**1.Windows中常用快捷键**

Win+e 打开文件资源管理器

Backspace 返回上一级文件

Win+d 显示/隐藏桌面

Win+r 打开运行（cmd进入终端/calc计算器/mspaint画图）

Alt+Tab 切换窗口（向右切换）

Alt+Shift+Tab 切换窗口（向左切换）

Alt+F4 关闭窗口

F2 重命名文件

**2.软件工程**

aba -> bug -> debug

IBM -> DOS -> Bill Gates -> Microsoft -> Windows

软件工程学科：

（1）软件过程

（2）软件测试

（3）软件质量

（4）软件标准

**3.软件过程——软件生命周期(重点)**

软件开发流程

**（1）软件定义期**

可行性研究阶段——《可行性研究报告》

技术、人力、时间、政策、回报率、风俗

需求分析阶段——《软件需求规约》

功能性需求、非功能性需求

**（2）软件开发期**

概要设计阶段

子系统、模块、模块的具体功能

详细设计阶段

主题内容、使用的框架、算法、数据库

编码实现阶段

设计师：出效果图

前端：把效果图转化成网页（html，css，js）

后台：为前端提供页面上所需要的数据

软件测试阶段

找项目中所出现的缺陷和问题

**（3）软件维护期**

项目部署阶段

部署到服务器

项目维护阶段

**4.学子商城模块及子系统**

后台：www.codeboy.com/admin/login.hrml

|  |  |
| --- | --- |
| 前台子系统 | 用户模块、商品模块、购物车模块 |
| 后台子系统 | 用户模块、商品模块、订单模块 |
| 移动端子系统 | 用户模块、商品模块、购物车模块 |

**5.服务器**

为客户端提供各种服务的功能强大的计算机

访问服务器：

（1）服务器的域名/IP地址

（2）每一个服务都有对应的端口号

（3）提供访问该服务器所使用的协议

例如：http://www.tmooc.cn：80/

|  |  |  |
| --- | --- | --- |
| http | www.tmooc.cn | 80（网页默认端口） |
| 协议 | 域名/IP地址 | 端口 |

当输入http://www.taobao.com是如何获取到淘宝首页的？

浏览器（输入域名/IP地址）->协议（http）->服务器（web服务—调取：文件服务、数据库）->反馈浏览器

访问自己电脑创建的web服务器

启动web服务器 /webServer/start.cmd

http://127.0.0.1:80 使用IP地址访问

http://localhost:80 使用域名访问

获取自己电脑在局域网的IP地址

在命令行输入ipconfig 查看IP地址

位于IPv4中

http://176.204.15.173:80 使用局域网IP地址访问服务器

**1.项目中如何保存数据**

内存（临时保存数据）、文件（Excel、文档）、云服务器、数据库服务器

**2.数据库的概述**

database：数据库，以特定的结构批量存储业务的数据的软件。

数据库发展史：

网状数据库->层次型数据库->**关系型数据库**（当前主流）->非关系型数据库

|  |
| --- |
| 关系型数据库中数据的逻辑结构 |
| Server=>Database=>Table=>Row=>Column |

关系型数据库种类：

SQlite：微型数据库，常用于移动设备

MYSQL（免费，后被Oracle收购）：开源中小型数据库，可用于各种操作系统——同一作者后编写MariaDB

PosrgreSQL：开源中小型数据库

SQL Server：只用于Windows系统，Micrsoft开发的中型数据库

Oracle：Oracle公司开发的中大型数据库，可用于各种操作系统

**3.使用MySQL数据库：**

Oracle分支：https://www.mysql.com/

Martin分支：https://Mariadb.org/

|  |
| --- |
| xampp是一个服务器套装，包含MySQL、Apache、PHP等，网址xampp.org |

**（1）MySQL的部署结构**

服务器软件：负责存储/维护数据——银行服务器机房

c:/xampp/mysql/bin/mysqld.exe

客户端软件：负责向服务器发起增删改查命令——ATM机

c:/xampp/mysql/bin/mysql.exe

**（2）使用MySQL客户端连接到服务器端**

进入：mysql.exe（客户端） -h127.0.0.1（ID） -P3306（大写P端口） -uroot（管理员用户、普通用户也可进入，权限更少） -p（密码）

【简化登录——管理员：mysql -uroot；其他用户：mysql -h127.0.0.1或mysql -P3306】

**4.MySQL常用管理指令**

退出：quit/exit（快捷键Ctrl+c）

显示数据库服务上所有的数据库：show databases**;**(后面加分号**;**结束)

进入指定的数据库：use 数据库名字;

查看所在的数据库：select database()**;**

显示当前数据库中所有的表：show tables**;**

查看表头：desc 表名**;**

创建新的数据库：create 新数据库名字**;**

**5.SQL命令**

SQL：Structured Query Language,结构化查询语言，用于操作关系型数据库服务器中的数据——增删改查。

SQL命令的两种执行方式：

**（1）交互模式**：客户端输入一行，点击回车，服务期执行一行。——适用于临时性的查看数据

**（2）脚本模式**：客户端把要执行的多行命令编写在一个脚本文件中一次性提交给服务器执行。——适用于批量的操作

创建脚本文档输入多个命令：02.sql（#后缀名不受影响）

打开客户端，将新建的脚本文档拉入"<"后面：mysql -uroot <C:\xampp\htdocs\CL\day01.docx\02.sql（#脚本存放路径）

**6.SQL语法规范**

（1）每天SQL语句一英文分号为结束，一条语句可以跨越多行，见到分号认为结束；

（2）假如某一条语句出现语法错误，则词条语句以后及后边所有的语句都不会执行；

（3）SQL命令不区分大小写，习惯上关键字（命令）用大写，非关键字用小写；

（4）SQL命令中可以使用单行注释（#...）和多行注释（/\*...\*/），注释内容不会被服务器所执行。

**7.日常开发中的常用的SQL语句：**

（1）丢弃指定的数据库jd，如果存在的话

DROP DATABASE IF EXISTS student;

（2）创建新的数据库

CREATE DATABASE student;

（3）进入刚刚创建的数据库

USE student;

（4）创建保存数据库的表student

CREATE TABLE student(

sid INT,

name VARCHAR(6),

sex VARCHAR(1),

score INT

);

（5）向数据表中插入数据

INSERT INTO student VALUES("2","kate","F","92"),("3","lily","F","93");

（6）查询数据表中的数据

SELECT \* from student;

（7）删除sid为3的数据（一定要加条件）

DELETE FROM student **WHERE sid='3'**;

（8）修改sid为4的数据，性别改为F，分数改为100

UPDATE student SET sex='F',score='100' **WHERE sid='4'**;

课后任务：

（1）复习今天内容，再次编写代码

（2）练习：

编写脚本文件xz.sql，先丢弃再创建数据库xz，进入数据库，创建数据表user，包含uid、uname、upwd（密码）、email、phone、sex、user\_NAME（真实姓名）、register\_time（注册时间）、is\_online（是否在线）

注册时间：'2018-12-28'

是否在线：'y'/'n'

插入5条数据，删除其中1条数据，修改2条数据，最后查询结果。

（3）预习MySQL中的列类型。

**1.标准的SQL语句分类**

|  |  |  |
| --- | --- | --- |
|  | DDL:Data Define Languge  定义数据 | DROP/CREATE/ALTER |
|  | DML:Data Munipulate Language  操作数据 | INSERT/DELETE/UPDATE |
|  | DQL:Date Query Language  查询数据 | SELECT |
|  | DCL：Date Control Language  控制用户权限 | GRANT(授权)/REVOKE（收权） |

**2.数据库中的中文乱码**

**（1）计算机存储英语字符**

ASCII：总共有128个，对所有的英文字符及其标点进行了编码。

Latin-1：总共有256个，兼容ASCII码，同时对欧洲符号进行了编码。（MySQL默认使用这种编码存储数据）

**（2）计算机中存储中文字符**

GB2312：对6千多常用的汉子进行了编码，兼容ASCII码。

GBK：对2万多汉子进行了编码，兼容ASCII码。

BIG5：台湾繁体字编码，兼容ASCII码，不兼容GB2312/GBK。

Unicode：对世界上主流国家的语言进行了编码，兼容ASCII码，不兼容GB系列，BIG5等。

具体分为UTF-8、UTF-16、UTF-32三种存储方案。

**（3）解决MySQL数据库中的乱码**

SQL脚本文件另存为的编码

客户端连接服务器使用的编码（脚本文档最上边加SET NAMES UTF8;）

数据库存储的编码(创建数据库时后面加CHARSET=UTF8 )

|  |
| --- |
| 设置editplus默认编码  Tools->preferences->file->Default Enconding |

练习：创建脚本文件02\_sina.sql，先丢弃在创建数据库sina，设置编码为UTF8，创建数据表news，包含nid、title（标题）、author（作者）、addtime（添加时间）、content（内容）

插入5条记录，更改第二条记录，更改第三条记录

**3.列类型**

创建数据表时，设置指定的列存储的数据类型

CREATE TABLE news（nid 列类型）

**（1）数值类型——引号可加、可不加**

|  |  |  |  |
| --- | --- | --- | --- |
| TINYINT | 微整型 | 占1字节 | -128~127 |
| SMALLINT | 小整型 | 占2字节 | -32768~32767 |
| MEDIUMINT | 中整型 | 占3字节 | -8388608~8388607 |
| INT | 整型 | 占4字节 | 11位（包含一个符号） |
| BIGINT | 大整型 | 占8字节 | 范围很大 |

1TB-> 1024GB->MB->KB->Byte

1Byte（字节）->8Bit（位）

1字符=8Byte

**浮点型**

FLOAT，单精度浮点型，占4字节，范围3.4e38，比INT的范围大得多，可能会产生误差——牺牲部分小数做指数，如3.4\*10^38。

DOUBLE，双精度浮点型，占8字节，范围比BGIINT大得多。

DECIMAL（M,D）,定点小数，不会产生误差，M代表总的有效位数，D代表小数点后的有效位数。

BOOL，布尔型，只有两个值TRUE/FALSE（不能使用引号）或者1/0，真正存储的时候，会使用TINYINT类型，存储的值位1/0。

**（2）日期时间类型——必须加引号**

DATE，日期类型'2018-01-01'

TIME，时间类型'14:30:20'

DATETIME，日期时间类型'2018-01-01 14:30:20'

**（3）字符串类型——必须加引号**

VARCHAR(M)，变长字符串，不会产生空间浪费，操作速度相对较慢，M的最大值为65535字符。

CHAR(M)，定长字符串会产生空间浪费，操作速度相对较快，M的最大值为255字符，用于存储手机号码、身份证等固定长度的字符。

TEXT(M)，大型变长字符串，最多存储2G。

|  |  |  |
| --- | --- | --- |
| 存储内容 | CHAR(5) | VARCHAR(5) |
| a | a\0\0\0\0（5个字符占一个字符，其余用0补齐） | a\0 |
| ab | ab\0\0\0 | ab\0 |
| 一 | 一\0\0\0\0 | 一\0 |

练习1：编写脚本文件03\_xzezi.sql，县丢弃再创建数据库xuezi，设置编码为DTF8，创建保存笔记本表laptop，包含lid，title（标题），price（价格），stockcount（库存量），shelftime（上架时间），isindex（是否在首页显示）

插入4条记录

练习2：编写脚本文件04-tedu.sql，丢弃数据库再创建数据库tedu，设置编码类型UTF8，创建保存部门数据的表dept，包含did、dname（部门名称）、empcount（员工数量），插入一下数据

10 研发部 3

20 市场部 2

30 运营部 2

创建保存员工数据的表emp，包含eid、ename、sex、birthday、salary（工资）、deptid（所属部门编号）

插入7条记录

**4.列约束**

MySQL可以对插入的数据进行特定的验证，只有满足条件才允许插入到数据表中，否则被认为是非法的插入。

例如：一个人的性别只能是男或女，一个人的年龄只能在0~100之间。

**（1）主键约束——PRIMARY KEY**

声明了主键列上的值不能出现重复，表中查询的记录会按照主键由小到大排序——加快查找速度，通常主键添加在编号列中，主键的列中不能使用NULL。

注意：一个表中只能出现一个主键。

|  |
| --- |
| 程序中NULL的含义：空的，空缺的  指应该有某个数据，暂时没有确定具体的值。  例如：一个刚入职的员工无法确定他的部门，未确定部门经理... |

**（2）唯一约束——UNIQUE**

声明了唯一约束的列上不能插入重复的值，但允许插入NULL，而且允许多次插入NULL。

说明：NULL和任何的值都不相等，甚至两个NULL也不相等。

练习：编写脚本文件05\_dangdang.sql，县丢弃在创建数据库dangdang，设置编码UTF8，创建保存图书的表book，包含bid、title、author、pubTime（出版时间）、publish（出版社）、isOnsale是否在售。

课后任务：

（1）复习今天内容，重新编写代码；

（2）编写脚本xz.sql，先丢弃再创建数据库xz，设置编码UTF8，进图该数据库；创建保存笔记本家族的表laptop\_family(fid,fname,laptopCount数量)

10 联想 2

10 戴尔 2

30 苹果 1

创建保存笔记本的表laptop（lid、title、price价格、spec规格、detail详细介绍、shelfTime上架时间、familyid所属家族编号）

插入5条记录

**（3）非空约束——**NOT NULL

声明了非空约束的列上不能插上NULL

**（4）默认值约束——**DEFAULT

可以使用DEFAULT关键字来声明默认值，有两种方式可以应用默认值：

INSERT INTO laptop VALUES（"5","戴尔",...DEFAULT,"20"）

INSERT INTO laptop(lid,title,price) VALUES("6","戴尔1","6666")

没有指定的都会自动调用默认值

**（5）检查约束——**CHECK

CREATE TABLE student(

age TINYINT CHECK(age>=18 AND age<=60)

);

MySQL不支持检查约束，会降低数据的插入速度。

**（6）外键约束——**FOREIGN KEY

声明了外键约束的列，取值必须再另一个表的**主键**上出现过，列类型也要保持一致，取值可以为NULL

FOREIGN KEY(familyid此表中需要约束的列) REFERENCES laptop\_family(fid)另一个表上的主键约束的列

**(7)自增列**

AUTO\_INCREMENT：自动增长，加入一个列上声明了一个自增列，无需手动复制，直接设置为NULL，会获取当前的最大值，然后加1插入。

注意：

自增列允许手动复制；

只是由于整数型的主键上。

练习：编写脚本文件，02\_tedu.sql，先丢弃在创建数据库tedu。

创建保存部门数据的表dept（did、dname部门名称）

10 研发部

20 市场部

30 运营部

40 测试部

创建保存员工数据的表emp（eid、ename、sex、birthday、salary、deptId所属部门编号）

插入15条记录，有一个员工没有部门，有一个部门没有员工

数据库中日期时间使用距离计算机元年（1970-1-1 0:0:0）的毫秒数，使用JS是可以获取任意的日起时间类型。要使用BIGINT列类型来存储。

**1.简单查询**

**（1）查询特定的列**

示例：查询出所有员工的姓名、工资、生日

SELECT ename,salary,birthday FROM emp;

练习：查询出所有员工的编号、姓名、性别、所属部门编号

SELECT eid,ename,sex,deptId FROM emp;

**（2）查询所有的列**

SELECT \* FROM emp;

**（3）给列起别名**

示例：查询所有员工的姓名和工资，使用汉字的别名显示

SELECT ename AS 姓名,salary AS 工资 FROM emp;

|  |
| --- |
| 注意：AS关键字是可以省略的，但是需要保留空格。 |

练习：查询所有员工的编号，姓名，性别，生日，使用汉字别名

SELECT eid编号,ename姓名,sex性别,birthday生日 FROM emp;

练习：查询所有员工的编号和姓名，使用一个英文字母作为别名

SELECT eid AS B,ename AS C FROM emp;

**（4）只显示不同的记录/合并相同的记录**

示例：查询出员工所在的部门都有哪些

SELECT DISTINCT deptId FROM emp;

练习：查询出公司都有哪些性别的员工

SELECT DISTINCT sex FROM emp;

**（5）在查询时执行计算**

示例：计算2\*15-18/6+3

SELECT 2\*15-18/6+3;

练习：查询出所有员工的姓名及其年薪

SELECT ename,salary\*12 FROM emp;

练习：假设每个员工的工资涨500，年终奖没人5000，计算年薪，同时起汉子别名

SELECT ename 姓名,(salary+500)\*12+5000 年薪 FROM emp;

**（6）查询的结果集排序**

示例：查询出所有部门，结果按照编号由小到大

SELECT \* FROM dept ORDER BY did ASC; #ascendant升序

示例：查询所有部门，结果按照序号由大到小

SELECT \* FROM dept ORDER BY did DESC; #descendant降序

练习：查询所有的员工信息，结果是按照工资从大到小排序

SELECT \* FROM emp ORDER BY salary DESC;

练习：查询素有的员工信息，结果按照年龄从大到小排序（生日从小到大）

SELECT \* FROM emp ORDER BY birthday ASC;

练习：查询所有员工的信息，结果按照年龄从小到大排序

SELECT \* FROM emp ORDER BY birthday DESC;

练习：查询所有的员工信息，结果按照姓名中的字母从a~z排序

SELECT \* FROM emp ORDER BY ename ASC;

|  |
| --- |
| ORDER BY可以按照数值、字符串、日期的排序  如果不指定排序规则，默认是升序排列（ASC）。 |

练习：查询所有的员工信息，结果按照工资的升序排列，如果工资相同按照姓名排列

SELECT \* FROM emp ORDER BY salary,ename;

练习：查询所有员工的信息，结果按照工资的降序排列，所有的女员工显示在前面。

SELECT \* FROM emp ORDER BY sex,salary DESC;

**（7）条件查询**

示例：查询出编号为5的员工所有的信息

SELECT \* FROM emp WHERE eid="5";

练习：查询出姓名为King的员工的编号、工资、生日

SELECT eid,salary,birthday FROM emp WHERE ename="King";

练习：查询出20号部门下所有员工的信息

SELECT \* FROM emp WHERE deptId="20";

练习：查询出所有的男员工信息

SELECT \* FROM emp WHERE sex="1";

练习：查询出工资在5000以上员工所有的信息

SELECT \* FROM emp WHERE salary>="5000";

|  |
| --- |
| > < >= <= = !=(不等于) |

练习：查询出1991-1-1后出生的员工

SELECT \* FROM emp WHERE birthday>"1991-1-1";

练习：查询出不在10号部门的员工所有信息

SELECT \* FROM emp WHERE deptId!=10;

练习：查询出没有明确部门的员工的所有信息

SELECT \* FROM emp WHERE deptId IS NULL;

练习：查询出由明确部门的员工所有信息

SELECT \* FROM emp WHERE deptId IS NOT NULL;

练习：查询出工资在6000以上的所有女员工所有信息

SELECT \* FROM emp WHERE salary>6000 AND sex=0;

练习：查询出工资在5000~8000之间的所有员工信息

SELECT \* FROM emp WHERE salary>=5000 AND salary<=8000;

SELECT \* FROM emp WHERE salary BETWEEN 5000 AND 8000;（BETWEEN包含=之间的数据）

练习：查询工资不在5000~8000之间员工的信息

SELECT \* FROM emp WHERE salary<5000 OR salary>8000;

SELECT \* FROM emp WHERE salary NOT BETWEEN 5000 AND 8000;

练习：查询出1990年之前和1993年以后出生的员工所有信息

SELECT \* FROM emp WHERE birthday>"1993-12-31" OR birthday<"1990-1-1";

练习：查询出1993年出生的员工所有信息

SELECT \* FROM emp WHERE birthday BETWEEN "1993-01-1" AND "1993-12-31";

练习：查询出20号和30号所有员工的信息

SELECT \* FROM emp WHERE deptId=20 OR deptId=30;

SELECT \* FROM emp WHERE deptId IN(20,30);（IN包含20或者30）

练习：查询出不在20号或者30号部门员工所有信息

SELECT \* FROM emp WHERE deptId NOT IN(20,30);

|  |
| --- |
| AND/OR  BETWEEN...AND/NOT BETWEEN...AND  IS NULL/IS NOT NULL  IN/NOT IN |

**（8）分页查询**

假如查询的结果集由太多的数据，一次显示不完，可以做成分页显示。

需要两个条件：当前的页码、每页的数据量。

SELECT \* FROM emp LIMIT start,count;

start：是一个数字，从结果集的那一条开始查询。

count：是一个数字，表示每一页的数据量。

|  |
| --- |
| 每页开始的算法：  start=(页码-1)\*count |

假设每一页显示5条记录

第一页：SELECT \* FROM emp LIMIT 0,5;

第二页：SELECT \* FROM emp LIMIT 5,5;

第三页：SELECT \* FROM emp LIMIT 10,5;

练习：假设每一页显示7条记录，写出前三页的SQL语句

第一页：SELECT \* FROM emp LIMIT 0,7;

第二页：SELECT \* FROM emp LIMIT 7,7;

第三页：SELECT \* FROM emp LIMIT 14,7;

|  |
| --- |
| 注意：start（开始）和count（每页的数量）不能是字符串的格式，必须是整数。 |

课后任务：

（1）复习今天内容，删除代码，保留住世重新编写SQL语句；

（2）练习：

查询出工资8000以上的女员工的姓名、性别、生日，结果集按照工资的降序排列，取前两个人。

（3）查看学子商城数据库说明书，尝试创建。

SELECT ename,sex,birthday FROM emp WHERE salary>8000 AND sex=0 ORDER BY salary DESC LIMIT 0,2;

**顺序WHERE -> ORDER BY -> LIMIT**

|  |
| --- |
| 1.cmd命令行source拖拽\_.sql文件  2.set charset gbk;  3.查询  （这样不会出现乱码）  项目经理博客nighthighness.site |

**（9）模糊查询**

示例：查询出员工的姓名含有字母e的所有信息

SELECT \* FROM emp WHERE ename LIKE "%e%";e前面和后面有任意个字符

练习：查询出姓名以e结尾的员工所有的信息

SELECT \* FROM emp WHERE ename LIKE "%e";e前面有任意多个字符，后面没有字符

练习：查询出姓名以e开头的员工所有的信息

SELECT \* FROM emp WHERE ename LIKE "e%";e后面有任意多个字符，前面没有字符

练习：查询出姓名中倒数第二个字符为e的员工所有信息

SELECT \* FROM emp WHERE ename LIKE "%e\_";"\_"表示e后面有1个字符

|  |
| --- |
| SQL中提供了两个模糊查询的字符  % 匹配任意多个字符 >=0  \_ 匹配任意1个字符 =1  注意：以上两个匹配不能使用（=），必须使用LIKE关键字。 |

**2、复杂查询**

**（1）聚合查询/分组查询**

示例：查询出所有员工的数量

SELECT COUNT(eid) FROM emp;

练习：使用员工的姓名来查询员工的数量

SELECT COUNT(ename) FROM emp;

练习：使用员工的部门编号来查询员工的数量

SELECT COUNT(deptId) FROM emp;

SELECT COUNT(\*) FROM emp; #推荐写法

|  |
| --- |
| 函数:是一个功能体，提供若干个数据，产生特定结果。  MySQL提供的聚合函数：  COUNT(...)/SUM(...)/AVG(...)/MAX(...)/MIN(...) |

练习：查询出所有员工的工资总和

SELECT SUM(salary) FROM emp;

练习：查询出男员工平均工资

SELECT AVG(salary) FROM emp WHERE sex=1;

练习：查询出女员工平均工资

SELECT SUM(salary)/COUNT(\*) FROM emp WHERE sex=0;

练习：查询工资最高的员工

SELECT MAX(salary) FROM emp;

练习：查询工资最低的员工

SELECT MIN(salary) FROM emp;

练习：查询年龄最大的员工

SELECT MIN(birthday) FROM emp;

|  |
| --- |
| 分组查询：只能查询分组条件和聚合函数 |

示例：查询出每个部门的员工数量（部门deptId为分组条件）

SELECT deptId,COUNT(\*) FROM emp GROUP BY deptId;

练习：查询出男女员工的平均工资、最高工资、最低工资（性别男女sex为分组条件）

SELECT sex,AVG(salary),MAX(salary),MIN(salary) FROM emp GROUP BY sex;

|  |
| --- |
| YEAR()=... 获取日期中的年份  MONTH()=... 获取日期中的月份 |

示例：查询出1991年出生的员工所有信息

SELECT \* FROM emp WHERE YEAR(birthday)=1991;

练习：查询3月份出生的员工

SELECT \* FROM emp WHERE MONTH(birthday)=3;

练习：查询tom出生年份

SELECT YEAR(birthday) FROM emp WHERE ename="tom";

**（2）子查询**

示例：查询出研发部所有的员工信息

步骤1：查询出研发部的部门编号——10

SELECT did FROM dept WHERE dname="研发部";

步骤2：查询10号部门都有那些员工

SELECT \* FROM emp WHERE deptId=10;

综合：SELECT \* FROM emp WHERE deptId=(

SELECT did FROM dept WHERE dname="研发部"

);

|  |
| --- |
| 总结：把一个SQL语句的查询结果作为另一个SQL语句的查询条件 |

练习：查询出比tom工资高的员工所有信息

SELECT \* FROM emp WHERE salary>(

SELECT salary FROM emp WHERE ename="tom");

练习：查询出和tom同一年出生的员工所有信息

SELECT \* FROM emp WHERE YEAR(birthday)=(

SELECT YEAR(birthday) FROM emp WHERE ename="tom"

);

**（3）多表查询/跨表查询**

示例：查询出所有的员工的姓名及其部门名称

SELECT ename,dname FROM emp,dept;

错误：产生笛卡尔积！

如何避免出现笛卡尔积，添加查询条件。

SELECT ename,dname FROM emp,dept WHERE deptId=did;

|  |
| --- |
| 上述查询语法是SQL-92，无法查询没有部门的员工，也不能查询没有员工的部门。  SQL-99中提出了新的夺标查询的方法。 |

内连接INNER JOIN...ON 和之前的查询结果一样！

SELECT ename,dname FROM emp INNER JOIN dept ON deptId=did;

