2024-2025-1 Web 期末纸质资料（个人整理）

2024 年 12 月 9 日星期一

复习课说明

期末考试全英文
题型分布
单选题 20 分
解答题每题 5 分、程序填空，sql 考核升级？30 分
程序阅读题，10 分，三题一共 30 分
第一道 html+javascript
第二道 php
第三道 php+mysql
编程题两道，10 分，两道 20 分

第一章

p4 internet IPV4 IPV6

对于人员，Internet 节点由名称标识;对于计算机，它们由数字地址标识。这种关系与程序中的变量名称（用于人员）与变量的数字内存地址（用于机器）之间的关系完全相似。 连接到 Internet 的计算机的 Internet 协议 （IP） 地址是一个唯一的 32 位数字。IP 地址通常写 （和认为） 为四个 8 位数字，用句点分隔。Internet 路由计算机分别使用这四个部分来决定邮件下一步必须到达目的地的位置。 组织被分配了 IP 块，然后他们又将这些 IP 块分配给需要 Internet 访问的计算机 — 现在几乎包括所有计算机。例如，可能会为小型组织分配 256 个 IP 地址，例如 191.57.126.0 到 191.57.126.255。可能会为非常大的组织（如国防部）分配 1600 万个 IP 地址，其中包括具有一个特定前 8 位数字的 IP 地址，例如 12.0.0.0 到 12.255.255.255。 尽管人们几乎总是在浏览器中输入域名，但 IP 也同样有效。例如， United Airlines （www.ual.com） 的 IP 为 209.87.113.93。因此，如果浏览器指向 http://209.87.113.93，它将连接到联合航空公司网站。 1998 年底，新的 IP 标准 IPv6 获得批准，尽管它仍未得到广泛使用。最显著的变化是将地址大小从 32 位扩展到 128 位。这一变化很快	For people, Internet nodes are identified by names; for computers, they are identified by numeric addresses. This relationship exactly parallels the one between a variable name in a program, which is for people, and the variable’s numeric memory address, which is for the machine. The Internet Protocol (IP) address of a machine connected to the Internet is a unique 32-bit number. IP addresses usually are written (and thought of) as four 8-bit numbers, separated by periods. The four parts are separately used by Internet-routing computers to decide where a message must go next to get to its destination. Organizations are assigned blocks of IPs, which they in turn assign to their machines that need Internet access—which now include virtually all comput-ers. For example, a small organization may be assigned 256 IP addresses, such as 191.57.126.0 to 191.57.126.255. Very large organizations, such as the Department of Defense, may be assigned 16 million IP addresses, which include IP addresses with one particular first 8-bit number, such as 12.0.0.0 to 12.255.255.255. Although people nearly always type domain names into their browsers, the IP works just as well. For example, the IP for United Airlines (www.ual.com) is 209.87.113.93. So, if a browser is pointed at http://209.87.113.93, it will be connected to the United Airlines Web site. In late 1998, a new IP standard, IPv6, was approved, although it still is not widely used. The most significant change was to expand the address size from 32 bits to 128 bits. This is a change that will soon be

就会变得必不可少，因为剩余未使用的 IP 地址的数量正在迅速减少。	essential because the number of remaining unused IP addresses is diminishing rapidly.

p6 DNS 域名服务器以及 WWW

由于人们处理和记住数字有困难，互联网中的机器也有文本名称。这些名称以主机机器的名称开始，后面跟着越来越大的机器集合，称为域。域名可能有两个、三个或更多。第一个域名紧接在主机名称之后，它是主机所在的域。第二个域名表示第一个域所在的域。最后一个域名标识主机所在组织的类型，这是网站名称中的最大域。 在美国，edu 是教育机构的扩展名，com 表示公司，gov 用于美国政府，org 用于其他各种组织。在其他国家，最大的域通常是该国的缩写，例如，se 用于瑞典，kz 用于哈萨克斯坦。考虑以下示例地址：movies.marxbros.comedy.com。 在这里，movies 是主机名称，marxbros 是 movies 的本地域，它是 comedy 域的一部分，而 comedy 又是 com 域的一部分。主机名称和所有域名一起称为完全限定域名（Fully Qualified Domain Name）。由于 IP 地址是互联网内部使用的地址，浏览器用户提供的完全限定域名必须转换为 IP 地址，才能通过互联网将消息传输到目标。这个转换过程由名为域名系统（DNS）的服务器系统完成。DNS 服务器为连接到互联网的机器群体提供服务，由负责该部分互联网的组织操作。如果名称服务器能够将完全限定域名转换为 IP 地址，它会执行该转换。如果不能，它会将完全限定域名发送给另一个名称服务器进行转换。像 IP 地址一样，完全限定域名必须是唯一的。图 1.1 显示了浏览器请求的完全限定域名如何在转换为 IP 后被路由到相应的 Web 服务器。 确定网站的 IP 地址的一种方法是使用 telnet 命令查询完全限定域名。在 1.7.1 节中有相关的说明。	Because people have difficulty dealing with and remembering numbers, machines on the Internet also have textual names. These names begin with the name of the host machine, followed by progressively larger enclosing collections of machines, called domains. There may be two, three, or more domain names. The first domain name, which appears immediately to the right of the host name, is the domain of which the host is a part. The second domain name gives the domain of which the first domain is a part. The last domain name identifies the type of organization in which the host resides, which is the largest domain in the site’s name. For organizations in the United States, edu is the extension for educational institutions, com specifies a company, gov is used for the U.S. government, and org is used for many other kinds of organizations. In other countries, the larg est domain is often an abbreviation for the country—for example, se is used for Sweden, and kz is used for Kazakhstan. Consider this sample address: movies.marxbros.comedy.com Here, movies is the host name and marxbros is movies’s local domain, which is a part of comedy’s domain, which is a part of the com domain. The host name and all the domain names are together called a fully qualified domain name. Because IP addresses are the addresses used internally by the Internet, the fully qualified domain name of the destination for a message, which is what is given by a browser user, must be converted to an IP address before the mes sage can be transmitted over the Internet to the destination. These conversions are done by software systems called name servers, which implement the Domain Name System (DNS). Name servers serve a collection of machines on the Inter net and are operated by organizations that are responsible for the part of the Internet to which those machines are connected. All document requests from browsers are routed to the nearest name server. If the name server can convert the fully qualified domain name to an IP address, it does so. If it cannot, the name server sends the fully qualified domain name to another name server for conver sion. Like IP addresses, fully qualified domain names must be unique. Figure 1.1 shows how fully qualified domain names requested by a browser are translated into IPs before they are routed to the appropriate Web server. One way to determine the IP address of a Web site is by using telnet on the fully qualified domain name. This approach is illustrated in Section 1.7.1. By the mid-1980s, a collection

到 1980 年代中期，基于 TCP/IP 的不同协议集已经开发出来，以支持各种互联网用途。最常见的协议包括 telnet（允许互联网中的一台计算机用户登录并使用另一台计算机），文件传输协议（ftp，用于在计算机之间传输文件），Usenet（作为电子公告板），以及 mailto（允许互联网中的一台计算机的用户向其他计算机的用户发送消息）。这些协议各自有不同的用户界面，仅适用于其设计目的，这限制了互联网的发展。用户必须学习所有不同的界面才能全面利用互联网。然而，很快就开发出了一种更好的方法：万维网（World Wide Web，简称 Web）。	of different protocols that run on top of TCP/ IP had been developed to support a variety of Internet uses. Among these proto cols, the most common were telnet, which was developed to allow a user on one computer on the Internet to log onto and use another computer on the Internet; File Transfer Protocol (ftp), which was developed to transfer files among com puters on the Internet; Usenet, which was developed to serve as an electronic bulletin board; and mailto, which was developed to allow messages to be sent from the user of one computer on the Internet to other users of other computers on the Internet. This variety of protocols, each having its own user interface and useful only for the purpose for which it was designed, restricted the growth of the Internet. Users were required to learn all the different interfaces to gain all the advantages of the Internet. Before long, however, a better approach was developed: the World Wide Web.

1989 年，由蒂姆·伯纳斯-李（Tim Berners-Lee）领导的一小组人，在欧洲核子研究中心（CERN）提出了一个新的互联网协议，并提出了用于该协议的文档访问系统。这个新系统的目的，是让全球的科学家能够通过互联网交换描述他们工作的文档。这个新系统的设计旨在让互联网上的用户可以搜索并检索位于任何多台不同文档服务器计算机上的文档。到 1990 年底，新的系统的基本思想已经完全开发并在 CERN 的 NeXT 计算机上实现。 1991 年，该系统被移植到其他计算机平台，并向全球发布。该系统使用超文本作为文档的形式，超文本是带有嵌入链接的文本，链接指向同一文档或另一文档，允许非顺序地浏览文本材料。超文本的概念早在 1980 年代中期就已被提出，并出现在 Xerox 的 NoteCards 和 Apple 的 HyperCard 中。从此，我们将只称这个新系统为万维网（Web）。 Web 上的信息单元有多个名称，最常见的包括页面、文档和资源。最合适的名称是“文档”，尽管这似乎只暗示了文本。页面（Pages）这一术语广泛使用，但它是误导性的，因为 Web 上的信息单元通常不仅仅是构成印刷媒体的单一页面。我们更倾向	In 1989, a small group of people led by Tim Berners-Lee at Conseil Européen pour la Recherche Nucléaire (CERN) or European Organization for Particle Physics proposed a new protocol for the Internet, as well as a system of docu ment access to use it.3 The intent of this new system, which the group named the World Wide Web, was to allow scientists around the world to use the Internet to exchange documents describing their work. The proposed new system was designed to allow a user anywhere on the Internet to search for and retrieve documents from databases on any number of different document-serving computers connected to the Internet. By late 1990, the basic ideas for the new system had been fully developed and implemented on a NeXT computer at CERN. In 1991, the system was ported to other computer platforms and released to the rest of the world. For the form of its documents, the new system used hypertext, which is text with embedded links to text, either in the same document or in another docu ment, to allow nonsequential browsing of textual material. The idea of hypertext had been developed earlier and had appeared in Xerox’s NoteCards and Apple’s HyperCard in the mid-1980s. From here on, we will refer to the World Wide Web simply as the Web. The units of information on the Web have been referred to by several different names; among them, the most common are pages, documents, and resources. Perhaps the best of these is documents, although that seems to imply only text. Pages is widely used, but it is misleading in that Web units of information often have more than one of the kind of pages that make up printed media. There is some merit to call ing these units resources, because that covers the possibility of

于将这些单元称为“文档”，尽管它们通常不仅仅是文本，还可能包含图像、声音录音或其他类型的媒体。当一个文档包含非文本信息时，它被称为超媒体。在抽象的层面上，Web 是一个庞大的文档集合，其中一些文档通过链接相互连接。这些文档是通过 Web 浏览器访问的（在 1.3 节介绍），由 Web 服务器提供（在 1.4 节介绍）。 理解互联网和 Web 的区别非常重要。互联网是一个由设备和计算机连接组成的网络，这些设备和计算机通过通信设备相互连接。而 Web 是一个安装在互联网大部分计算机上的软件和协议集合。这些计算机中的一些运行 Web 服务器，提供文档；大多数运行 Web 客户端或浏览器，向服务器请求文档并将其显示给用户。在 Web 被开发之前，互联网已经是非常有用的，且即使没有 Web，互联网依然有用。然而，现在大多数互联网用户都是通过 Web 来使用互联网的。	nontextual informa tion. This book will use documents and pages more or less interchangeably, but we prefer documents in most situations. Documents are sometimes just text, usually with embedded links to other documents, but they often also include images, sound recordings, or other kinds of media. When a document contains nontextual information, it is called hypermedia. In an abstract sense, the Web is a vast collection of documents, some of which are connected by links. These documents are accessed by Web browsers, intro duced in Section 1.3, and are provided by Web servers, introduced in Section 1.4. It is important to understand that the Internet and the Web are not the same thing. The Internet is a collection of computers and other devices connected by equipment that allows them to communicate with each other. The Web is a col lection of software and protocols that has been installed on most, if not all, of the computers on the Internet. Some of these computers run Web servers, which pro vide documents, but most run Web clients, or browsers, which request documents from servers and display them to users. The Internet was quite useful before the Web was developed, and it is still useful without it. However, most users of the Internet now use it through the Web

p11

统一资源标识符（URI）用于标识互联网中的资源（通常是文档）。URI 有两种不同的用途：一种是命名资源，在这种情况下它们通常被称为 URI，尽管更准确地应该称为统一资源名称（URN）。URI 的另一种常见形式是提供资源的路径或位置，这时它们被称为统一资源定位符（URL）。URI 和 URL 的通用形式相似，因此它们常常被混淆。我们将在第 8 章中使用 URI 来命名用于 XML Schema 的命名空间，而在本章中，我们只讨论 URL。 所有的 URL 具有相同的一般格式：scheme:object-address。其中，scheme 通常是通信协议。常见的协议包括：http、ftp、gopher、telnet、file、mailto 和 news。不同的协议使用不同形式的对象地址。在本书中，我们关注的是 HTTP 协议，它支持 Web。该协议用于请求和发送超文本标记语言（HTML）文档。对于 HTTP 协议，URL 的对象地址形式如下 //fully-qualified-domain-name/path-to-d	Uniform (or universal)8 Resource Identifiers (URIs) are used to identify resources (often documents) on the Internet. URIs are used for two different purposes, to name a resource, in which case they are often called URIs, even though they could be more accurately called Uniform Resource Names (URNs). The more commonly used form of URIs is to provide a path to, or location of, a resource, in which case they are called Uniform Resource Locators (URLs). The general forms of URIs and URLs are similar, and URIs are often confused with URLs. We will use URIs in Chapter 8 to name namespaces for use with XML Schema. In this chapter, we only deal with URLs. All URLs have the same general format: scheme:object-address The scheme is often a communications protocol. Common schemes include http, ftp, gopher, telnet, file, mailto, and news. Different schemes use object addresses that have different forms. Our interest here is in the HTTP protocol, which supports the Web. This protocol is used to request and send Hypertext Markup Language (HTML) documents. In the case of HTTP, the form of the object address of a URL is as follows: //fully-qualified-domain-name/path-to-document

ocument 另一个我们关心的协议是 file 协议。file 协议表示文档位于运行浏览器的机器上。这种方法对于测试将要发布到 Web 上的文档非常有用，因为它可以在不让其他浏览器访问的情况下进行测试。当使用 file 协议时，完全限定域名被省略，URL 的形式如下： file://path-to-document 由于本书的重点是 HTML 文档，因此后续的 URL 讨论仅限于 HTTP 协议。主机名是存储文档（或提供访问该文档，尽管它存储在其他计算机上）的服务器计算机的名称。发送到主机机器的消息必须指向适当的进程，这些进程通过其关联的端口号进行识别。Web 服务器进程的默认端口号是 80。如果服务器配置为使用其他端口号，则需要将该端口号附加到主机名上。例如，如果 Web 服务器配置为使用端口 800，则 URL 中的主机名必须附加:800。URL 中不能包含嵌入的空格。还有一组特殊字符，包括分号、冒号和与号（;、:、&），这些字符不能出现在 URL 中。如果需要在 URL 中包含空格或这些不允许的特殊字符，必须将该字符编码为百分号（%）后跟该字符的两位十六进制 ASCII 代码。例如，如果“San Jose”是域名，它必须写成 San%20Jose（20 是空格的十六进制 ASCII 代码）。有关 URL 的所有详细信息，可以参见 http://www.w3.org/Addressing/URL/URI_Overview.html。	Another scheme of interest to us is file. The file protocol means that the document resides on the machine running the browser. This approach is useful for testing documents to be made available on the Web without making them vis ible to any other browser. When file is the protocol, the fully qualified domain name is omitted, making the form of such URLs as follows: file: //path-to-document Because the focus of this book is HTML documents, the remainder of the discussion of URLs is limited to the HTTP protocol. The host name is the name of the server computer that stores the document (or provides access to it, although it is stored on some other computer). Messages to a host machine must be directed to the appropriate process running on the host for handling. Such processes are identified by their associated port numbers. The default port number of Web server processes is 80. If a server has been configured to use some other port number, it is necessary to attach that port number to the host name in the URL. For example, if the Web server is configured to use port 800, the host name must have :800 attached. URLs can never have embedded spaces.9 Also, there is a collection of spe cial characters, including semicolons, colons, and ampersands (;, :, &), that cannot appear in a URL. To include a space or one of the disallowed special characters, the character must be coded as a percent sign (%) followed by the two-digit hexadecimal ASCII code for the character. For example, if San Jose is a domain name, it must be typed as San%20Jose (20 is the hexadecimal ASCII code for a space). All the details characterizing URLs can be found at http://www.w3.org/Addressing/URL/URI_Overview.html.

