚疼医脚的局部编程方法。但是本章的内容实际上还有很多可以深入的方法，例如JQuery的动画使用，JavaScript的设计模式等，这些对于今后的编程都是莫大的好处。