外连接：

a:左外连接LEFT OUTER JOIN...ON

SELECT ename,dname FROM emp LEFT JOIN dept ON deptId=did;

|  |
| --- |
| 显示左侧表中所有的记录，OUTER可以省略 |

b:右外连接RIGHT OUTER JOIN...ON

SELECT ename,dname FROM emp RIGHT JOIN dept ON deptId=did;

|  |
| --- |
| 显示右侧表中所有的记录，OUTER可以省略 |

全连接 FULL JOIN

|  |
| --- |
| 显示左侧和右侧表中所欲的记录——MySQL不支持全连接 |

MySQL中使用UNION/UNION ON代替全连接

UNION——合并相同的记录

UNION ALL——不合并相同的记录

(SELECT ename,dname FROM emp LEFT JOIN dept ON deptId=did)

UNION

(SELECT ename,dname FROM emp RIGHT JOIN dept ON deptId=did);

学习一门编程语言步骤

（1）了解背景知识：历史、现状、特点、应用场景

（2）搭建开发环境，编写hello world

（3）变量和常量

（4）数据类型

（5）运算符

（6）逻辑结构

（7）通用小程序

（8）函数和对象

（9）第三方的库、框架

（10）实用的项目

**1.JS概述**

**（1）历史**

1995年，JS最早出现在Netscape的浏览器中；

1997年，JS的第一个标准ECMA-262；

2009年，JS遵循了CommonJS规范，开始向服务器端发展。

**（2）现状**

JS既可以运行在客户端，也可以运行在服务器端。

**（3）特点**

编译执行，编译一行执行一行。

弱类型语言

基于对象

跨平台

**（4）应用领域**

用于开发浏览器的交互效果。

用于创建web服务器，操作数据库等服务器相关内容。

**2.JS的执行环节**

**（1）浏览器自带JS解释器**

创建01.js、01.html，把01.js嵌入到01.html中

|  |
| --- |
| <script src='01.js'></script> |

**（2）NODEJS下的JS解释器**

运行 -> cmd -> 查看NODEJS版本node -v

使用NODEJS执行js文件

node C:/xammp/.../o1.js

**3.JS语法规范**

（1）区分大小写

（2）每行代码结尾的分号可加可不加，提倡都加上

（3）分为单行注释（//...）和多行注释（/\*...\*/）

**4.变量**

变量就是用于存储数据的容器。

x=1——将数值1存储在x中

**（1）声明变量**

使用var关键字，例如var ename='tom';

**（2）变量的命名规则**

变量名称可以是字母、数字、下划线、美元符号$；不能以数字开头。

多个英文连词之间使用下划线命名或者驼峰命名方式

emp\_name、empName

不能使用关键字或者保留字作为变量名称。

**（3）一次性声明多个变量,用逗号隔开**

var a=5,...,c=7;

console.log(a,...,c);

练习：一次性声明多个变量，分别保存语文、数学、总成绩；打印总成绩。

var chinese=50,

math=60,

total=a+b;

console.log(total);

**（4）变量的注意**

var a;

console.log(a);

声明了变量未赋值，此时的值为undefined

console.log(a);->报错：a is not defined

使用了未声明的变量会报错。

**5.常量**

一旦声明就无法重新赋值（常用大写）

const PI=3.14;

console.log(PI);

练习：声明常量，分别保存春节、端午节、中秋节的日期

const SPRING='2019-1-1',DW='2019-5-5',MIDDLE='2019-8-20';

console.log(SPRING,DW,MIDDLE);

课后任务：

声明变量保存圆的半径，声明常量保存圆周率，使用常量圆的面积和周长。

var r=5;

const PI=3.14;

const len=2\*PI\*r;

const area=PI\*r\*r;

console.log(len,area);

//结果是31.400000000000002

//因为是由于计算甲需要转成二进制，计算完再转回十进制，中间出现了误差。

使用变量保存商品的单价为15，数量5；单价18，数量20，计算商品的总价。

**1.数据类型**

分为原始类型和引用类型

原始类型分为数值型、字符串型、布尔型、未定义型(undefined)、空(null)

**（1）数值型numeber**

分为整数和浮点数

整型在内存中占4个字节，浮点型占8个字节

//八进制，以0开头

//0 1 2...7 10 11...

var num2=010;

console.log(num2);

->8

//十六进制，以0x开头，a~f代表10~15 不区分大小写

//0 1 2...7 8 9 a b c d e f 10 11...

var num3=0x10;

console.log(num3);

->16

//浮点型分为普通小数和指数小数

1234.56 ->1.23456e3 ->1.12345\*10^3

0.0000034 ->3.4e-6 ->3.4\*10^-6

**（2）字符型string**

数据被引号所包含就是字符串类型，不区分单双引号。

Var str=12;

Console.log(str);

Console.loge(typeof(str));

->12

->string

//typeof(数据) 检测数据的类型

数值型->number 字符串->string

console.log('文'.charCodeAt());

//'…'.charCodeAt() 查看某个字符的Unicode码

**（3）布尔型boolean**

在程序中表示真或者假的结果

true/false

常用于一些是否的结果，例如：是否为会员、是否注册、是否在线

isMembber=true isOline=false

练习：检测true或者false的数据类型

var isMember=true;

console.log(isMember);

console.log(typeof(isMember));

->true

->boolean

**（4）未定义型undefined**

声明了变量未赋值，结果就是未定义型，只有一个值undefined。

**（5）空object**

常用于和引用类型使用，只有一个值null。

var student=null;

console.log(student);

->null

console.log(typeof(student));

->object

**2.数据类型转换**

**（1）隐式转换**

数值型+字符串：数值型被转换成了字符串型

console.log(1+'2');

console.log(typeof(1+'2'));

->12

->string

数值型+布尔型：布尔型不转换成了数值型true->1 false->0

console.log(3+true);

console.log(typeof(3+true));

->4

->numeber

布尔型+字符串型：布尔型被转换成了字符串型

console.log(true+'hello');

console.log(typeof(true+'hello'));

->truehello

->string

**JS中加号（+）的作用**

执行加法运算，执行字符串的拼接

练习：var a=3,b=true,c='tedu';

console.log(a+b+c);

console.log(b+c+a);

console.log(c+a+b);

->4tedu

->truetedu3

->tedu3true

**（2）强制转换**

将任意数据转换成数值型

var a=Number('1');

console.log(a);

console.log(typeof(a));

console.log(a+2)

->1

->number

->3

var b=Number('1c'); //含有非数字

console.log(b);

->NaN //Not a Number

Number(undefined);->NaN

Number(null);->0

Number(true);->1 false->0

练习：计算'3'-1、'2a'/5、true-null的结果，并查看数据库类型

var exp1='3'-1;

console.log(exp1);

console.log(typeof(exp1));

->2

->number

var exp2='2a'/5

console.log(exp2);

console.log(typeof(exp2));

->NaN

->number

var exp3=true-null;

console.log(exp3);

console.log(typeof(exp3));

->1

->number

|  |
| --- |
| 结论：在执行减法、乘法、除法运算的时候，如果数据不是数值型，则自动调用Numeber转为数值型。 |

将任意的数据转为整型

parseInt(数据)

parseInt(‘3.9a’);

->3

转换规则：如果要转换的数据是字符串，从开头开始查找数字，遇到小数点或者非数字，则返回前边的数字；否则开头不是数字，返回NaN。如果是布尔型、未定义型、空，返回null。

将任意数据转为浮点型

parseFloat(‘3.9a’);

->3.9

转换规则：和parseInt基本上一直，只是可以转成浮点型。

将数值型和布尔型转换为字符串型

toString()

var a=2;

a=a.toString(); //重新为变量换值

console.log(typeof(a));

->string //'２'

十进制转换为其他进制：

　var num=255;

　num=num.toString(16); //转换成16进制,可以设置进制（2/8）

console.log(num);

->ff

**3.运算符**

由运算符所连接的操作数据，所组成的结构称为表达式。

1. **算术运算符**

+ - \* / % ++ --

% 取余

console.log(3%2);

->1

++ 自增，在原来的基础之上加1

-- 自减，在原来的基础之上减1

var a=1;

a++;

console.log(a);

->2

var a=1;

console.log(a++);

->1 //先在控制台打印a的值，再执行自增a+1=2，此时a的值是2

console.log(a); //验证a的值

->2

console.log(++a);

->2 //先执行自增，再在控制台打印a的值

练习：计算以下代码的执行结果

var b=3;

console.log(b++ + ++b);

// 3 + 5

//先拿出b的值3，然后再执行自增

//先让b的值自增4，自增后变成了5，然后再拿出来运算

->8

console.log(++b + b++);

// 4 + 4

//先让b的值3自增，自增b的值后变成了4，然后再拿出来运算

//先拿出b的值4，再执行自增

->8

**（2）比较运算符**

> < >= <= ==（等于） != ===（全等于） !==（不全等于）

console.log(3=='3'); //比较值，类型不管

->true

console.log(3==='3'); //比较值和类型，都相同返回true，否则false

->false

console.log(3>'10'); //数字和字符串比较，字符串转换成了数值

->false

console.log('3'>'10'); //两个字符串比较，比较的是第一个字符Unicode码

->true

'张无忌'>'张翠山' ->false

如果是个字符相同，会继续往后比较

数值和字符串比较，字符串自动转为数值型,自动调用Number转为数值型。

示例：console.log(3>'10a'); //'10a'->NaN, NaN在和任何值比较> < ==都是false；包括自身比较，返回false //NaN==NaN->false

console.log(3<'10a');

console.log(3=='10a');

->false

->false

->false

console.log(3!='10a');->true

**（3）逻辑运算符**

&& 并且，关联的两个条件都是true，结果才是true，否则false

|| 或者，关联的两个条件只要有一个为true，结果为true，否则为false

! 取反 !true->false !false->true

示例：查看一个人的工资是否在5000~8000之间

var salary=7200;

console.log(salary>=5000 && salary<=8000);

->true

示例：老人65岁以上或者儿童12岁以下参观半价

var age=70;

console.log(age>=65 || age<=12);

->true

示例：取3<5的反向

console.log(!(3<5));

->false

练习：声明两个变量来保存用户名和密码，如果用户名是'root'，密码是'123456'，打印true，否则false

var uname='rooooot';

var upwd='123456';

console.log(uname==='root' && upwd==='123456');

->false

练习：声明一个变量来保存年龄，如果年龄90岁以上，或者3岁以下，打印true，否则false

var age=93;

console.log(age>=90 || age<=3);

->true

**逻辑短路：**

&& 当第一个条件是false，就不需要继续执行第二个条件

|| 当第一个条件是true，就不需要继续执行第二个条件

练习：以下两个语句是否会报错？

var num=3;

num>5 && console.log(a); //不会报错

num<1 || console.log(a); //报错，a未被定义a is not defined

练习：声明一个变量保存年龄，如果满十八岁，打印成年人

var age=19;

age>=18 && console.log('成年人');

->成年人

**（4）位运算符（了解）**

在执行运算的时候，把数字转成二进制再进行运算

1 10 11 100 101 110 111 1000 1001 1010…

2 2^2=4 2^3=8

将11010、101101用十进制表示？

11010=10000+1000+10

16 +8 +2

->26

101101=100000+1000+100+1

32 +8 +4 +1

->45

将25用二进制表示？

25=16+8+1=10000+1000+1=11001

**按位与（&）** 上下两位都是1，结果是1，否则是0

示例：3&5

console.log(3&5);

/\*

011

101

得到001

\*/

->1

**按位或（|）**上下两位含有1，结果是1，否则是0

示例：22|37

22=16+4+2=10000+100+10=10110

37=32+4+1=100000+100+1=100101

console.log(22|37);

/\*

010110

100101

得到110111

32+16+4+2+1

\*/

->55

**按位右移（>>）** 删除二进制的最后的n位数

示例：console.log(5>>1)

/\*

101

删除后面1位，10

\*/

->2

**按位左移（<<）** 二进制的最后添加n个0

示例：console.log(10<<2)

/\*1010

后面添加2个0，101000=100000+1000=32+8=40

\*/

->40

课后任务：

声明一个变量保存年份，判断这个年份是否为闰年，如果是打印'闰年'。

闰年的判断条件：

能够被4整除且不能被100整除；

能够被400整除。

var year=2000;

(year%100!=0 && year%4==0 || year%400==0) && console.log('闰年');

->闰年

**（5）赋值运算符**

= += -= \*= /= %=

练习：声明变量保存商品价格为20，计算打八折的结果

var price=20;

price\*=0.8; //在原来的基础之上乘以0.8，再赋值到price，price=price\*0.8

console.log(price);

->16

练习：假如本金10000，存款利率3.5%，计算一年利息是多少？

var money=10000;

money\*=0.035;

console.log(money);

-> 350.00000000000006

**（6）三目运算符**

需要有3个数据或者表达式

一个条件有两种结果

格式：条件表达式 **？** 表达式1 **:** 表达式2;

如果条件表达式为true，执行表达式1

如果条件表达式为false，执行表达式2

练习：判断一个人的饿年龄，满18岁打印成年人，未满18岁打印未成年人

var age=17;

age>=18 ? console.log('成年人') : console.log('未成年人');

->未成年人

练习：声明2个变量，分别保存数字，比较2个数字，打印最大值

var a=10;

var b=15;

a>b ? console.log(a) : console.log(b);

->15

练习：声明连个变量，分别保存用户名和密码，如果用户名是root并且密码是123456，打印登录成功，否则登录失败

var uname='ro',upwd='123456';

(uname==='root' && upwd==='123456') ? console.log('登录成功') : console.log('登录失败');

->登录失败

1. **浏览器端函数**

alert() 弹出警示框（消息框）

prompt() 弹出提示框（输入框），需要使用变量来接受输入的值，数据类型是字符串

<script src='04\_browser.js'></script> JS文件嵌入浏览器

var password=prompt('input password');

1. **流程控制**

程序 = 数据 + 算法

程序执行方式分为3种：顺序执行、选择执行、循环执行。

1. **if语句**

|  |
| --- |
| 语句1;  if(条件表达式){  语句2;  };  语句3; |

注意：如果if后的大括号中只有一行语句，大括号可以省略。

|  |
| --- |
| 在if语句的条件表达式中，有一些值默认就是false：  undefined、0、false、NaN、''(空，中间没有空格) |

示例：满30减25

var total=38;

//判断是否满30，如果满30减25

if (total>=30)

{

total-=25;

}

console.log(total);

->8

练习：声明变量保存年龄，如果查过50岁，在原来的基础上减5岁，打印年龄

var age=45;

if (age>50)

{

age-=5;

}

console.log(age);

->45

逻辑短路编写：

var age=65;

(age>50) && (age-=5); //逻辑短路和if相同效果

console.log(age);

->60

练习：声明变量保存用户名和明码，如果用户名是root，密码是123456，打印登录成功

var uname='root',upwd='123456';

if (uname=='root' && upwd=='123456')

{

console.log('登录成功')

}

->登录成功

练习：声明一个变量来不保存个人签名，如果签名为空，打印'这家伙很懒，什么也没有留下'，有签名就正常打印。

var signa='';

if (signa=='')

{

signa=('这个家伙很懒，什么也没有留下');

}

console.log(signa);

->这个家伙很懒，什么也没有留下

var signa='';

if (!signa) //!false->true(undefined、0、false、NaN、''相同效果)

{

signa=('这个家伙很懒，什么也没有留下');

}

console.log(signa);

->这个家伙很懒，什么也没有留下

**（2）if-else语句**

|  |
| --- |
| 语句1;  if(条件表达式){  语句2;  }else{  语句3;  } |

//判断一个人是否为成年人

var age=20;

if(age>=18){

console.log('成年人');

}else{

console.log('未成年人');

}

->成年人

//三目运算（逻辑相对简单）和if-else作用一样

age>=18 ? console.log('成年人') : console.log('未成年人');

->成年人

练习：声明变量保存用户名和明码，如果用户名是root，密码是123456，正确'登录成功'，错误'登录失败'

var uname='roooot',upwd='123456';

if(uname=='root' && upwd=='123456'){

console.log('登陆成功');

}else{

console.log('登录失败');

}

->登录失败

练习：使用弹出提示框分别输入商品的单价和数量，如果总价满500打八折，当前余额为1000，如果余额超过总价打印‘支付成功’，否则打印‘余额不足’

//弹出提示框分别输入单价和数量

var price=prompt('input price');

var num=prompt('input num');

//计算总价

var total=price\*num; //隐式转换成数值型

//如果总价满500打八折

if(total>=500){

total\*=0.8;

}

console.log('money'+total);

//余额

var money=1000;

//如果余额足够支付

if(money>=total){

console.log('pay success');

}else{

console.log('par error');

}

**if-else嵌套**

|  |
| --- |
| 语句0;  if（条件表达式1）{  语句1;  }else if（条件表达式2）{  语句2;  }else … if（条件表达式n）{  语句n;  }else{  语句n+1; //以上所有条件表达式都为false  } |

//判断一个人的政治面貌

var type='少先队员';

if(type=='群众'){

console.log('这个人是群众');

}else if(type=='党员'){

console.log('这个是党员');

}else if(type=='团员'){

console.log('这个人是团员');

}else{

console.log('一个非法的政治面貌');

}

->一个非法的政治面貌

练习：根据订单的状态码来打印订单状态，

1-等待付款 2-等待发货 3-运输中 4-已签收 5-已取消 其他-无法追踪

var now=7;

if(now==1){

console.log('等待付款');

}else if(now==2){

console.log('等待发货');

}else if(now==3){

console.log('运输中');

}else if(now==4){

console.log('已签收');

}else if(now==5){

console.log('已取消');

}else{

console.log('无法追踪');

}

->无法追踪

练习：根据分数，打印状态

90以上-优秀 80~90-良好 70~80-中等 60-70-及格 60以下-不及格

var score=20;

if(score>=90){

console.log('优秀');

}else if(score>=80){ //包含score<90，不用写

console.log('良好');

}else if(score>=70){

console.log('中等');

}else if(score>=60){

console.log('及格');

}else{

console.log('不及格')

}

->不及格

**（3）switch-case**

是一种特殊的多项分支语句，可以根据一个表达式的值，来执行不同的语句。

|  |
| --- |
| 语句0;  switch(表达式){  case 1: //表达式的值等于case后面的值，执行语句1  语句1;  break; //阻止程序继续往后执行其他的语句  …  case n:  语句n;  break;  default: //如果以上所有条件都是false  语句n+1;  } |

注意：在case中，表达式和值比较时候使用的是全等于（===），要求值和类型同时满足，结果才是true。

//根据订单状态码打印订单状态

var status='2';

switch(status){ //表达式为字符串型

case 1: //case后面值为数值型

console.log('等待服务');

break;

case 2:

console.log('等待发货');

break;

case 3:

console.log('运输中');

break;

case 4:

console.log('已签收');

break;

default:

console.log('无法追踪');

}

->无法追踪

练习：使用变量保存星期的值（0~6）,根据值打印对应的中文星期

var week=1;

switch(week){

case 0:

console.log('星期日');

break;

case 1:

console.log('星期一');

break;

case 2:

console.log('星期二');

break;

case 3:

console.log('星期三');

break;

case 4:

console.log('星期四');

break;

case 5:

console.log('星期五');

break;

case 6:

console.log('星期六');

break;

default:

console.log('输入错误');

}

示例：

var status=1;

switch(status){

case 1:

case 2:

case 3:

console.log('人数100'); //相同的结果，case可以写在一起

break;

case 4:

console.log('人数101');

break;

default:

console.log('无法追踪');

}

**对比if-else嵌套和switch-case语句：**

相同点：两者都可以用于多项分支语句

不同点：if-else可以判断相等或者不相等的情况，适用于范围更广；switch-case只能用于全等于的比较，在结构更清晰，执行效率相对较高。

**（4）循环执行**

循环：就是一遍又一遍执行**相同或者相似**的代码。

循环的两个要素：

循环条件：控制循环的次数

循环体：执行的相同或者相似代码

**while循环**

|  |
| --- |
| while(循环条件){ //循环条件是一个布尔型的值  循环体  } |

示例：//打印10次hello，循环10次 0~9

var i=0; //初始值，只初始化数据，并未参加循环

while(i<10){ //循环条件

console.log('hello');

i++; //增量 i需要变化起来

}

练习：//打印100~200之间的所有整数

var i=100;

while(i<=200){

console.log(i);

i++;

}

课后作业：

1. 复习今天内容，整理思维导图；
2. 练习：

使用switch-case判断一个的成绩标准

分数/10

80~90 8.0-8.9 取整8

使用while打印1~100之间所有的奇数

使用while计算1~100之前所有整数的和

（3）预习while、do-while、for循环、循环嵌套

var score=89;

var score=parseInt(score/10);

switch(score){

case 9:

case 10:

console.log('优秀');

break;

case 8:

console.log('良好');

break;

case 7:

console.log('中等');

break;

case 6:

console.log('及格');

break;

default:

console.log('不及格');

}

var i=1;

while(i<=100){ //循环条件

if(i%2==1){ //判断是否为基数，和2取余结果是1

console.log(i);

}

i++; //增量

}

var i=1;

var sum=0;

while(i<=100){

sum+=i; //把每一个值（i）加到sum中，sum=sum+i

i++;

}

console.log(sum);

->5050

练习：打印50~150之间所有的偶数

var i=50;

while(i<=150){

if(i%2==0){ //判断是否为偶数，和2取余结果为0

console.log(i);

}

i++;

}

练习：打印30 27 24 21 18 15

var i=30;

while(i>=15){

console.log(i);

i-=3;

}

练习：计算1~200之前所有能被3整除的数字的和

var i=1;

var sum=0;

while(i<=200){

if(i%3==0){

sum+=i //sum=sum+i

}

i++;

}

console.log(sum);

->6633

**break：强制结束任何形式的循环**

var i=1;

while(true){

if(i==11){

break; //强制结束，当i=11的时候，结束

}

console.log(i);

i++;

}

->1…10

练习：使用无限循环计算1~10之间所有整数相乘的结果。

var i=1;

var jc=1;

while(true){

if(i==11){

break;

}

jc\*=i;

i++;

}

console.log(jc);

练习：使用变量提前存出一个数字，循环弹出提示框输入值，开始猜大了，弹出警示框：big，如果猜小了，弹出警示框：small，否则弹出警示框提示：right，并且结束循环。

var i=10;

while(true){

var a=prompt('input num');

if(a>i){ //bug——字符串和数字比较，字符串转换成数字，调用Number，字符串含有非数字，返回NaN->false

//先判断是否为NaN，如果是提示输入数字（后面再说）???

a.isNaN()判断a是否为NaN

alert('big');

}else if(a<i){

alert('small');

}else{

alert('right');

break;

}

}

**do-while循环**

|  |
| --- |
| do{  循环体  }while(循环条件); |

特点：不管循环条件你是否为true，都会先执行一遍循环体，再去判断条件。

示例：使用do-while循环1~10

var i=1;

do{

console.log(i);

i++;

}while(i<=10);

练习：打印1~100所有奇数

var i=1;

do{

if(i%2==1){

console.log(i);

}

i++;

}while(i<=100);

练习：打印1~100所有整数的和

var i=1;

var sum=0;

do{

sum+=i; //把循环产生的整数加到sum中

i++;

}while(i<=100);

console.log(sum);

练习：打印200~300所有能被7整除的数字和

var i=200;

var sum=0;

do{

if(i%7==0){

sum+=i;

}

i++;

}while(i<=300);

console.log(sum);

->3479

练习：打印100 95 90 85 80 75 70

var i=100;

do{

console.log(i);

i-=5;

}while(i>=70);

练习：声明一个变量保存密码：123456，无限循环弹出提示框输入值，如果输入正确，弹出警示框：login suc，结束循环。

var pwd='123456';

do{

var a=prompt('input password') //输入值

if(a===pwd){ //输入值和密码比较

alert('login suc');

break;

}

}while(true);

另一种写法：

var pwd='123456';

do{

var a=prompt('input password');

}while(a!=pwd); //在循环天骄中判断密码是否正确，如果输入的值，和密码不同（true）；和密码相同（false），结束循环。

**for循环**

|  |
| --- |
| for(初始值**;**循环条件**;**增量){  循环体  } |

//while循环1~10

var i=1; //初始值

while(i<=10){ //循环条件

console.log(i);

i++; //增量

}

示例：for循环1~10

for(var i=1;i<=10;i++){

//步骤

console.log(i);

//

}

while循环和for循环，执行的顺序相同：初始值->循环条件->循环体->增量

练习：打印20~60之间的所有奇数

for(var i=20;i<=60;i++){

if(i%2==1){

console.log(i);

}

}

练习：打印80 77 74 71 …50

for(var i=80;i>=50;i-=3){

console.log(i);

}

练习：计算1~20之间所有偶数的乘积

var jc=1;

for(var i=1;i<=20;i++){

if(i%2==0){

jc\*=i;

}

}

console.log(jc);

->3715891200

练习：计算200~400之间所有的能被3整除的数字和

var sum=0;

for(var i=200;i<=400;i++){

if(i%3==0){

sum+=i;

}

}

console.log(sum);

->20100

练习：打印本世纪所有的闰年

for(var i=2000;i<2100;i++){

if(i%4==0 && i%100!=0 || i%400==0){ //闰年的判断条件

console.log(i);

}

}

练习：本金10000，每年的利率%3.5，计算五年后本金和利息一共多少？

var money=10000;

for(var i=1;i<=5;i++){

money=money+money\*0.035; //一年后的本金=前一年的本金+利息

//money=money\*(1+0.035)或money\*=1.035

}

console.log(money);

->11876.86

练习：打印九九乘法表第五行

for(var i=1, str='';i<=5;i++){ //可以同时声明多个变量

str+=i + '\*5=' + (i\*5) + ' ';

//拼接成12345 拼接\*5= 拼接乘积 拼接空格

}

console.log(str);

//for不同写法，求1~100整数和

for(var i=1,sum=0;i<=100;i++){ //声明多个变量

sum+=i;

}

console.log(sum);

var i=1,sum=0; //初始值

for(;i<=100;i++){ //初始值放在外面，必须保留分号(:)

sum+=i;

}

console.log(sum);

var i=1,sum=0; //初始值

for(;i<=100;){

sum+=i;

i++; //增量

}

console.log(sum);

var i=1,sum=0;

for(;;){ //默认循环条件就是true

if(i==101){

break; //循环条件

}

console.log(i);

i++; //增量

}

for(var i=1,j=5;i<=10,j>=1;i++,j--){ //如果循环条件有多个，最后一个起作用

console.log(i,j);