p13 MIME 是个什么玩意

浏览器需要某种方式来确定它从 Web 服务器接收到的文档的格式。如果不知道文档的格式，浏览器将无法渲染它，因为不同的文档格式需要不同的渲染软件。这些文档的格式通过多用途互联网邮件扩展（MIME）来指定。 MIME 是为了指定通过互联网邮件发送的不同类型文档的格式而开发的。这些文档可能包含各种类型的文本、视频数据或声音数据。由于 Web 的需求类似于互联网邮件，因此 MIME 被采纳为指定通过 Web 传输的文档类型的方式。Web 服务器会在提供给浏览器的文档开	A browser needs some way to determine the format of a document it receives from a Web server. Without knowing the form of the document, the browser would not be able to render it, because different document formats require dif ferent rendering software. The forms of these documents are specified with Mul tipurpose Internet Mail Extensions (MIMEs) MIME was developed to specify the format of different kinds of documents to be sent via Internet mail. These documents could contain various kinds of text, video data, or sound data. Because the Web has needs similar to those of Inter net mail, MIME was adopted as the way to specify document types transmitted over the Web. A Web server

始处附加 MIME 格式规范。当浏览器从 Web 服务器接收到文档时，它使用包含的 MIME 格式规范来确定如何处理该文档。

例如，如果内容是文本，MIME 代码告诉浏览器它是文本，并且还指明它是什么类型的文本。如果内容是声音，MIME 代码则告诉浏览器它是声音，并提供声音的具体表示方式，以便浏览器可以选择一个程序来播放传输的声音。

MIME 规范的格式为：type/subtype。最常见的 MIME 类型是 text、image 和 video。最常见的 text 子类型是 plain 和 html。一些常见的 image 子类型是 gif 和 jpeg。一些常见的 video 子类型是 mpeg 和 quicktime。

每个 Web 服务器的配置文件中都存储有 MIME 规范的列表。本文接下来的部分中，当我们提到文档类型时，我们指的是文档的 type 和 subtype。服务器通过使用文件名扩展名作为键来查找类型表，以确定文档的类型。例如，扩展名为.html 的文件告诉服务器，在发送文档之前，应该附加 text/html 类型。浏览器也会维护一张转换表，通过文件名扩展名查找文档类型。但是，这张表仅在服务器未指定 MIME 类型时使用，这在一些较老的服务器中可能会发生。在其他情况下，浏览器从服务器提供的 MIME 头部获取文档类型。

有时会使用实验性的子类型。实验性子类型的名称以 x- 开头，如 video/x-msvideo。任何 Web 提供商都可以通过将其名称添加到 MIME 规范列表中，向 Web 服务器添加实验性子类型。例如，Web 提供商可能有一个手工制作的数据库，想通过 Web 共享给其他人。当然，这也引发了浏览器如何显示该数据库内容的问题。通常，Web 提供商需要提供一个程序，浏览器在需要显示该数据库内容时可以调用该程序。这些程序可以是外部的，称为辅助应用程序（helper applications），或者是插入到浏览器中的代码模块，称为插件（plug-ins）。

每个浏览器都有一组它能够处理的 MIME 规范（文件类型）。所有浏览器都能处理 text/plain（未格式化文本）和 text/html（HTML 文件）等类型。有时，

attaches an MIME format specification to the beginning of the document that it is about to provide to a browser. When the browser receives the document from a Web server, it uses the included MIME format specification to determine what to do with the document.

If the content is text, for example, the MIME code tells the browser that it is text and also indicates the particular kind of text it is. If the content is sound, the MIME code tells the browser that it is sound and then gives the particular representation of sound so the browser can choose a program to which it has access to produce the transmitted sound

MIME specifications have the following form: type/subtype The most common MIME types are text, image, and video. The most common text subtypes are plain and html. Some common image subtypes are gif and jpeg. Some common video subtypes are mpeg and quicktime.

A list of MIME specifications is stored in the configuration files of every Web server. In the remainder of this book, when we say document type, we mean both the type and subtype of the document. Servers determine the type of a document by using the file name exten sion as the key into a table of types. For example, the extension .html tells the server that it should attach text/html to the document before sending it to the requesting browser.10 Browsers also maintain a conversion table for looking up the type of a docu ment by its file name extension. However, this table is used only when the server does not specify an MIME type, which may be the case with some older servers. In all other cases, the browser gets the document type from the MIME header provided by the server

Experimental subtypes are sometimes used. The name of an experimental sub type begins with x-, as in video/x-msvideo. Any Web provider can add an experimental subtype by having its name added to the list of MIME specifica tions stored in the Web provider's server. For example, a Web provider might have a handcrafted database whose contents he or she wants to make available to others through the Web. Of course, this raises the issue of how the browser can display the database. As might be expected, the Web provider must supply a program that the browser can call when it needs to display the contents of the database. These programs either are external to the browser, in which case they are called helper applications, or are code modules that are inserted into the browser, in which case they are called plug-ins.

Every browser has a set of MIME specifications (file types) it can handle. All can deal with text/plain (unformatted text) and text/html (HTML files), among

某些浏览器无法处理特定的文档类型，尽管该类型被广泛使用。在这种情况下，浏览器会像处理实验性类型一样处理这些情况。浏览器通过检查浏览器配置文件来确定它需要的辅助应用程序或插件，配置文件提供了文件类型与所需的辅助应用程序或插件之间的关联。如果浏览器没有它需要的应用程序或插件来渲染文档，则会显示错误消息。 浏览器还可以向服务器指示它更喜欢接收的文档类型，正如第 1.7 节中讨论的那样。	others. Sometimes a particular browser cannot handle a specific document type, even though the type is widely used. These cases are handled in the same way as the experimental types described previously. The browser determines the helper application or plug-in it needs by examining the browser configuration file, which provides an association between file types and their required helpers or plug-ins. If the browser does not have an application or a plug-in that it needs to render a document, an error message is displayed. A browser can indicate to the server the document types it prefers to receive, as discussed in Section 1.7.

p16 HTTP 是什么，方法 1.7.1,1.7.2

所有的 Web 通信事务都使用相同的协议：超文本传输协议（HTTP）。当前版本的 HTTP 是 1.1，正式定义为 RFC 2616，已于 1999 年 6 月批准。RFC 2616 可在万维网联盟（W3C）的网站上找到：http://www.w3.org。本节提供了对 HTTP 的简要介绍。HTTP 由两个阶段组成：请求和响应。每次浏览器与 Web 服务器之间的 HTTP 通信（无论是请求还是响应）都由两部分组成：头部和正文。头部包含通信的相关信息，正文包含通信的数据（如果有的话）。

HTTP 请求的一般形式如下：

1. HTTP 方法 域名部分的 URL HTTP 版本
2. 头部字段
3. 空行
4. 消息正文

以下是 HTTP 请求第一行的示例：

GET /storefront.html HTTP/1.1

HTTP 定义了少数几种请求方法，实际上通常使用的更少。表 1.1 列出了最常用的几种方法。

方法	描述
GET	返回指定文档的内容
HEAD	返回指定文档的头部信息
POST	执行指定文档，使用其中的封装数据
PUT	用封装的数据替换指定文档
DELETE	删除指定文档

在表 1.1 中列出的这些方法中，GET 和 POST 是最常用的。POST 最初设计用于诸如将新闻文章发布到新闻组这样的任务。现在，

All Web communications transactions use the same protocol: the Hypertext Transfer Protocol (HTTP). The current version of HTTP is 1.1, formally defined as RFC 2616, which was approved in June 1999. RFC 2616 is available at the Web site for the World Wide Web Consortium (W3C), http://www .w3.org. This section provides a brief introduction to HTTP. HTTP consists of two phases: the request and the response. Each HTTP communication (request or response) between a browser and a Web server con sists of two parts: a header and a body. The header contains information about the communication; the body contains the data of the communication if there is any.

The general form of an HTTP request is as follows:

1. HTTP method Domain part of the URL HTTP version
2. Header fields
3. Blank line
4. Message body

The following is an example of the first line of an HTTP request: GET /storefront.html HTTP/1.1 Only a few request methods are defined by HTTP, and even a smaller num ber of these are typically used. Table 1.1 lists the most commonly used methods.

Method	Description
Get	Returns the content of a specified document
HEAD	Returns the header information for a specified document
POST	Executes a specified document,using the enclosed data
PUT	Replaces a specified document with the enclosed data
DELETE	Deletes a specified document

Among the methods given in Table 1.1, GET and POST

它最常见的用途是发送表单数据从浏览器到服务器，并请求服务器上运行一个程序来处理这些数据。HTTP 通信的第一行之后是任意数量的头部字段，大多数字段是可选的。头部字段的格式是字段名后跟冒号和字段值。头部字段分为四类：

1. 一般：用于一般信息，如日期
2. 请求：包含在请求头中的字段
3. 响应：用于响应头中的字段
4. 实体：在请求和响应头中都使用的字段

一个常见的请求字段是 Accept 字段，它指定浏览器对于请求文档的 MIME 类型的偏好。如果浏览器愿意接受多个格式的文档，可以指定多个 Accept 字段。例如，我们可以看到以下几种：

Accept: text/plain

Accept: text/html

Accept: image/gif

可以使用通配符字符“*”代替 MIME 类型的一部分，这时该部分可以是任意内容。例如，如果可以接受任何类型的文本，Accept 字段可以这样表示：

Accept: text/*

Host: host name 请求字段指定了主机名。Host 字段是 HTTP 1.1 的要求。If-Modified-Since: date 请求字段表示只有在文件自指定日期以来被修改过时，才返回文件。如果请求中有正文，则必须使用 Content-length 字段来指定正文的长度，表示响应正文的字节数。POST 方法请求需要这个字段，因为它们会向服务器发送数据。请求的头部之后必须跟一个空行，用来将头部和请求正文分开。使用 GET、HEAD 和 DELETE 方法的请求没有正文，因此这些方法的请求通过空行来表示请求的结束。

与 Web 服务器通信不一定需要使用浏览器；也可以通过 telnet 命令来实现。假设你在任何常用操作系统的命令行中输入以下命令：

>telnet blanca.uccs.edu http

这条命令会创建一个到 blanca.uccs.edu 服务器上 http 端口的连接。服务器的响应是：

Trying 128.198.162.60 ...

Connected to blanca

Escape character is '^]'.

此时与服务器的连接已建立，可以输入如下 HTTP 命令：

GET /~user1/respond.html HTTP/1.1

Host: blanca.uccs.edu

are the most fre quently used. POST was originally designed for tasks such as posting a news article to a newsgroup. Its most common use now is to send form data from a browser to a server, along with a request to execute a server-resident program on the server that will process the data. Following the first line of an HTTP communication is any number of header fields, most of which are optional. The format of a header field is the field name

followed by a colon and the value of the field. There are four categories of header fields:

1. General: For general information, such as the date
2. Request: Included in request headers
3. Response: For response headers
4. Entity: Used in both request and response headers One common request field is the Accept field, which specifies a preference of the browser for the MIME type of the requested document. More than one Accept field can be specified if the browser is willing to accept documents in more than one format. For example, we might have any of the following: Accept: text/plain Accept: text/html Accept: image/gif A wildcard character, the asterisk (*), can be used in place of a part of a MIME type, in which case that part can be anything. For example, if any kind of text is acceptable, the Accept field could be as follows: Accept: text/* The Host: host name request field gives the name of the host. The Host field is required for HTTP 1.1. The If-Modified-Since: date request field specifies that the requested file should be sent only if it has been modified since the given date. If the request has a body, the length of that body must be given with a Content-length field, which gives the length of the response body in bytes. The POST method requests require this field because they send data to the server. The header of a request must be followed by a blank line, which is used to separate the header from the body of the request. Requests that use the GET, HEAD, and DELETE methods do not have bodies. In these cases, the blank line signals the end of the request. A browser is not necessary to communicate with a Web server; telnet can be used instead. Consider the following command, given at the command line of any widely used operating system: >telnet blanca.uccs.edu http This command creates a connection to the http port on the blanca.uccs . edu server. The server responds with the following:11 Trying 128.198.162.60 ... Connected to blanca Escape character is ’^]’.

The connection to the server is now complete, and HTTP commands such as the following can be given: GET

这个请求的响应头部会在第 1.7.2 节给出。

HTTP 响应的一般形式如下：

1. 状态行
2. 响应头字段
3. 空行
4. 响应正文

状态行包含使用的 HTTP 版本、一个三位数的响应状态码，以及对该状态码的简短文本说明。例如，大多数响应都以以下内容开始：

HTTP/1.1 200 OK

状态码的首位数字为 1、2、3、4 或 5。表 1.2 显示了这些数字所表示的五类的含义。

首位数字	类别
1	信息性
2	成功
3	重定向
4	客户端错误
5	服务器错误

其中一个用户最不愿意看到的状态码是 404 Not Found，它意味着请求的文件没有找到。200 OK 是用户希望看到的状态码，表示请求已被成功处理。500 状态码意味着服务器遇到了问题，无法完成请求。状态行之后，服务器会发送响应头部，可能包含几行关于响应的信息，每行的格式为字段。响应头中唯一必须包含的字段是 Content-type。

以下是第 1.7.1 节给出请求的响应头部的示例：

HTTP/1.1 200 OK

Date: Sat, 25 July 2009 22:15:11 GMT

Server: Apache/2.2.3 (CentOS)

Last-modified: Tues, 18 May 2004 16:38:38 GMT

ETag: "1b48098-16c-3dab592dc9f80"

Accept-ranges: bytes

Content-length: 364

Connection: close

Content-type: text/html, charset=UTF-8

响应头部后必须跟一个空行，这和请求头部的格式相同。空行之后是响应数据。在前面的示例中，响应正文将是 HTML 文件 respond.html。HTTP 1.1 的默认行为是保持连接一段时间，以便客户端可以在短时间内发送多个请求，而无需重新建立与服务器的连接。

/~user1/respond.html HTTP/1.1 Host: blanca.uccs.edu

The header of the response to this request is given in Section 1.7.2.

The general form of an HTTP response is as follows: 1. Status line 2. Response header fields 3. Blank line 4. Response body The status line includes the HTTP version used, a three-digit status code for the response, and a short textual explanation of the status code. For example, most responses begin with the following: HTTP/1.1 200 OK The status codes begin with 1, 2, 3, 4, or 5. The general meanings of the five categories specified by these first digits are shown in Table 1.2

First Digit	Category
1	Informational
2	Success
3	Redirection
4	Client error
5	Server error

One of the more common status codes is one users never want to see: 404 Not Found, which means the requested file could not be found. Of course, 200 OK is what users want to see, because it means that the request was handled with out error. The 500 code means that the server has encountered a problem and was not able to fulfill the request. After the status line, the server sends a response header, which can contain several lines of information about the response, each in the form of a field. The only essential field of the header is Content-type.