}

练习:1/10+2/9+3/8...10/1

for(var i=1,j=10,sum=0;i<=10,j>=1;i++,j--){

sum+=(i/j);

}

console.log(sum);

->22.218650793650795

}

**break和continue**

break：结束（跳出）循环，后续不会再执行其他的循环

continue：跳过本次循环，继续下一次循环（执行增量）

练习：打印1234 678910

for(var i=1;i<=10;i++){

if(i==5){

continue; //当i=5的时候，跳过后面的循环体，去执行增量

}

console.log(i);

练习：计算1~100之间所有偶数的和，要求遇到奇数就跳过。

for(var i=1,sum=0;i<=100;i++){

if(i%2==1){

continue; //奇数不会加到sum中

}

sum+=i;

}

console.log(sum);

->2550

练习：打印1~100之间，1 2 5 7 10 11 13 14 17 19 …98（去除了能被3或者4整除的整数）

for(var i=1;i<=100;i++){

if(i%3==0 || i%4==0){

continue;

}

console.log(i);

}

练习：打印出本世纪前10个闰年

for(var i=2000,count=0;i<=2100;i++){

if(count==10){ //当count=10的时候，结束循环

break;

}

if(i%4==0 && i%100!=0 || i%400==0){

console.log(i);

count++; //计算闰年个数，是闰年就让count加1

}

}

**循环嵌套**

**三个循环之间可以相互嵌套**

练习：打印

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

for(var i=1,str='';i<=j;i++){

str+='\*';

}

for(var j=1;j<=4;j++){

console.log(str);

}

练习：打印

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

for(var j=1;j<=5;j++){ //外层循环，j代表行数，控制循环的行数

for(var i=1,str='';i<=j;i++){ //内层循环，i代表列数，控制循环的列数,循环条件：i小于等于当前的行号j

str+='\*';

}

console.log(str);

}

课后任务：

（1）复习今天内容整理思维导图

（2）使用循环嵌套完成九九乘法表

for(var j=1;j<=9;j++){

for(var i=1,m='';i<=j;i++){

m+=i+'\*'+j+'='+(i\*j)+' ';

}

console.log(m);

}

计算1~100之间的和，当和大于4000的时候，并且打印当前的以及i的值。

for(var i=1,sum=0;i<=100;i++){

sum+=i; //每次sum加完后，就立即判断

if(sum>4000){

break;

}

}

console.log(sum,i);

->4005 89

（3）预习JS中自定义函数

**函数**

Numeber()/parseInt()/parseFloat()/alert()/prompt()系统函数

JS分为系统函数和自定义函数

**1．自定义函数**

function：功能体，可以接受若干个数据，返回处理的结果。——用于封装重复执行的代码。（饺子机）

**（1）创建普通函数/方法**

|  |
| --- |
| function 函数名称(){  函数体——重复执行的代码  } |

调用时候：函数名称()

function printHello(){ //创建函数，使用function

console.log('hello'); //函数体

console.log('word');

console.log('css');

}

printHello(); //调用函数——执行函数体中代码

printHello(); //调用函数——执行函数体中代码

练习：使用函数来封装计算1~100之间所有整数的和，调用三次

function getSum(){

for(var i=1,sum=0;i<=100;i++){

sum+=i;

}

console.log(sum);

}

getSum();

getSum();

getSum();

**（2）创建带有参数的函数**

|  |
| --- |
| function 函数名称(参数列表){ //形参  函数体  } |

调用：函数名称(参数列表) //实参

形参：创建函数时的参数，也称为形式上参数，用于接收数据。——相当于声明未赋值的变量

实参：调用函数时的参数，这次才是实际传递的参数。——实参个数如果少于形参，未赋值的参数就是undefined

示例：计算任意两个数字相加

function add(a,b){ //a,b就是形参（相当于两个只是声明的变量，未赋值），用于接收数据

console.log(a+b);

}

add(3,7); //3,7就是实参（相当于给变量a,b赋值），实际的数据

add(5,6);

练习：创建函数getSum，传递一个参数，计算1~的任意数字所有数字的和

function getSum(n){

for(var i=1,sum=0;i<=n;i++){

sum+=i;

}

console.log(sum);

}

getSum(100);

getSum(1000);

getSum(10000);

->5050

->500500

->50005000

练习：创建函数getRun,传递2个参数，计算任意两个年份之间的闰年个数

function getRun(m,n){

for(var i=m,count=0;i<n;i++){

if(i%4==0 && i%100!=0 || i%400==0){

count++;

}

}

console.log(count);

}

getRun(2000,2100);

->25

**（3）带有返回值的函数**

|  |
| --- |
| function 函数名称(参数列表){  函数体  return 返回值;  } |

调用：函数名称();

return：表示函数执行后的返回结果，后面返回值写什么就返回什么。

注意：

使用return目的，为了保存函数的调用结果。

return执行后，再后的所有命令都无法执行，return有终止函数的执行。

如果函数中写了return后面没有加任何返回值或者函数中没有写return，函数结果返回undefined。

示例：计算任意两个数字的和，并保存

function add(a,b){

//console.log(a+b); //只能打印结果，未保存

return a+b; //把a+b的结果返回出来，并存储到str中

}

var str=add(3,4); //str用于保存return返回函数的结果

console.log(str); //查看str中是否保存这连个数字的和

->7

练习：创建函数，传递两个参数，返回任意两个数字的最大值。

function getMax(a,b){

//return a>b ? a : b; //三目运算

if(a>b){

return a;

}else{

return b;

}

}

var str=getMax(1,2)

console.log(str);

//console.log(gitMax(1,2));写成一步

->2

练习：创建函数，传递3个参数，返回任意3个数字的最大值

function getMax(a,b,c){

//return a>b>c ? a : (b>c ? b : c); //三目运算

if(a>b>c){

return a;

}else if(b>c){

return b;

}else{

return c;

}

}

var str=getMax(3,4,5);

console.log(str);

练习：创建函数，传递一个订单参数，根据订单状态码返回对应的中文

1. 待付款 2-待发货 3-运输中 4-已签收 5-已取消 其他-无

function getStatus(a){

switch(a){

case 1:

return '待付款';

//break;可以省略

case 2:

return '待发货';

case 3:

return '运输中';

case 4:

return '已签收';

default:

return '其他';

}

}

var str=getStatus(7);

console.log(str);

**对比return、break和continue**

return：用于函数中，终止函数的执行，并返回结果。

break：用于终止循环、终止switch语句。

continue：跳过某一次循环，还会继续下一次循环。

练习：创建函数isRun，传递一个参数（年份），返回是否为闰年，是->true,不是->false，判断一年的天数

function isRun(y){

if(y%4==0 && y%100!=0 || y%400==0){

return true;

}else{ //else可以省略，return执行后，下面的语句不执行

return false;

}

}

var res=isRun(2004);

//console.log(res);

//根据res判断天数

if(res){

console.log(366);

}else{

console.log(365);

}

练习：创建函数，传递2个参数，计算任意年份之间的个数

function isRun(year){

if(year%4==0 && year%100!=0 || year%400==0){

return true;

}

return false;

}

function getRun(n1,n2){

for(var i=n1,count=0;i<=n2;i++){

if(isRun(i)){ //判断i是否为闰年，如果是count加1

count++;

}

}

return count;

}

var res=getRun(2000,2100);

console.log(res);

**2.变量的作用域**

**（1）作用域**

变量或者函数的可访问范围

全局作用域：在全局作用域下声明的变量可以在任意位置访问到。

函数作用域：在函数作用域下声明的变量只能在函数内部访问到。

示例1：

var b=2; //b,全局变量

function fn(c){

var a=1; //a,c，局部变量

console.log(b); //->2

console.log(c); //->3

}

console.log(a); //无法访问函数作用域内的变量

fn(3);

console.log(c); //无法访问函数作用域内的变量

特点：在函数中，不加var声明的变量就是全局变量。

示例2：

var a=1;

function fn(){

a=3; //加var变成局部变量->1

}

fn();

console.log(a);

->3

|  |  |  |
| --- | --- | --- |
|  | 全局作用域 | 函数作用域 |
| 加var | 全局变量，存在变量声明提升 | 局部变量，存在变量声明提升 |
| 不加var | 全局变量，不存在变量声明提升 | 全局变量，不存在变量声明提升 |

**（2）变量提升**

程序在执行前，查找当前作用域下使用var声明的变量，把变量声明提升到当前作用域最前面；赋值还是在原来的位置。

示例1：

console.log(a);

var a=1; //(var a;)提升到最前面，赋值(a=1)还是在此处。

->undefined

示例2：

var c=1;

function fn(){

console.log(c); //先调用当前作用域下的var（由内而外）

var c=3; //var c;被提升到局部作用域最前面，c=3赋值还是在原处

}

fn();

->undefined

示例3：

var d=1; //全局变量

function fn(d){ //d是参数，局部变量

//d=5； //调用函数时，先执行此步

d=3; //给参数重新赋值，对全局没有影响

console.log(d);

}

fn(5);

console.log(d);

->3

->1

**3.函数的作用域**

**（1）作用域**

函数和变量一样，也存在全局作用域和函数作用域；

全局下的函数可以在任意位置调用，函数中创建的函数只能在函数内部调用。

示例1：

function fn1(){

console.log(1);

function fn2(){ //函数作用域下创建的函数，只能在当前作用域下调用

console.log(2);

}

fn2(); //只能在fn1的函数作用域内调用

}

//fn2();->报错 //不能在全局作用域下调用fn2

fn1();

->2

示例2：

var a=1;

function fn1(){

function fn2(){

function fn3(){

console.log(a);

}

fn3();

}

fn2();

}

fn1();

->1

（2）函数提升：JS在程序执行前，检查使用function关键字创建的函数，提升到所在作用域的最前面。

示例：

add(2,3);

function add(a,b){

console.log(a+b);

}

->5

**4.递归（了解，运行速度慢）**

在函数内部调用自身

递归在使用的时候，要有结束的条件，结合return使用。

示例：浏览器弹出3次警示框

var count=0;

function say(){

count++;

alert('long ago');

if(count==3){ //当count为3的时候，结束函数

return;

}

say(); //函数内调用自身

}

say();

练习：使用递归计算1~任意数字之间的和

function getSum(n){

if(n==1){ ////跳出条件，当n为1的时候，返回1，并结束函数

return 1;

}

//算法

return n+getSum(n-1);

}

console.log(getSum(100));

/\*递归原理

前5个数的和等于5加前4个数的和

5+getSum(4)

5+4+getSum(3)

5+4+3+getSum(2)

5+4+3+2+getSum(1)

前1个数的和=1，结束

5+4+3+2+1

\*/

练习：斐波那契数列（从第三次开始的数字等于前两数字之和），第n个数字是多少？

1 1 2 3 5 8 13 21 …

function dg(n){

if(n==1 || n==2){

return 1; //当n为第一个数字或者第二个数字的时候，返回1。

}

return dg(n-1)+dg(n-2);

}

console.log(dg(3));

课后任务：

创建函数，传递1个参数（n），计算！n+！（n-1）+…+!2+!1

!3=3\*2\*1（阶乘）

第种写法：

function getjc(n){

for(var i=1,jc=1;i<=n;i++){

jc\*=i; //先求i的阶乘

}

return jc;

}

console.log(getjc(3));

function getsum(n){

for(var i=1,sum=0;i<=n;i++){

sum+=getjc(i); //再求i阶乘的和

}

return sum;

}

console.log(getsum(3));

第种写法

function jc(n){

for(var i=1,product=1,sum=0;i<=n;i++){

product\*=i;

sum+=product;

}

return sum;

}

var x=jc(4);

console.log(x);

**5.匿名函数：**没有名字的函数 function(…){…}

|  |
| --- |
| 创建函数——函数声明  function add(){…} |

1. **创建函数——函数表达式**

|  |
| --- |
| var add=function(形参列表){  函数体  return 值;  }  把匿名函数赋值给了变量，变量名称就是函数的名称  调用：add(实参列表) |

**对比函数声明和函数表达式的区别**

函数声明存在函数提升，可以写在当前作用域的任意合法位置。

函数表达式不存在函数提升（找不到function开头的函数），必须先创建再调用。

示例：

var add2=function(a,b){ //把匿名函数放到变量add2中，此时的add2自动成为了函数名称

console.log(a+b);

}

add2(4,9);

练习：使用函数表达式，计算1~任意数字之间所有偶数的和

var add=function(n){

for(i=1,sum=0;i<=n;i++){

if(i%2==0){

sum+=i;

}

}

return sum;

}

console.log(add(100));

**（2）匿名函数的自调用：**把函数自己当做函数名字

|  |
| --- |
| (function(形参列表){  函数体; //是一个函数作用域  })(实参列表); |

使用目的：为了创建一个函数作用域，防止污染。

示例：

(function(a,b){ //创建的括号，写形参

console.log(a+b);

})(3,4); //调用的括号，写实参

->7

**（3）回调函数：**把匿名函数以实参的形式传递。

|  |
| --- |
| function fn(a){ //传递实参后的a=function(){函数体}  //要执行传递的匿名函数的函数体中代码，需要调用匿名函数，而现在匿名函数已经赋值给a。  console.log(a()); //所以使用a()调用匿名函数;  }  fn(function(){  return 1;  }); //传递实参(匿名函数) |

练习：创建函数add，接收两个参数（都是匿名函数，函数体重返回一个数字），打印连个数字的和。

function getSum(a,b){

console.log(a()+b());

}

getSum(function(){

return 1;

},function(){

return 2;

});

**6.全局系统函数**

parseInt()——将数据转为整形

var a=parseInt('3.9a');

console.log(a); //console.log(parseInt('3.9a'));

parseFloat()——将数据转换成浮点型

encodeURI()——对URI中的中文字符进行编码

decodeURI()——对已编码的URI进行解码

示例：

var uri='http://www.codeboy.com/search?kw=电脑';

console.log(uri);

//对URI进行编码

uri=encodeURI(uri);

console.log(uri);

//对已编码的URI进行解码

uri=decodeURI(uri);

console.log(uri);

isNaN()—检测一个数据是否为NaN，是返回true，不是返回false

isFinite()——检测数据是否为有限值,是有限值返回true，不是有限值返回false

1/0 ->Infinity无穷 -1/0 ->-Infinity负无穷

eval()——执行字符串中的表达式

var str='1+2';

console.log(str);

console.log(eval(str));

->1+2

->3

练习：使用弹出提示框，输入一组JS表达式，使用eval执行字符串表达式

var str=prompt('input');

console.log(eval(str));

**对象（object）**

对象是一组属性（property）和方法（method）的集合。

对象要具体到某一个。

有哪些对象？

一部手机：属性有颜色、尺寸、型号、、、（侧重外观）

方法有打电话、发短信、玩游戏、、、（侧重功能）

一辆汽车：属性有颜色、品牌、大小、类型、、、

方法有载人、听音乐、、、

一本书：属性有名称、作者、页码、、、

方法有学习、垫桌角、、、

万物皆对象。

**1.JS中的对象**

原生(内置)对象：JS提供的

宿主对象：根据JS不同的执行环境来划分

自定义对象：自己创建的

**（1）自定义对象——对象字面(直接)量**

使用大括号{}创建空对象

属性名和属性值之间用冒号隔开；

多组属性之间用逗号隔开；

属性名中的引号可加可不加，如果含有特殊字符必须加引号。

示例：创建一个手机对象

var phone={

color:'黑色', //添加属性——属性名:属性值，多组属性用逗号隔开

size:'5.9',

brand:'huawei',

'made-in':'china' //特殊字符时，属性名必须加引号(避免)

};

console.log(phone);

练习：创建一个部门对象，属性有部门编号，部门名称

var dept={

did:10,

dname:'测试部'

}

console.log(dept);

**（2）访问对象中的属性**

对象.属性名

对象['属性名']

如果访问的属性值不存在，则属性值为undefined

练习：创建一个员工对象，属性有编号，姓名，性别，生日，工资，部门编号

var emp={

eid:1,

ename:'tom',

sex:'man',

birthday:'2010-01-01',

salary:9000,

deptId:10

}

console.log(emp);

//访问对象中的属性值

console.log(emp.ename);

console.log(emp['birthday']);

//访问一个不存在的属性->undefined

console.log(emp.abc);

//修改某一个属性的值

emp.salary=12000;

console.log(emp);

//添加一个新的属性

emp['addr']='beijing';

console.log(emp);

**（3）使用内置构造函数创建对象**

new Object()创建一个空对象，后面需要单独添加每一个属性，括号内不能直接添加。

示例：

var car=new Object();

//添加属性，使用访问对象中的属性

//属性有颜色、品牌、大小

car.color='黑色';

car['brand']='byd';

car['size']='越野';

console.log(car);

**（4）遍历属性**

|  |
| --- |
| for(var key in phone){  key //所有的属性名  phone //通过属性名来找到对应的属性值  phone[key] //通过属性名key来找到对应属性值  } |

示例：

var phone={

color:'白色',

size:5.9,

brand:'小米'

}

for(var key in phone){ //新建key来遍历所有的属性名

//console.log(key); //通过属性名key，获取属性

console.log(key,phone[key]); //使用属性名key获取属性值

}

练习：创建一个用户对象，属性有编码，用户名，密码，邮箱，手机；使用for-in便利对象。

var emp={

eid:1,

ename:'tom',

wpwd:'123456',

email:'3131@qq.com',

phone:1324444444

}

for(var key in emp){

console.log(key,emp[key]);

}

**（5）对象的方法**

|  |
| --- |
| var person={  name:'tom',  say:function(){  this.name //在方法中调用属性通过this，this指向当前的对象  }  } |

示例：

var person={

name:'tom',

age:20,

say:function(){ //方法对应的是函数，say就是方法名

console.log('我叫'+this.name+',今年23岁'); //this自动指向person对象

}

}

person.say(); //调用对象中的方法

**（6）检测属性是否存在**

'属性名' in 对象

存在->true 不存在->false

对象.hasOwnProperty('属性名');

存在->true 不存在->false

对象.属性名===undefined

存在->false 不存在->true

示例：

var phone={

color:'黑色',

size:5.1,

brand:'华为'

};

//检测属性是否存在

console.log('color' in phone);->true存在

console.log(phone.hasOwnProperty('size'));->true存在

//不存在的属性值是undefined

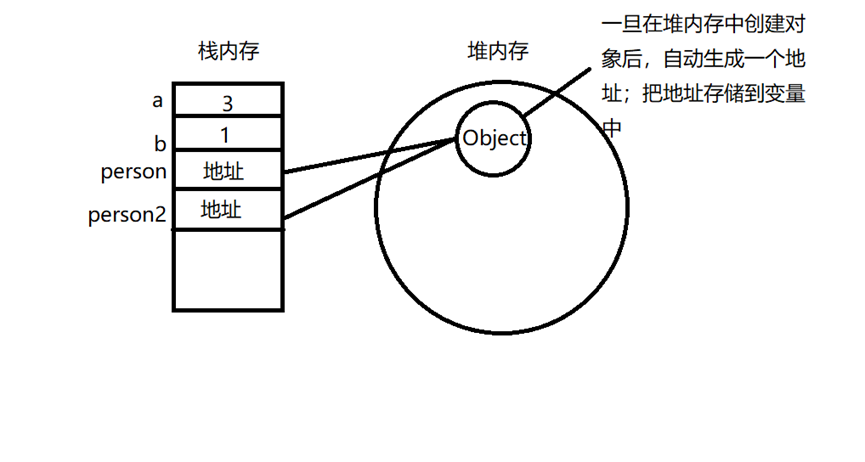
console.log(phone.brand1===undefined);->true不存在

**（7）原始类型和引用类型的存储**

原始类型存储在栈内存中。

引用类型存储在堆内存中，自动生成一个地址，保存在栈内存中。如果要销毁，只要设置为null即可。

示图：

示例：

原始类型：

var a=1;

var b=a; //把a的值赋（拷贝）给了b

a=3; //只改变a的值，b的值没有改变

console.log(b);->1

引用类型：

var person={

name:'tom',

age:18

}

var person2=person; //把person对象的地址赋值（拷贝）给了person2，两个变量指向同一个对象，所以改变任何一个对象都会改变

//改变person的属性值

person.name='汤姆';

console.log(person2); ->person2改变

//改变person2的属性值

person2.age=28;

console.log(person); ->person也改变

//释放（销毁）堆内存中的数据，只要堆内存中的数据没有被应用，则会自动销毁

person=null;

person2=null;

课后任务：

创建一个圆对象，属性有半径、圆周率；添加两个方法，计算圆的周长（getlength），计算圆的面积（getArea）。

var circle={

r:3,

PI:3.14,

getlength:function(){

return 2\*this.PI\*this.r;

},

getArea:function(){

return this.PI\*this.r\*this.r;

}

}

console.log(circle.getlength());

console.log(circle.getArea());

**1.数组**

数组是由多个元素（数据）组成集合。（每个元素就代表一个数据）

**（1）创建数组——数组字面量**

var a=[元素1,元素2,…]

访问数组中的元素，使用下标，下标是从0开始

访问数组中的元素：数组[下标]

示例：

var score=[

90, ->下标为0

80, ->下标为1

88, ->下标为2

];

console.log(score);

//访问数组中的元素——下标（从0开始计数）

console.log(score[1]);

//改变下标为2的值为100

score[2]=100;

//添加下标为3的值为96

score[3]=96;

console.log(score);

**（2）创建数组——内置构造函数**

①var arr=new Array(元素1,元素2,…);

var sourse=new Array('语文','数学','英语','德语');

console.log(sourse[0]);

②var arr=new Array(5) 数组初始化的长度为5，可加可不加，后期可以添加更多的元素，不能为负数。

var ball=new Array(5);

ball[0]='大前锋';

ball[1]='小前锋';

ball[2]='中锋';

ball[3]='控球后卫';

ball[4]='得分后卫';

ball[5]='替补';

console.log(ball);

**（3）数组的长度**

获取长度：数组名称.length ->值

在末尾添加一个新的元素：数组名称[数组名称.length]

数组最后一个元素的下标：数组.length-1

示例：

var sourse=new Array('语文','数学','英语','德语');

//获取数组的长度

console.log(sourse.length); ->4

//查看数组的数据类型

console.log(typeof(sourse)); ->object

//末尾添加元素

course[course.length]='物理';

console.log(sourse);

//获取最后一个元素

console.log(sourse[sourse.length-1]);

练习：创建数组，在末尾添加4个国家的名称，打印最后一个元素

var country=new Array('中国','巴基斯坦');

country[country.length]='美国';

country[country.length]='法国';

country[country.length]='英国';

country[country.length]='德国';

console.log(country);

console.log(country[country.length-1]);

**（4）数组的分类**

索引数组：以数字作为下标（数组保存一组相同形式的数据）

var names=[];

names[0]='tom';

names[1]='jerry';

关联数组：以字符串作为下标，只能单独添加元素（这种最好用对象保存，元素信息不同）——不推荐使用这种数组

var emp=[];

emp['ename']='kate';

emp['birthday']='2000-1-1';

console.log(emp);

练习：创建一个数组，包含所有员工的数据，姓名，性别，生日

//数组包含多个对象（每个员工是一个对象）

var emps=[

{ename:'tom',sex:'男',birthday:'1991-1-1'},

{ename:'kate',sex:'女',birthday:'1995-1-1'}

];

console.log(emps);

**（5）遍历数组**

遍历数组的元素，使用的遍历下标

for-in

|  |
| --- |
| for(var key in 数组){  key 代表下标  数组[key] 下标对应的元素  }  既可以遍历索引数组，也可以遍历关联数组 |

for循环

|  |
| --- |
| for(var i;i<数组.length;i++){  i 代表下标  数组[i] 下标对应的元素  }  只能遍历索引数组（常用） |

示例：

var person=['tom','jerry','kate','king'];

for(var key in person){

//console.log(key); //遍历数组元素的下标

console.log(person[key]);

}

//使用for循环遍历数组元素下标

for(var i=0;i<person.length;i++){ //i是元素下标

console.log(i,person[i]);

}

练习：创建关联数组，包含手机的颜色、尺寸、品牌，使用for-in遍历该数组

var phone=[]

phone['color']='白色';

phone['size']=5;

phone['brand']='华为';

for(var key in phone){ //下标是字符串，不能使用for循环遍历

console.log(phone[key]);

}

练习：创建一个数组，保存若干个成绩，使用变量数组元素，让每个成绩加10分。

var score=[90,100,98,83,76,82];

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

//console.log(score[i]); //遍历所有数组

score[i]+=10; //score[i]=score+10

}

console.log(score);

练习：创建数组，保存若干个汽车品牌，把品牌名称为‘宝马’替换为‘BMW’

var brand=['BYD','奇瑞','哈弗','宝马','宾利'];

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

if(brand[i]=='宝马'){

//替换为BWM

brand[i]='BMW';

}

}

console.log(brand);

练习：创建函数getTotal，传递一个参数（数组），包含若干个员工的工资，返回工资的平均值。

function getTotal(salary){

for(var i=0,sum=0;i<salary.length;i++){

sum+=salary[i];

}

return sum/salary.length;

}

console.log(getTotal([6000,9000,7000,8500]));

//var arr=[6000,9000,7000,8500]

//var fn=getTotal(arr);

//console.log(fn);

练习：创建函数getCount，传递2个参数（数组，包含所有员工姓名；字符串），返回字符串在数组中出现的次数。

function getCount(names,name){

for(var i=0,count=0;i<names.length;i++){

if(names[i]==name){

count++;

}

}

return count;

}

console.log(getCount(['tom','jerry','kate','tom','tom'],['tom']));

练习：创建函数getMax，传递一个成绩参数（数组，包含若干个数字），返回最大值

function getMax(score){

var max=score[0]; //初始化一个变量用于存储最大值，假设第一个元素就是最大值

for(var i=1;i<score.length;i++){ //遍历第2个元素到最后

if(max<score[i]){ //用max和每个元素进行比较

max=score[i]; //如果max小于任何元素，则把该元素放入max中

}

}

return max;

}

console.log(getMax([70,68,92,43,85]));

**（6）数组中的方法（api）**

api-应用程序编程接口，预定义好的函数/方法

arr.toString() 将数组中的元素转为字符串，按照逗号分隔。

arr.join('-') 将数组中的元素转为字符串，可以指定分隔符。

arr.concat(arr1,arr2,…) 拼接两个或者更多个数组。

arr.slice(start,end) 截取数组中的元素，start开始的下标，end结尾的下标，不包含end本身；如果是负数表示倒数。

arr.splice(start,count,value1,…) 删除数组中的元素，start是开始的下标，count是删除的长度，value1…在删除的位置插入的元素。（返回删除的元素）

arr.reverse() 翻转数组中的元素

arr.sort() 默认按照Unicode码，从小到大排序

|  |
| --- |
| 对数字排序  arr.sort(function(a,b){  return a-b; //从小到大  //return b-a; //从大到小  }); |

arr.push(value1,…) 在数组的末未添加元素，返回数组的长度

arr.pop() 在数组的末尾删除1个元素，返回删除的元素

arr.unshift(value1,…) 在数组开头添加元素，返回数组的长度

arr.shift() 在数组的开头删除1个元素，返回删除的元素

示例：

var arr=['tom','jerry','king'];

console.log(arr.toString());

console.log(arr.join('|'));

var arr1=['kate','lucy'];

var arr2['张三','李四'];

console.log(arr.concat(arr1,arr2)); //拼接数组

练习：创建3个数组，每个元素含有一个字母，拼接3个数组，再将数组转为字符串。

var arr1=['a1','a2','a3'];

var arr2=['b1','b2','b3'];

var arr3=['c1','c2','c3'];

var newArr=arr1.concat(arr2,arr3);

console.log(newArr.toString());

console.log(newArr.join('|'));

练习：创建数组，每个元素有1个字母组成a~h，分别截取bc和fg，拼接成一个新数组

var arr=['a','b','c','d','e','f','g','h'];

var arr1=arr.slice(1,3);

var arr2=arr.slice(-3,-1);

console.log(arr1.concat(arr2));

练习：创建数组，保存a~h，每个字母为一个元素，删除c，d，替换f为m，在下标为1的位置插入元素z。

var arr=['a','b','c','d','e','f','g','h'];

arr.splice(2,2);

arr.splice(-3,1,'m');

arr.splice(1,0,'z');

console.log(arr);

练习：

var arr=[80,73,9,61,100];

console.log(arr.sort()); //默认按照首位的Unicode码，由小到大排序

//借助回调函数——匿名函数以参数传递

console.log(arr.sort(function(a,b){

return a-b; //由小到大排序

}));

console.log(arr.sort(function(a,b){

return b-a; //由大到小排序

}));

练习：

var arr=['推荐','娱乐','财经'];

//在数组末未添加元素

//console.log(arr.push('热点')); //->4

//在数组的末尾删除1个元素

//console.log(arr.pop()); //->财经（删除的元素）

//在数组的开头添加元素

//arr.splice(0,0,'历史');

//console.log(arr.unshift('历史')); //->4

//在数组的开头删除1个元素

console.log(arr.shift()); //->推荐（删除的元素）

console.log(arr);

**冒泡排序**

由小到大排序

var arr=[23,9,78,6,45];

//外层循环：控制循环的轮数

for(var j=0;j<arr.length-1;j++){

//循环轮数4轮 每轮对比次数

//0 4

//1 3

//2 2

//3 1

// 次数=长度-1-j

//内存循环：控制循环次数

for(var i=0;i<arr.length-1-j;i++){ //i是下标

//用当前的元素和下一个元素进行比较

//i i+1

if(arr[i]>arr[i+1]){ //如果当前元素大于下一个元素，则位置交换

var c=arr[i];

arr[i]=arr[i+1];

arr[i+1]=c;

}

}

}

console.log(arr);

**（7）二维数组**

在数组中的每个元素也是数组。

|  |
| --- |
| var arr=[['a','b','c'],['d','f'],['e'],…] |

访问二维数组中的元素 arr[下标][下标]

示例：

//所有的省市

var pro=['北京','天津','四川','广东'];

//二维数组（和前一个数组下标对应）

var city=[

['海淀区','西城区','东城区','朝阳区','丰台'],

['和平区','河东区','河西区'],

['成都','南充','宜宾','眉山'],

['广州','深圳','佛山','惠州']

];

//访问元素

console.log(city[0][1],city[0][2]);

console.log(city[3][1]);

//数组也是对象，属于引用类型

var arr1=['a','b','c'];

var arr2=arr1; //把arr1的地址赋值给了arr2

arr2[0]='d'; //通过arr2改变数组第一个元素，arr1也改变

console.log(arr1);

课后练习：

翻转数组中的元素（用遍历数组元素），不能使用reverse，['a','b','c','d']

顺着遍历：var arr=['a','b','c','d'];

for(var i=0,arr1=[];i<arr.length;i++){

arr1[i]=arr[(arr.length-1)-i];

}

console.log(arr1);

倒着遍历：var arr=['a','b','c','d'];

for(var i=arr.length-1,arr1=[];i>=0;i--){

arr1[(arr.length-1)-i]=arr[i];

}

console.log(arr1);

**1.字符串操作**

包装对象：目的就是让原始类型的数据也可以像引用类型数据一样，具有属性和方法。(本质还是字符串)

JS中提供了3种包装对象：String、Number、Boolean

将任意数据转为字符串：

构造函数：var str=new String(1) 返回object

普通函数：var str=String(1) 返回string

示例：将任意的数据包装成字符串对象

var str1=new String(1); //将数字1转成字符串1

//console.log(str1);

//console.log(typeof(str1)); //->object

//console.log('1'+2); //'12'

//console.log(str1+2); //'12'

var str2=String(2);

console.log(str2);

console.log(typeof(str2)); //->string

console.log(str2+2); //'22'

**（1）转义字符——\**

装换字符本身的意义

\n 将字符n转成了换行符

\' 将引号转成普通的引号字符

\t 将字符t转成了制表符（tab键的效果）

练习：打印 welcome to chi\na

console.log('welcome to chi\\na');

**（2）字符串api**

str.length 查看字符串的长度。

str.charAt(1)/str[1] 查看下标对应的字符。

str.charCodeAt() 查看某一个字符的Unicode码。

str.indexOf('@') 查找某个字符第一次出现的下标，找不到返回-1

str.lastIndexOf('a') 查找最后一个字符出现的下标，找不到返回-1。

str.toUpperCase() 将英文字母转为大写。

str.toLowerCase() 将英文字母转为小写。

str.slice(start,end) 截取元素，start是开始的下标，end是结束的尾标，不包含end本身。

str.substring(start,end) 截取元素，start是开始的下标，end是结束的尾标，不包含end本身。

atr.substr(start,count) start是开始的下标，count是要截取的长度。

str.split('-') 按照指定的字符'-'，将字符串分割为数组。

示例：

var str='javascript';

console.log(str.length);

console.log(str.charAt(4));

console.log(str[4]);

练习：声明变量存储邮箱，检验是否为合法格式（查看是否有@），如果是打印true，不是打印false

var email='87453689@qq.com';

if(email.indexOf('@')>=0){

console.log('true');

}else{

console.log('false');

}

if(email.indexOf('@')==-1){

console.log(false);

}else{

console.log(true);

}

练习：初始化4个英文字母（有大写小写），保存到变量中，循环弹出提示框，输入验证码（不区分大小写），如果输入正确，结束循环

var code='abCD';

do{

var str=prompt('input word');

if(str.toUpperCase()==code.toUpperCase()){

alert('right');

break;

}

}while(true);

示例：截取字符串

var str='javascript';

//console.log(str.slice(4));

console.log(str.substr(4));

//console.log(str.slice(4,7));

console.log(str.substr(4,3));

//console.log(str.slice(-3));

console.log(str.substr(-3));

//console.log(str.slice(-3,-1));

console.log(str.substr(-3,2));

练习：声明变量来保存一个邮箱，从中分别截取出用户名和域名

var email='tom345325@sina.com';

var i=email.indexOf('@')

var str1=email.slice(0,i);

console.log(str1);

var str2=email.slice(i+1);

console.log(str2);

//使用split截取邮箱用户名和域名

var email='tom345325@sina.com';

var arr=email.split('@');

console.log(arr[0],arr[1]);

练习：声明变量保存身份证号，从中截取出生年与日，获取性别（倒数第二位，奇数为男），打印：1988年09月09日 男

var sid='515512198809092577';

var year=sid.substr(6,4);

var month=sid.substr(10,2);

var day=sid.substr(12,2);

var sex=sid.substr(-2,1);

if(sex%2==1){

var s='男';

}else{

var s='女';

}

//三目运算var s=sex%2==0 ? '女' : '男'

console.log(year+'年'+month+'月'+day+'日 '+s);

练习：wElCome 转为首字母为大写->Welcome

var str='wElCome';

var first=str.slice(0,1).toUpperCase();

var last=str.slice(1).toLowerCase();

console.log(first+last);

//函数形式

function transforWord(word){

return word.slice(0,1).toUpperCase()+

word.slice(1).toLowerCase();

}

console.log(transforWord('wElCome'));

**（3）遍历字符串**

只能使用for循环

var str='javascript';

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

console.log(str[i],str.charAt(i));

}

练习：查看javascript中有多少个a?

for(var i=0,count=0;i<str.length;i++){

if(str[i]=='a'){

count++;

}

}

console.log(count);

**（4）匹配模式（掌握）**

作用：用于查找并替换字符串

str.replace(/value1/ig,value2) 查找并替换，value1要查找的字符串，value2要替换的字符串；value1既可以使用字符串形式，也可以使用正则表达式形式 /value/ig

i-ignore 忽略大小写

g-global 全局查找

str.match(/tom/gi) 查找匹配的字符串，返回数组（使用length可以查看出现的次数）。

str.search(/tom/i) 查找匹配的第一个字符串，返回下标，找不到返回-1。

示例：

var str='tom是一个程序员,TOM出生在中国,TOM上学的时候经常逃课,tom还没有女朋友';

//查找所有的tom并替换成中文汤姆

//console.log(str.replace('tom','汤姆')); 只改了第一个

//正则表达式写法

console.log(str.replace(/tom/gi,'汤姆'));

**2.Math对象**

Math对象不需要使用new来创建，可以直接使用。

Math.PI 获取圆周率

Math.abs(-3) 取绝对值

Math.ceil(3.1) 向上取整 ceil-天花板

Math.floor(5.9) 向下取整 floor-地板

Math.round(5.49) 四舍五入取整

Math.max(…) 获取一组数字的最大值

Math.min(…) 获取一组数字的最小值

Math.pow(x,y) 计算x的y次幂

Math.random() 取随机(默认取值范围0~1，包含0，不包含1)要得到1~任何数，如果向上取整，可能会出现0，所以只有加1再向下取整。

练习：随机获取0~9之间的整数

var num=Math.random()\*10

num=Math.floor(num)

console.log(num);

课后任务：

一句英文的首字母大写，其余小写：hoW aRe YOU

var str='hoW aRe YOU';

var arr=str.split(' ');

//遍历数组元素，获取每个单词

for(var i=0,str1='';i<arr.length;i++){

s1=arr[i].slice(0,1);

s2=arr[i].slice(1);

arr[i]=s1.toUpperCase()+s2.toLowerCase();

}

//将数组按照空格转成字符串

console.log(arr.join(' '));

练习2：创建数组，包含a~z和0~9，在数组中随机取4个字符，放在一个新数组中

var arr=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',0,1,2,3,4,5,6,7,8,9];

//准备一个新数组，用于储存随即找到的字符

for(var i=0,arr1=[];i<4;i++){

var num=Math.random()\*arr.length;

arr[i]=arr[Math.floor(num)];

//把获取的随机字符，放到新数组中

arr1.push(arr[i]); //arr1[i]=arr[i];

}

console.log(arr1);

**1.Date对象**

用于日期和时间进行储存和计算

**（1）创建date对象**

var d1=new Date('2019/1/16 10:0:0'); //打印时间往前推了8小时，中国在东8区

var d2=new Date(2019,0,16,10,0,0); //月份对应0~11

存储当前系统时间（自己电脑或者服务器）的date对象

var d3=new Date();

存储到数据库的时间（距离计算机元年（1970-1-1 0:0:0）的毫秒数）

var d4=new Date(1000\*60\*60\*24); //距离计算机元年1天的时间

2个date对象相减，获得相差的毫秒数。

**（2）获取对象中的信息（api）**

d.getFullYear/Month/Date/Hours/Minutes/Seconds/Milliseconds/Day/Time()

示例：

var d1=new Date('2019/1/16 10:40:50');

//获取距离计算机元年的毫秒数（计算机存储的时间）

console.log(d1.getTime());

//获取年，月，日，时，分，秒，毫秒，星期

console.log(d1.getFullYear());

console.log(d1.getMonth()); //对应0~11

console.log(d1.getDate());

console.log(d1.getHours());

console.log(d1.getMinutes());

console.log(d1.getSeconds());

console.log(d1.getMilliseconds());

console.log(d1.getDay()); //对应0~6（从星期天开始）

练习：获取2019/2/5 0:0:0，打印：2019年02月05月0时0分0秒 星期二

var d1=new Date('2019/2/5'); //默认0:0:0

var y=d1.getFullYear();

var m=d1.getMonth()+1;

var d=d1.getDate();

var h=d1.getHours();

var mi=d1.getMinutes();

var s=d1.getSeconds();

var day=d1.getDay();

var arr=['星期日','星期一','星期二','星期三','星期四','星期五','星期六'] //将day作为下标

//判断是否小于10

if(m<10){

//在数字前拼接字符0

m='0'+m;

}

if(d<10){

d='0'+d;

}

var rq=y+'年'+m+'月'+d+'日'+h+'时'+mi+'分'+s+'秒 '+arr[day];

console.log(rq);

练习：当前时间距离春节还有？天？小时？分？秒

var now=new Date();

var target=new Date('2019/2/5');

//两个对象之间毫秒差

var ms=(Math.abs(target-now)); //直接时间相减，获取相差的毫秒数

//var d=d2.getTime()-s.getTime();

var s=Math.floor(ms/1000); //通过毫秒获得秒

//通过相差的秒数获取天数

var d=Math.floor(s/(24\*60\*60));

//相差的秒数和一天的秒数取余=不满一天的秒数

var h=s%(24\*60\*60)

h=Math.floor(h/(60\*60));

//通过相差的秒数获取分钟数

var m=s%(60\*60)

m=Math.floor(m/60);

//通过相差的秒数获取秒数

var s1=s%60;

var days='距离2019年春节还有'+d+'天'+h+'时'+m+'分'+s1+'秒';

console.log(days);

**（3）设置(修改)date中的信息**

d.setFullYear/Month/Date/Hours/Minutes/Seconds/Milliseconds/Time()

月份 0~11

设置距离计算机元年秒数，有可能会覆盖所有的日期时间；

不能设置星期，星期由日期决定的。

示例：

//当前的date对象

var d=new Date('2019/1/16 14:33:45');

//三年前，获取当前的年份，在这个基础之上减3

d.setFullYear('2016');

d.setMonth(11); //0~11

d.setDate(5);

d.setMilliseconds(888);

//无法设置星期

//设置距离计算机元年的毫秒数

d.setTime(140000000000);

console.log(d);

**（4）转为本地时间**

d.toLocaleString() //年-月-日 时:分:秒

d.toLocaleDateString() //年-月-日

d.toLocaleTimeString() //时:分:秒

练习：创建2019-2-5 10:20:30日期时间的date对象，打印15天以后，8小时后的本地字符串

var d=new Date('2019/2/5 10:20:30');

d.setDate(d.getDate()+15); //获取日期加15

d.setHours(d.getHours()+8);

console.log(d.toLocaleString());

var ms=d.getTime();

var ms1=ms+15\*24\*60\*60\*1000+8\*60\*60\*1000

var d1=new Date(ms1);

console.log(d1.toLocaleString());

**（5）拷贝一个date对象**

var d1=new Date(date对象);

拷贝后会生成一个新的对象，和之前的对象没有关系。

练习：创建一个对象来保存一个员工的入职时间‘2018-11-16’，3年后到期，计算到期时间，合同到期前一个月续签合同，假设是周末提前到周五，计算续签时间，打印入职时间和到期时间，续签时间的本地。

var d=new Date('2018/11/16');

var d1=new Date(d); //入职时间

var d2=new Date(d1); //到期时间

d2.setFullYear(d2.getFullYear()+3);

var d3=new Date(d2) //续签时间

d3.setMonth(d3.getMonth()-1);

console.log(d3.getDay());

if(d3.getDay()==6){

d3.setDate(d3.getDate()-1); //周六提前1天

}else if(d.getDay()==0){

d3.setDate(d3.getDate()-2); //周天提前2天

}

console.log('入职时间'+d1.toLocaleDateString());

console.log('到期时间'+d2.toLocaleDateString());

console.log('续签时间'+d3.toLocaleDateString());

**2.Number对象**

var n1=new Number(值)；

将任意的数据转为数值型，返回对象。

var n2=Number(值);

将任意的数据转为数值型，返回数值型数据。

**（1）获取计算机能存储的最大、最小值（了解）**

Number.MAX\_VALUE

Number.MIN\_VALUE

**（2）保留小数点后n位**

num.toFixed(n)

示例：

var num1=0.1+0.2;

var num2=2\*3.14\*5;

//保留小数点后n位

console.log(num1.toFixed(1));

console.log(num2.toFixed(2));

**3.Boolean对象**

将任意的数据转为bool型。

false的5种情况：''、NaN、undefined、0、null

var b1=new Boolean(值);

将任意的数据转为布尔型，返回对象。

var b2=Boolean(值);

将任意的数据转为布尔型，返回布尔型数据。

**4.常见的报错**

**（1）出现的错误**

语法错误：错误的使用中文、缺少括号等——程序不执行。

var a=1；

SyntaxError: Invalid or unexpected token

无效或意外的字符

引用错误：使用了未声明的变量——影响后续代码的执行。

console.log(b);

ReferenceError: b is not defined

b未被定义

类型错误：错误的使用了括号——影响后续代码的执行。

var a=1;

a();

TypeError: a is not a function

a不是一个函数

范围错误：参数的使用超出范围——影响后续代码的执行。

var arr=new Array(-3);

RangeError: Invalid array length

无效的数组长度

**（2）错误处理**

尝试执行

|  |
| --- |
| try{  尝试执行，可能出现错误的代码  }catch(err){  err：捕获到的错误信息  处理错误的代码，执行其他代码  } |

try{

//可能出现错误的代码

var arr=new Array(-1);

}catch(err){ //catch捕获 err捕获的错误信息

console.log(err);

//处理错误

var arr=new Array(1);

}

console.log(arr);

课后任务：

双色球：随机取1~33之间的6个数字，不能重复，并且从小到大排列，去1~16之间一个数字，和之前的6个数字组成一个新数组。splice()

var arr=[];

for(var i=1;i<=33;i++){

arr.push(i); //得到1~33之间所有数组

}

for(var i=0,newarr=[];i<6;i++){ //随机取6个数

var num=Math.floor(Math.random()\*arr.length); //长度不能写33，删除元素后数值变化，只能使用arr.length

newarr.push(arr[num]);

arr.splice(num,1); //每次取完后在arr中删除此元素

}

newarr.sort(function(a,b){

return a-b;

});

var blue=Math.floor(Math.random()\*16+1); //向上取整可能会出现0，加1向下取整就不会出现0

newarr.push(blue);

console.log(newarr);

**5.ES6新特征**

JS由ECMA（欧洲计算机制造厂商联合会）维护、执行

ECMAScript 6（第6版本）

**（1）函数参数**

|  |
| --- |
| 设置参数的默认值  function add(a,b=0){  console.log(a+b);  }  add(5);  如果没有给参数传递值，则自动使用默认值。  如果给了参数，则会覆盖默认值。 |

**（2）块级作用域-let(局部作用域)**

使用let关键字声明的变量，不存在变量的提升，只能在块级作用域下使用，不能被外部访问。

块级作用域：{}、if、else、while、for等大括号下都是。

**（3）箭头函数**

是回调函数的另一种写法，和之前的匿名函数写法不能完全划等号。

|  |
| --- |
| sort((a,b)=>{  return a-b;  }); |

大括号中函数体只有一行代码，并且是return形式，大括号和return都可以省略：

sort((a,b)=>a-b);

示例：

var arr=[4,7,8,1,4,9];

arr.sort((a,b)=>{

return b-a; //大括号中只有一行代码，大括号和return都可以省略

});

console.log(arr);

简化为：

arr.sort((a,b)=>b-a);

console.log(arr);

**（4）模板字符串**——用于字符拼接

|  |
| --- |
| `在反引号之间可以写任意形式的代码${JS语法}` |

练习：创建一个对象来保存员工的姓名、性别、生日、工资，使用模板字符串打印以下内容

var emp={

name:'tom',

sex:1,

birthday:'1999-1-1',

salary:9000

};

console.log(`

姓名:${emp.name}

性别:${emp.sex==1 ? '男' : '女'}

生日:${emp.birthday}

工资:${emp.salary}

`);

1. **nodeJS概述**

nodejs是基于谷歌v8引擎的JS解释器，运行在服务器端。

http://nodejs.org（官网）

http://nodejs.cn（中文版，可以在网上去学习下）

**对比JS和nodeJS**

运行环境：js运行在浏览器端；nodejs运行在服务器端。

兼容性：由于js有多个浏览器，解释器不相同，存在兼容性；而nodejs只有一种解释器，不存在兼容性。

对象：都有原生对象，宿主对象，自定义对象；nodejs的宿主对象很多，js只有DOM和BOM。

作用：js用于网页中交互效果；nodejs用于服务器中创建web服务器，操作数据库，文件操作等。

**2.nodejs的执行方式**

脚本模式：node c:/xammp/nodejs…/01.js

交互模式：node回车 进入交互模式——退出：按两次 Ctrl+c或者 输入.exit

**3.全局对象**

**nodejs:global（global可以省略不写）**

在交互模式下，声明的变量或者创建的函数都属于是在全局对象下，可以使用global来访问。例如：var a=1; global.a

在脚本文件下，量声明的变或者创建的函数不属于全局对象下，也不能使用global来访问。

**js:window**

在js下，文件中创建的函数和变量都属于全局对象下，可以使用window来访问。

练习：在浏览器下运行脚本文件，在文件中声明变量，然后使用全局对象window来访问。

var a=1;

console.log(window.a);

function add(){

console.log(2);

}

window.add();

**（1）console**(控制台)**对象**

global.console.log() 打印日志

global.console.info() 打印消息

global.console.warn() 打印警告

global.console.error() 打印错误

global.console.time('自定义字符串') 开始计时

global.console.timeEed('自定义字符串') 结束计时

自定义字符串开始和结束计时必须保持一致

练习：使用计时功能查看for、do-while、while循环1000000次

global.console.time('while'); //开始计时

var i=1;

while(i<=100000){

i++;

}

global.console.timeEnd('while'); //结束计时

**（2）process**(操作)**对象**

用于查看当前计算机的进程

process.arch 查看CPU架构64位 ->x64

process.platform 查看当前操作平台 ->win32

process.env 查看当前环境变量

此电脑->属性->高级系统设置->高级->环境变量->系统变量->Path

process.pid 查看进程编号 ->7124

process.kill(7124) 关闭进程

process.version 查看当前nodejs版本号（或者node -v）

**（3）Buffer**(缓冲器)**对象**

缓冲区：在内存中保存数据的区域，网络传输时的资源。

创建Buffer缓冲区并保存数据

Buffer.alloc(5,'abcde') //大小为5个字节,不设置存储内容，将默认存储5个空字符''。

将Buffer数据转为普通字符串

buf.toString()

练习：创建Buffer，设置大小为10，保存1一个字母c，查看该数据

var buf=Buffer.alloc(10,'c');

console.log(buf.toString());

->cccccccccc

**4.全局函数**

parseInt/parseFloat/encodeURI/decodeURI/isNaN/isFinite/eval()…

（1）**一次性定时器**

|  |
| --- |
| setTimeout(回调函数,间隔时间) //间隔时间：毫秒数  当间隔时间到了，会自动执行一次回调函数；  clearTimeout(timer);  清除一次性定时器 |
| 也可写成周期性定时器  function fn(){  setTimeout(fn,1000);  console.log(123)  }  fn(); |
| 智能定时器(自动控制最佳时间，执行fn)——兼容性差  function fn(){  requestAnimationFrame(fn);  console.log(123)  }  fn(); |

示例：

global.setTimeout(function(){

console.log('炸弹来了');

},3000);

global.setTimeout(()=>{

console.log('炸弹来了');

},3000);

clearTimeout(timer);

**（2）周期性定时器**

|  |
| --- |
| setInterval(回调函数,间隔时间)  每隔一段时间，执行一次回调函数  clearInterval(timer);  清除周期性定时器 |

练习：使用周期性定时器打印3次hello

var count=0;

var timer=setInterval(()=>{

console.log('hello');

count++;

if(count==3){

clearInterval(timer);

}

},3000);

**（3）立即执行定时器**

|  |
| --- |
| process.nextTick(回调函数)  在整个主程序的最后执行（相当于：2018年的年末） |
| setImmediate(回调函数)  在事件队列的开头（相当于：2019年的年初）  clearImmediate(timer);  清除立即执行定时器 |

**5.模块**

模块是一个独立的功能体（积木块）

在nodejs下分为3个模块：核心模块、自定义模块、第三方模块

**（1）引入文件模块**

每一个文件都是一个独立的模块，里面的变量只能在内部使用，默认不能被外部访问。

以下代码中红色部分是NODEJS自动为每个文件添加的：(第一行代码出错可以看到)

|  |
| --- |
| (function(exports,require,module,\_\_filename,\_\_dirname){  程序员写的代码  }) |
| \_\_dirname 当前模块的绝对路径(directory 目录)  \_\_filename 当前模块的绝对路径和模块名称  require 用于引入一个其他模块，返回对象  module 指代当前的模块  module.exports 当前模块的导出（公开）对象，可以提供其他模块使用的属性和方法  exports 等价于module.exports |

示例：

|  |  |
| --- | --- |
| 脚本01.js  //(function (exports, require, module, \_\_filename, \_\_dirname) {  //程序员写的代码，自动被函数所包含  console.log(\_\_dirname); //当前模块(文件)的绝对路径  console.log(\_\_filename); //当前模块的绝对路径和名称  //需要引入其他模块  var obj=require('./02.js'); // ./表示同一级目录下，如果后缀名是.js，可以省略后缀名  //require('C:/xampp/htdocs/TL/nodeJS/day1.17/02.js');  console.log(obj);  console.log(obj.a);  obj.add();  obj.say();  //}) | 脚本02.js  //(function (exports, require, module, \_\_filename, \_\_dirname) {  console.log('02js');  var a=1;  //把a公开出去，引入的才能使用  //module.exports要公开的内容（要导出的内容）  module.exports.b=2;  module.exports.a=a;  function add(){  console.log(3);  }  //公开一个函数  //把函数add公开，而不是把函数的调用结果add()公开出去  module.exports.add=add;  //直接公开一个函数  module.exports.say=function(){  console.log(4);  }  //}) |