The following is the response header for the request given near the end of Section 1.7.1: HTTP/1.1 200 OK Date: Sat, 25 July 2009 22:15:11 GMT Server: Apache/2.2.3 (CentOS) Last-modified: Tues, 18 May 2004 16:38:38 GMT ETag: "1b48098-16c-3dab592dc9f80" Accept-ranges: bytes Content-length: 364 Connection: close Content-type: text/html, charset=UTF-8 The response header must be followed by a blank line, as is the case for request headers. The response data follows the blank line. In the preceding exam ple, the response body would be the HTML file, respond.html. The default operation of HTTP 1.1 is that the connection is kept open for a time so that the client can make several requests over a short span of time without needing to reestablish the communications connection with the server

第二章

全掌握 2.2 2.3 2.4 2.5 但不包括 2.5.3　2.6 2.7 2.8 2.9 其中 2.9.3 的 select 菜单要会写（ppt 没讲的不用看）

HTML 基本框架和标签、格式化 P38-P48

基本文档结构	基本标签
<!DOCTYPE html> <head> </head> <body> </body> </html>	最大的标题 . . . . 最小的标题 这是一个段落。 （ 换行 ） <hr>（ 水平线 ） <!-- 这是注释 -->
<img src="*url*" alt="*some_text*"> #图片	

Here is the table in text format:

Character	Entity	Meaning
&	&amp;	Ampersand
<	&lt;	Is less than
>	&gt;	Is greater than
"	&quot;	Double quote
'	&apos;	Single quote (apostrophe)
¼	&frac14;	One-quarter
½	&frac12;	One-half
¾	&frac34;	Three-quarters
°	&deg;	Degree
(space)		Nonbreaking space
©	&copy;	Copyright
€	&euro;	Euro

Web 搜索引擎使用 meta 元素提供的信息在其索引中对 Web 文档进行分类。因此，如果文档的作者希望该文档得到广泛曝光，则会包含一个或多个元元素，以确保 Web 搜索引擎能够找到该文档。 同时，其他格式化输出的形式如下表所示。	Web search engines use the information provided by the meta element to categorize web documents in their indexes. Therefore, if the author of the document wants the document to gain widespread exposure, they will include one or more meta elements to ensure that web search engines can find the document. At the same time, other formatting outputs are shown in the table below.

格式化输出	
粗体文本 <code>计算机代码</code> 强调文本 斜体文本 <kbd>键盘输入</kbd> <pre>预格式化文本</pre> 更小的文本 重要的文本	<abbr>（缩写） <address>（联系信息） <bdo>（文字方向） <blockquote>（从另一个源引用的部分） <cite>（工作的名称） <del>（删除的文本） <ins>（插入的文本） <sub>（下标文本） <sup>（上标文本）

列表、表格 P52~P69

有序列表	无序列表
  - 第一项 - 第二项 	 - 项目 - 项目
定义列表	**表格**
<dl> <dt>项目 1</dt> <dd>描述项目 1</dd> <dt>项目 2</dt> <dd>描述项目 2</dd> </dl>	<table border="border"> <tr> <th>表格标题</th> <th>表格标题</th> </tr> <tr> <td>表格数据</td> <td>表格数据</td> </tr> </table>

Form 表单 P70~P82

空表单	包含所有类型控件的表单
<form action = ""> </form>	<form action="demo_form.php" method="post/get"> <input type="text" name="email" size="40" maxlength="50"> <input type="password"> <input type="checkbox" name=”xxx” value=”xxx” checked="checked"> <input type="radio" checked="checked"> <input type="submit" value="Send"> <input type="reset"> <input type="hidden"> <select>

	<option>苹果</option> <option selected="selected">香蕉</option> <option>樱桃</option> </select> <textarea name="comment" rows="60" cols="20"></textarea> </form>

例一（教材例子）	例二（教材例子）	例三（教材例子）
<form action = ""> With size = 1 (the default) <select name = "groceries"> <option> milk </option> <option> bread </option> <option> eggs </option> <option> cheese </option> </select> </form>	<form action = "handler"> <textarea name = "aspirations" rows = "3" cols = "40"> (Be brief and concise) </textarea> </form>	<form action = ""> <input type = "submit" value = "Submit Form" /> <input type = "reset" value = "Reset Form" /> </form>

音视频组件 P83~ P84

浏览器选择它可以播放的第一个音频文件，并跳过 audio 元素的内容。如果它无法播放源元素中出现的任何音频文件，则它只显示其内容。遗憾的是，不同的浏览器能够播放不同的音频容器/编解码器组合。Firefox 3.5+ 浏览器支持 Ogg/Vorbis 和 Wav/Wav 容器/编解码器音频文件。Chrome 3.0+ 浏览器支持 Ogg/Vorbis 和 MP3/MP3 容器/编解码器音频文件。IE9+ 浏览器支持 MP3/MP3 容器/编解码器音频文件。Safari 3.0+ 浏览器支持 Wav/Wav 容器/编解码器音频文件。	The browser selects the first audio file it can play and skips the content of the audio element. If it cannot play any of the audio files in the source elements, it will only display its content. Unfortunately, different browsers can play different audio container/codec combinations. Firefox 3.5+ browsers support Ogg/Vorbis and Wav/Wav container/codec audio files. Chrome 3.0+ browsers support Ogg/Vorbis and MP3/MP3 container/codec audio files. IE9+ browsers support MP3/MP3 container/codec audio files. Safari 3.0+ browsers support Wav/Wav container/codec audio files.

```
<audio attributes>
<source src = "filename1">
...
<source src = "filenamen">
Your browser does not support the audio element
</audio>
```

Web 上最常用的视频容器是 MPEG-4（.mp4 文件）、Flash 视频（.flv 文件）、Ogg（.ogv 文件）、	The most commonly used video containers on the web are MPEG-4 (.mp4 files), Flash Video (.flv files), Ogg (.ogv files), WebM

WebM（.webm 文件）和音频视频交错（.avi 文件）。Web 上最常用的视频编解码器是 H.264（也称为 MPEG-4 高级视频编码或 MPEG-4 AVC），它可以嵌入到 MP4 容器中，Theora 可以嵌入到任何容器中，VP8 可以嵌入到 WebM 容器中。除了视频数据之外，视频容器还存储音频数据，因为大多数视频都伴随着音频。Web 上使用的三种最常见的容器/视频编解码器/音频编解码器组合是带有 Theora 视频编解码器和 Vorbis 音频编解码器的 Ogg 容器、带有 H.264 视频编解码器和 AAC 音频编解码器的 MPEG-4 容器，以及带有 VP8 视频编解码器和 Vorbis 音频编解码器的 WebM 容器。 IE9+ 浏览器支持 MPEG-4 视频容器，Firefox 3.5+ 浏览器支持 Ogg 视频容器，Firefox 4.0+ 浏览器支持 Ogg 和 WebM 视频容器，Chrome 6.0+ 浏览器支持所有三种最常见的视频容器，Safari 3.0+ 浏览器支持 MPEG-4 视频容器。	(.webm files), and Audio Video Interleave (.avi files). The most commonly used video codecs on the web are H.264 (also known as MPEG-4 Advanced Video Coding or MPEG-4 AVC), which can be embedded in MP4 containers, Theora, which can be embedded in any container, and VP8, which can be embedded in WebM containers. In addition to video data, video containers also store audio data, as most videos are accompanied by sound. The three most common container/video codec/audio codec combinations used on the web are the Ogg container with Theora video codec and Vorbis audio codec, the MPEG-4 container with H.264 video codec and AAC audio codec, and the WebM container with VP8 video codec and Vorbis audio codec. IE9+ browsers support the MPEG-4 video container, Firefox 3.5+ browsers support the Ogg video container, Firefox 4.0+ browsers support both Ogg and WebM video containers, Chrome 6.0+ browsers support all three common video containers, and Safari 3.0+ browsers support the MPEG-4 video container.

```
<video attributes>
<source src = "filename1">
...
<source src = "filenamen">
Your browser does not support the video element
</video>
```

Time 组件 P88

section 元素用于封装文档的各个部分，例如书籍的章节或论文的单独部分。页脚元素可以包含一个或多个部分。 article 元素用于封装来自某些外部源（如论坛中的帖子或报纸文章）的文档的自包含部分。article 元素可以包括页眉、页脚和部分。当文档由几个单独编写的部分放在一起时，文章元素很方便。aside 元素用于与文档的主要信息相切的内容。在印刷品中，此类内容通常放在侧边栏中。 nav 元素用于封装导航部分;即，指向文档不同部分的链接列表。nav 元素清楚地标记了文档中用于访问其他文档的部分。它们对于使用文本到语音转换阅读器“查看”文档的视障用户特别有用。	The <section> element is used to encapsulate distinct parts of a document, such as chapters of a book or separate sections of a paper. The <footer> element can contain one or more sections. The <article> element is used to encapsulate a self-contained portion of a document, such as posts from forums or newspaper articles. The <article> element can include a header, footer, and sections. It is particularly useful when a document is composed of several individually written parts. The <aside> element is used for content that is tangentially related to the main content of the document. In print media, this content is often found in sidebars. The <nav> element is used to encapsulate a navigation section; that is, a list of links pointing to different parts of the document. The <nav> element clearly marks sections of a document used for accessing other documents. These elements are especially helpful for visually impaired users using text-to-speech readers to "view" the document.

```
<time datetime = "2011-02-14T08:00" pubdate = "pubdate">
February 14, 2011 8:00am MDT
</time>
```

datetime 元素的值可以包含时区偏移量，例如+14:00 或-12:00。这表示相对于协调世界时（UTC）的时间差。例如，+14:00 表示时间比 UTC 早 14 小时，-12:00 表示比 UTC 晚 12 小时。 偏移量使用“+”或“-”号表示，后面跟随时差值。例如，示例中的时间值 2011-02-14T08:00-06:00 表示 2011 年 2 月 14 日上午 8 点，时区比 UTC 晚 6 小时（-06:00）。 日期时间不能表示公元前的年份，负年份（如-500）也是不被允许的。示例中提到的“不可接受的年份”是指像“大约 1900 年”这种模糊的日期是不被允许的，HTML 中的日期时间必须是明确且精确的。 在使用 HTML 日期时间元素时，必须精确地包含时间和时区信息，且不支持表示公元前的时间或不精确的年份。	The value of the <datetime> element can include a timezone offset, such as +14:00 or -12:00. This indicates the time difference relative to Coordinated Universal Time (UTC). For example, +14:00 means the time is 14 hours ahead of UTC, while -12:00 means it is 12 hours behind UTC. The offset is indicated with a "+" or "-" sign, followed by the time difference. For example, the time value "2011-02-14T08:00-06:00" represents 8:00 AM on February 14, 2011, with a timezone that is 6 hours behind UTC (-06:00). Dates and times cannot represent years before Christ (BC), and negative years (such as -500) are not allowed. The term "unacceptable years" refers to vague dates, such as "around 1900," which are not permissible. The date and time in HTML must be precise and unambiguous. When using the HTML <datetime> element, it is necessary to precisely include the time and timezone information. The representation of dates before the Common Era (BCE) or imprecise years is not supported.

第三章

p97 3.2 三种样式表格的层级结构 3.3 3.4 3.5 如何设置黑体（font 的一些方法）

样式 P99

外部样式表的链接必须出现在文档的头部。如果外部样式表驻留在 Web 服务器计算机上，则只需将其路径地址作为 href 值给出。外部样式表的一个例子出现在第 3.6 节中。 因为将 CSS 与标记分开是很好的，所以最好使用外部样式表。然而，由于本书中的示例文档都相对较小，并且我们希望将 CSS 保留在使用位置附近以便于引用，因此我们在示例中几乎总是使用文档级样式表。最好不要用内联样式，因为复用性不高。	The link to an external style sheet must appear in the head of the document. If the external style sheet is hosted on a web server, only its path address needs to be provided as the href value. An example of an external style sheet is provided in Section 3.6. Since separating CSS from markup is a good practice, it is generally recommended to use external style sheets. However, because the example documents in this book are relatively small and we prefer to keep the CSS close to its usage for ease of reference, we almost always use document-level style sheets in the examples. Inline styles should be avoided as they have low reusability.

文档级 CSS	外部 CSS

<style type = "text/css"> /* Styles for the initial paragraph */ /* Styles for other paragraphs */ </style>	<link rel = "stylesheet" type = "text/css" href = "http://www.cs.usc.edu/styles/wbook.css" />

7 种选择器 P100~P104

一般选择器，组件名称打头。

General selectors, starting with the component name.

html	css
这是一个需要高亮的段落。 这是一个需要高亮的容器。	p{background-color: yellow;} div,h1,h2{属性值列表}

类选择器用于选取具有特定 class 属性的 HTML 元素。使用类选择器，可以为多个元素设置相同的样式，使得样式复用更加方便。在 CSS 中，类选择器以 .（点）符号开头，后跟类名。

Class selectors are used to select HTML elements with a specific class attribute. By using class selectors, you can apply the same styles to multiple elements, making style reuse more convenient. In CSS, class selectors start with a dot (.) followed by the class name.

html	css
这是一个需要高亮的段落。 这是一个需要高亮的段落。	p.highlight {background-color: yellow;} p.height{属性值列表}

泛型选择器（通用选择器）同一类不同组件

Universal selector (wildcard selector) for different components of the same class.

html	css
这是一个需要高亮的段落。 这是一个需要高亮的容器。	.highlight {background-color: yellow;}

ID 选择器用于选取具有特定 id 属性的唯一元素。在一个 HTML 文档中，id 属性的值必须是唯一的。ID 选择器在 CSS 中以 #（井号）开头，后跟 ID 名称。

ID selectors are used to select a unique element with a specific id attribute. In an HTML document, the value of the id attribute must be unique. The ID selector in CSS begins with a # (hash symbol), followed by the ID name.

html	css
<h1 id="main-title">网站主标题</h1>	main-title {font-size: 36px;color: blue;}

伪类用于选择处于特定状态或位置的元素，它们以 :（冒号）开头。伪类可以捕获无法用简单选择器表达的复杂信息，例如鼠标悬停状态、选中状态、元素的第几个子节点等。

Pseudo-classes are used to select elements that are in a specific state or position, and they begin with a colon (:). Pseudo-classes can capture complex information that cannot be expressed with simple selectors, such as the hover state, the selected state, or the nth-child of an element.

常见伪类		
动态伪类	结构性伪类	其他伪类

:hover：当鼠标指针悬停在元素上时。 :active：当元素被激活（通常是被点击）时。 :focus：当元素获得焦点时（如文本输入框被选中）。	:first-child：选择父元素的第一个子元素。 :last-child：选择父元素的最后一个子元素。 :nth-child(n)：选择父元素的第 n 个子元素。	:visited：已访问过的链接。 :disabled：被禁用的表单元素。

html	css
访问示例网站 <button>提交</button> - 项目 1 - 项目 2 - 项目 3 	/* 当鼠标悬停在链接上时，改变文字颜色 */ a:hover {color: red;} /* 当按钮被点击时，改变背景颜色 */ button:active {background-color: green;} /* 为列表中的第一个项目设置样式 */ li:first-child {font-weight: bold;} /* 为已访问的链接设置颜色 */ a:visited {color: purple;}

通用选择器使用 * 符号，表示选择所有元素。它可以单独使用，也可以与其他选择器组合，用于选择特定范围内的所有元素，下面举用例。

The universal selector uses the * symbol, which selects all elements. It can be used alone or combined with other selectors to select all elements within a specific range. Here's an example.