课后任务：

创建2个模块main.js（主模块），circle（功能模块）；在功能模块下创建2个函数，传递1个参数，获取远的周长和面积，导出这两个函数，在主模块中引入并调用2个函数

|  |  |
| --- | --- |
| //引入circle.js  var circle=require('./01\_circle'); //可以省略.js  var area=circle.getArea(5);  console.log(area.toFixed(2));  var len=circle.getLength(5);  console.log(len.toFixed(2)); | //导出circle  module.exports.getArea=function(r){  return Math.PI\*Math.pow(r,2);  }  module.exports.getLength=function(r){  return 2\*Math.PI\*r;  } |

**（2）引入目录模块**

|  |  |  |
| --- | --- | --- |
|  | 以路经开头 | 不以路径开头 |
| 文件模块 | require('./circle.js')  常用于用户自定义的模块 | require('url')  常用于引入官方提供的核心模块 |
| 目录模块 | require('./02\_2')  自动引入目录下的index.js，如果要引入其他文件需要创建package.json文件，设置main属性值。 | require('04\_2')自动到当前目录下寻找node\_modules中寻找模块，如果找不到，会一直网上一级寻找，知道最顶层目录。  常用于引入第三方模块。 |

练习：创建模块03\_1.js，引入当前目录下的03\_2目录模块；在03\_2下有一个test.js，到处函数add（传递2个参数），计算2个数字相加

|  |  |  |
| --- | --- | --- |
| 脚本03\_1.js | 文件夹03\_2 | |
| 文档package.json | 脚本test.js |
| //引入目录模块  var obj=require('./03\_2');  console.log(obj);  console.log(obj.add(2,3)); | {  "main":"test.js"  } | function add(a,b){  return a+b;  }  module.exports.add=add; |

练习：在05目录下创建05\_1.js,引入不以路径开始的05\_2，该目录中含有文件hello.js（打印一句话）

|  |  |  |
| --- | --- | --- |
| 脚本05\_1.js | 文件夹node\_modules | |
| 文件夹05\_2 | |
| 文件package.json | 脚本hello.js |
| //引入目录模块  require('05\_2'); | {  "main":"hello.js"  } | console.log('hello'); |

**（3）npm下载第三方包**

http://www.npmjs.com

npm在nodejs安装的时候自动安装了

npm -v 查看npm版本号

**切换目录：**

cd 完整路径（cd:change directory改变目录）

进入到要安装的目录，按住shift，单击右键，选择‘在此处打开powershell窗口’。

**使用npm安装第三方模块：**

进入需要将npm安装所在的目录

npm install 模块名称

**（4）核心模块**

是由nodejs官方提供的模块，可以直接引用，不需要创建或者安装。

**查询字符串模块**——querystring

http://www.codeboy.com/detail.html**?**lid=5&name=dell

parse() 将查询字符串解析为对象

stringify() 将对象转换成查询字符串

练习：使用查询字符串模块来获取数据

price=5000&pid=17&name=手机

const querystring=require('querystring'); //引入查询字符串模块——核心模块

var str='price=5000&pid=17&name=手机';

var obj=querystring.parse(str); //使用查询字符串模块解析

console.log(obj);

console.log(obj.price,obj.pid,obj.name);

var str2=querystring.stringify(obj); //将对象转换成查询字符串

console.log(str2);

**URL模块**

http://www.codeboy.com:80/admin/login.html?lid=5&name=dell

parse() 将URL解析为对象

protocol 协议

hostname 服务器域名或IP地址

port 服务器的端口

pathname 文件在服务器上的位置

query 查询字符串

format() 将一个对象转换成URL

示例：

var obj1={

protocol:'http',

hostname:'www.codeboy.com',

port:80,

pathname:'/admin/login.html',

query:{

lid:5,

name:'dell'

}

}

//将一个对象转成url

var str1=url.format(obj1);

console.log(str1);

练习：获取URL中查询字符串的数据cid

const querystring=require('querystring');

const url=require('url');

var str='http://www.tmooc.cn:8080/web/1812.heml?course=nodejs&cid=6';

var obj=url.parse(str); //解析URL

console.log(obj);

console.log(obj.query);

var obj2=querystring.parse(obj.query); //再将字符串解析

console.log(obj2.cid);

**同步和异步**

同步会阻止后续代码的执行，只有当前的代码执行完，才会继续执行后边的代码；同步方法是通过返回值来得到结果。fs.statSync

异步不会阻止后续代码的执行，把执行放到最后的实践队列中，是通过回电函数来获取结果。fs.stat

**fs模块**

**（1）查看文件和目录状态**

fs.stat(path,callback)/fs.statSync(path)

path 要查看的文件路径

callback 回调函数

err 如果文件有错误对应的错误信息

stats 文件的状态信息

isFile() 是否为文件

isDirectory() 是否为目录

示例：

//引入文件系统模块

const fs=require('fs');

异步函数：

fs.stat('05',(err,stats)=>{ //通过回调函数来获取状态信息

//err 如果查看失败的错误信息

//stats 文件的状态信息

if(err){

throw err; //打印错误的同时，终止往后执行

}

//console.log(stats);

//查看是否为文件

console.log(stats.isFile());

//查看是否为目录

console.log(stats.isDirectory());

});

同步函数：

var res=fs.statSync('05');

console.log(res);

console.log(res.isFile());

**（2）创建目录**

fs.mkdir(目录的路径，回调函数)/fs.mkdirSync(目录的路径)

示例：

const fs=require('fs');

//当前目录下创建mydir目录

fs.mkdir('mydir',(err)=>{

if(err){

throw err;

}; //可简写if(err) throw err;

console.log('目录创建成功');

});

fs.mkdirSync('mydir2');

**（3）删除目录**

fs.rmdir(目录的路径,回调函数)/fs.rmdir(目录的路径)

示例：

const fs=require('fs');

fs.rmdir('mydir',(err)=>{

if(err) throw err;

console.log('删除成功');

});

fs.rmdirSync('mydir2');

**（4）读取目录**

fs.readdir(目录的路径,回调函数)/fs.readdirSync(目录路径)

示例：

fs.readdir('node\_modules',(err,result)=>{

if(err) throw err;

console.log(result);

});

var result2=fs.readdirSync('node\_modules');

console.log(result2);

**（5）判断目录或者文件是否存在**

fs.existsSync(路径) 存在->true 不存在->false

**（6）创建文件/写入文件**

fs.writeFile(路径,添加的数据,回调函数)/fs.writeFileSync(路径,数据)

如果文件不存在，会创建文件；吐过文件存在，会清空内容，然后再写入。

fs.appendFile(路径,添加的数据,回调函数)/ fs.appendFileSync(路径,数据)

如果文件不存在，会创建文件；如果文件存在，会在末尾追加写入。

练习：使用文件操作，同步方法创建文件date.csv，写入对象

const fs=require('fs');

var emp={

eid:1,

ename:'tom',

sex:'男'

};

for(var key in emp){

console.log(emp[key]);

fs.appendFileSync('date.csv',emp[key]+','); //csv是Excel格式

}

**（7）读取文件**

fs.readFile(文件路径,回调函数)/fs.readFileSync(路径)

读取的数据格式是Buffer

示例：

const fs=require('fs');

fs.readFile('num.txt',(err,result)=>{

if(err) throw err;

//result读取的数据是Buffer格式

console.log(result.toString());

});

**（8）删除文件**

fs.unlink(路径,回调函数)/fs.unlinkSync(路径)

练习：判断num.txt文件是否存在，如果不存在则创建改文件，并写入数据0；读取文件num.txt中的内容，让读取的值加1，打印出来，把读取的值再次写入num.txt中

var filename='num.txt';

if(!fs.existsSync(filename)){

fs.writeFileSync(filename,0);

}

var data=fs.readFileSync(filename);

var num=data.toString();

num++; //自动换换为数值型

console.log(num);

fs.writeFileSync(filename,num);

课后任务：

创建01\_1.js模块，引入不以路径开头的目录模块01\_2,该模块下有文件time.js，导出一个函数，函数中每隔3秒钟打印时间'年-月-日 时：分：秒'，在01\_1.js中调用该函数

|  |  |  |
| --- | --- | --- |
| 脚本01\_1.js | 文件夹node\_modules | |
| 文件夹01\_2 | |
| 文件package.json | 脚本time.js |
| //引入目录模块  var obj=require('01\_2');  console.log(obj);  obj.timer(); | {  "main":"time.js"  } | function timer(){  setInterval(()=>{  var now=new Date();  console.log(now.toLocaleString());  },3000);  }  module.exports.timer=timer; |

**1.http协议**

是浏览器和web服务器之间的通信协议

**（1）通用头信息（General Headers）**

Request URL：请求的URL，获取服务器的哪些内容

Request Method：请求的方法

GET/POST

Status Code：响应的状态码

200：服务器端成功的响应

302：响应的重定向，跳转到另一个URL

404：浏览器端错误

5\*\*：服务器端错误

Remote Address：请求的服务器的IP地址和端口

**（2）响应头信息（Response Headers）**

Connection：连接方式

keep-alive：持续连接

Content-type：响应的文件类型

Content-Encoding：响应时的压缩类型

gzip

Transfer-Encoding：传输方式

chunked：分段传输

Location：响应重定向的时候，跳转的URL

**（3）请求头信息（Request Headers）**

Accept：浏览器端接受的文件类型有哪些

Accept-Encoding：接受的压缩类型

gzip，deflate

Accept-Language：接受的语言

User-Agent：当前使用的浏览器

**（4）请求主体(Form data)**

可有可无，用于客户端向服务器端传数据

post请求方法才会用。

**2.http模块**

既可以模拟浏览器向web服务器发请求，也可以创建web服务器

1. **模拟浏览器**

http.get(url,(res)=>{ });

get：请求的方法

url：请求的URL

callback：回调函数，获取服务器响应的内容

res：响应的对象

res.statusCode：响应的状态码

res.on('data',(buf)=>{ })：使用事件来获取服务器端响应的数据，参数buf就是响应的数据，格式为buffer

示例：

//引入http协议

const http=require('http');

//模拟浏览器向服务器发请求

//get获取服务器端内容

//参数1：请求URL

//参数2：获取服务器端响应

http.get('http://www.tmooc.cn',(res)=>{

//res:response是服务器端响应的对象

//获取响应的状态码

console.log(res.statusCode);

//获取响应的内容

//当有数据传输的时候，就会触发data事件

//把接收的数据放到buf中

res.on('data',(buf)=>{

console.log(buf.toString());

});

});

**（2）创建web服务器**

const http=require('http')：引入http

var server=http.createServer()：创建web服务器

server.listen(8080)：分配端口，监听8080端口

server.on('request',(req,res)=>{ })：接收浏览器的请求，是一个事件，只要有请求，自动执行。

* req：请求的对象

req.url：请求的URL，端口后的部分，默认 /

req.method：请求的方法，直接在浏览器地址栏输入，使用get

req.headers：请求的头信息

* res：响应的对象

res.write(文本)：响应文本

res.writeHead(状态码,{对象})：设置相应的状态码和头信息

res.end()：告诉浏览器响应结束

示例：

//引入http

const http=require('http');

//创建web服务器

var server=http.createServer();

//分配端口：监听9007

server.listen(9007,()=>{

console.log('web服务器创建成功');

});

//接收浏览器的请求

//当有请求发生的时候，自动执行

server.on('request',(req,res)=>{

//req：Request请求的对象

console.log(req.url);

console.log(req.method);

console.log(req.headers);

//res：Response响应的对象

console.log('有一个请求发生');

//跳转到另一个网页

res.writeHead(302,{

Location:'http://www.tmooc.cn',

server:'1812'

});

//响应文本

res.write('这是wb1812');

//响应结束

res.end();

});

练习：创建web服务器，监听8081端口，使用事件来接受浏览器请求，打印请求的方法和URL，接上一个练习，当请求URL为 /list，响应一行文本'this is list'；档请求URL为 /search，条转到'http://www.tmooc.cn'

http://127.0.0.1:8081/admin/login.html

http://localhost:8081/memmber/shopping.html

const http=require('http');

var server=http.createServer();

server.listen(8081);

server.on('request',(req,res)=>{

console.log(req.method);

console.log(req.url);

switch(req.url){

case '/list':

res.write('this is list');

break;

case '/search':

res.writeHead(302,{

Location:'http://www.tmooc.cn'

});

break;

}

res.end();

});

课后练习：

使用http模块创建web服务器，根据请求的URL响应不同的内容

/login 响应文本 this is login page

/index 响应文本 this is home page

/study 跳转到tmooc

/ 跳转到/index

/notfound 响应文本 404 not found

如果以上都没有，跳转到/notfound

const http=require('http');

var server=http.createServer();

server.listen(8080);

server.on('request',(req,res)=>{

switch(req.url){

case '/login':

res.write('this is login page');

break;

case '/index':

res.write('this is home page');

break;

case '/study':

res.writeHead(302,{

Location:'http://www.tmooc.cn'

});

break;

case '/':

res.writeHead(302,{

Location:'/index'

});

break;

case '/notfound':

res.write('404 not found');

break;

default:

res.writeHead(302,{

Location:'/notfound'

});

}

res.end();

});

**1.express框架**

基于nodejs，用于创建web服务器的框架

http://www.expressjs.com.cn

npm install express

|  |
| --- |
| const express=require('express'); //引入express模块  var server=express(); //创建web服务器  server.listen(8080); //监听端口 |

**（1）express中的路由**

浏览器向web服务器发请求，web服务器要根据请求的方法和请求的URL来做出相应。

路由三要素：请求的方法、请求的URL、响应

* 请求对象req

req.url 请求的URL

req.method 请求的方法

req.headers 请求的头信息

req.query 获取查询字符串中的数据，返回的格式为对象

* 响应对象res

res.send() 响应文本，只能响应一次，如果是数字认为是状态码

res.sendFile() 响应文件，必须使用绝对路径(\_\_dirname)

res.redirect() 跳转，响应重定向

练习：创建路由，请求方法为get，请求URL为/login，打印‘有一个用户登录’

const express=require('express'); //引入express模块

var server=express(); //创建

server.listen(8080); //监听

//路由

//请求方法：get，请求的URL：/list

server.get('/ list',(req,res)=>{

console.log('有一个用户登录');

//res.send(200); //->ok，响应的状态码，避免出现

res.send(`这是商品列表,<br>

商品的列表

`);

//res.sendFile(\_\_dirname+'/list.html');

//跳转

//res.redirect('http://www.tmooc.cn');

});

**（2）get和post请求**

get请求以查询字符串的形式传递数据，服务器端使用req.query获取数据，格式为对象。

post请求现阶段只能通过表单提交传递数据，服务器端使用事件形式获取数据，格式为查询字符串，需要借助查询字符串模块解析为对象。

|  |
| --- |
| req.on('data',(buf)=>{  buf.toString(); //buf是buffer数据，转换后为查询字符串  }) |

示例：

|  |
| --- |
| 03\_post.js |
| const express=require('express');  var server=express();  server.listen(8080);  //路由，获取注册页面  server.get('/reg',(req,res)=>{  //准备注册页面，响应到浏览器  res.sendFile(\_\_dirname+'/reg.html');  });  //根据表单的请求的方法和请求的URL来写对应的路由  //方法：get URL：/myreg  server.get('/myreg',(req,res)=>{  //获取get传递的数据  console.log(req.query);  res.send('注册成功');  });  //根据表单的提交，写对应路由  //请求方法：post 请求的URL：/mylogin  server.post('/mylogin',(req,res)=>{  //获取post请求的数据  //当有请求的数据时候，触发data事件  req.on('data',(buf)=>{  //console.log(buf.toString());  var mydata=buf.toString();  //解析对象  var obj=querystring.parse(mydata);  console.log(obj);  });  res.send('登录成功');  }); |
| reg.html |
| <h1>请注册</h1>  <!--  name：用于服务器端区分接收的数据  form：用于提交表单，需要设置项服务器端请求的方法和请求的URL  method：用于设置请求的方法 get/post  action：用于设置请求的URL  <input type="test" name="uname"> 文本框  <br> 换行  <input type="submit"> 提交按钮  -->  <form method="get" action="/myreg">  用户：<input type="test" name="uname"><br>  密码：<input type="text" name="upwd"><br>  手机：<input type="text" name="phone"><br>  <input type="submit">  </form>  <h1>请登录<h1>  <form method="post" action="/mylogin">  用户：<input type="test" name="uname"><br>  密码：<input type="text" name="upwd"><br>  <input type="submit">  </form> |

对比get和post两种请求方法

get 传递数据安全性相对低，但是速度比较快。搜索、检索…

post传递数据安全性相对高，但是速度比较慢。注册、登录…

**（3）使用路由传递数据**

设置路由中接收数据的名称lid

|  |
| --- |
| server.get('/detail/:lid',(req,res)=>{  req.params //获取路由传递的数据，格式为对象  }); |

浏览器请求

http://127.0.0.1:8080/detail/5

5就是传递的数据，会被lid所接收

练习：创建购物车路由，方法（get），URL（/shopping），传递商品的编号（lid）、价格（price）、名称（name）

const express=require('express');

var server=express();

server.listen(8080);

//商品详情的路由

//方法：get URL：/shopping

//:lid 设置接收数据的名称为lid

server.get('/shopping/:lid/:price/:name',(req,res)=>{

//获取路由传递的数据

console.log(req.params);

res.send('这是购物车');

});

**（4）路由器**

路由在使用过程中，在不同模块下的路由可能出现相同的URL，把同一个模块下的路由挂载到特定的前缀。

路由器就是一个js文件，把同一模块下的路由写在一起

|  |
| --- |
| const express=require('express');  var router=express.Router(); //创建一个空的路由器  router.get('/list',(req,res)=>{ }); //往路由中添加路由  module.exports=router; //导出路由器对象 |

在web服务下使用路由器

|  |
| --- |
| const productRouter=requiire('./product.js');  server.use('/product',productRouter); //把商品路由器挂载到/product下，路由访问形式/product/list |

示例：把商品下的路由挂载到/product下，访问形式/product/list；创建用户路由器（user.js）,添加路由用户列表（list），把该路由器在web服务器引入，并挂载到/user下

|  |
| --- |
| 05\_router.js |
| //引入商品路由器模块  const productRouter=require('./product.js');  //引入用户路由器模块  const userRouter=require('./user.js');  const express=require('express');  var server=express();  server.listen(8080);  //把商品路由器挂载到/product下  //参数1：挂载的URL  //参数2：要挂在的路由器  server.use('/product',productRouter);  //把用户浏览器挂载到/uers下  server.use('/user',userRouter); |
| product.js |
| //商品路由器模块  //引入express  const express=require('express');  //创建路由器，空对象  var router=express.Router();  //往路由器中添加商品模块下的所有的路由  router.get('/list',(req,res)=>{  res.send('这是商品列表');  });  //导出路由器  module.exports=router; |
| user.js |
| const express=require('express');  var router=express.Router();  router.get('/list',(req,res)=>{  res.send('这是用户列表');  });  //导出路由器  module.exports=router; |

**（5）中间件**

作用是为主要的业务逻辑所服务。

分为5个：应用级中间件、路由级中间件、内置中间件、第三方中间件、错误处理中间件

**应用级中间件**

每一个中间就就是一个回调函数，需要配合其他的中间件或者路由使用。

server.use((req,res,next)=>{ }) 拦截所有的路由，每个路由都会调用回调函数。

server.use('/login', (req,res,next)=>{ }) 拦截路由中URL为/login的路由。

练习：创建路由（get /view），初始化num为0，在中间件中让num加1，在路由中将num响应到浏览器中

const express=require('express');

var server=express();

server.listen(8080);

//拦截/view的路由

//参数1：要拦截的路由的URL

var num=0;

server.use('/view',(req,res,next)=>{

num++;

next();

});

server.get('/view',(req,res)=>{

//send中的内容需要为字符串格式

res.send(num+''); //num+空字符是隐式转换，num.toString()

});

浏览器端：http://127.0.0.1:8080/view

->1（刷新，数字自增）

**路由级中间件**

用于服务器中将路由器挂载到特定的URL

server.use('/user',userRouter)

**内置中间件**

在express下只有一个内置中间件

server.use(repress.static('要托管的目录'))

托管静态资源到某一个目录下，如果浏览器要请求静态资源，则自动会到目录下寻找，无需添加路由。

静态资源：html、css、js、img…

示例：

const express=require('express');

var server=express();

server.listen(8080);

//托管静态资源（html、css、js、img...）到指定的目录

//如果浏览器请求静态资源文件，则会自动到该目录下寻找，无需再单独的添加路由

//使用内置中间件

server.use(express.static('public'));

//往public添加图片，使用浏览器请求图片http://127.0.0.1:8080/tmooc.png

//托管静态资源到files目录下，如果两个目录下有相同的文件，查看显示哪一个目录？

server.use(express.static('files'));

先使用中间件的目录中文件，如果public没有，才会使用files中的文件。

**第三方中间件**

body-parser

作用：将post请求的数据解析为对象

使用步骤

|  |
| --- |
| const bodyParser=require('body-parser');  server.use(bodyParser.urlencoded({  extended:false  }))  urlencoded：可以将数据解析为对象  extended:false: 不使用第三方的qs模块，默认会使用querystring将查询字符串解析为对象 |

在路由中获取post请求的对象

req.body 返回对象

|  |
| --- |
| req.params 获取路由传递的数据  req.query 获取get请求传递的数据  req.body 获取post请求传递的数据 |

练习：创建web服务器，在托管静态资源到public下，在public下创建longin.html文件，包含用户名和密码，点击提交（post,/mylogin），创建对应的路由。

|  |
| --- |
| 03\_post\_data.js |
| //引入body-parser中间件  const bodyParser=require('body-parser');  const express=require('express');  var server=express();  server.listen(8080);  //使用内置中间件，请求静态文件  server.use(express.static('public'));  //中间件将post请求的数据直接解析为对象  server.use(bodyParser.urlencoded({  extended:false //不使用第三方的qs模块，默认会使用querystring将查询字符串解析为对象  }));  //根据表单提交，写对应的路由  server.post('/mylogin',(req,res)=>{  //获取post请求的数据  console.log(req.body);  res.send('登录成功');  }); |
| public |
| 2.html |
| <h1>注册用户</h1>  <form method="post" action="/mylogin">  用户：<input type="next" name="uname"><br>  密码：<input type="next" name="upwd"><br>  <input type="submit">  </form> |

课后任务：

创建web服务器，请求得到html文件，在文本框中输入省份证号，点击查询，在中间件中提取身份证的年月日和性别，在路由中响应到浏览器中。

|  |
| --- |
| 01\_homework.js |
| const express=require('express');  var server=express();  server.listen(8080);  //创建路由，响应html文件  server.get('/check',(req,res)=>{  res.sendFile(\_\_dirname+'/check.html');  });  //中间件，拦截/mycard  var year,month,date,sex;  server.use('/mycard',(req,res,next)=>{  var obj=req.query;  console.log(obj);  //提取年月日  year=obj.cid.substr(6,4);  month=obj.cid.substr(10,2);  date=obj.cid.substr(12,2);  //提取性别  sex=obj.cid.substr(-2,1);  next();  });  //根据表单的请求，写对应的路由  server.get('/mycard',(req,res)=>{  //console.log(req.query);  //console.log(typeof(req.query.cid));  res.send(`您的生日为${year}年${month}月${date}日  性别是${sex%2==1 ? '男' : '女')}  `);  }); |
| check.html |
| <h1>输入身份证号</h1>  <!--  method="get" action="/mycard"需要将请求的数据发送至路由  -->  <form method="get" action="/mycard">  身份证号：<input type="text" name="cid"><br>  <input type="submit">  </form> |

**1.MySQL模块**

**（1）普通链接**

var connection=mysql.createConnection({ //创建连接对象

服务器的IP

端口

用户名

密码

使用数据库名称

});

connection.connect();

connection.query(sql,callback); 执行sql语句，callback回调函数，用于获取sql语句的执行结果

connection.end(); 执行完多有的sql语句，关闭连接

示例：

//引入MySQL

const mysql=require('mysql');

//创建连接——对象

var connection=mysql.createConnection({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu' //连接成功后要使用的数据库

});

//执行连接——连接成功

connection.connect();

//执行SQL语句

connection.query('SELECT \* FROM emp',(err,result)=>{

//err sql语句执行是吧的错误信息

//result sql语句的执行结果

if(err) throw err;

console.log(result);

});

//关闭连接

connection.end();

**（2）连接池**

var pool=mysql.createPool({

参数同上

connectionLimit:15

}); 创建连接池

pool.query(sql,callback); 执行sql语句，callback回调函数，用于获取sql语句的执行结果

示例：

//引入mysql模块

const mysql=require('mysql');

//创建连接池对象

var pool=mysql.createPool({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu',

connectionLimit:20 //设置连接池的大小，默认15个

});

//执行sql语句

//查询数据——返回数组

pool.query('SELECT \* FROM dept',(err,result)=>{

if(err) throw err;

console.log(result);

});

//删除数据——返回对象，如果affectedRows大于0，则有数据被删除，否则没有任何数据被删除

pool.query('DELETE FROM dept WHERE did=40',(err,result)=>{

if(err) throw err;

console.log(result);

});

//更改数据——返回对象，如果affectedRows大于0，则有数据被更改，否则没有任何数据被更改

//占位符(?)，在第二个参数的数组中添加数据，作用：防止在字符串中套字符串，防止SQL注入

pool.query('UPDATE dept SET dname=? WHERE did=?',['人力部',30],(err,result)=>{

if(err) throw err;

console.log(result);

});

//插入数据——返回对象，如果affectedRows大于0，则有数据插入，否则没有任何数据插入

pool.query('INSERT INTO dept VALUES(?,?)',[40,'运营部'],(err,result)=>{

if(err) throw err;

console.log(result);

});

//可以将对象作为数据插入

//更改数据不能使用对象

var deptobj={

did:50,

dname:'生产部'

}

pool.query('INSERT INTO dept SET ?',[deptobj],(err,result)=>{ //插入对象使用SET

if(err) throw err;

console.log(result);

});

练习：创建web服务器，在public下建add.html，将请求的数据插入数据库tedu中的dept表中

|  |
| --- |
| 06\_add.js |
| const mysql=require('mysql');  const express=require('express');  var server=express();  server.listen(8080);  //创建连接池  var pool=mysql.createPool({  host:'127.0.0.1',  port:'3306',  user:'root',  password:'',  database:'tedu',  connectionLimit:20  });  //托管静态文件到public下  server.use(express.static('public'));  //根据表单的请求，对应的路由  server.get('/add',(req,res)=>{  console.log(req.query);  pool.query('INSERT INTO dept SET ?',[req.query],(err,result)=>{  if(err) throw err;  if(result.affectedRows>0){  res.send('部门插入成功');  }  });  }); |
| public |
| add.html |
| <h1>输入部门数据</h1>  <form method='get' action="/add">  部门编号：<input type="text" name="did"><br>  部门名称：<input type="text" name="dname"><br>  <input type="submit">  </form> |

课后练习：

创建web服务器，托管静态资源到public，创建文件emp\_add.html，点击提交（post /add）将数据添加到tedu下的emp表中：（没有eid是因为，建表是设置默认值为null，自增列，会自动添加null）

姓名：

性别：

生日：

工资：

部门：

提交

|  |
| --- |
| 01\_homework.js |
| const mysql=require('mysql');  const bodyParser=require('body-parser');  const express=require('express');  var server=express();  server.listen(8080);  //托管静态文件  server.use(express.static('public'));  //创建连接池  var pool=mysql.createPool({  host:'127.0.0.1',  port:'3306',  user:'root',  password:'',  database:'tedu',  connectionLimit:20  });  //配置body中间件  server.use(bodyParser.urlencoded({  extended:false  }));  //根据请求的方法和URL写对应的路由  server.post('/add',(req,res)=>{  console.log(req.body);  //把数据插入MySQL数据库  pool.query('INSERT INTO emp SET ?',[req.body],(err,result)=>{  if(err) throw err;  if(result.affectedRows>0){  res.send('添加成功');  }  });  }); |
| emp\_add.html |
| <h1>请输入员工信息</h1>  <form method="post" action="/add">  姓名：<input type="text" name="ename"><br>  性别：<input type="text" name="sex"><br>  生日：<input type="text" name="birthday"><br>  工资：<input type="text" name="salary"><br>  部门：<input type="text" name="deptId"><br>  <input type="submit">  </form> |

浏览器输入：http://127.0.0.1:8080/emp\_add.html

李然 QQ：1535212067

**一、课程安排**

|  |
| --- |
| 1.HTML5基础（2~3天）  2.AJAX（3天）  3.项目1（2天）  mysql、js、nodejs、html、ajax  从前端页面对用户数据的增删改查  4.CSS3（4+2天）美化html  5.Bootstrap框架 CSS框架，响应式布局，简化css开发（4天）  6.项目三天 |

20天课程遇到的问题

|  |
| --- |
| 1.知识点细碎，调理型低  2.单词量巨大，关键很长  3.键速  4.笔记 |

**1.web与internet**

internet全球计算机互联网 俗称：互联网，交换网，交互网

www服务：word wide web万维网服，访问网站的服务

BBS：论坛

Email：邮件服务

telnet：远程服务

FTP：上传下载

这些服务都是运行在internet

**2.internet上的应用程序**

**（1）C/S**

client：客户端

server：服务器

代表应用：QQ、微信、手游、.exe

**（2）B/S**

browser：浏览器

server： 服务器

所有网站

**（3）区别**

cs需要升级，bs不需要升级

**3.web的运行原理**

web运行在internet之上的一种B/S的应用程序，俗称网站

internet为web应用运行提供网络环境

web的工作原理：基于浏览器和服务器以及通信协议，来实现的数据传输和展示。

通信协议：规范了数据是如何和打包和传递的

**服务器**：

（1）功能

储存数据

接收用户的请求并给出相应

提供程序的运行环境

具备一定的安全功能

（2）服务器的产品

APACHE(php)

Tomcat(java)

IIS(.net)

（3）服务器技术

php、java、.net、nodejs、python

**浏览器**：

（1）功能

代表用户发送请求

作为HTML、CSS、JS的解析器

以图形化的界面展示给用户

（2）浏览器的产品

chrome、safari、Firefox、Opera、IE--edge

（3）浏览器

HTML5、CSS3、JS

**二、HTML5快速入门**

**1、什么是HTML**

Hyper Text Markup Language 超文本标记语言

普通文本 a——普通字母

超文本 <a>百度</a>——超链接

标记：超文本的组成形式，具有自己的功能

语言：有自己的语法结构

**2.HTML的特点**

（1）以.html或者html.为后缀

（2）有浏览器解析执行

（3）用带有尖括号的"标记"标识

（4）在网页中可以嵌套js脚本

**3.HTML的基础语法**

**（1）标记**

又称：元素、标签、节点

标记在使用时，必须使用尖括号<>括起来

标记的分类：

双标记（封闭类型标记）

<标签> </标签>

<a></a> <button></button>

单标记（非封闭类型标记）

<标签>或者<标记/>

示例：

<img> 或者写成<img/>

练习：创建01\_first.html

在文件中按以下要求写出标记

写一对html标记

写一对head标记

写一对title标记

**（2）标签的嵌套**

在一堆标签中出现另一对标签，从而形成功能的层叠

|  |
| --- |
| <a href="http://www.baidu.com">  <b>百度</b> <!--b加粗-->  </a>  语法：  <标记1>  <标记2>  <标签3>  </标记2>  </标记1>  嵌套层级不要出错 |

练习：把01\_first.html中的三个要素进行嵌套

html在最外面（爷爷）

head在中间（父亲）

title在最里面（儿子）

|  |
| --- |
| **editplus快捷键：**  Ctrl+d：删除当前行  Ctrl+Alt+：复制当前行到下一行  Alt+：移动当前行  Ctrl+/：注释符 |

**（3）属性和值**

HTML5允许通过属性和值对标签进行修饰

<p align="center">汝乃天骄</>

属性必须在开始标记的内部

<标签 属性="值"></标签>

多属性设置

<标签 属性="值1" 属性="值2"></标签>

标准属性（通用属性）

所有元素都支持的属性

id： 定义元素在页面中的唯一标示

title：鼠标悬停在元素上所提示的文本

示例：<p align="center" title="你那么牛">汝乃天骄</>

style：css中，定义行内样式

示例：<span style="color:pink">哈哈</span>\_web呵呵

class：css中，引用类型选择器

**（4）注释**

<!--注释内容-->

注释不能嵌套注释

标记的<>中，不能出现注释

总结：学习HTML5标签

就是学习固定的标签名，及其功能

学习固定的标签属性，及其功能

学习标签的的嵌套关系

**三、HTML文档的结构**

**1．HTML文档的构成**

**（1）文档类型声明**

<!doctype html> H5版本

**（2）网页结构**

一个页面有且只有一对根节点<html></html>

在它内部还有2个元素

<head></head> 网页的头部，定义全局的信息

<body></body> 网页的主体，定义网页的内容

|  |
| --- |
| 练习：创建02\_ex.html编写最简洁的html页面结构  <!doctype html>  <html>  <head></head>  <body></body>  </html> |

**（3）head元素，是其他头元素的容器**

<meta /> 元数据，修饰网页的数据

|  |
| --- |
| <head>  <title>网页标题</title>  <meta charset="utf-8"/>  <meta name="description" content="内容描述">  <meta name="Keywords" content="关键字">  <style>定义网页的内部样式</style>  <link> 引用网页的外部样式  <script>定义或者引用js文件</script>  </head> |

**（4）body定义网页的主体**

bgcolor="pink" 背景颜色

text="red" 文本颜色

**四、文本标记**

**1．标题**

在页面中以醒目的方式显示文本

语法: <h1></h1>

<h2></h2>

<h3></h3>

<h4></h4>

<h5></h5>

<h6></h6>

特点：字体大小有变化h1最大，h6最小

加粗显示字体

单独成行，上下有垂直距离

属性：align——设置标记内容水平对齐方式

取值：left（默认）/center/right

**2.段落标签paragraph**

以段落的形式显示一段文字

<p></p>

特点：单独成行，上下有空白间距

属性：align

取值：left/center/right

练习：使用标题标签和段落标签模拟个人简历

**3.换行标记**

<br> 或者 <br/>

**4.水平线（分割线）**

<hr> 或者 <hr/>

属性：size="15px" 水平线的尺寸（粗细），单位为px的数值

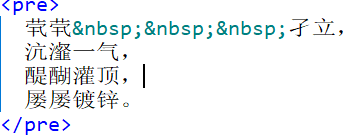
width="50%" 水平线的宽度，单位为px的数值或者%

align="right" 水平线的对齐方式（left/center/right）

color="pink"> 水平线的颜色，合法颜色的值

**5.预格式化标签**

<pre></pre>

保留HTML代码中的回车和空格效果

**6.特殊字符**

&nbsp; 空格

&lt; 小于<

&gt; 大于>

&copy; 版权©

&reg; 注册诸®

&yen; 人民币￥

&times; 关闭的小叉×

练习：© 北京金桥科技有限公司版权所有 <京ICP>备12003709号-3

&copy;北京科技有限公司 &lt;京JD&gt;

**7.文本样式**

载体加粗：

<strong></strong>

<b></b>——不建议用

斜体：

<em></em>

<i></i>——不建议用

删除线：

<del></del>

<s></s>——不建议用

上标和下标：

2<sup>3</sup> 上标（p的肚子在上）

2<sub>3</sub> 下标（b的肚子在下）

下划线：

<u></u>

**8.分区元素**

**（1）块分区，用于页面中布局**

<div></div>

特点：单独成行

**（2）行分区，用于处理一行文本中的不同样式**

<span></span>

特点：与其他行内元素共用一行

示例：<span style="color:pink">哈哈</span>\_web嘿嘿

**9.元素的显示方式**

**（1）行内元素（行级元素）**

在页面中多个元素在一行中显示

从左往右排列

ex:span、i、em、b、strong、u、s、del、sup、sub

**（2）块级元素**

在网页中独占一行的元素

从上往下

ex:div、p、h1~h6、结构标记

**（3）行内块元素**

表现方式如同行内元素，但是同时具备块级元素的特征

**（4）table**

表格的宽高，根据表格具体内容来决定

内容多，表格就宽一些，高一些

是一种单独表现方式

课后练习：

|  |
| --- |
| 使用h、p、div完成个人简历  <!doctype>  <html>  <head>  <title>个人简历</title>  <meta charset="utf-8">  </head>  <body bgcolor="silver">  <div>  <h1 align="center">个人简历</h1>  </div>  <hr size="15px" color="orange">  <div>  <h3 align="center">基本信息</h3>  <hr size="15px" width="20%" align="center" color="green">  <p style="color:red">姓名：  <span style="color:purple">tom</span></p>  <p style="color:red">性别：  <span style="color:purple">男</span></p>  <p style="color:red">年龄：  <span style="color:purple">18</span></p>  <p style="color:red">籍贯：  <span style="color:purple">四川宜宾</span></p>  <p style="color:red">现居地：  <span style="color:purple">四川成都</span></p>  <p style="color:red">政治面貌：  <span style="color:purple">党员</span></p>  </div>  <hr size="15px" color="orange">    <div>  <h3 align="center">工作经验</h3>  <hr size="15px" width="20%" align="center" color="green">  <p>第一份工作：</p>  <p>第二份工作：</p>  <p>第三份工作：</p>  </div>  <hr size="15px" color="orange">    <div>  <h3 align="center">获得成就</h3>  <hr size="15px" width="20%" align="center" color="green">  <p>证书：</p>  <p>项目：</p>  </div>  <hr size="15px" color="orange">  <div>  <h3 align="center">个人评价</h3>  <hr size="15px" width="20%" align="center" color="green">  <p>沉着冷静、学习能力强、善于沟通</p>  </div>  <hr size="15px" color="orange">  </body>  </html> |

**一、图像和链接**

**1.使用**

|  |
| --- |
| <img>或者<img/>  属性 src="url/图片路径" |

**2.URL**

|  |
| --- |
| Uniform Resource Locator 统一资源标识符/统一资源定位器  俗称：路径  绝对路径：从文件所在的最高级的目录下开始查找资源文件，所在的完整的路径。  目录位置：服务器所在位置的绝对路径下，我们电脑server是键在xz下面，所以xz就是服务器的根目录。 |

**3.路径的表现形式**

**（1）绝对路径——完整的路径**

通信协议+主机名+目录结构+文件具体名称

例如：http://cdn.tmooc.cn/bsfile//imgad///7D4B2196A.png

使用网络资源的时候，使用绝对路径。

优点：不占用服务器的存储空间

缺点：不稳定

本地资源的绝对路径：

<img src="C:\Users\web\Desktop\web1812\day02\05.png">

项目中不使用，需要每个浏览器所在的电脑，硬盘存储了该文件。

**（2）相对路径——参照物，当前html文件**

同级目录：直接写资源名称

src="02.png"

子集目录：先直接写兄弟文件夹名称，再用 / 调用兄弟的孩子

src="image/06.png"

父级目录：先找父亲../，再直接调用父亲的兄弟文件

src="../03.png"

|  |
| --- |
| 如图： |

**（3）image属性**

src(source源)：设置资源路径

alt：图片资源出错时，显示的文本

width：设置图片的宽度，以px为单位的数字

hight：设置图片的高度，以px为单位的数字

注意：如果设置图片的宽高，不符合原始图片的宽高比，会出现图片的失真效果。

解决失真：只设置宽或者高，另一个不设置，另一个属性自动适应。

练习：

01\_ex中，显示5张图片，分别使用5种方式显示

<!doctype>

<html>

<head>

<title></title>

<meta charset="utf-8">

</head>

<body>

<img src="http://cdn.tmooc.cn/bsfile//imgad///247B30A693DC4477BB4B40A7D4B2196A.png">

<img src="02.png">

<img src="./image/06.png">

<img src="../03.png" width="500px">

<img src="../../000.png" alt="我是一个大头兵">

</body>

</html>

**4.链接**

**（1）语法**

<a>链接路径</a>

属性：href：链接的路径

target：指定打开连接的方式

取值：\_self：默认值，在当前标签页面打开新链接

\_blank：在新的标签页中打开链接

<a target="\_blank" href="http://www.tmooc.cn">TMOOC</a>

**（2）a标签的其他表现方式**

下载资源

<a helf="03.zip">下载</a>

链接到电子邮件

<a href="mailto:1234556@qq.com">新建邮件</a>

执行调用JavaScript脚本

<a href="javascript:say()"></a>

返回当前网页的顶部

<a href="#">返回顶部</a>

**（3）锚点**

什么是锚点？

网页中的一个记号，

使用超链接直接跳转到记号处。

定义锚点（锚点的位置）

1.使用a标签的name属性定义锚点

<a name="锚点名称"></a>

2.使用任意标签的id属性定义锚点

<any id="锚点名称">

链接的锚点（点击就跳转到锚点位置）

1.在a标签href属性中添加#锚点名称，链接到当前页的锚点

<a href="#锚点名称">跳转到锚点</a>

2.链接到其他页的锚点？

<a href="网页的url.html#锚点名称"></a>

练习：

03\_ex.html

在网页中分别用div和h2定义两个锚点

通过a标记链接到以上两个锚点

然后再其他网页中用p元素定义一个锚点

锚点名称随意，通过03\_ex中的a标签链接到p元素这个锚点，另外，一个定义了3个锚点，本页面2个，其他页面1个。

要求锚点的定义，两种方式都要用

|  |
| --- |
| 03\_ex.html |
| <!doctype>  <html>  <head>  <title></title>  <meta charset="utf-8">  </head>  <body>  <div id="web">web课程</div>  <p id="yw">运维课程</p>  </body>  </html> |
| 03html |
| <!doctype>  <html>  <head>  <title></title>  <meta charset="utf-8">  <script>  function say(){  alert("新年快乐");  }  </script>  </head>  <body>  <a href="javascript:say()"></a>  <!--新建邮件-->  <a href="mailto:1234556@qq.com">新建邮件</a>  <br>  <a href="03.zip">下载</a>  <br>  <!--链接到网页-->  <a target="\_blank" href="http://www.tmooc.cn">TMOOC</a>  <br>  <p id="doudou">斗破苍穹</p>  <a href="#">返回顶部</a>  <br>  <a href="03\_ex.html">03\_ex</a>  </body>  </html> |

**二．表格**

作用：按照一定的结构来显示数据。

数据都保存在表格中。

按照数据从上往下（或者从左往右来显示数据）。

**1.表格的语法**

**（1）标记**

|  |
| --- |
| 表： <table>  行： <tr>  列： <td></td>  </tr>  </table> |

**（2）属性和值**

table

border：设置表格边框

bordercolor：设置表格表框颜色

width：设置表格宽度

height：设置表格高度

align：设置表格本身水平方向对齐方式

bgcolor：设置表格背景颜色

cellpadding：设置单元格内边距，内容到边框之间的距离

cells pacing：设置单元格外边距，边框与边框之间的距离

border-collapse:collapse;列之间无空隙

练习：

05\_ex.html

做一个4\*4表格，设置表格尺寸400px\*400px

背景颜色为黄色

表格边框1px

表格水平居中

表格内边框5px，表格外边框10px

|  |
| --- |
| 建表快捷键：table>tr\*4>td\*4 按tab键，创建相同内容 |

tr

align：设置当前行内容的水平对齐方式

valign：设置当前行内容的垂直对齐方式 top/middle/bottom

bgcolor：设置当前行的背景颜色

td

width/height：设置当前列的宽高，但是有可能会影响其他列

align/valign：设置当前列的内容对齐方式

bgcolor：设置当前列背景颜色

注意：传统的表格，每行的列数都是统一的。

一行中所有的单元格高度都是相同的，默认情况下以最高的单元格为准。

每行中，相同的列宽度都是相同的，默认情况下以最宽的单元格为准。

**（3）不规则的表格**

colspan（col—column）：合并列

rowspan：合并行

夸列

colspan="n"

从指定单元格的位置处，横向向右合并n个单元格（包含自己），被合并的单元格要删除。

跨行

rowspan="n"

从指定单元格位置处开始，纵向向下合并n个单元格（包含自己），被合并的单元格要删除（被删除的单元格在其他tr中）。

**（4）可选标签 table>tr>td**

表格标题

<caption> </caption>

如果要设置标题，必须在table标签的后面

行/列的标题

<th></th>

th与td的效果和属性基本一致

th有加粗和文本居中

**（5）复杂的表格应用**

行分组，可以将联系的几行，划分到一个组中，进行统一管理

表头行： <thead></thead>

表主体： <tbody></tbody>

表尾： <tfoot></tfoot>

表格嵌套

表格中的所有嵌套只能放在td中。

|  |
| --- |
| <!doctype>  <html>  <head>  <title>不规则表格</title>  <meta charset="utf-8">  </head>  <body>  <table border="1px">  <caption>一周菜单</caption>  <thead>  <tr>  <th>日期</th><!--th：加粗居中，其余效果和td一样-->  <th>午餐</th>  <th>晚餐</th>  </tr>  </thead>  <tbody>  <tr>  <th>周1</th>  <td>肌肉</td>  <td>鸭肉</td>  </tr>  <tr>  <th>周2</th>  <td>鹅肉</td>  <td>鸡腿</td>  </tr>  <tr>  <th>周3</th>  <td>香肠</td>  <td>火腿肠</td>  </tr>  <tr>  <th>周4</th>  <td>腊肉</td>  <td>火腿肠</td>  </tr>  <tr>  <th>周5</th>  <td>火锅肉</td>  <td>  <table>  <tr>  <td>烤羊腿</td>  <td>烤骆驼</td>  </tr>  <tr>  <td>烤板筋</td>  <td>烤鱿鱼</td>  </tr>  </table>  </td>  </tr>  </tbody>  <tfoot>  <tr>  <td colspan="3">吃胖的目标</td>  </tr>  </tfoot>  </table>  </body>  </html> |

**一．列表（重点）**

**1.列表**

按照数据从上往下（或者从左往右来显示数据）

可以在数据之前增加标识项

**2.列表的组成**

列表是由"列表类型"和"列表项"组成的

**（1）列表类型**

有序列表ol

<ol>——order list

<li>…</li>——list item

</ol>

ol标签的直接子元素只能有li标签。

属性1：type：指定标识项的类型

取值：1、a、A、i、I

属性2：start：指定起始编号，取值为无单位数字

无序列表ul

<ul>——unorder list

<li>…</li>

</ul>

属性：type：定义列表项的标识类型

取值：none：什么都没有

disc：实心圆

circle：空心圆

square：实心方块

**3.嵌套列表**

|  |
| --- |
| 被嵌套列表要放在li中  <ul>  <li>黑猫警长  <ul>  <li>黑猫</li>  <li>白猫</li>  </ul>  </li>  </ul> |

**4.定义列表**（名词解释的效果）

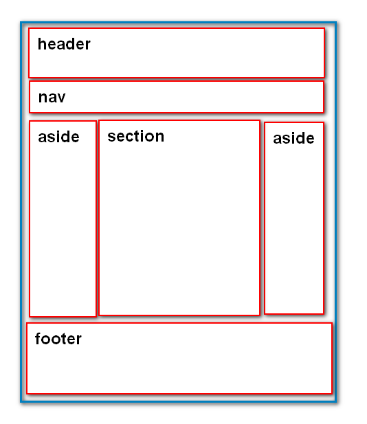
|  |
| --- |
| <dl>  <dt>需要解释的文字</dt>  <dd>具体解释的内容…</dd>  </dl> |

**二．结构标记**

用于描述整个网页的结构，取代div的布局，提升标记语意性

**常用的标记**

|  |
| --- |
| 1.<header></header>——定义网页或者某区域头部内容  2.<nav></nav>——定义网页导航链接的部分  3.<section></section>——网页的主体内容  4.<aside></aside>——侧边栏  5.<footer></footer>——定义网页中底部内容  6.<article></article>——定义与文字相关的内容，比如论坛，帖子，微博，评论… |

如图：

**三．表单（重点和难点）**

**1.作用**

提供可视化的输入控制，收集用户信息并提交给服务器。

**2.表单的组成部分**

前端部分：提供表单控件，与用户交互的可视化控制。

后端部分：对提交的数据进行处理（接口）。

**3.表单**

<form></form>

属性（3个）：

（1）action：定义表单被提交时发送动作，就是定义服务器处理程序的地址、接口、URL。默认提交给本html的地址

（2）method：指定表单数据提交的方式

取值：get（默认值）

明文提交，待提交的数据会显示在浏览器的地址栏上。

提交数据有大小限制，最大2kb。

向服务器要数据时使用。

post

隐式提交，提交的数据不会显示在地址栏中

没有大小限制

把数据传递给服务器时使用

delete

put

（3）enctype：指定表单数据的编码方式，允许将什么格式的数据提交个服务器。

取值：text/plain（默认值）：只能将普通字符提交给服务器（不能带特殊符号，比如说 &就是特殊符号）。

application/x-www-form-urlencoded：允许将任意字符提交给服务器（不能提交文件）。

multipart/form-data：允许提交文件。

**4.表单控件**

能够与用户进行交互的可视化元素

**4.1 分类：**

input 元素

textarea 多行文本域

select和option 下拉选择框

其他元素

**4.2 input 元素**

在页面中提供各式各样的输入控件

<input type="控件类型" name="uid"> input除了单选和多选，都是行内块元素。

属性 type：指定输入控件的类型。

name：为控件定义名称，必须写，name的值是提供给服务器使用的，与服务器对应。

value：控件提交的值，提交给服务器使用。

disabled：禁用，无值属性，提交的时候，该控件不提交。

取值：disabled、disabled="disabled"、disabled="true"

**4.3 input详解（type的9个属性值——重点）**

**（1）文本框和密码框**

文本框：type="text"

密码框：type="password"

属性：maxlength：指定限制输入的最大字符数，无单位数字。

readonly：只读，无值属性，只能看不能写，但是允许被提交。

placeholder：占位符，默认显示在空间上做提示的文本

**（2）按钮**

提交按钮 type="submit"

自带提交功能，将表单中的数据收集整理，提交个服务器。

属性 value：定义按钮上显示的文字

重置按钮 type="reset"

将表单内容恢复到初始化状态

属性 value：定义按钮上显示的文字

普通按钮 type="button"

通过事件，调用js代码

属性 value：定义按钮上显示的文字

提交按钮

<button></button> 默认有提交功能（HTML5中新增）

设置button为普通按钮：type="button"

**（3）单选按钮和复选按钮**

单选按钮 type="radio"

属性 name：除了作为提交时，等号左边内容，还起到分组作用。

value：单选按钮想要提交正确的值，需要在value中定义，不写都提交的是on。

checked：设置默认被选中，无值属性

复选框 type="checkbox"

属性 name：用来分组，name做分组的时候，为了存储多个数据，要使用数组hobby[]。

value：用来赋值

**（4）隐藏域——**想提交数据，但是不想展示给用户看

type="hidden"

<input type="hidden" name="id" value="10001">

**（5）文件选择器**

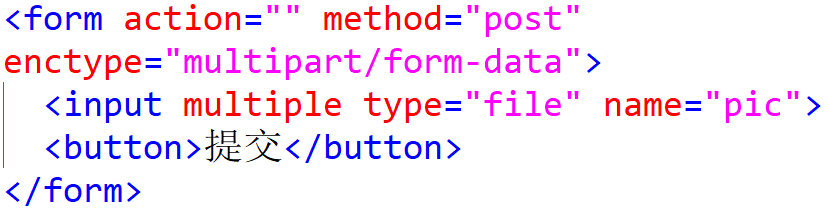
type="file"