单独使用	与其他选择器组合使用
css * {margin: 0;padding: 0;}	.container * {color: blue;}
这将重置所有元素的外边距和内边距。	这将选择 .container 内的所有元素，并将文字颜色设置为蓝色。

限定范围的通用选择器

Scoped Universal Selector" or "Universal Selector with a Scoped Range.

html	css

 标题
 段落内容。
 	
 - 列表项 1
 - 列表项 2
 	
	/* 仅选择 .content 内的所有元素 */
	.content * {
	font-family: Arial, sans-serif;
	}

这段 CSS 将 .content 容器内的所有元素的字体设置为 Arial。	

总结（言简意赅）

类选择器用于选择具有特定 class 属性的元素。允许多个元素共享相同的样式规则。可以为一个元素指定多个类，使用空格分隔。 ID 选择器用于选择具有特定 id 属性的唯一元素。id 属性的值在整个 HTML 文档中必须是唯一的。主要用于需要单独样式的元素或在 JavaScript 中进行 DOM 操作。	Class selectors are used to select elements with a specific class attribute. They allow multiple elements to share the same style rules. You can assign multiple classes to a single element, separated by spaces. ID selectors are used to select a unique element with a specific ID attribute. The value of the ID attribute must be unique within the entire HTML document. ID selectors are mainly used for elements that

伪类用于选择元素的特定状态或位置。以 ：开头，后跟伪类名称。常用于交互样式（如鼠标悬停、点击）、表单状态和结构性选择。 通用选择器用于选择所有元素或特定范围内的所有元素。常用于重置默认样式或批量设置样式。与其他选择器组合使用时，可以限定选择范围，提高选择器的精确度。 **具体有什么属性可以设置，看附录 A。** **颜色属性值可**以指定为颜色名称、六位十六进制数字或 RGB 形式。 RGB 形式只是单词 rgb 后跟一个括号内的列表，其中包含 0 到 255 范围内的三个十进制数字或三个百分比值。这些数字或百分比分别指定红色、绿色和蓝色的级别。例如，作为三个值中的第一个值的 0 或 0% 将指定颜色中不包含红色。值 255 或 100% 将指定最大红色量。十六进制数字前面必须带有井号 (#)，如#43AF00。例如，紫红色（红色和蓝色的混合物）可以指定为 **fuchsia、rgb(255, 0, 255)、#FF00FF**	need individual styling or for DOM manipulation in JavaScript. Pseudo-classes are used to select elements based on their specific state or position. They start with a colon (:), followed by the pseudo-class name. Pseudo-classes are commonly used for interactive styles (e.g., hover, click), form states, and structural selection. Universal selectors are used to select all elements or all elements within a specific range. They are often used to reset default styles or to apply styles in bulk. When combined with other selectors, the scope of selection can be limited to improve precision. For specific properties that can be set, refer to Appendix A. Color property values can be specified as color names, six-digit hexadecimal numbers, or RGB format. The RGB format consists of the word rgb, followed by a list inside parentheses containing three decimal numbers or percentage values, each ranging from 0 to 255. These values represent the levels of red, green, and blue, respectively. For example, 0 or 0% as the first value specifies no red in the color, while 255 or 100% specifies the maximum amount of red. Hexadecimal numbers must begin with a hash (#), such as #43AF00. For example, fuchsia (a mix of red and blue) can be specified as fuchsia, rgb(255, 0, 255), or #FF00FF.

字体 P104~112

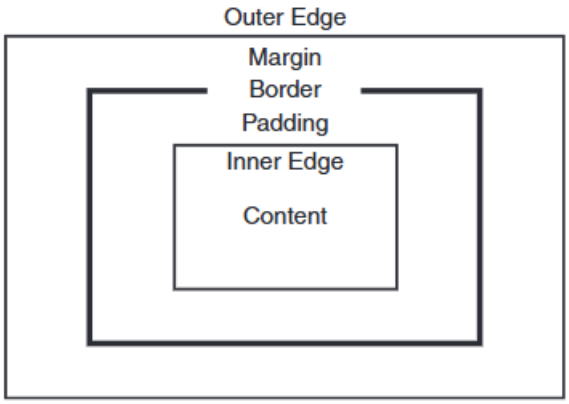
Serif	衬线字体	常见的示例是 Times New Roman 和 Garamond
Sans-serif	无衬线字体	示例为 Arial 和 Helvetica
Cursive	草书字体	示例为 Caflisch Script 和 Zapf-Chancery
Fantasy	幻想字体	例为 Critter 和 Cottonwood
Monospace	等宽字体	示例为 Courier 和 Prestige

```
<!DOCTYPE html>
<!-- fonts.html
An example to illustrate font properties
-->
<html lang = "en">
<head>
<title> Font properties </title>
<meta charset = "utf-8" />
<style type = "text/css">
p.major {font-size: 1.1em;font-style: italic;font-family: 'Times New Roman';}
p.minor {font: bold 0.9em 'Courier New';}
h2 {font-family: 'Times New Roman';
font-size: 2em; font-weight: bold;}
h3 {font-family: 'Courier New'; font-size: 1.5em;}
</style>
</head>
<body>
<p class = "major">
If a job is worth doing, it's worth doing right.
</p>
<p class = "minor">
Two wrongs don't make a right, but they certainly
can get you in a lot of trouble.
</p>
<h2> Chapter 1 Introduction </h2>
<h3> 1.1 The Basics of Computer Networks </h3>
</body>
</html>
```

盒式模型 P121 • P130

Border 的属性

```
<style type="text/css">
    td, th {border: thin solid black;}
    table {border: thin solid black;
        border-collapse: collapse;
        border-top-width: medium;
        border-bottom-width: thick;
        border-top-color: red;
        border-bottom-color: blue;
        border-top-style: dotted;
        border-bottom-style: dashed;}
    p {border: thin dashed green;}
</style>
```



边距是元素边框与其相邻元素之间的空间。当没有边框时，边距加上填充就是元素内容与其相邻元素之间的空间。在这种情况下，填充和边距之间可能看起来没有区别。但是，当元素具有背景时，会存在差异：背景延伸到内边距中，但不会延伸到边距中。

边距属性被命名为边距，它适用于所有四个元素的边、左边距、右边距、上边距和下边距。填充属性称为 padding，适用于所有四个边：padding-left、padding-right、padding-top 和 padding-bottom。

Margin is the space between an element's border and its adjacent elements. When there is no border, the margin plus the padding forms the space between the element's content and its neighboring elements. In this case, padding and margin may appear indistinguishable. However, when the element has a background, the difference becomes noticeable: the background extends into the padding area but does not extend into the margin area.

The margin property is simply called **margin**, and it applies to all four edges of an element: left margin, right margin, top margin, and bottom margin. The padding property is called **padding**, and it applies to all four sides: padding-left, padding-right, padding-top, and padding-bottom.

```
<!DOCTYPE html>
<!-- back_image.html
        An example to illustrate background images -->
<html lang="en">
<head>
    <title>Background images</title>
    <meta charset="utf-8" />
    <style type="text/css">
        body {background-image: url(../images/plane1.jpg);
            background-size: 375px 300px;}
        p {margin-left: 30px;
            margin-right: 30px;
            margin-top: 50px;
            font-size: 1.1em;}
    </style>
```

```
</head>
<body>
    <p>
The Cessna 172 is the most common general aviation airplane in the world. It is an all-metal, single-engine
piston, high-wing, four-place monoplane. It has fixed gear and is categorized as a non-high-performance aircraft. The current model is the 172R.
        <br /><br />
The wingspan of the 172R is 36'1". Its fuel capacity is 56 gallons in two tanks, one in each wing. The takeoff
weight is 2,450 pounds. Its maximum useful load is 837 pounds. The maximum speed of the 172R at sea level is 142 mph. The plane is powered by a 360-cubic-inch gasoline engine that develops 160 horsepower. The climb rate of the 172R at sea level is 720 feet per minute.
    </p>
</body>
</html>
```

Span，样式优先级 P128~P132

在许多情况下，我们希望将特殊的字体属性应用于不到一整段的文本。例如，让一行中的单词或短语以不同的字体大小或颜色显示通常很有用。 标签就是为此目的而设计的。与大多数其他标签不同， 的内容没有默认布局。	In many cases, we want to apply special font properties to only part of a paragraph of text. For example, it is often useful to display a word or phrase within a line in a different font size or color. The tag is designed for this purpose. Unlike most other tags, the content of a does not have a default layout.

```
<style type = "text/css" >
.spanred {font-size: 2em;
font-family: Ariel; color: red;}
</style>
<p>
It sure is fun to be in
<span class = "spanred"> total </span>
control of text
</p>
```

当两个或多个级别的样式表为同一元素上的同一属性指定不同的值时。这种特殊类型的冲突是通过三个不同级别的样式表的优先级来解决的。**内联样式表优先于文档和外部样式表，文档样式表优先于外部样式表。** 冲突解决是一个多阶段的排序过程。该过程的第一步是从样式表的三个可能级别收集样式规范。这些规范按照样式表级别的相对优先级排序。接下来，所有可用的规范（来自样式表的规范、来自用户的规范和来自浏览器的规范）根据以下规则列表	When two or more levels of stylesheets specify different values for the same property on the same element, this special type of conflict is resolved through the priority of the three different levels of stylesheets. Inline styles take precedence over document and external stylesheets, and document styles take precedence over external stylesheets. Conflict resolution is a multi-stage sorting process. The first step is to gather style rules from the three possible levels of stylesheets. These rules are sorted according to the relative priority of the stylesheet levels. Next, all available rules (from the stylesheet, from the user, and from the browser) are sorted

按来源和权重排序，其中第一个具有最高优先级： **具有用户来源的重要声明** **带有作者来源的重要声明** **带有作者来源的正常声明** **具有用户来源的正常声明** **任何带有浏览器（或其他用户代理）来源的声明** 请注意，用户来源规范被认为具有最高优先级。这种方法的基本原理是，此类规范通常是由于用户能力下降（最常见的是视力障碍）而声明的。如果在第一次排序后存在冲突，解决冲突的下一步是按特异性排序。这种排序基于以下规则，其中第一个具有最高优先级： **Id 选择器** **类和伪类选择器** **上下文选择器** **通用选择器**	by source and weight according to the following list, where the first has the highest priority: • Important declarations from the user's source • Important declarations from the author's source • Normal declarations from the author's source • Normal declarations from the user's source • Any declarations from the browser (or other user agents) Note that user-source rules are considered the highest priority. The basic principle behind this approach is that such rules are typically declared due to user needs (most commonly visual impairments). If conflicts remain after the first sorting, the next step in conflict resolution is sorting by specificity. This sorting is based on the following rules, where the first has the highest priority: • ID selectors • Class and pseudo-class selectors • Contextual selectors • Universal selectors

第四章

p142 4.3 简介嵌入的方式要掌握 4.4 4.5 4.6 4.7 4.8 4.8.3 数组的方法 4.9

JS 脚本引用 P142

```
<script type = "text/javascript" src = "tst_number.js" >
</script>
```

在 JavaScript 中，标识符或名称与其他常见编程语言中的标识符或名称类似。它们必须以字母、下划线 (_) 或美元符号 ($) 开头。后续字符可以是字母、下划线、美元符号或数字。

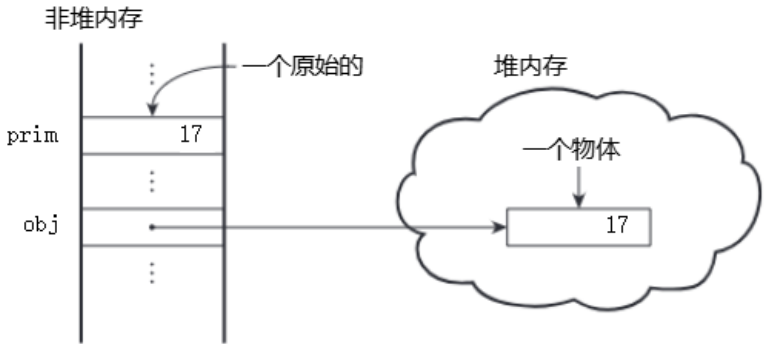
In JavaScript, identifiers or names are similar to identifiers or names in other common programming languages. They must begin with a letter, an underscore (_), or a dollar sign ($). Subsequent characters can be letters, underscores, dollar signs, or numbers.

JS 的保留字（均小写）				
break	**delete**	**function**	**return**	**typeof**
case	**do**	**if**	**switch**	**var**
catch	**else**	**in**	**this**	**void**
continue	**finally**	**instanceof**	**throw**	**while**
default	**for**	**new**	**try**	**wit**

JS 基本类型、基本类及其方法 P145~P156

JavaScript 有五种基本类型：数字、字符串、布尔值、未定义和空。每个基本值都有这些类型之一。 JavaScript 包含与 Number、String 和 Boolean 类型密切相关的预定义对象，分别命名为 Number、String 和 Boolean。 这些对象称为包装对象。每个都包含一个存储相应基本类型值的属性。包装对象的目的是提供便于与基本类型的值一起使用的属性和方法。对于 Number 来说，这些属性更有用；对于字符串，这些方法更有用。由于 JavaScript 会在 Number 类型原始值和 Number 对象之间以及 String 类型原始值和 String 对象之间强制转换值，因此可以对相应原始类型的变量使用 Number 和 String 的方法。事实上，在大多数情况下，您可以简单地将 Number 和 String 类型值视为对象。 下面的示例显示了基元和对象之间的区别。假设 prim 是一个值为 17 的原始变量，obj 是一个属性值为 17 的 Number 对象。图 4.1 显示了 prim 和 obj 的存储方式。	JavaScript has five basic types: numbers, strings, booleans, undefined, and null. Each basic value is of one of these types. JavaScript also includes predefined objects closely related to the Number, String, and Boolean types, named Number, String, and Boolean, respectively. hese objects are called wrapper objects. Each one contains a property that stores the corresponding basic type value. The purpose of the wrapper objects is to provide properties and methods that make it easier to work with the values of the basic types. For Number, these properties are more useful; for strings, these methods are more useful. Since JavaScript automatically converts between the primitive values of the Number type and the Number object, as well as between the primitive values of the String type and the String object, you can use the methods of Number and String on variables of the respective primitive types. In fact, in most cases, you can treat the values of Number and String types as objects. The following example shows the difference between primitives and objects. Suppose prim is a primitive variable with the value of 17, and obj is a Number object with the property value of 17. Figure 4.1 shows how prim and obj are stored.

```
var counter,
index,
pi = 3.14159265,
quarterback = "Elway",
stop_flag = true;
```



Number 类属性	
Number.MAX_VALU	可表示的最大正数。
Number.MIN_VALUE	可表示的最小正数（即最接近 0 的正数，不是最小的负数）。
Number.NaN	表示“不是数字”的特殊值。
Number.NEGATIVE_INFINITY	表示负无穷大；溢出时返回该值。
Number.POSITIVE_INFINITY	表示正无穷大；溢出时返回该值。
Number.EPSILON	表示 1 与大于 1 的最小浮点数之间的差。
Number.MAX_SAFE_INTEGER	在 JavaScript 中安全表示的最大整数。
Number.MIN_SAFE_INTEGER	在 JavaScript 中安全表示的最小整数。
Number.PI	π

Number 类方法	
Number.isFinite()	检查传入的值是否为有限数。

Number.isInteger()	检查传入的值是否为整数。
Number.isNaN()	检查传入的值是否是 NaN。
Number.isSafeInteger()	检查传入的值是否在安全整数范围内。
Number.parseFloat()	将字符串解析成浮点数。
Number.parseInt()	将字符串解析成整数。

String 类及其方法			
方法	**描述**	**示例代码**	**输出**
charAt(index)	返回指定位置的字符	'Hello'.charAt(1)	'e'
charCodeAt(index)	返回指定位置字符的 Unicode 编码	'Hello'.charCodeAt(1)	101
concat(str1, str2)	连接两个或多个字符串，返回新的字符串	'Hello'.concat(' ', 'World')	'Hello World'
includes(substring)	检查字符串是否包含指定的子字符串	'Hello'.includes('World')	false
endsWith(substring)	检查字符串是否以指定的子字符串结束	'Hello'.endsWith('lo')	true
indexOf(substring)	返回子字符串首次出现的位置	'Hello'.indexOf('e')	1
lastIndexOf(substring)	返回子字符串最后一次出现的位置	'Hello'.lastIndexOf('e')	1
slice(start, end)	提取字符串的指定部分，返回新字符串	'Hello World'.slice(0, 5)	'Hello'
split(separator)	根据分隔符将字符串拆分为数组	'Hello World'.split(' ')	['Hello', 'World']
substring(start, end)	返回字符串中指定位置的子字符串	'Hello World'.substring(0, 5)	'Hello'
toLowerCase()	将字符串转换为小写	'Hello'.toLowerCase()	'hello'
toUpperCase()	将字符串转换为大写	'Hello'.toUpperCase()	'HELLO'
trim()	去除字符串两端的空白字符	' Hello '.trim()	'Hello'
trimStart()	去除字符串开头的空白字符	' Hello '.trimStart()	'Hello '
trimEnd()	去除字符串末尾的空白字符	' Hello '.trimEnd()	' Hello'
replace(searchValue, newValue)	将匹配的子字符串替换为新值	'Hello World'.replace('World', 'JS')	'Hello JS'
repeat(count)	返回重复指定次数的新字符串	'Hello'.repeat(3)	'HelloHelloHello'
startsWith(substring)	判断字符串是否以指定的子字符串开头	'Hello'.startsWith('He')	true
padStart(targetLength, padString)	用指定字符填充字符串开头以达到指定长度	'5'.padStart(3, '0')	'005'
padEnd(targetLength, padString)	用指定字符填充字符串末尾以达到指定长度	'5'.padEnd(3, '0')	'500'