<input multiple type="file" name="pic">

注意：需要form表单<form action="" method="post" enctype="multipart/form-data">

属性 multiple：无值属性，可同时上传多个文件，name用数组。

如图：



**4.4 文本域——**允许录入多行数据的大文本框

<textarea name="addr" cols="10" rows="5"></textarea>

属性 name：

cols：指定文本域列数

rows：指定文本域行数

readonly：

**4.5 下拉选择框**

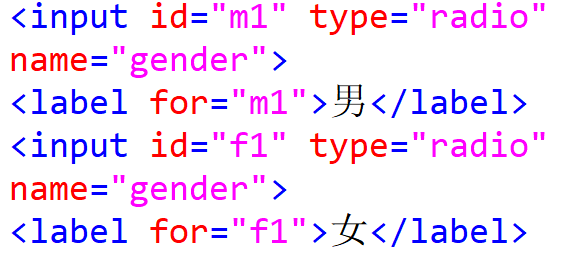
|  |
| --- |
| <span>晚上吃的菜</span>  <select name="tname">  <option>回锅肉</option>  <option value="hg">火锅</option>  </select>  name写在select中  **（1）提交的值分两种情况：**  option没有写value，那么select的value就是option当中的内容。  option有写value，那么select的value就是option的value值。  **（2）select属性**  size：设置显示option的个数，默认值是1个，下拉选择列表；如果设置值大于1，显示形式变成滚动选择列表。  multiple：设置多选，如果使用multiple，需要name设置是数组。  **（3）option属性**  selected：设定默认选中 |

**4.6 其他元素**

**（1）label关联文本与表单控件**

<label>需要关联的文本</label>，可以代替span使用

属性 for：用于需要绑定控件id

如图：

**（2）为控件分组**

|  |
| --- |
| <fieldset> 为控件分组  <legend></legend> 为分组指定名称  </fieldset> |

**（3）浮动框架**——在一个网页中**引入另一个网页**，实现代码重用

|  |
| --- |
| <iframe src="03\_form.html" frameborder="0" width="500px" height="500px"></iframe> |

src：要引入的网页.html路径

属性：width/height

framborder：浮动框架的边框，默认为1（不写时）

**4.7 新表单元素（10个）**

HTML5新版中新提出的表单控件

（1）电子邮件

<input type="email">

提交时，会验证数据是否符合email的格式规范，并且@后面必须有字符。

（2）搜索类型

<input type="search">

增加了快速删除功能

（3）URL类型

<input type="url">

提交时，验证数据是否符合url的格式（http）

（4）电话号码

<input type="tel">

在移动设备中，显示拨号键盘

（5）数字类型

<input type="number" value="10" max="20" min="5" step="5">

只能接收数字，同时能够灵活调整数字

属性 value：控制的值

max：接受的最大值

min：接受的最小值

step：每次调整数字时，数字的变化

（6）范围类型

提供了一个滑块组件，允许用户在范围内选取值。

<input type="range" max="100" min="50" name="ran" step="5">

属性：min/man/step/value初始值

（7）颜色类型

提供了一个颜色的拾取器

<input type="color">

（8）日期类型

<input type="date">

（9）月份类型

<input type="month">

（10）周类型

<input type="week">

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <script>  function show(){alert(111111)}  </script>  </head>  <body>  <!--下拉选择框 -->  <form action="">  <span>晚上翻谁的牌子</span>  <select name="tname[]" size="2" multiple>  <option value="dc">貂蝉</option>  <option value="wzj">王昭君</option>  <option selected>杨贵妃</option>  <option>西施</option>  </select>  <button>提交</button>  </form>  <hr>  <!-- 文本域 -->  <form action="">  <textarea name="addr" cols="10" rows="5"></textarea>  <button>提交</button>  </form>  <hr>  <!--隐藏域，可以提交数据，但是不让用户看见 -->  <form action="" method="post" enctype="multipart/form-data">  <input type="hidden" name="id" value="10001">  <input multiple type="file" name="pic">  <button>提交</button>  </form>  <hr>  <!-- 单选和复选 -->  <form action="">  <input type="radio" name="gender" value="1">男  <input type="radio" name="gender" value="0">女  <input type="radio" name="gender" value="x" checked>随便  <button>提交</button>  </form>  <hr>  <!--复选框 -->  <form action="">  <input type="checkbox" name="hobby[]" value="eat">吃  <input type="checkbox" name="hobby[]" value="drink">喝  <input type="checkbox" name="hobby[]" value="play">玩  <input type="checkbox" name="hobby[]" value="happy">乐  <button>提交</button>  </form>  <hr>  <!--文本框和按钮 -->  <form action="/list">  <input type="text" name="uname" placeholder="请输入您的用户名"><br>  <input type="password" name="upwd" maxlength="6"><br>  <input type="submit" value="登录"><br>  <input type="reset" value="复原"><br>  <input onclick="show()" type="button" value="点点我试试"><br>  <button type="button">提交</button>  </form>  </body>  </html> |

**一．HTTP协议**

**1.URL**

完整结构：

<scheme>://<user>:<pwd>@<host>:<port>/<path>;<params>?<query>#<frag>

scheme：方案、协议，以哪种方式获取服务器资源

不区分大小写，常见的协议：http/https/ftp

常见的协议和默认端口号：(端口总个数2^16,六万多个)

|  |  |  |
| --- | --- | --- |
| 常见的协议 | 默认端口号 | 协议的基本作用 |
| ftp | 21 | 负责文件的上传下载 |
| ssh | 22 | 安全的远程登录 |
| telnet | 23 | 远程登录 |
| smtp | 25 | 邮件的传输 |
| dns | 53 | 域名解析 |
| http | 80 | 超文本传输 |
| pop3 | 110 | 邮件的接收 |
| https | 443 | 加密传输的http |

user：用户名

pwd： 密码

host：主机名称 127.0.0.1、localhost

port：端口，每一项服务都在服务器上对一个监听的端口号。

path：路径，服务器上资源的本地文件夹名称。

params：参数，某些方案会要求带参数，其中session存储登录状态/cookie存储用户名和密码。

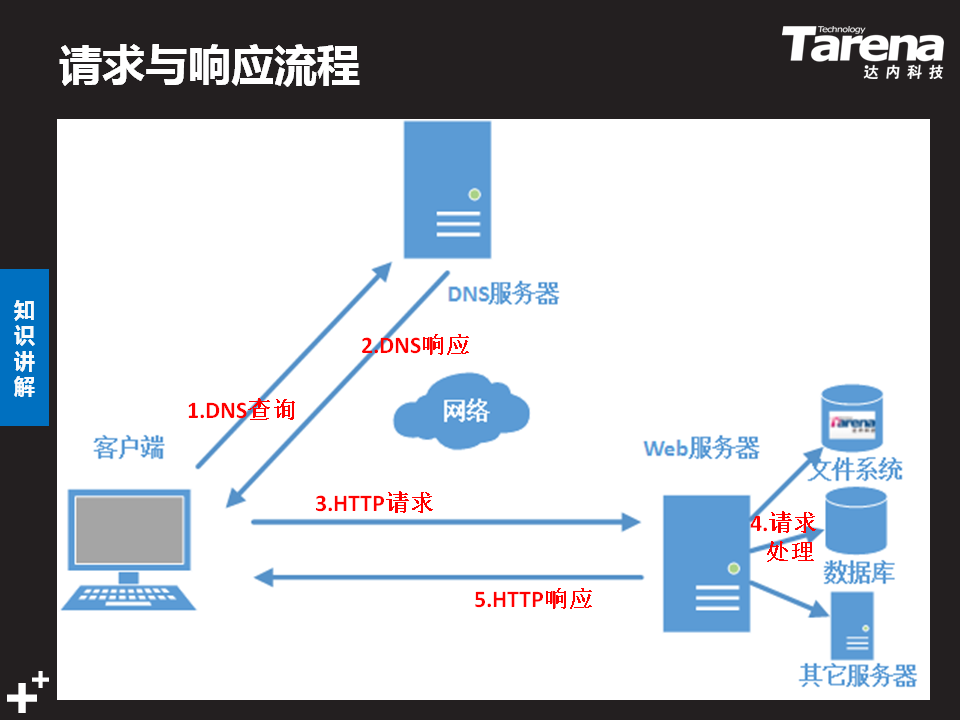
query：查询字符串。形式->?名称=值&名称=值

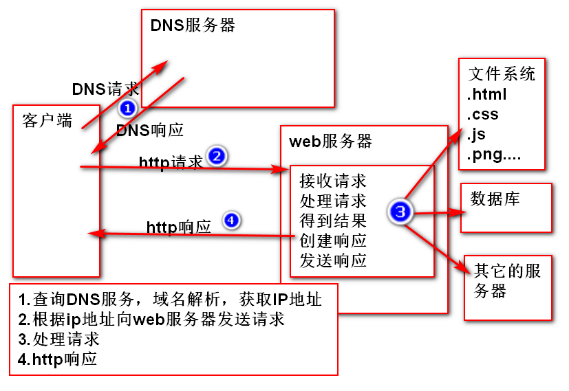
frag： 锚点

**2.HTTP协议**

HTTP：Hyper Text Transfer Protocol超文本传输协议

规范了数据是如何打包和传递的

 正在使用的版本：http/1.1

**3.详解**

**4.请求消息：Request Message**

客户端（浏览器）发送给服务器的数据

由三部分组成：请求起始行，请求头，请求主体

**4.1 请求起始行**

（1）请求的方法

get：客户端向服务器要数据的时候使用。

特点：靠地址栏传递查询字符串，大小限制2kb，无请求主体。

post：客户端向服务器提交数据时使用。

特点：靠请求主体传递数据

delete：表示客户端可以删除服务器的内容（一般禁用）。

put：客户端可以存放数据（文件）到服务器（一般禁用）。

connect：测试连接

trace：追踪请求路径

option：选项，预请求

head：客户端只获取响应的消息头

（2）请求的URL

（3）协议的版本号

http/1.1

**4.2 请求头（Request Header）**

（1）Host：www.tmooc.cn

告诉服务器，客户端请求的是哪个主机。

（2）Connection：keep-alive

告诉服务器，进行持久连接。

（3）User-Agent（用户代理）

告诉服务器自己（浏览器）的信息。

（4）Accept-Encoding

告诉服务器，自己能够接收的压缩文件类型。

（5）Accept-Language：zh-CN,zh;q=0.9

告诉服务器，自己能够接收的语言类型。

（6）Referer（引用页）：http://www.tmooc.cn/

告诉服务器，请求来自于哪个网页。

**4.3 请求主体（Form Data）**

存储参数，get无请求主体，post靠请求主体传递数据。

**5.响应消息：Response Message**

服务器发送给客户端的数据

由三部分组成：响应起始行，响应头，响应主体

**5.1 响应起始行**

（1）协议版本号 http/1.1

（2）响应状态码 200

告诉浏览器，服务器的响应状态是什么。

1XX：100~199 提示信息

2XX：成功响应（200 OK）

3XX：301 永久重定向

302 临时重定向

304 请求未被修改，命中了缓存

4XX：404 请求资源不存在（NOT FOUND）

403 权限不够（Forbidden）

405 请求方法不允许（Method not Allowed）

5XX：500 服务器内部错误

（3）原因短句

对响应状态码的简短解释说明。

**5.2 响应头**

（1）Date：Wed, 30 Jan 2019 07:15:38 GMT

格林威治时间：1970.1.1 00:00:00 +8（中国东8区）

（2）Connection：keep-alive

告诉浏览器，已经开启持续连接。

（3）Content-Type: text/html

告诉浏览器，响应主体的类型。

text/html： 响应回来的数据是html文本。

text/css： 响应回来的数据是css样式文件。

text/plain：响应回来的数据是普通字符。

application/javascript：响应回来的数据是js代码。

application/xml： 响应回来的数据是xml格式的字符串。

application/json：响应回来的数据是json格式的字符串。

image/png…jpg：响应回来的数据是图片。

（3）Content-Encoding: gzip

告诉浏览器，响应回来文件压缩类型。

**5.3 响应主体**

响应回来的数据（Rsponse中）

**6.缓存**

客户端将服务器响应回来的数据进行自动保存，

当再次访问时，直接使用保存的数据。

**（1）缓存优点：**

减少冗余的数据传输，节省了客户端流量；

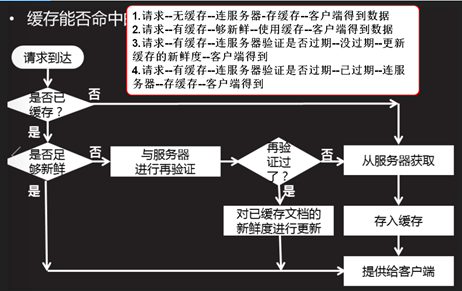
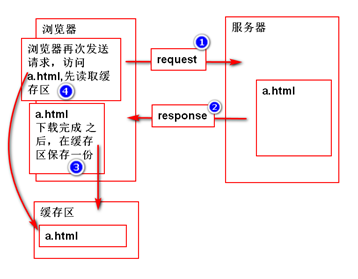
节省了服务器宽带；

降低了对服务器资源的消耗和运行要求；

降低了由于远程距离传输造成的延时加载。

**（2）缓存的具体工作流程**

如图：



**（3）与缓存相关的信息头**

http/1.1的用法

Cache-Control：max-age=处于新鲜的秒数

取值 0： 不缓存

3600：缓存的新鲜秒数为1小时

从服务器将文档传到客户端之时起，此文档处于新鲜的秒数，这是一个相对值。

Expires http/1.0

指定过期的确切时间点，是一个具体的时间点。

expires：Tue,31 Jul 2019 02:00:00 GMT

一般情况，使用Cache-Control的相对时间

**（4）在网页中添加缓存（前端）**

<meta http-equiv="信息头" content="值">

例如：<meta http-equiv="Cache-Control" content="3600">

**7.HTTP性能优化**

**（1）http连接过程**

发送请求->建立连接->处理请求->访问资源->构建响应->发送响应->记录日志

**（2）http连接性能优化**

减少连接创建的次数（开启持久连接）；

减少请求的次数；

提高服务器端的运行速度；

尽可能的减少响应数据的长度；

**（3）安全的http协议——https**

https：安全版的http

SSL：微数据通讯提供安全支持

客户端发送请求信息时，在SSL层加密；

服务器端接收到的加密文件，在SSL层解密；

得到请求明文，对请求进行处理。

服务器发送响应数据，在SSL层加密；

客户端收到加密文件，在SSL层解密；

得到响应明文，解析响应内容。

**二．DOM操作**

学习简易的DOM

目的：获得控件对象，得到控件中的数据；

获得控件对象，把数据在控件中显示。

**1.完成的JavaScript的组成**

（1）js核心 ECMA Script

（2）DOM：Document Object Model 文档对象模型

（3）BOM：Browser Object Model 浏览器对象模型

**2．使用js获取页面的某个元素**

var elem=document.getElementById("id值");

|  |
| --- |
| <div id="d1">  <p>这是DOM的测试文本</p>  </div>  <script>  //1.使用DOM获取div对象  var elem=document.getElementById("d1");  console.log(elem);  -><p>这是DOM的测试文本</p>  //2.获取对象中的数据（div中间的文档）  console.log(elem.innerHTML);  ->这是DOM的测试文本  //3.把数据放到对象中显示  //elem.innerHTML="aaaaa";  elem.innerHTML="<h1>aaaaa</h1>";  </script> |

H5为我们简化了上面写法，当元素有id时，可以简化为：

直接使用元素的id值，当做元素对象（div中间的内容，相当于上面的elem的内容。）

|  |
| --- |
| <body>  <div id="d2">H5简化了getElementById</div>  <script>  //使用id简写获取对象  console.log(d2.innerHTML);  </script>  </body> |

**3.通过对象获取元素的内容，只有双标签才有内容**

获取：console.log(elem.innerHTML);

赋值：elem.innerHTML="新的值";

注意：如果新的值中有正确的HTML标签，此标签会被浏览器渲染显示。

|  |
| --- |
| <script>  var names=["hanmeimei","lilei","tom"];  var age=[19,20,21];  var gender=["女","男","男"];  var html="<table border='1px'><tr><th>姓名</th><th>年龄</th><th>性别</th></tr>";  for(var i=0;i<age.length;i++){  html+="<tr>";  html+="<td>"+names[i]+"</td>";  html+="<td>"+age[i]+"</td>";  html+="<td>"+gender[i]+"</td>";  html+="</tr>";  }  html+="</table>";  d1.innerHTML=html;  </script> |

**4.获取单标签的值（value）**

|  |
| --- |
| <body>  <input type="text" id="t1" value="abc">  <br>  <script>  //改变input的值  t1.value="123";  //获取input的值  console.log(t1.value);  </script>  </body> |

**5.事件**

允许通过用户的行为激发的操作，就是事件。

onclick：点击button就会激发事件

|  |
| --- |
| <body>  <!-- 将input输入内容显示在div中 -->  <input type="text" id="msg1">  <div id="msg2">显示数据</div>  <button onclick="show()">显示</button>  <script>  function show(){  msg2.innerHTML=msg1.value; //msg2是双标签，获取数据使用innerHTML，msg1是单标签，获取数据使用value。  }  </script>  </body> |

6.**其他事件**

（1）文本框或者密码框，获得焦点事件 onfocus

（2）文本框或者密码框，失去焦点事件 onblur

（3）body的事件，onload 当页面一加载就要执行

|  |
| --- |
| <body>  <div>  用户名：<input type="text" id="uname" onfocus="show\_uname()" onblur="check\_uname()">  <span id="uname\_msg"></span>  </div>  <div>  密&nbsp;码：<input type="password" id="upwd" onfocus="show\_upwd()" onblur="check\_upwd()">  <span id="upwd\_msg"></span>  </div>  <script>  function show\_uname(){  //用户名获得焦点  uname\_msg.innerHTML="用户名3~12位";  }  </script>  <script>  function show\_upwd(){  upwd\_msg.innerHTML="密码名6~12位";  }  </script>  <script>  //用户名失去焦点，验证用户名是否为空  function check\_uname(){  //1.得到用户框的值  var userName=uname.value;  //2.验证用户框中值是否为空  if(userName==''){  uname\_msg.innerHTML="用户名不能为空";  }else{  uname\_msg.innerHTML="通过";  }  }  </script>  <script>  function check\_upwd(){  var userPwd=upwd.value;  if(userPwd==''){  upwd\_msg.innerHTML="密码名不能为空";  }else{  upwd\_msg.innerHTML="通过";  }  }  </script>  </body> |

**一．Ajax**

**1.同步和异步**

1.1 同步 Synchronous

在一个任务进行中时，不能开启其他的任务。

同步访问：浏览器在向服务器发送请求时，浏览器只能等待服务器的响应，不能做其他事情。

出现的场合：

（1）地址栏输入网址，访问网页。

（2）a标签的跳转。

（3）form，submit的提交。

1.2 异步 Asynchronous

在一个任务进行中时，可以开启其他任务。

异步访问：浏览器在向服务器发送请求时，不耽误用户在网页上的其他操作。

出现的场合：

（1）用户名验证

（2）聊天室

（3）搜索建议

（4）股票走势图

**2.什么是Ajax**

Asynchronous Javascript And XML

ajax本质：

使用js提供的异步对象（XMLHttpRequest） xhr—记忆小黄人

异步的向服务器发送请求

并接收响应回来的数据（是部分数据，不是完整的页面）

ajax，可以无刷新的效果更改页面的布局内容

ajax原理：

相当于在用户和服务器之间添加了一个中间层，使用户操作与服务器响应异步化，把以前的一些服务器负担的工作转嫁到客户端，利用客户端闲置的处理能力来处理问题。

**3.异步ajax请求的步骤（4步）**

（1）创建异步对象

（2）绑定监听事件（接收相应）

（3）创建请求（打开链接）

（4）发送请求

**3.1 创建异步对象（XMLHttpRquest）**

（1）标准创建异步对象

var xhr=new XMLHttpRequest();

（2）IE8以下版本的创建方式(了解)

var xhr=new ActiceXObject('Microsoft.XHMHttp');

|  |
| --- |
| <body>  <script>  //1.标准创建异步对象  var xhr=new XMLHttpRequest();  console.log(xhr);  //IE8以下版本的创建方式(了解)  var xhr=ActiceXObject('Microsoft.XHMHttp');  //兼容性写法，需要判断window.XMLHttpRequest是否有值，如果有值就用标准创建，如果没值(null)就用IE8版本以下创建。  if(window.XMLHttpRequest){  var xhr=new XMLHttpRequest();  }else{  var xhr=new ActiceXObject('Microsoft.XHMHttp');  };  </script>  </body>  课程只需要记住标准创建。 |

**3.2 打开连接，创建请求**

xhr.open("method","接口",isAsyn);

method：请求方式，string类型。 "get" "post"

url：请求地址，string类型，接口

isAsyn：boolean是否要使用异步处理，true（使用异步访问），false（使用同步访问）

**3.3 发送请求**

xhr.send(body);

body：请求主体 form data

如果没有请求主体，使用null，或者不写——get提交。

**3.4 绑定监听事件（接收相应）**

（1）接收响应数据，使用xhr.onreadystatechange事件，监听xhr.readyState属性的值，每变化一次，此事件就会被调用一次，调用4次（0->1,1->2,2->3,3->4）。

（2）xhr.readyState属性

用于表示xhr对象的请求状态，5个请求状态

0：请求尚未初始化（3.1步）

1：已经打开服务器连接，正在发送请求（3.3步）

2：接收响应头

3：接收响应主体

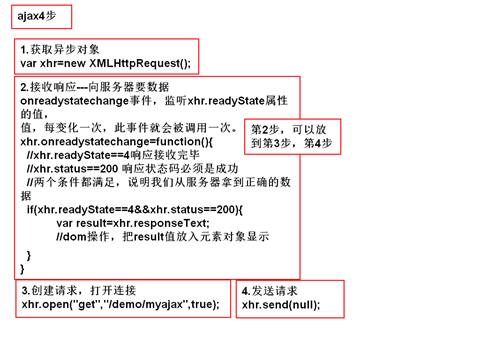
4：接收响应数据成功（响应接收完毕）

（3）xhr.status 响应的状态码（需要成功状态的状态码200）

（4）xhr.responseText 服务器响应的数据

xhr.getAllResponseHeaders()->获取所有响应头信息

xhr.getResponseHeader(响应头名称)->获取对应的响应头

示图：

常见错误：

|  |
| --- |
| 跨域错误，html文件是本地打开，不是从服务器打开 |
| 404：接口写错了，或者资源文件路径错了 |
| 服务器代码写错 500的错误 |

示例：

|  |
| --- |
| app.js |
| //引入中间件  const bodyParser=require('body-parser');  //创建web服务器  const express=require('express');  //引入demo路由  const demo=require('./routes/demo.js');  var server=express();  server.listen(8080);  //托管静态资源到myex下  server.use(express.static('myex'));  //将demo路由挂在到/demo下  server.use('/demo',demo); |
| demo.js |
| const express=require('express');  //使用express创建路由器  var router=express.Router();  //1.创建路由myajax  router.get('/myajax',(req,res)=>{  res.send('这是我第一个ajax程序');  });  //把路由器对象导出  module.exports=router; |
| 02.html |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  </head>  <body>  //点击"显示"会在div中显示响应的内容  <button onclick="show()">显示</button>  <div id="d1"></div>  <script>  function show(){  //1.创建异步对象  var xhr=new XMLHttpRequest();  //2.绑定监听器  xhr.onreadystatechange=function(){  // xhr.readyState==4 意味着接收响应成功  // xhr.status==200 意味着响应数据成功  if(xhr.readyState==4 && xhr.status==200){  var result=xhr.responseText;  console.log(result);  d1.innerHTML=result;  }  }  //3.打开连接（创建请求）  xhr.open("get","/demo/myajax",true);  //4.发送请求  xhr.send(null);  }  </script>  </body>  </html> |

**post方法请求**

|  |
| --- |
| <body>  <div>  用户名称：<input type="text" id="uname">  用户密码：<input type="password" id="upwd">  </div>  <button onclick="login()">登录</button>  <script>  function login(){  var xhr=new XMLHttpRequest();  xhr.onreadystatechange=function(){  if(xhr.readyState==4 && xhr.status==200){  var result=xhr.responseText;  alert(result);  }  }  xhr.open("post","/demo/loginpost",true);  //由于默认的请求头中，设置请求主体内容必须为普通字符（不带特殊符号）  //但是传参主题中有特殊符号  //我们要重新设置请求内容的类型  xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  //请求主体  var formdata="uname="+uname.value+"&upwd="+upwd.value;  console.log(formdata);  xhr.send(formdata);  }  </script>  </body> |

**封装ajax**

|  |
| --- |
| function ajax({url,type,data,dataType}){  return new Promise(function(open,err){  //1. 创建xhr对象  var xhr=new XMLHttpRequest();  //2.绑定监听事件  xhr.onreadystatechange=function(){  if(xhr.readyState==4&&xhr.status==200){  if(dataType!==undefined  &&dataType.toLowerCase()==="json")  var res=JSON.parse(xhr.responseText)  else  var res=xhr.responseText    open(res);  }  }  if(type.toLowerCase()=="get"&&data!=undefined){  url+="?"+data;  }  //3.打开连接  xhr.open(type,url,true);  if(type.toLowerCase()==="post")  //增加：设置请求消息头  xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  //4.发送请求  if(type.toLowerCase()=="post"&&data!==undefined)  xhr.send(data);  else  xhr.send(null);  })  }  /\*  (async function(){  //ES7  var res=await ajax({  url:"http://localhost:3000/index/",  type:"get",  dataType:"json"  })  //ES6  .then(res=>{  ... ...  })  })();  \*/ |

**一、JSON数据类型**

1．js对象的数据格式

var 对象名称={

属性1：值1,

属性2：值2,

属性3：值3

}

2.JSON的数据格式

JSON JavaScript object notation

js 对象 表现方式

js对象表示法：以js对象的数据表现出来的字符串。

JSON是字符串，不能直接使用js代码操作。

3.JSON的语法

（1）json中用一对{}来表示对象；

（2）对象中的属性名称，必须使用""括起来（不要使用单引号），

值如果是字符串，也必须使用""括起来。

（3）json是一个字符串，最外层加单引号。

|  |
| --- |
| 将JSON字符串转换成js对象/数组  服务器的响应，把数据自动的装换成了json字符串  我们在接收到响应后，需要把json字符串解析成js对象/数组  JSON.parse(result) |