字符串的匹配方法、模式符修饰方法			
方法	**描述**	**示例代码**	**输出**
match()	根据正则表达式查找匹配的子字符	'Hello World'.match(/o/g)	['o', 'o']

	串，返回匹配数组或 null		
matchAll()	返回所有匹配项的迭代器，包括捕获组	[...'Hello World'.matchAll(/o/g)]	[['o'], ['o']]
search()	根据正则表达式搜索字符串，返回第一个匹配项的索引	'Hello World'.search(/World/)	6
replace()	用新字符串替换匹配的子字符串	'Hello World'.replace('World', 'JS')	'Hello JS'
replaceAll()	替换所有匹配的子字符串（支持正则或字符串）	'Hello World World'.replaceAll('World', 'JS')	'Hello JS JS'
split()	使用指定的分隔符将字符串分割为数组	'Hello, World, JS'.split(/, /)	['Hello', 'World', 'JS']
test()	检查字符串是否与正则表达式匹配，返回布尔值	/World/.test('Hello World')	true
exec()	使用正则表达式匹配字符串，返回匹配数组或 null	/World/.exec('Hello World')	['World']
修饰符	**描述**	**示例代码**	**输出**
g	全局匹配，匹配所有符合条件的子字符串	'Hello World'.match(/o/g)	['o', 'o']
i	忽略大小写匹配	'Hello World'.match(/hello/i)	['Hello']
m	多行匹配，`^` 和 `$` 还可以匹配行的开头和结尾	'World\nHello'.match(/^World/m)	['World']

JS 的 Date 类及其方法			
方法	**描述**	**示例代码**	**输出**
Date()	返回当前日期和时间的新日期对象	new Date()	当前日期和时间
getDate()	返回月份中的某一天（1-31）	new Date().getDate()	28（假设）
getDay()	返回星期几（0-6），0 表示星期日	new Date().getDay()	2（假设）
getFullYear()	返回四位的年份	new Date().getFullYear()	2024
getHours()	返回小时（0-23）	new Date().getHours()	14（假设）
getMinutes()	返回分钟（0-59）	new Date().getMinutes()	45（假设）
getSeconds()	返回秒数（0-59）	new Date().getSeconds()	30（假设 14:45:30）
getMilliseconds()	返回毫秒数（0-999）	new Date().getMilliseconds()	500（假设）
getMonth()	返回月份（0-11），0 表示一月	new Date().getMonth()	8（假设当前是 9 月）
getTime()	返回自 1970-01-01 以来的毫秒数	new Date().getTime()	1695414930000（假）
getTimezoneOffset()	返回当前时区与 UTC 时间的差值，以分钟为单位	new Date().getTimezoneOffset()	-480（假设时区是 UTC+8）

setDate(day)	设置日期（1-31）	let d = new Date(); d.setDate(15)	将日期设置为 15 号
setFullYear(year, month, day)	设置完整年份及可选的月份和日期	let d = new Date(); d.setFullYear(2025, 1, 15)	将日期设置为 2025 年 2 月 15 日
setHours(hours, min, sec, ms)	设置小时及可选的分钟、秒、毫秒	let d = new Date(); d.setHours(10, 30)	将时间设置为 10:30
setMilliseconds(ms)	设置毫秒数	let d = new Date(); d.setMilliseconds(123)	将毫秒设置为 123
setMonth(month, day)	设置月份及可选的日期	let d = new Date(); d.setMonth(3, 5)	将日期设置为 4 月 5 日
setTime(milliseconds)	设置自 1970-01-01 以来的毫秒数	let d = new Date(); d.setTime(1695414930000)	将日期设置为指定时间戳
toDateString()	返回日期的字符串表示形式（不包含时间部分）	new Date().toDateString()	'Tue Sep 26 2024'
toISOString()	返回 ISO 格式的完整字符串表示形式	new Date().toISOString()	'2024-09-26T06:45:30.000Z'
toLocaleDateString()	返回根据本地格式化的日期字符串	new Date().toLocaleDateString()	'9/26/2024'
toLocaleTimeString()	返回根据本地格式化的时间字符串	new Date().toLocaleTimeString()	'14:45:30'
toTimeString()	返回时间的字符串表示形式（不包含日期部分）	new Date().toTimeString()	'14:45:30 GMT+0800 (China Standard Time)'
toUTCString()	返回 UTC 时间的字符串表示形式	new Date().toUTCString()	'Tue, 26 Sep 2024 06:45:30 GMT'

方法	描述	示例代码	输出
alert()	显示一个警告框，只有确定按钮，没有返回值	alert('This is an alert message')	弹出警告框，显示 'This is an alert message'
confirm()	显示一个带有确定和取消按钮的对话框，返回布尔值	confirm('Are you sure?')	用户点击“确定”返回 `true`，点击“取消”返回 `false`
prompt()	显示一个输入框，要求用户输入值，并返回输入的字符串（或 `null`）	prompt('Enter your name', 'Default Name')	用户输入文本后返回该文本，取消或关闭对话框返回 `null`

JS 的显式类型转换、隐式类型转换、强制类型转换			
类型转换	描述	示例代码	输出
显式类型转换	使用 JavaScript 内置函数或运算符明确转换数据类型	String(123)	'123'
		Number('123')	123
		Boolean(1)	true
隐式类型转换	在运算中 JavaScript 自动将不同类型数据转换为兼容的类型	'123' + 1	'1231'
		123 == '123'	true

		null + 1	1
强制类型转换	强制将值转换为期望的类型，通常发生在使用 `==` 时	'123' == 123	true
		'0' == false	true
		[] == ''	true

JS 的垃圾回收机制？

JS 类区别、数组类、Math 类 P165~P166

因此，在 Java 中，构造函数会初始化成员，但不会创建它们。然而，在 JavaScript 中，new 运算符创建一个空白对象，即没有属性的对象。此外，JavaScript 对象没有类型。构造函数创建并初始化属性。 **JS 在实例化对象后还可以加入其他原本未定义的东西，所以上面写的是“初始化”而不是“创建”。** 对象可以嵌套。在 JavaScript 中，**数组是具有一些特殊功能的对象**。数组元素可以是原始值或对其他对象（包括其他数组）的引用。	In Java, a constructor initializes members but does not create them. However, in JavaScript, the new operator creates a blank object, i.e., an object without properties. Additionally, JavaScript objects have no predefined type. The constructor creates and initializes the properties. In JavaScript, after an object is instantiated, you can add properties that were not originally defined, so the term “initialize” is used rather than “create.” Objects can be nested. In JavaScript, arrays are objects with some special features. Array elements can be primitive values or references to other objects (including other arrays).

```
var my_list = new Array(1, 2, "three", "four");
var your_list = new Array(100);
```

JS 数组类及其方法

方法	描述	示例代码	输出
push()	向数组末尾添加一个或多个元素，并返回数组的新长度	let arr = [1, 2]; arr.push(3)	[1, 2, 3]（返回 3）
pop()	删除数组最后一个元素，并返回该元素	let arr = [1, 2, 3]; arr.pop()	3（arr 变为 [1, 2]）
shift()	删除数组第一个元素，并返回该元素	let arr = [1, 2, 3]; arr.shift()	1（arr 变为 [2, 3]）
unshift()	向数组开头添加一个或多个元素，并返回数组的新长度	let arr = [2, 3]; arr.unshift(1)	[1, 2, 3]（返回 3）
concat()	合并两个或多个数组，并返回新的数组	let arr1 = [1]; let arr2 = [2, 3]; arr1.concat(arr2)	[1, 2, 3]
slice()	返回数组的一个子集，不修改原数组	let arr = [1, 2, 3, 4]; arr.slice(1, 3)	[2, 3]
splice()	删除或替换数组的元素，并返回被删除的元素	let arr = [1, 2, 3, 4]; arr.splice(1, 2)	[2, 3]（arr 变为 [1, 4]）
indexOf()	返回数组中指定元素的第一个索引，找不到则返回 -1	let arr = [1, 2, 3]; arr.indexOf(2)	1
lastIndexOf()	返回数组中指定元素的最后一个索引，找不到则返回 -1	let arr = [1, 2, 2, 3]; arr.lastIndexOf(2)	2

forEach()	对数组中的每个元素执行一次提供的函数（不返回值）	[1, 2, 3].forEach(x => console.log(x))	输出 1 2 3（依次输出）
map()	对数组中的每个元素执行提供的函数，并返回一个新数组	[1, 2, 3].map(x => x * 2)	[2, 4, 6]
filter()	创建一个包含所有通过测试的元素的新数组	[1, 2, 3].filter(x => x > 1)	[2, 3]
reduce()	将数组中的元素通过一个函数累积为单个值	[1, 2, 3].reduce((sum, x) => sum + x, 0)	6
reverse()	反转数组中元素的顺序	let arr = [1, 2, 3]; arr.reverse()	[3, 2, 1]
sort()	对数组元素进行排序，默认按字符串 Unicode 码点升序	let arr = [3, 1, 2]; arr.sort()	[1, 2, 3]
join()	将数组的所有元素连接成一个字符串	let arr = [1, 2, 3]; arr.join('-')	'1-2-3'
find()	返回数组中第一个满足提供的函数的元素	[1, 2, 3].find(x => x > 1)	2
findIndex()	返回数组中第一个满足提供的函数的元素索引	[1, 2, 3].findIndex(x => x > 1)	1
some()	测试数组中是否至少有一个元素通过提供的函数测试	[1, 2, 3].some(x => x > 2)	true
every()	测试数组中是否所有元素都通过提供的函数测试	[1, 2, 3].every(x => x > 0)	true
flat()	按指定深度递归合并数组中的所有子数组	[1, [2, [3]]].flat(2)	[1, 2, 3]
flatMap()	首先对数组中的每个元素应用一个函数，然后将结果合并为一个新数组	[1, 2].flatMap(x => [x, x * 2])	[1, 2, 2, 4]

JS Math 类及其方法

方法	描述	示例代码	输出
Math.abs(x)	返回数值的绝对值	Math.abs(-5)	5
Math.ceil(x)	向上取整，返回大于或等于给定数的最小整数	Math.ceil(4.3)	5
Math.floor(x)	向下取整，返回小于或等于给定数的最大整数	Math.floor(4.7)	4
Math.round(x)	返回四舍五入后的整数	Math.round(4.5)	5
Math.max(x, y, ...)	返回一组数中的最大值	Math.max(10, 20, 30)	30
Math.min(x, y, ...)	返回一组数中的最小值	Math.min(10, 20, 30)	10
Math.pow(x, y)	返回 x 的 y 次幂	Math.pow(2, 3)	8
Math.sqrt(x)	返回数值的平方根	Math.sqrt(9)	3
Math.random()	返回 0 和 1 之间的随机数	Math.random()	0.123... (随机数)
Math.trunc(x)	返回数值的整数部分（去除小数部分）	Math.trunc(4.9)	4
Math.sign(x)	返回数值的符号，1 表示正数，-1 表示负数，0 表示零	Math.sign(-10)	-1
Math.log(x)	返回数值的自然对数（底为 e）	Math.log(1)	0
Math.log10(x)	返回数值的以 10 为底的对数	Math.log10(1000)	3
Math.exp(x)	返回 e 的 x 次幂	Math.exp(1)	2.718...
Math.sin(x)	返回数值的正弦值（参数为弧度）	Math.sin(Math.PI / 2)	1

Math.cos(x)	返回数值的余弦值（参数为弧度）	Math.cos(0)	1
Math.tan(x)	返回数值的正切值（参数为弧度）	Math.tan(Math.PI / 4)	1
Math.asin(x)	返回数值的反正弦值，结果在 -π/2 到 π/2 之间	Math.asin(1)	π/2
Math.acos(x)	返回数值的反余弦值，结果在 0 到 π 之间	Math.acos(1)	0
Math.atan(x)	返回数值的反正切值，结果在 -π/2 到 π/2 之间	Math.atan(1)	π/4
Math.PI	返回圆周率常数 π	Math.PI	3.14159...
Math.E	返回自然常数 e	Math.E	2.718...
Math.LN10	返回 10 的自然对数	Math.LN10	2.302...
Math.LN2	返回 2 的自然对数	Math.LN2	0.693...
Math.SQRT2	返回 2 的平方根	Math.SQRT2	1.414...
Math.SQRT1_2	返回 1/2 的平方根	Math.SQRT1_2	0.707...

数组的长度可以通过 length 属性进行读取和写入，该属性是由 Array 构造函数为每个数组对象创建的。因此，可以通过分配 length 属性将数组的长度设置为您喜欢的任何长度，如下例所示：my_list.length = 100 现在，my_list 的长度是 1002，无论它之前是什么。为 length 属性赋值可以延长、缩短或不影响数组的长度。 只有数组中分配的元素才实际占用空间。例如，如果使用 100 到 150（而不是 0 到 50）的下标范围比较方便，则可以创建长度为 151 的数组。但是，如果仅对索引为 100 到 150 的元素赋值，则数组将需要 51 个元素的空间，而不是 151 个。数组的长度属性不一定是分配的元素数量。例如，以下语句将 new_list 的 length 属性设置为 1002，但 new_list 可能没有具有值或占用空间的元素。（属实是幽默了，这内存浪费的） 为了支持 JavaScript 的动态数组，所有数组元素都是从堆中动态分配的。将值分配给先前不存在的数组元素会创建该元素。	The length of an array can be read and written using the length property, which is created by the Array constructor for each array object. Therefore, you can set the array's length to any desired value by assigning a value to the length property, as shown in the following example: my_list.length = 100. Now, my_list has a length of 100, no matter what it was before. Assigning a value to the length property can extend, shorten, or have no effect on the array's length. Only the elements of an array that are actually assigned values occupy space. For example, if it is more convenient to use an index range of 100 to 150 (rather than 0 to 50), you can create an array with a length of 151. However, if you only assign values to the elements with indexes from 100 to 150, the array will only require space for 51 elements, not 151. The length property does not necessarily reflect the number of allocated elements. For example, the following statement sets the length property of new_list to 1002, but new_list may not have elements that have values or occupy space. To support JavaScript's dynamic arrays, all array elements are dynamically allocated from the heap. Assigning values to previously non-existent array elements will create those elements.

JS 函数 P172

```
function fun() {
document.write("This surely is fun! <br/>");
}
ref_fun = fun; // Now, ref_fun refers to the fun object
fun(); // A call to fun
ref_fun(); // Also a call to fun
```

JS 正则表达式 P178

JavaScript 具有基于正则表达式的强大模式匹配功能。 JavaScript 中有两种模式匹配方法：一种基于 RegExp 对象的方法，另一种基于 String 对象的方法。这两种方法使用的正则表达式相同，并且基于 Perl 编程语言的正则表达式。

JavaScript has powerful pattern matching capabilities based on regular expressions. There are two pattern matching methods in JavaScript: one based on the RegExp object and the other based on the String object. Both methods use the same regular expressions, which are based on the regular expressions of the Perl programming language.

第五章

DOM 一定要掌握 ineerHTML 获得标签里面的内容
5.3 四种获取方法 5.4.1 参照 ppt 5.7.2 密码框？、、

DOM(Do) P200~P202

在大多数情况下，JavaScript 中的属性名称与 HTML 中相应的属性名称相同。 DOM（Document Object Model，文档对象模型）是一种平台和语言无关的接口，允许程序和脚本动态地访问和更新文档的内容、结构和样式。 DOM 将 HTML 或 XML 文档表示为一个对象树，使开发者可以使用编程语言（如 JavaScript）对网页的元素进行增删改查操作。	In most cases, property names in JavaScript are the same as their corresponding attribute names in HTML. However, there are some differences, particularly with respect to naming conventions, which arise due to JavaScript's case-sensitivity and HTML's lowercase attribute names. DOM (Document Object Model) is a platform- and language-independent interface that allows programs and scripts to dynamically access and update the content, structure, and style of documents. The DOM represents HTML or XML documents as a tree of objects, allowing developers to perform operations like adding, deleting, modifying, and querying elements on a webpage using programming languages such as JavaScript.

方法	示例代码	输出
getElementById(id)	document.getElementById('myDiv')	返回 ID 为'myDiv'的元素
getElementsByClassName(class)	document.getElementsByClassName('myClass')	返回类名为'myClass'的元素集合
getElementsByTagName(tag)	document.getElementsByTagName('div')	返回所有 元素的集合
querySelector(selector)	document.querySelector('.myClass')	返回类名为'myClass'的第一个元素
querySelectorAll(selector)	document.querySelectorAll('.myClass')	返回类名为'myClass'的所有元素
createElement(tagName)	document.createElement('div')	创建一个新的 元素
createTextNode(text)	document.createTextNode('Hello World')	创建一个文本节点'Hello World'
appendChild(node)	document.body.appendChild(newDiv)	将 newDiv 元素添加到 body 元素末尾

removeChild(node)	document.body.removeChild(oldDiv)	从 body 中移除 oldDiv
replaceChild(newNode, oldNode)	document.body.replaceChild(newDiv, oldDiv)	用 newDiv 替换 oldDiv
getAttribute(attr)	document.getElementById('myDiv').getAttribute('class')	返回元素的 class 属性值
setAttribute(attr, value)	document.getElementById('myDiv').setAttribute('class', 'newClass')	设置元素的 class 为'newClass'
removeAttribute(attr)	document.getElementById('myDiv').removeAttribute('class')	移除元素的 class 属性
write(htmlString)	document.write(' Hello World ')	将' Hello World '写入文档
addEventListener(event, func)	document.getElementById('myDiv').addEventListener('click', handleClick)	为元素添加点击事件监听器
removeEventListener(event, func)	document.getElementById('myDiv').removeEventListener('click', handleClick)	移除元素的点击事件监听器
getElementByName(name)	document.getElementsByName('username')	返回所有 name 为'username'的元素集合

示例

html

```
<!DOCTYPE html>
<html>
<head>
  <title>示例页面</title>
</head>
<body>
  <p id="greeting">Hello, World!</p>
</body>
</html>
```

javascript

```
document.getElementById('greeting').textContent = '你好，世界！';
```

DOM 并非特定于某种编程语言，它定义了一组标准接口，这些接口描述了如何访问和操作文档的结构。不同的编程语言需要将这些抽象接口映射到自己的语言特性中，这就是“绑定”。例如，JavaScript、Java、Python 等语言都有自己的 DOM 绑定，使得开发者可以使用这些语言与 DOM 交互。

The DOM is not specific to any one programming language; it defines a set of standard interfaces that describe how to access and manipulate the structure of documents. Different programming languages need to map these abstract interfaces to their own language features, which is known as "binding." For example, languages such as JavaScript, Java, Python, etc., each have their own DOM bindings, enabling developers to interact with the DOM using the specific features and syntax of these languages.

示例

javascript

```
var element = document.createElement('div');
element.textContent = '这是一个新的 DIV 元素';
document.body.appendChild(element);
```

python

```
from xml.dom.minidom import Document
doc = Document()
element = doc.createElement('div')
text = doc.createTextNode('这是一个新的 DIV 元素')
```

	element.appendChild(text) doc.appendChild(element)

实际的 DOM 规范由一组接口组成，其中每个文档树节点类型都有一个接口。这些接口类似于 Java 接口和 C++ 抽象类。它们定义与其各自节点类型关联的对象、方法和属性。

DOM 规范定义了各种节点类型（如元素、属性、文本节点等）的接口，每个接口都描述了该节点类型应具备的属性和方法。这些 DOM 接口类似于 Java 中的接口和 C++ 中的抽象类，它们只是定义了规范，但不包含具体实现。

The DOM specification consists of a set of interfaces, where each document tree node type has its corresponding interface. These interfaces are similar to Java interfaces and C++ abstract classes. They define the objects, methods, and properties associated with each node type.

The DOM specification defines interfaces for various node types, such as elements, attributes, text nodes, and so on. Each interface describes the properties and methods that the corresponding node type should have. These DOM interfaces are analogous to Java interfaces and C++ abstract classes in that they define the specification but do not contain the actual implementation.

示例	
Element 接口：	**javascript**

属性：
tagName：标签名
attributes：属性列表
方法：
getAttribute(name)：获取属性值
setAttribute(name, value)：设置属性值
appendChild(node)：添加子节点
	var element = document.createElement('a');
	element.setAttribute('href', 'https://www.example.com');
	element.textContent = '点击这里访问示例网站';
	document.body.appendChild(element);

借助 DOM，用户可以用编程语言编写代码来创建文档、在其结构中移动以及更改、添加或删除元素及其内容。 DOM 的功能： DOM 中的文档具有树状结构，但文档中可以有多于一棵树（尽管这是不常见的）。由于 DOM 是一个抽象接口，因此它并不规定将文档实现为树或树的集合。因此，在实现中，文档元素之间的关系可以用几种不同的方式表示。 HTML 文档在 DOM 中被表示为一个节点树，根节点是 document。每个元素节点可能有子节点，形成层级关系。虽然大多数文档只有一棵节点树，但在某些特殊情况下（如使用 DocumentFragment），可能存在多棵树。 DOM 作为抽象接口，并不限定具体	With the help of the DOM, users can write code in a programming language to create documents, move within their structure, and modify, add, or remove elements and their content. **Functionality of the DOM:** **Tree Structure**: The document in the DOM has a tree structure, but there can be more than one tree in the document (though this is uncommon). Since the DOM is an abstract interface, it does not specify how to implement the document as a tree or a collection of trees. In the implementation, the relationships between document elements can be represented in different ways. **Representation of HTML Documents in the DOM**: HTML documents in the DOM are represented as a node tree, with the root node being document. Each element node may have child nodes, forming a hierarchical relationship. While most documents only have a single tree, in some special cases (such as when using DocumentFragment), there may be multiple trees. **Abstract Interface**: The DOM, as an abstract interface, does not limit

的实现方式。不同的浏览器或解析器可能以不同的数据结构实现 DOM IE8+、FX3+ 和 C10+ 提供了一种查看所显示文档的 DOM 结构的方法。	the implementation method. Different browsers or parsers may implement the DOM using different data structures. **Viewing the DOM Structure**: IE8+, Firefox 3+, and Chrome 10+ provide a method to view the DOM structure of the displayed document. This can be helpful for debugging or understanding how the browser parses and builds the structure of HTML content.

IE8+：Internet Explorer 8 及以上版本
FX3+： Firefox 3 及以上版本
C10+： Chrome 10 及以上版本

事件一览 P203

Events	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

各事件对应继承的控件

Attributes	Tag	Description
onblur	<a>	The link loses focus.
onblur	<button>	The button loses focus.
onblur	<input>	The input element loses focus.
onblur	<textarea>	The text area loses focus.
onblur	<select>	The selection element loses focus.
onchange	<input>	The input element is changed and loses focus.
onchange	<textarea>	The text area is changed and loses focus.
onchange	<select>	The selection element is changed and loses focus.
onclick	<a>	The user clicks the link.
onclick	<input>	The input element is clicked.

ondblclick	Most elements	The user double-clicks the left mouse button.
onfocus	<a>	The link acquires focus.
onfocus	<input>	The input element acquires focus.
onfocus	<textarea>	A text area acquires focus.
onfocus	<select>	A selection element acquires focus.
onkeydown	<body>, form elements	A key is pressed.
onkeypress	<body>, form elements	A key is pressed and released.
onkeyup	<body>, form elements	A key is released.
onload	<body>	The document is finished loading.
onmousedown	Most elements	The user clicks the left mouse button.
onmousemove	Most elements	The user moves the mouse cursor within the element.
onmouseout	Most elements	The mouse cursor is moved away from being over the element.
onmouseover	Most elements	The mouse cursor is moved over the element.
onmouseup	Most elements	The left mouse button is unclicked.
onreset	<form>	The reset button is clicked.
onselect	<input>	Any text in the content of the element is selected.
onselect	<textarea>	Any text in the content of the element is selected.
onsubmit	<form>	The Submit button is pressed.
onunload	<body>	The user exits the document.

控件响应事件 P205

body 元素最常创建的事件是 load 和 unload。

```
<body onload="load_greeting();">
```

事件的用法如下：（一个静态，一个动态）

```
<input type="button" id="myButton" onclick="myButtonHandler();" />
document.getElementById("myButton").onclick = myButtonHandler;
```

当然，onclick 事件可以用上面的其他事件来代替，事件依旧是在 JS 中处理。

正则化 P219

在 JavaScript 中，如果你想给出一个 **正则化式子**，通常指的是正则表达式（Regular Expression）及其应用。正则表达式是用来匹配字符串的一种模式。匹配一个具体的字符，例如，’a’ 匹配字符 ’a’。

元字符：用于指定匹配规则		**字符类：用方括号 [] 包围的字符集合**	
.	匹配任意单个字符。	[a-z]	匹配小写字母。
^	匹配字符串的开始。	\d	匹配数字（[0-9]）。
$	匹配字符串的结束。	\w	匹配字母、数字或下划线。
*	匹配前面的字符零次或多次。	\s	匹配任何空白字符（空格、制表符、换行符等）。
+	匹配前面的字符一次或多次。		
?	匹配前面的字符零次或一次。		
量词：指定匹配的数量		**分组和捕获：用() 包围的表达式作为一个单独的单元**	
{n}	匹配前一个字符恰好 n 次。	(abc)	匹配字符串'abc'。
{n,}	匹配前一个字符至少 n 次。	(a|b)	匹配字符'a'或'b'。
{n,m}	匹配前一个字符 n 到 m 次。		

正则表达式的创建方式

```
let regex = /abc/;  // 匹配字符串 "abc"
let regex = new RegExp("abc");  // 匹配字符串 "abc"
```

常见的正则化式子

匹配邮箱地址	let regex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;
匹配电话号码（如美国号码）	let regex = /^\(\d{3}\) \d{3}-\d{4}$/;
匹配日期（格式为 YYYY-MM-DD）	let regex = /^\d{4}-\d{2}-\d{2}$/;
匹配 URL	let regex = /^(https?|ftp):\/\/[^\s/$.?#].[^\s]*$/i;
匹配数字（整数或浮动小数）	let regex = /^-?\d*(\.\d+)?$/;

在 JavaScript 中使用正则表达式

使用 .test() 方法
let regex = /^[a-zA-Z0-9]+$/;
console.log(regex.test("abc123")); // true
console.log(regex.test("abc@123")); // false
	使用 .exec() 方法
	let regex = /(\d{3})-(\d{3})-(\d{4})/;
	let str = "123-456-7890";
	let result = regex.exec(str);
	console.log(result); // ["123-456-7890", "123", "456", "7890"]

替换操作 .replace()
let regex = /\d+/g;
let str = "There are 123 apples and 456 oranges.";
let result = str.replace(regex, "number");
console.log(result); // "There are number apples and number oranges."	

第七章

基本不考，不太用关心

XML (eXtensible Markup Language) 是一种可扩展的标记语言，它定义了一种规则，用于编码文档，使它们可以被计算机和人类轻松地读取和处理。XML 是一个用于表示结构化数据的标准，广泛应用于 Web 服务、配置文件、数据交换等场景。 XML 主要用于表示数据和传递数据。它**不关心数据的呈现方式，仅关心数据的结构**。使用 XML，用户可以为自己的数据定义标签，并且标签的含义是用户自定义的。因此，它非常灵活，广泛用于存储和交换数据。 XML 的语法规则非常严格，主要包括以下几个方面： ● 文档必须有一个根元素。一个 XML 文件中只能有一个根元素，根元素可以包含其他元素。 ● 元素标签必须成对出现。每个开始标签（如 <tag>）必须有一个匹配的结束标签	XML (eXtensible Markup Language) is an extensible markup language that defines a set of rules for encoding documents so that they can be easily read and processed by both computers and humans. XML is a standard for representing structured data, widely used in scenarios such as web services, configuration files, and data exchange. XML is primarily used for representing and transmitting data. It is not concerned with how the data is presented, but with its structure. Using XML, users can define their own tags for their data, and the meaning of these tags is user-defined. As a result, it is highly flexible and is widely used for storing and exchanging data. The syntax rules of XML are very strict and include the following key aspects: • **Document must have a root element**: An XML file can only have one root element, which can contain other elements. • **Element tags must appear in pairs**: Every opening tag (e.g., <tag>) must have a corresponding closing tag (e.g., </tag>). • **Element attributes**: Elements can contain attributes, which are usually used to provide additional information.

（如 </tag>）。 ● 元素的属性。元素可以包含属性，属性通常用于提供附加的信息。 ● 标签区分大小写。XML 是区分大小写的，<Tag> 和 <tag> 是两个不同的标签。 ● 必须有正确的嵌套关系。元素必须按照正确的顺序进行嵌套，不能有交叉。 ● 文件必须符合 UTF-8 编码。XML 文件默认使用 UTF-8 编码，可以通过声明编码来指定不同的编码方式。	• **Tags are case-sensitive**: XML is case-sensitive, meaning that <Tag> and <tag> are considered two different tags. • **Correct nesting must be maintained**: Elements must be nested in the correct order, and there should be no overlapping or crossing of elements. • **File must be UTF-8 encoded**: XML files are typically encoded in UTF-8 by default, although a different encoding can be specified by declaring the encoding at the beginning of the document.

基本结构：

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book>
        <title>XML for Beginners</title>
        <author>John Doe</author>
        <year>2020</year>
    </book>
    <book>
        <title>Advanced XML</title>
        <author>Jane Smith</author>
        <year>2021</year>
    </book>
</library>
```

XML 名称空间用于区分不同 XML 文档中可能重复的元素和属性。它避免了在大型文档或跨文档的情况下，标签名称冲突的风险。名称空间通常使用 URI 来标识。**XML 名称空间通过 xmlns 属性来声明**，一般在根元素上进行声明。声明的 URI 可以是任意字符串，但通常使用 URL。

XML namespaces are used to distinguish between elements and attributes that might have the same name in different XML documents. They help prevent conflicts of tag names, especially in large documents or when documents are combined. Namespaces are typically identified using a URI (Uniform Resource Identifier).XML namespaces are declared using the xmlns attribute, usually on the root element. The URI declared can be any string, though it is typically a URL.

```
<book xmlns="http://example.com/book">
    <title>XML for Beginners</title>
</book>
```

使用前缀来区分不同的名称空间：

Use prefixes to distinguish between different namespaces:

```
<book xmlns:fiction="http://example.com/fiction"
      xmlns:nonfiction="http://example.com/nonfiction">
    <fiction:title>XML for Beginners</fiction:title>
    <nonfiction:title>Advanced XML</nonfiction:title>
</book>
```

XML Schema（或称 XSD，即 XML Schema Definition）是一种用于定义 XML 文档结构的语言。它允许开发者定义 XML 文档中可以包含哪些元素、属性以及它们的顺序和数据类型。使用 XML Schema 可以确保 XML 文档的数据结构是正确的，并且符合特定规则。

XML Schema (or XSD, which stands for XML Schema Definition) is a language used to define the structure of an XML document. It allows developers to specify which elements and attributes can appear in an XML document, as well as their

order and data types. Using XML Schema ensures that the data structure of an XML document is correct and adheres to specific rules.

XSD 的基本结构

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="year" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML 用于存储和传输结构化数据，灵活且人类可读。XML 语法包含严格的结构和规则，例如元素的嵌套、标签的成对出现等。XML 文档结构包括根元素、子元素、属性、文本内容等。XML 名称空间用于区分不同来源的元素和属性，避免命名冲突。XML Schema 定义 XML 文档的结构和数据类型，用于验证 XML 文档的正确性。XML 的灵活性和可扩展性使其在很多应用中都发挥了重要作用，尤其是在跨平台的数据交换中。

XML is used for storing and transmitting structured data, and is both flexible and human-readable. The XML syntax includes strict structures and rules, such as element nesting, paired tags, etc. The structure of an XML document includes the root element, child elements, attributes, text content, and more. XML namespaces are used to distinguish elements and attributes from different sources, preventing naming conflicts. XML Schema defines the structure and data types of an XML document, and is used to validate the correctness of the XML document. The flexibility and extensibility of XML make it a key player in many applications, especially in cross-platform data exchange.

第九章

Php

9.3 9.4 9.5 9.6 9.7 9.8 9.10 表单处理（考大题的地方）

PHP 保留字 P360~P361

Table 9.1 The reserved words of PHP

and	else	global	require	virtual
break	elseif	if	return	xor
case	extends	include	static	while
class	false	list	switch	
continue	for	new	this	
default	foreach	not	true	
do	function	or	var	

PHP 允许通过三种不同的方式来编写注释。单行注释可以使用 # 或 //，这与 JavaScript 中的注释方式相同。多行注释使用 /* 和 */ 来括起来，就像许多其他编程语言一样。

PHP 语句以分号 ; 结束。大括号 {} 用于构成控制结构的复合语句。除非作为函数定义的主体，否则复合语句不能是一个块（block）。(块可以定义局部作用域的变量，但复合语句不能。)

PHP 的变量 P361

标量类型（Scalar Types）标量类型用于存储单一的值，可以是整数、浮点数、字符串或布尔值。

整数（Integer）用于存储没有小数的数字。可以是正整数、负整数或零。

```
$age = 25;  // 整数
$negative = -10;  // 负整数
$zero = 0;  // 零
```

浮点数（Float / Double）用于存储带有小数点的数字。PHP 中浮点数也被称为 double。

```
$price = 19.99;  // 浮点数
$temperature = -10.5;  // 负浮点数
```

字符串（String）用于存储文本数据，字符串由一对引号包围，可以是单引号或双引号。

```
$name = "John Doe";  // 双引号字符串
$message = 'Hello, World!';  // 单引号字符串
```

布尔值（Boolean）用于表示 true 或 false 的值。主要用于条件判断和逻辑运算。

```
$is_active = true;  // 布尔值 true
$is_deleted = false;  // 布尔值 false
```

复合类型（Compound Types）复合类型用于存储多个值或更复杂的数据结构。

数组（Array）数组可以存储多个值，可以是数字、字符串或其他类型的数据。PHP 支持两种类型的数组：索引数组和关联数组。

索引数组：使用数字索引访问元素。

```
$fruits = array("Apple", "Banana", "Orange");  // 索引数组
echo $fruits[1];  // 输出 "Banana"
```

关联数组：使用命名的键值对访问元素。

```
$person = array("name" => "John", "age" => 25);  // 关联数组
echo $person["name"];  // 输出 "John"
```

对象（Object）对象是一个类的实例，类可以定义属性（变量）和方法（函数）。PHP 支持面向对象编程（OOP），通过 class 和 new 关键字创建和使用对象。

```
class Person {
    public $name;
    public $age;
    function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
    function greet() {
        return "Hello, my name is " . $this->name;
    }
}
$person = new Person("John", 25);  // 创建对象
echo $person->greet();  // 输出 "Hello, my name is John"
```

特殊类型（Special Types）

NULL 是一个表示“无值”的特殊类型。一个变量如果没有被赋值，它的值就是 NULL。NULL 也可以显式赋值给变量。

```
$var = NULL;  // 变量被设置为 NULL
```

资源（Resource）

资源是一种特殊的数据类型，通常用于数据库连接、文件句柄等外部资源。

资源类型在 PHP 中并不常见，通常是由函数返回的。

```
$file = fopen("example.txt", "r");  // 打开文件并返回资源类型
```

一些有用的预定义函数 P363

函数	描述	示例代码
Math.floor()	向下取整，返回小于或等于给定数字的最大整数。	Math.floor(4.7) 返回 4
Math.ceil()	向上取整，返回大于或等于给定数字的最小整数。	Math.ceil(4.2) 返回 5
Math.round()	四舍五入，返回最接近给定数字的整数。	Math.round(4.5) 返回 5 Math.round(4.4) 返回 4
Math.random()	生成一个介于 0 和 1 之间的随机浮动小数。	Math.random() 返回 0.827（示例）
Math.abs()	返回数字的绝对值。	Math.abs(-5) 返回 5
Math.min()	返回给定数值中的最小值。	Math.min(3, 5, 7, 2) 返回 2
Math.max()	返回给定数值中的最大值。	Math.max(3, 5, 7, 2) 返回 7
Math.pow()	返回一个数的指定次方。	Math.pow(2, 3) 返回 8
Math.sqrt()	返回一个数的平方根。	Math.sqrt(16) 返回 4
Math.log()	返回指定数字的自然对数（以 e 为底）。	Math.log(10) 返回 2.302585092994046
Math.random() + Math.floor()	生成指定范围内的随机整数（模拟 rand()）。	Math.floor(Math.random() * 100) + 1 返回 1 到 100 之间的随机整数

字符串方法 P364

方法	描述	示例代码
String.length	返回字符串的长度（字符数）。	'hello'.length 返回 5
String.charAt()	返回指定位置的字符。	'hello'.charAt(1) 返回 'e'
String.indexOf()	返回指定子字符串在字符串中首次出现的位置，如果不存在则返回 -1。	'hello'.indexOf('l') 返回 2
String.lastIndexOf()	返回指定子字符串在字符串中最后一次出现的位置。	'hello'.lastIndexOf('l') 返回 3
String.includes()	判断字符串是否包含指定的子字符串，返回 true 或 false。	'hello'.includes('ell') 返回 true
String.startsWith()	判断字符串是否以指定的子字符串开始，返回 true 或 false。	'hello'.startsWith('he') 返回 true
String.endsWith()	判断字符串是否以指定的子字符串结束，返回 true 或 false。	'hello'.endsWith('lo') 返回 true
String.slice()	提取字符串的指定部分，返回一个新的字符串。	'hello'.slice(1, 4) 返回 'ell'
String.substring()	提取字符串的指定部分，返回一个新的字符串，参数顺序不敏感。	'hello'.substring(1, 4) 返回 'ell'
String.replace()	替换字符串中的子字符串，返回一个	'hello'.replace('e', 'a') 返回 'hallo'

方法	描述	示例代码
	新的字符串。	
String.toUpperCase()	将字符串中的所有字母转换为大写。	'hello'.toUpperCase() 返回 'HELLO'
String.toLowerCase()	将字符串中的所有字母转换为小写。	'HELLO'.toLowerCase() 返回 'hello'
String.trim()	移除字符串两端的空格。	' hello '.trim() 返回 'hello'
String.split()	将字符串分割为一个数组，指定分隔符。	'apple,banana,orange'.split(',') 返回 ['apple', 'banana', 'orange']
String.repeat()	返回一个新字符串，表示原字符串重复指定次数。	'hello'.repeat(3) 返回 'hellohellohello'
String.concat()	合并多个字符串并返回一个新的字符串。	'hello'.concat(' ', 'world') 返回 'hello world'
String.match()	用正则表达式检索字符串，返回匹配的结果。	'hello123'.match(/\d+/) 返回 ['123']
String.search()	返回字符串中匹配正则表达式的位置，未找到返回 -1。	'hello123'.search(/\d+/) 返回 5
String.replaceAll()	替换字符串中所有匹配的子字符串（需要支持 ES2021+）。	'hello hello'.replaceAll('hello', 'hi') 返回 'hi hi'
String.padStart()	在字符串的开头填充指定的字符，直到字符串达到指定的长度。	'5'.padStart(3, '0') 返回 '005'
String.padEnd()	在字符串的结尾填充指定的字符，直到字符串达到指定的长度。	'5'.padEnd(3, '0') 返回 '500'
String.localeCompare()	比较两个字符串，根据语言环境返回 -1, 0 或 1。	'apple'.localeCompare('banana') 返回 -1

获取字符串中的某个字符:

charAt() 和 [] 都可以获取字符串中的某个字符。

```
let str = 'hello';
console.log(str.charAt(1));    // 'e'
console.log(str[1]);                // 'e'
```

去除两端的空格:

```
let str = '      hello      ';
console.log(str.trim());    // 'hello'
```

分割字符串:

```
let str = 'apple,banana,orange';
let arr = str.split(',');    // ['apple', 'banana', 'orange']
```

字符串拼接:

```
let str1 = 'hello';
let str2 = 'world';
console.log(str1.concat(' ', str2));    // 'hello world'
```

PHP 中的标量类型转换 P364

在 PHP 中，标量类型转换指的是不同数据类型之间的自动或手动转换。PHP 作为一种弱类型语言，自动进行

类型转换，同时也允许开发者进行显式转换。以下是 PHP 中标量类型转换的几个重要概念。

隐式类型转换是 PHP 自动进行的转换。PHP 会根据操作的上下文，自动将不同类型的值转换为需要的类型。

字符串转换为数字：如果字符串中包含数字字符，PHP 会将其自动转换为数字进行计算。

整数转换为浮点数：当进行包含浮点数运算的操作时，PHP 会自动将整数转换为浮点数。

布尔值转换为数字或字符串：false 会被转换为 0，true 会被转换为 1。在字符串上下文中，false 会转换为空字符串，而 true 会转换为字符串 "1"。

隐式类型转换的一个关键特征是 PHP 会根据操作需求自动进行转换，开发者通常不需要显式干预。

显式类型转换是开发者通过明确的语法告诉 PHP 进行类型转换。这种转换要求开发者显式地转换一个类型为另一个类型。

PHP 输出结果方式 P366

方法	描述	示例代码
echo	输出一个或多个字符串。	echo "Hello, world!";
print	输出一个字符串，返回值为 1。通常用于输出一个值。	print "Hello, world!";
printf()	格式化输出。允许按照指定格式输出变量。	printf("Hello, %s!", "world");
print_r()	输出一个变量的可打印信息，适用于数组和对象。	print_r([1, 2, 3]);
var_dump()	输出一个变量的类型和值，适用于详细调试，显示变量的结构。	var_dump([1, 2, 3]);
var_export()	输出一个变量的可解析的 PHP 代码。通常用于输出数组或对象的结构。	var_export([1, 2, 3]);
htmlspecialchars()	将特殊字符转换为 HTML 实体，防止 HTML 注入。	echo htmlspecialchars('Test'); // 输出 Test
nl2br()	将字符串中的换行符 (\n) 转换为 HTML 标签。	echo nl2br("Hello\nworld!");
ob_start()	启用输出缓冲区，通常与 ob_get_contents() 和 ob_end_flush() 配合使用。	ob_start(); echo "Hello, world!"; ob_end_flush();
flush()	刷新输出缓冲区并将其内容发送到浏览器。	flush(); // 在输出缓冲区启用的情况下使用

功能	代码示例

声明变量	$variable = 10; // 声明变量并赋值
	$name = "John"; // 字符串类型变量
	$float = 3.14; // 浮动类型变量

输出内容	echo "Hello, World!"; // 输出字符串 print "This is a test."; // 输出字符串
if-else 语句	$age = 18; if ($age >= 18) { echo "You are an adult.";} else { echo "You are a minor.";}
switch 语句	$day = "Monday";

功能	代码示例
	switch ($day) {case "Monday":echo "Start of the week!";break; case "Friday":echo "End of the week!";break;default: echo "It's a regular day.";}
for 循环	phpfor ($i = 0; $i < 5; $i++) { echo $i . " ";}
while 循环	php$i = 0;while ($i < 5) { echo $i . " "; $i++;}
foreach 循环	php$fruits = ["Apple", "Banana", "Orange"];foreach ($fruits as $fruit) { echo $fruit . " ";}
包含文件	phpinclude "header.php"; // 包含一个外部文件 require "footer.php"; // 必须包含的文件
return 语句	phpfunction add($a, $b) { return $a + $b;}$result = add(5, 3);echo $result; // 输出 8
break 和 continue	phpfor ($i = 0; $i < 10; $i++) { if ($i == 5) { break; // 跳出循环 } echo $i . " ";}phpfor ($i = 0; $i < 10; $i++) { if ($i == 5) { continue; // 跳过当前循环的剩余部分，进入下一次循环 } echo $i . " ";}
isset()和 empty()	php$var = "Hello";if (isset($var)) { echo "Variable is set.";}if (empty($var)) { echo "Variable is empty.";}
switch 和 case	php$fruit = "apple";switch ($fruit) { case "apple": echo "Apple selected!"; break; case "banana": echo "Banana selected!"; break; default: echo "No fruit selected!";}
函数声明和调用	phpfunction greet($name) { echo "Hello, " . $name;}greet("Alice"); // 调用函数
对象和类	phpclass Person { public $name; public $age; function __construct($name, $age) { $this->name = $name; $this->age = $age; } function greet() { echo "Hello, my name is " . $this->name . " and I am " . $this->age . " years old."; }}$person = new Person("John", 30);$person->greet(); // 调用类中的方法

PHP 数组类型

类型	代码示例	输出示例
索引数组	$fruits = ["Apple", "Banana", "Orange"];echo $fruits[0];	"Apple"
关联数组	$person = ["name" => "John", "age" => 30, "city" => "New York"];echo $person["name"];	"John"
多维数组	$contacts = ["John" => ["phone" => "123-456", "email" => "john@example.com"], "Alice" => ["phone" => "789-101", "email" => "alice@example.com"]];echo $contacts["John"]["email"];	"john@example.com"

PHP 数组常用方法

方法	描述	代码示例	输出示例
count()	返回数组中的元素数量	$fruits = ["Apple", "Banana", "Orange"];echo count($fruits);	3
array_push()	向数组末尾添加一个或多个元素	$fruits = ["Apple", "Banana"];array_push($fruits, "Orange", "Grape");print_r($fruits);	["Apple", "Banana", "Orange", "Grape"]
array_pop()	移除数组末尾的元素，并返回该元素	$fruits = ["Apple", "Banana", "Orange"];$last = array_pop($fruits);echo $last;print_r($fruits);	"Orange" / ["Apple", "Banana"]
array_shift()	移除数组开头的元素，并返回该元素	$fruits = ["Apple", "Banana", "Orange"];$first = array_shift($fruits);echo $first;print_r($fruits);	"Apple" / ["Banana", "Orange"]

方法	描述	代码示例	输出示例
array_unshift()	将一个或多个元素添加到数组的开头	$fruits = ["Banana", "Orange"];array_unshift($fruits, "Apple", "Grape");print_r($fruits);	["Apple", "Grape", "Banana", "Orange"]
array_merge()	将两个或多个数组合并成一个数组	$array1 = ["Apple", "Banana"];$array2 = ["Orange", "Grape"];$merged = array_merge($array1, $array2);print_r($merged);	["Apple", "Banana", "Orange", "Grape"]
array_slice()	返回数组的一个部分，指定开始位置和长度	$fruits = ["Apple", "Banana", "Orange", "Grape"];$slice = array_slice($fruits, 1, 2);print_r($slice);	["Banana", "Orange"]
array_splice()	删除数组中的一部分并替换为其他元素	$fruits = ["Apple", "Banana", "Orange", "Grape"];array_splice($fruits, 1, 2, ["Lemon", "Mango"]);print_r($fruits);	["Apple", "Lemon", "Mango", "Grape"]
array_key_exists()	检查数组中是否存在指定的键	$person = ["name" => "John", "age" => 30, "city" => "New York"];if (array_key_exists("age", $person)) { echo "Age is: " . $person["age"]; }	"Age is: 30"
in_array()	检查某个值是否存在于数组中	$fruits = ["Apple", "Banana", "Orange"];if (in_array("Banana", $fruits)) { echo "Banana is in the array."; }	"Banana is in the array."
array_flip()	交换数组中的键和值	$array = ["a" => "Apple", "b" => "Banana"];$flipped = array_flip($array);print_r($flipped);	["Apple" => "a", "Banana" => "b"]
array_map()	通过回调函数操作数组中的每个元素	$numbers = [1, 2, 3, 4];$squared = array_map(function($num) { return $num * $num; }, $numbers);print_r($squared);	[1, 4, 9, 16]
array_filter()	使用回调函数过滤数组中的元素	$numbers = [1, 2, 3, 4, 5];$even = array_filter($numbers, function($num) { return $num % 2 == 0; });print_r($even);	[2, 4]
array_sort()	排序数组（如 sort(), asort(), ksort()等）	$fruits = ["Banana", "Apple", "Orange"];sort($fruits);print_r($fruits);	["Apple", "Banana", "Orange"]

方法	功能	代码示例	输出示例
count()	获取数组长度	$arr = [1, 2, 3]; echo count($arr);	3
sort()	数组升序排序	$arr = [3, 1, 2]; sort($arr); print_r($arr);	[1, 2, 3]
rsort()	数组降序排序	$arr = [3, 1, 2]; rsort($arr); print_r($arr);	[3, 2, 1]
asort()	值排序（关联数组）	$arr = ["b" => 2, "a" => 1]; asort($arr); print_r($arr);	["a" => 1, "b" => 2]
ksort()	键排序（关联数组）	$arr = ["b" => 2, "a" => 1]; ksort($arr); print_r($arr);	["a" => 1, "b" => 2]
in_array()	检查值是否存在	$arr = [1, 2, 3]; echo in_array(2, $arr);	true
array_search()	查找值的索引	$arr = [1, 2, 3]; echo array_search(2, $arr);	1

方法	功能	代码示例	输出示例
foreach	遍历数组	$arr = ["name" => "John", "age" => 30]; foreach ($arr as $key => $value) { echo "$key: $value\n"; }	"name: John" / "age: 30"

数组的键名和类型

方法	描述	代码示例	输出示例
数组键名类型	键名可以是整数、字符串、布尔值。布尔值会转换为整数 0 或 1。	$arr = [true => "yes", false => "no"];	[1 => "yes", 0 => "no"]
键类型转换	浮动类型会转换为整数，对象会调用 __toString() 方法作为键名。	$arr = [1.0 => "one", 1 => "two"];	[1 => "two"]
array_filter()	过滤数组中的元素，返回符合条件的元素。	$arr = [1, 2, 3, 4]; $filtered = array_filter($arr, function($item) { return $item % 2 == 0; });	[2, 4]
array_map()	通过回调函数操作数组中的每个元素。	$arr = [1, 2, 3]; $squared = array_map(function($item) { return $item * $item; }, $arr);	[1, 4, 9]
array_reduce()	将数组元素按顺序归约为一个值，通常用于求和、求积等。	$arr = [1, 2, 3, 4]; $sum = array_reduce($arr, function($carry, $item) { return $carry + $item; });	10
array_fill()	填充数组，生成指定长度的数组并填充相同的值。	$arr = array_fill(0, 5, "apple");	["apple", "apple", "apple", "apple", "apple"]
range()	生成一个范围数组，从一个数到另一个数。	$arr = range(1, 10);	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
&（引用）	数组按引用传递，修改一个数组会影响另一个数组。	$arr = [1, 2, 3]; $arr2 = &$arr;	影响原数组值
unserialize(serialize())	数组深拷贝，生成一个完全独立的新数组。	$arr = [[1, 2], [3, 4]]; $arr_copy = unserialize(serialize($arr));	[[1, 2], [3, 4]]

模式匹配 P385

函数	描述	示例代码	输出示例
preg_match()	匹配一个正则表达式模式，返回匹配成功（1）或失败（0）。	$pattern = "/\d+/"; $string = "There are 100 apples"; if (preg_match($pattern, $string, $matches)) { echo "Matched: " . $matches[0]; }	Matched: 100
preg_match_all()	匹配所有符合正则表达式的内容，返回所有匹配的结果。	$pattern = "/\d+/"; $string = "I have 100 apples and 200 oranges"; if (preg_match_all($pattern, $string, $matches)) { print_r($matches); }	Array ([0] => Array ([0] => 100 [1] => 200))
preg_replace()	将正则表达式匹配的内容替换为指定的字符串。	$pattern = "/\d+/"; $string = "There are 100 apples and 200 oranges"; $replacement = "X"; $result = preg_replace($pattern, $replacement, $string); echo $result;	There are X apples and X oranges

函数	描述	示例代码	输出示例
preg_split()	按照正则表达式的模式将字符串分割成数组。	$pattern = "/\s+/"; $string = "This is a test"; $result = preg_split($pattern, $string); print_r($result);	Array ([0] => This [1] => is [2] => a [3] => test)
preg_quote()	将字符串中的特殊字符转义成合法的正则表达式格式。	$pattern = preg_quote("$100", "/"); $string = "I have $100 in my wallet"; if (preg_match("/$pattern/", $string)) { echo "Found the amount"; }	Found the amount

表单处理 P386

在 PHP 中，处理表单提交通常使用 $_GET 或 $_POST 超全局数组来获取提交的数据。处理过程包括接收表单数据、验证输入、处理逻辑以及返回响应。以下是一个典型的表单提交处理流程的示例：

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>表单示例</title>
</head>
<body>

<form action="process_form.php" method="POST">
    <label for="name">姓名:</label>
    <input type="text" id="name" name="name" required>

    <label for="email">电子邮箱:</label>
    <input type="email" id="email" name="email" required>

    <label for="age">年龄:</label>
    <input type="number" id="age" name="age" required>

    <input type="submit" value="提交">
</form>

</body>
</html>
```

process_form.php

```
<?php
// 检查表单是否已提交
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // 获取表单数据
    $name = isset($_POST['name']) ? $_POST['name'] : '';
    $email = isset($_POST['email']) ? $_POST['email'] : '';
    $age = isset($_POST['age']) ? $_POST['age'] : '';
    // 简单的输入验证
    if (empty($name) || empty($email) || empty($age)) {
        echo "所有字段都必须填写。";
    } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "电子邮箱格式无效。";
    } elseif (!is_numeric($age) || $age <= 0) {
        echo "年龄必须是一个有效的数字且大于零。";
    } else {
        // 输入有效，进行处理（例如存储到数据库、发送邮件等）
        echo "表单提交成功！";
        echo "姓名: $name";
        echo "电子邮箱: $email";
        echo "年龄: $age";
        // 你可以在这里进行进一步的处理，比如将数据保存到数据库
    }
} else {
    echo "无效的请求方式。";
}
?>
```

表单的 method="POST" 表示提交数据将通过 HTTP POST 方法传递到 process_form.php。action="process_form.php" 指定了处理表单的 PHP 文件。使用 $_POST['name']、$_POST['email'] 和 $_POST['age'] 来获取用户提交的表单数据。

要确保用户输入的数据是合法的，避免注入攻击。在输出用户输入的数据时，**使用 htmlspecialchars() 来避免跨站脚本（XSS）攻击。**echo "姓名: " . htmlspecialchars($name)。echo "电子邮箱: " . htmlspecialchars($email);

防止跨站请求伪造（CSRF）攻击，可以使用 CSRF 令牌验证。

在表单中添加一个隐藏字段，存储一个随机生成的 CSRF 令牌，然后在处理表单时验证这个令牌。

```
// 在表单中生成 CSRF 令牌
session_start();
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
?>
<form action="process_form.php" method="POST">
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['csrf_token']; ?>">
    <!-- 其他表单字段 -->
</form>
```

然后在 process_form.php 中验证令牌：

```
session_start();
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // 检查 CSRF 令牌
    if ($_POST['csrf_token'] != $_SESSION['csrf_token']) {
        die("CSRF 校验失败！");
    }
    // 处理表单数据...
}
```

第十三章

要求 mysql 会用语句
p575 php 访问 mysql 重点掌握
p572 13.5

SQL 基础（原书写的更细一点，可以直接看） P563

PHP 连接 Mysql 数据库 P572

使用 mysqli 扩展连接 MySQL 数据库

mysqli 是 PHP 提供的一个接口，用于与 MySQL 数据库进行交互。它支持面向对象和过程化的方式。

面向对象的方式	过程化的方式

<?php
// 数据库连接配置
$servername = "localhost"; // 数据库主机名
$username = "root"; // 数据库用户名
$password = ""; // 数据库密码
$dbname = "test"; // 数据库名
// 创建数据库连接
$conn = new mysqli($servername, $username, $password,
	<?php
	// 数据库连接配置
	$servername = "localhost";
	$username = "root";
	$password = "";
	$dbname = "test";
	// 创建数据库连接
	$conn = mysqli_connect($servername, $username,

```
$dbname);
// 检查连接是否成功
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}
echo "连接成功";
// 关闭数据库连接
$conn->close();
?>
```

```
$password, $dbname);
// 检查连接是否成功
if (!$conn) {
    die("连接失败: " . mysqli_connect_error());
}
echo "连接成功";
// 关闭数据库连接
mysqli_close($conn);
?>
```

使用 PDO 扩展连接 MySQL 数据库

PDO（PHP Data Objects）是 PHP 提供的一种更通用的数据库访问方法，支持多种数据库管理系统（如 MySQL、PostgreSQL、SQLite 等）。PDO 提供了一个一致的接口，支持面向对象的编程风格。

使用 PDO 连接数据库

```
<?php
// 数据库连接配置
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "test";
try {
    // 创建 PDO 实例
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);

    // 设置错误模式为异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    echo "连接成功";
}
catch(PDOException $e) {
    echo "连接失败: " . $e->getMessage();
}
?>
```

执行 SQL 查询

使用 mysqli 执行查询

SELECT 查询

```
<?php
// 连接数据库
$conn = new mysqli("localhost", "root", "", "test");
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}
// 执行 SELECT 查询
$sql = "SELECT id, name FROM users";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    // 输出每行数据
```

INSERT 查询

```
<?php
// 连接数据库
$conn = new mysqli("localhost", "root", "", "test");
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}
// 执行 INSERT 查询
$sql = "INSERT INTO users (name, email) VALUES ('John Doe',
'john@example.com')";
if ($conn->query($sql) === TRUE) {
    echo "新记录插入成功";
```

```
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["name"]. "";
    }
} else {
    echo "0 结果";
}
$conn->close();
?>
```

```
} else {
    echo "错误: " . $sql . "" . $conn->error;
}
$conn->close();
?>
```

使用 PDO 执行查询

SELECT 查询

```
<?php
// 连接数据库
try {$conn = new PDO("mysql:host=localhost;dbname=test", "root", "");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // 执行 SELECT 查询
    $stmt = $conn->prepare("SELECT id, name FROM users");
    $stmt->execute();
    // 设置结果集为关联数组
    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
    // 输出每行数据
    foreach ($result as $row) {
        echo "id: " . $row["id"] . " - Name: " . $row["name"] . "";
    }
}
catch(PDOException $e) {
    echo "连接失败: " . $e->getMessage();
}
?>
```

INSERT 查询

```
<?php
// 连接数据库
try {
    $conn = new PDO("mysql:host=localhost;dbname=test", "root", "");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // 执行 INSERT 查询
    $stmt = $conn->prepare("INSERT INTO users (name, email) VALUES (:name, :email)");
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':email', $email);
    // 插入数据
    $name = "Jane Doe";
    $email = "jane@example.com";
    $stmt->execute();
    echo "新记录插入成功";
}
catch(PDOException $e) {
    echo "连接失败: " . $e->getMessage();
}
?>
```

关闭数据库连接

mysqli 使用 mysqli_close($conn) 来关闭连接。

PDO 连接在脚本结束时会自动关闭，因此通常不需要显式调用关闭连接。如果需要显式关闭，可以将 PDO 对象设为 null： $conn = null;

错误处理

mysqli 可以通过 connect_error 或 error 属性来检查错误。

PDO 使用 setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION) 设置异常模式，这样在出错时会抛出异常，可以使用 try-catch 语句捕获并处理。

mysqli 是针对 MySQL 专门设计的扩展，适用于 MySQL 数据库，支持过程化和面向对象编程。PDO 是一种更通用的数据库访问方式，支持多种数据库，适用于需要跨平台或跨数据库的应用。在选择时，mysqli 更适用于仅使用 MySQL 数据库的项目，而 PDO 更适合需要与多个数据库系统交互的应用。

因为 PHP 是动态类型的，所以它没有类型声明。事实上，没有办法也不需要声明变量的类型。每次为变量赋值时都会设置变量的类型。未赋值的变量（有时称为未绑定变量）的值为 NULL，这是 NULL 类型的唯一值。如果在表达式中使用未绑定变量，则 NULL 将被强制转换为由使用上下文指定的值。如果上下文指定一个数字，则 NULL 被强制为 0；如果上下文指定字符串，则 NULL 被强制为空字符串。

PHP 有一个整数类型，称为整数。该类型对应于 C 及其后继类型的 long 类型，这意味着它的大小是运行程序的机器中的一个字的大小。

PHP 中的字符是单字节；不支持 Unicode。没有字符类型。

附录

附录 A CSS 属性表

文本属性		背景属性	
color	文本颜色	background-color	背景颜色
font-size	字体大小	background-image	背景图片
font-family	字体	background-position	背景图片位置
font-weight	字体粗细	background-size	背景图片大小
font-style	字体样式（如斜体）	background-repeat	背景图片重复方式
text-align	文本对齐方式	background-attachment	背景图片是否随页面滚动
text-decoration	文本修饰（如下划线）	background-clip	背景绘制的区域
line-height	行高	background-origin	背景图片定位的起点
letter-spacing	字母间距		
word-spacing	单词间距		
text-transform	字母大小写转换		
text-indent	首行缩进		

边框属性		布局属性	
border	边框简写	display	显示方式（如 block、inline、flex）
border-width	边框宽度	position	定位方式（如 absolute、relative、fixed）
border-style	边框样式（实线、虚线等）	top / right / bottom / left	定位偏移
border-color	边框颜色	z-index	层叠顺序
border-radius	圆角	float	浮动
border-top / border-right / border-bottom / border-left	分别设置每个边的边框	clear	清除浮动
visibility	可见性	overflow	溢出处理方式

尺寸属性

width	宽度
height	高度
max-width / min-width	最大/最小宽度

max-height / min-height	最大/最小高度
padding	内边距
margin	外边距
box-sizing	定义如何计算元素的宽高

列表属性

list-style	列表样式简写
list-style-type	项目符号的类型（如圆点、数字等）
list-style-position	项目符号的位置
list-style-image	项目符号的图像

表格属性

border-collapse	边框合并模式
border-spacing	单元格间距
table-layout	表格布局方式
caption-side	标题的位置
empty-cells	是否显示空单元格

动画与过渡

transition	过渡效果简写
transition-property	过渡属性
transition-duration	过渡持续时间
transition-timing-function	过渡的速度曲线
animation	动画简写
animation-name	动画名称
animation-duration	动画持续时间
animation-timing-function	动画速度曲线
animation-delay	动画延迟

弹性盒模型（Flexbox）属性

display: flex	启用弹性布局
flex-direction	子元素排列方向
justify-content	主轴上的对齐方式
align-items	交叉轴上的对齐方式
flex-wrap	是否换行
align-content	多行内容的对齐方式
order	子元素的排列顺序
flex-grow	子元素的放大比例
flex-shrink	子元素的缩小比例
flex-basis	主轴上元素的初始大小

网格布局（Grid）属性

display: grid	启用网格布局
grid-template-columns	定义列的宽度
grid-template-rows	定义行的高度
grid-column / grid-row	元素的列/行范围
grid-gap	网格间距

justify-items	网格项的水平对齐方式
align-items	网格项的垂直对齐方式

其他常用属性

opacity	不透明度
cursor	鼠标指针样式
box-shadow	盒子阴影
text-shadow	文本阴影
filter	图像滤镜效果
transform	2D/3D 转换
perspective	3D 透视效果