**二．XML**

**1.什么是XML**

eXtensible Markup Language

可拓展 标记 语言

xml的标签：是没有预定义过的，需要自己定义

xml的属性：是没有预定义过的，需要自己定义

xml的作用：用作数据传递的，而不是数据展示的

**2.语法规范**

（1）XML的最顶端做声明

<?xml version="1.0" encoding="utf-8"?>

（2）XML的标记，必须成对出现，没有单标记

<name></name> 正确

<name> 错误写法

（3）XML标记严格区分大小写

<Name></Nname> 正确

<Name><name> 错误写法

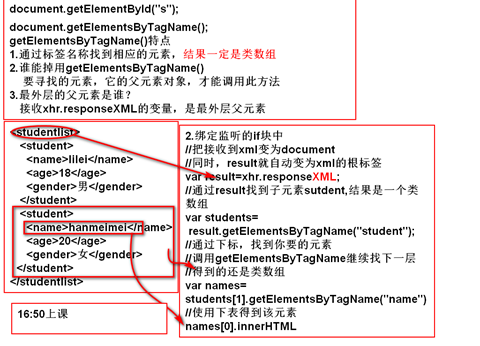
（4）XML允许嵌套

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <studentlist>  <sutdent>  <name>lilei</name>  <age>18</age>  <gender>男</gender>  </sutdent>  </studentlist> |

（5）每个标记允许自定义属性，格式与HTML小童，但是必须有双引号

<name num="10"></name>

（6）每个XML文件，有且只有一个根元素

3. 使用ajax请求student.xml

**一．CSS3概述**

1.什么是css？

Cascading Style Sheels

层叠样式表，级联演示表，简称样式表。

2.作用

设置html网页中元素的样式。

3.HTML和CSS的关系

html负责网页的搭建，内容的展示。

css负责网页的修饰，样式的构建。

4.CSS与HTML属性的使用原则

W3C建议我们尽量使用css的方式取代html属性。

CSS样式：

（1）达到演示代码的高度重用

（2）方便后期维护，提高了可维护性

**二．CSS的语法规范**

**1.使用CSS样式的方式**

**（1）行内样式（内联样式）**

将css样式定义在HTML开始标记中

<any style="样式的声明"></any>

样式声明的格式：样式属性:样式的值;样式属性:样式值;…

常用的样式属性

.设置文本颜色

color：合法的颜色值

.设置字体大小

font-size：以px为单位的数字

.设置背景颜色

background：合法的颜色

内联样式在项目中基本不用，样式代码重用性低，内联样式优先级最高，只在学习和测试中使用。

**（2）内部样式**

在网页头部中添加<style></style>标签

在style中定义样式

<head>

<style>

样式规则

</style>

</head>

样式规则：选择器{样式声明}

**选择器**：规范了网页中哪些元素能够使用此样式（条件）

内部样式，只能在同一个HTML中重用样式代码，项目中使用较少，学习和测试中使用较多。

**（3）外部样式**

独立于任何网页的位置处，创建一个样式文件（css为后缀），在css文件中保存样式规则。

<link rel="stylesheet" href="css文件的url">

使用步骤：

创建样式表文件

在css样式表文件中编写样式规则

在网页的头部引入css文件

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <!-- 内部样式 -->  <style>  /\*样式规则 选择器{样式声明}\*/  h3{color:blue !important;font-size:40px}  </style>  <!-- 引入外部样式 -->  <link rel="stylesheet" href="mydemo.css">  </head>  <body>  <!-- 外部样式 -->  <h4>茕茕孑立</h4>  <!-- 内部样式 -->  <h3>苍茫的天涯是我的爱</h3>  <h3>万水千山花正开</h3>  <!-- 内联样式 -->  <p style="color:red;font-size:30px;background:pink">杨家小树林</p>  <h2 style="color:red;font-size:30px;background:yellow">TF小白条</h2>  </body>  </html> |

**2.CSS样式特征**

**（1）继承性**

大部分的css效果可以被直接继承的，

必须是层级嵌套结构，子继承父或者更高层。

**（2）层叠性**

可以为一个元素定义多个样式规则，规则中属性不冲突，可以同时引用到当前元素上。

**（3）优先级**

如果样式声明冲突时，按照样式规则的优先级去应用。

默认的优先级，由高往低：

内联样式

就近优先（内部样式和外联样式），离标签最近的优先

浏览器默认（缺省）样式

**（4）调整优先级**

!important

放在样式属性的值之后，与值用空格隔开。

示例：h2{color:blue !important;}

作用：调整样式的优先级

**三．基础选择器（重点）**

**1.选择器的作用**

规范了页面中的哪些元素能够使用定义好的样式。

**2.选择器详解**

**（1）通用选择器**

\*{样式声明}

可以与网页中所有元素匹配

\*效率极低，不建议使用

\*{margin:0;padding:0} 所有内外边距清零（0可以省略单位px）

**（2）元素选择器（标签选择器）**

设置页面中某种元素的默认样式

标记名称{样式声明}

body{color:#666}

**（3）类选择器**

定义页面上某个或者某类元素的样式（公共的样式，谁想用谁就能用的样式）。

通过元素的class属性来引用

语法：

声明

.类名{样式声明}

点不能省略。

类名的命名规则：类名不能以数字开头；不能包含特殊字符（除了-,\_）。

使用

<any class="类名"></any> 此处没有点

* 特殊用法

多类选择器

让元素引用多个类型选择器

<any class="类名1 类名2 类名3 …"></any>

分类选择器

将元素选择器和类选择器联合使用。

语法：元素选择器.类选择器{样式声明}。

符合元素选择器和类选择器两个条件的标签，才算匹配。

特点：

a．指向更精准；

b．优先级权值更高。

**（4）id选择器**

设置指定id元素的样式（专属定制）

语法：#id值{样式声明}

类选择器的权值比id选择器权值低，所以id选择器优先级高。

示例：

<style>

#s1{background:pink}

</style>

<span id="s1">踽踽独行</span>

**（5）群组选择器**

将多个选择器放在一起定义公共的样式。

语法：选择器1,选择器2,选择器3,…{样式声明}

示例：div,#s1, .bg-danger{background:red;}（相当于把每个选择器分开一样）

**（6）后代选择器（找后代）**

通过元素的后代关系匹配元素

后代：一级或者多于一级的嵌套。

语法：选择器1 选择器2 选择器3 …{样式声明}

示例：div p span{color:purple;}

**（7）子代选择器（找儿子）**

通过元素的子代关系匹配元素

子代：一级嵌套

语法：选择器1>选择器2>选择器3>…{样式声明}

示例：div>p>span{color:purple;}

**（8）伪类选择器**

用于匹配元素的不同状态下的样式

所有的伪类选择器都是以冒号(:)开头。

选择器:伪类{样式声明}

链接伪类

<a id="a1" href="http://www.tmooc.cn">tmooc</a>

#a1:link{color:red;} 匹配元素尚未访问时的状态

#a1:visited{color:green;} 匹配元素访问后的状态

动态伪类

<img src="image/01.png" alt="">

img:hover{height:100px;width:256px;} 匹配鼠标悬停时状态

#a1:active{color:orange;} 匹配元素激活时的状态

如果4个伪类作用到同一个a标签上，会发生冲突，必须按照一定的顺序编写4个伪类，爱恨(love hate)原则：

L(link)OV(visited)E H(hover)A(active)TE

input:focus{} 匹配元素获取焦点时的状态

**（9）选择器权值**

标识当前选择器的重要程度，权值越大，优先级越高。

|  |
| --- |
| !important >1000  内联样式 1000  id选择器 100  类和伪类选择器 10  元素选择器 1  通用选择器 0  继承的样式 无 |

总结：

当一个选择器中含有多个选择器，需要将所有的选择器的权值进行相加，然后进行比较，权值大的优先级高。

分组选择器，权值单独计算，不相加。

如果样式后面，添加了!important，该元素获取最高优先级。内联样式不能加!important。

权值相同，就近原则。

选择器权值计算，不会超过其最大的数量级（1万个元素选择器权值相加，也不会大于10）。

示例：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  #a1:link{color:black;text-decoration:none;}  #a1:visited{color:orange;}  #a1:hover{color:red;}  #a1:active{color:green;}  input{color:gray;font-style:italic;}  input:active{font-style:normal;}  input:focus{color:red;}  div .c2 #d3{color:red;}/\*1+10+100=111权值相加\*/  div #d3 span{color:green;}/\*1+100+1=102权值相加\*/  </style>  </head>  <body>  <div class="c1" id="d1">  <p class="c2" id="d2">  <span class="c3" id="d3">ssssssssssss</span>  </p>  </div>  <input type="text">  <a id="a1" href="http://www.tmooc.cn">无名之辈</a>  </body>  </html> |

**四．尺寸和边框**

**1.尺寸属性**

**（1）作用**：改变元素的宽度和高度

**（2）属性**

宽度：width

最大宽度：max-width 元素宽度随着屏幕变化而变化，但是不会超过某个数值。

最小宽度：min-width 元素宽度随着屏幕变化而变化，但是不会小于某个数值。 高度：hight

最大高度：max-height

最小高度：min-height

取值px为单位的数字（学习中这么写）

**关于不同元素的宽高解释**

|  |
| --- |
| 块级元素：  不写宽，默认占父元素的100%。  不写高，默认自适应内部内容。  行内元素：  宽高不生效，默认自适应内部内容。  总结：想要设置宽高，元素必须是非行内元素。 |

**2.长度单位详解**

（1）像素px

（2）in 英寸 1in=2.54cm

（3）pt 磅值 1pt=1/72in 多用于表示字体大小

（4）cm 厘米

（5）mm 毫米

（6）em 相对于父元素数值乘以的倍数(2em就是父元素数值乘2)

（7）rem 相对于html标签的数值乘以的倍数

（8）% 相对父元素百分比

**3.页面中允许设置尺寸的元素**

（1）所有的块级元素都允许设置尺寸。

（2）行内块元素都允许设置尺寸，input都属于行内块（单选和复选框除外）。

（3）大部分行内元素不允许设置宽高，本身具有width/height属性的元素，可以设置，如：img，table。

**4.颜色单位（合法颜色值）**

（1）颜色的英文单词（red、green、blue、yellow、…）

（2）以#开头，6位16进制（#006699）

#00 66 99

red green blue——>最大f为白，最小0为黑

转为十进制：00->0\*16^1=0 + 0\*16^0=0 = 0

66->6\*16^1=0 + 6\*16^0=0 = 102

99->9\*16^1=0 + 9\*16^0=0 = 153

（3）如果#aabbcc——>简写为#abc

（4）rgb(r,g,b);——>r,g,b取值0~255

（5）rgb(r%,g%,b%);——>0~255百分比

（6）rgba(r,g,b,alpha)——>alpha:透明度 取值：0~1

**5.溢出的处理**

内容过多，元素区域小的时候，就会产生溢出效果。

H5/C3默认纵向溢出

属性：overflow/overflow-x水平方向/over-flow-y垂直方向

取值：visible 默认值 可见的 溢出可见

hidden 溢出部分隐藏

scroll 不管是否溢出，都有滚动条，溢出方向滚动条可用

auto 溢出的方向才有滚动条

**6.边框**

**（1）边框属性——简写方式**

语法：border:width style color;

width：边框的宽度，以px单位的数字

style：边框的样式

取值：solid 实线

dotted 虚线（点）

dashed 虚线（线）

double 双实线

color：边框的颜色，合法的颜色值/transparent 透明

注意：简写方式设置4条边框

取消边框：border:none;/border:0;

|  |
| --- |
| 快捷键：  div# container直接按Tab键  div.c1直接按Tab键 |

**（2）单边定义，单独设置一条边的三个属性**

border-top:width style color;

border-right:width style color;

border-bottom:width style color;

border-left:width style color;

**（3）单属性设置，必须设置style**

border-width:值; 不写，默认父元素

border-style:值; 必须值

border-color:值; 不写，默认黑色

**（4）单边单属性设置**

border-方向-属性:值;

…（12个）

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  #d1{  width:0;  border:10px solid transparent;  border-top-color:#666;  }  #container{  width:0;  border:20px solid #f00;  border-right-color:#0f0;  border-bottom-color:#00f;  border-left-color:#0ff;  }  </style>  </head>  <body>  <div id="d1"></div>  <div id="container"></div>  </body>  </html> |

**（5）边框倒角（圆角）**

将直角设置成圆角

属性：border-radius

取值：以px/%为单位的数字——数字为半径值

50%或者半径为宽高一半时，是个圆圈

单角设置

左上角：border-top-left-radius

右上角：border-top-right-radius

右下角：border-bottom-right-radius

左下角：border-bottom-left-radius

|  |
| --- |
| #d1{  width:200px;height:200px;  border:3px solid #f00;  /\* 左上 右上 右下 左下\*/  /\* border-radius:60% 5% 60% 5%;\*/  border-top-left-radius:100%;  /\* border-top-right-radius:20%;\*/  /\* border-bottom-left-radius:20%;\*/  border-bottom-right-radius:100%;  } |

**（6）边框的阴影**

属性：box-shadow

取值：h-shadow v-shadow blur spread color inset

h-shadow：水平方向阴影的偏移，必须值

+：往右 -：往左

v-shadow：垂直方向阴影的偏移，必须值

+：往下 -：往上

blur：阴影的模糊距离，值越大越模糊，可选值

spread：阴影的尺寸，可选值

color：阴影颜色，可选值

inset：将默认的外部阴影改为内部阴影，可选值

练习：

宇智波家族标志：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  </head>  <style>  body{  background:black;  }  #d1{  width:300px;height:300px;  overflow:hidden;  background:#f00;  border-radius:50%;  /\* box-shadow:0px 170px 0px 0px #f00 inset;\*/  }  #d2{  width:260px;height:260px;  margin-top:160px;  background:#fff;  border:20px solid #000;  border-radius:50%;  }  #d3{  width:20px;height:120px;  margin-left:140px;  background:#fff;  }  </style>  <body>  <div id="d1">  <div id="d2"></div>  </div>  <div id="d3"></div>  </body>  </html> |

**（7）轮廓**

轮廓指的是边框的边框，绘制于边框外的线条。

语法：outline:width style color;

去掉轮廓：outline:0/none; ——input中常用

斜体：font-style:italic

非斜体：font-style:normal

无下划线：text-decoration:none

**一．框模型，盒子模型（重点）**

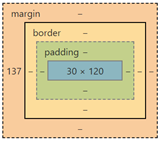
定义了元素框处理元素的内容，内边框，外边框，以及边框的一种计算方式。

计算一个元素在页面上占地大小的算法。

box model

框模型默认的计算方式：

元素实际占地宽度=左外边距+左边框+左内边距+内容区域宽度+右内边距+右边框+右外边距

 元素实际占地高度=上外边距+上边框+上内边距+内容区域高度+下内边距+下边框+下外边距

外边距：围绕在元素边框外的间距；默认不能被其他元素占用，外边距是透明的；更改元素外边距，元素会发生位移效果。

内边距：元素边框到内容区域之间的距离。

内容区域：样式中设置宽高的大小。

**1.外边距**

（1）margin：定义4个方向的外边框

属性：margin-top/right/bottom/left 定义某一个方向的外边框

取值：以px为单位的数字，%

margin-left：取值+：元素往右移动 -：元素往左移动

margin-top：取值+：元素往下移动 -：元素往上移动

auto：自动计算块级元素的外边距，控制块级元素在水平方向居中对齐。

* 简写方式：

margin:value; 设置4个方向的外边距。

margin:auto; auto对垂直外边距无效，等同于margin:0 auto;

margin:v1 v2; v1：上下外边距 v2：左右外边距。

margin:0 auto; 设置块级元素水平居中的写法。

margin:v1 v2 v3; v1：上 v2：左右 v3：下。

margin:v1 v2 v3 v4; v1：上 v2：右 v3：下 v4：左。

**（2）自带外边距的元素**

h1~h6 p ol ul dl pre body

一般开发样式第一行\*{margin:0;padding:0;}

|  |  |  |
| --- | --- | --- |
| 行内元素和块级元素区别 | | |
|  | 行内元素 | 块级元素 |
| 尺寸 | 无效 | 有效 |
| 排列方式 | 与其他行内元素共用一行 | 独占一行 |
| margin | 上下外边距无效 | 有效 |

**（3）外边距的特殊效果**

外边距合并

当两个垂直的外边距相遇时，他们将合并成一个，

最终的距离，取决于两个外边距中较大的值。

行内元素和行内块元素的外边距问题

行内元素上下边距无效。

行内块设置垂直外边距，整行元素都跟着位移。

外边距溢出

特殊情况下，为子元素设置上外边距，会作用到父元素上。

特殊情况：

a.父元素没有上边框；

b.父元素内，子元素上边与父元素上边重合时；

c.为这个子元素设置上外边距时。

解决方案：

a.为父元素添加上边框(border-top)，弊端：影响父元素实际占地高度。

b.为父元素设置上内边距，弊端：影响父元素实际占地高度。

c.在父元素第一个子元素位置处，添加一个空<table></table>

**2.内边距**

边框和内容区域之间的距离。

内边距变化，感觉是元素大小再变化，但是元素的内容区域尺寸是没有变化的。

内边距的改变会改变元素实际的占地尺寸。

**（1）语法**

padding:值; 设置4个方向的内边距

padding-top/right/bottom/left

取值：以px为单位的数据，%。注意：padding没有auto。

**（2）简写方式**

padding:v1 v2; v1：上下外边距 v2：左右外边距。

padding:0 auto; 设置块级元素水平居中的写法。

padding:v1 v2 v3; v1：上 v2：左右 v3：下。

padding:v1 v2 v3 v4; v1：上 v2：右 v3：下 v4：左。

**3.box-sizing**

指定盒子模型计算方式，不按照默认的盒子模型去计算了。

box-sizing:值;

取值：content-box; 默认的盒子模型计算方式（只设置的内容区域）。

border-box; 设置width包含：左边框+左内边距+内容区域宽度+右内边距+右边框（改变内边距，不会改变占地宽高）。

元素实际占地宽度：左外+width+右外

元素实际占地高度：上外+height+下外

**一．背景**

**1.背景颜色**

属性：background-color:合法的颜色值; transparent

注意：背景颜色和背景图默认都是从边框位置处开始填充。

**2.背景图片**

属性：background-image:

取值：url(路径);

示例：background-image:url(../image/01.png);

**3.背景图片的平铺**

属性：background-repeat:

取值：repeat; 默认平铺

no-repeat; 不平铺

repeat-x; 水平方向平铺

repeat-y; 垂直方向平铺

**4.背景图片的尺寸**

语法：background-size:width height;

取值：width height以px为单位的数字

width% height%;

cover; 覆盖，背景图完全把元素覆盖住，背景图显示不完全没关系。

contain; 包含，元素要包含完整的整张图片，元素没有被占满没关系（可能有空白区域）。

**5.背景图片的固定**

将背景图片固定在页面某个位置，一直在可视区域中显示，不会随着滚动条滚动。

属性：background-attachment:

取值：scroll; 默认

fixed; 固定

**6.背景图定位**

属性：background-position:

取值：以px为单位的数字 x y;

x% y%;

关键字 x;left/center/right

y:top/center/bottom

**7.背景的简写方式**

在属性中，指定背景的多个属性值

background：color url() repeat attachment position;

**二．渐变(gradient)**

**1.什么是渐变？**

渐变指的是多种颜色平缓变化的一种显示效果。

**2.渐变的主要因素**

色标：一种颜色，及其出现的位置

一个渐变至少两个色标

**3.渐变的分类**

（1）线性渐变，以直线的方式来填充渐变色。

（2）径向渐变，以圆形的方式实现渐变色。

（3）重复渐变，将线性渐变或者径向渐变，重复几次实现渐变。

**3.1.线性渐变**

语法：background-image:liner-gradient(方向,色标1,色标2,,…);

background-image:liner-gradient(angle,color-point1 n1,color-point2 n2,…);

（1）angle 标识渐变的方向/角度

取值：关键字

to top 从小往上

to right 从左往右

to bottom 从上往下

to left 从右往左

角度值

0deg 等同于to top

90deg 等同于to right

180deg 等同于to bottom

270deg 等同于to left

（2）color-point色标，标识颜色及其位置

合法颜色 位置;

位置：以px为单位的数字，%

**3.2.径向渐变**

语法：background-image:radial-gradient(size at position, color-point1 n1,color-point2 n2,…);

size：半径，以px为单位的数字

position：圆心所在的位置

取值：以px为单位的数字x y

x% y%;

关键字 x：left/center/right

y：top/cengter/bottom

**3.3.重复渐变**

（1）重复的线性渐变

语法：background-image:repeating-linear-gradient(angle, color-point1,color-point2,…);

（2）重复的径向渐变

语法：background-image:repeating-radial-gradient(size at position,color-point1,color-point2,…);

注意：在重复渐变中，位置一定要给绝对值数组（以px为单位的数字）

**4.关于浏览器对渐变的兼容性问题**

各个浏览器新版本都支持渐变；

对于不支持的旧版本，可以通过添加浏览器内核前缀的方式，让浏览器支持。

chrome&Safari：-webkit-

IE： -ms-

Firefox： -moz-

Oprah： -o-

注意：如果使用浏览器兼容的写法，

线性渐变的方向，需要直接写起始位置的单词，而不再是to …

示例：

background-image:-webkit-linear-gradient(left,#006699 0%,#990066 100%);

|  |
| --- |
| * **写CSS的思路**   1.布局，大概位置，尺寸（大概位置：水平居中，浮动，定位，margin）  2.背景，边框  3.文本样式，文本对齐  4.微调  注意：元素从上往下，从左往右，从外往里，一个一个元素写CSS |

**一．文本格式化（重点）**

**1.字体的属性（以font开头）**

**（1）字体大小**

属性：font-size:

取值：以px/pt为单位的数字

**（2）字体类型**

语法：font-family:黑体,mv boli;

取值，用逗号隔开多个字体类型

**（3）字体加粗**

属性：font-weight:

取值：bold;

normal;

无单位数字，必须100整倍数，400~1000

**（4）字体样式**

属性：font-style:

取值：italic; 斜体

normal; 正常

**（5）小型大写字母**

属性：font-variant:

取值：small-caps; 小型大写

normal; 正常

**（6）字体属性简写方式**

语法：font:style variant weight size family;

简写方式，最少要写size和family

font:size family;

**2.文本格式化（大部分以text开头）**

**（1）文本颜色**

color:合法的颜色值;

**（2）文本的排列（水平方向）**

属性：text-align:

取值：left/center/right/justify(两端对齐);

**（3）行高**

特点：定义一行数据的高度，如果行高的高度大于字体本身的大小时，那么该行文本将在指定的行高内，呈现垂直居中显示的效果。

作用：把行高的高度设置为与容器高度一致，文字会在容器垂直居中（只针对一行文字的情况使用）。

属性：line-height:

取值：以px为单位的数字。

无单位数字，行高是字体大小的倍数 \* 该数字。

**（4）线条修饰**

属性：text-decoration:

取值：none 无线条修饰

underline 下划线

line-through 删除线

overline 上划线

（5）首行的缩进

属性：text-indent:

取值：以px为单位的数字

（6）文本阴影

属性：text-shadow:

取值：h-shadow v-shadow blur color;

（7）超出显示省略号...

text-overflow: ellipsis;

ellipsis 当对象内文本溢出时显示省略标记（...）

(8)超出文本自动换行

word-wrap:break-word;

**二．表格**

**1.表格常用样式属性**

（1）尺寸 width/height

（2）边框 border（table最外有边框，td没有设置）

（3）边距 padding有效，td的margin无效

（4）文本格式属性

（5）背景属性

（6）vertical-align:单元数据的垂直对齐方式

取值：top/middle/bottom

注意：普通文本的垂直排序，不能使用vertical-align;

vertical-align只在table、img中使用

|  |
| --- |
| 总结：  表格是一种特殊的表现方式  尺寸是根据数据多少而决定的，如果给表格设置了宽高：  内容少，设置的的宽高放得下内容，就按照设置的尺寸显示；  内容多，设置的宽高放不下内容，按照内容的大小显示尺寸。  表格在页面中的渲染方式  整张表格读取到内存中，一次性渲染在页面上。 |

**2.表格特有的样式属性**

**（1）边框合并**

属性：border-collapse:

取值：separate 默认值，边框分离状态

collapse 边框合并状态

**（2）边框边距**

属性：border-spacing:

取值：以px为单位的数字

一个值，同时设置水平和垂直的间距。

两个值，第一个设置水平间距，第二个设置垂直间距。

注意：表格必须保持在边框分离状态border-collapse:separate;

**（3）表格标题的位置**

属性：caption-side:

取值：top; 默认值，在表格顶部

bottom; 在表格底部

**（4）设置表格的显示规则**

用来帮助浏览器决定如何布局一张表格，也就是指定td尺寸的计算方式。

属性：table-layout:

取值：auto 默认值，自动布局

td的尺寸，实际上由内容决定，内容多余，表格撑大

fixed 固定表格布局

td的尺寸，由设置的宽高为准，内容多余，放在表格外

|  |  |
| --- | --- |
| 表格自动布局 | 表格固定布局 |
| 单元格td大小，是内容决定的 | 单元格td大小，取决于设置的尺寸 |
| 缺：表格复杂时，加载速度较慢，先读取到内存中，然后一次性渲染 | 优：任何情况下都会加速加载表格 |
| 优：自动布局比较灵活 | 缺：固定布局不够灵活 |
| 适用于不确定每列大小并且不复杂的表格使用 | 适用于确定每列大小时使用 |

|  |
| --- |
| 1.设置大体位置；  2.背景，边框；  3.文字所有颜色；  4.微调。 |

**一.定位**

**1.什么是定位？**

设置元素在页面中的位置。

**2.分类**

（1）普通流定位

（2）浮动定位

（3）相对定位

（4）绝对定位

（5）固定定位

**3.普通流定位**（默认文档流定位）

不同元素显示方式不一样，但是都有默认的显示效果

块级/行内/行内块/table…

又称：默认文档流定位，页面中元素们的默认显示方式。

（1）每个元素在页面中都有自己的空间。

（2）每个元素都是从父元素的左上角开始显示/渲染。

（3）块级元素都是按照从上往下逐个排列，每个元素独占一行。

（4）行内元素是多个元素在同一行中显示，从左往右逐个排列。

**4.浮动定位（重点\*\*\*\*\*）**

**（1）特点**

元素一旦浮动，脱离默认文档流，不占页面空间，后面非浮动元素会上前补位。

浮动元素会停靠在父元素的左边或者右边，或者其他已经浮动的元素的边缘。

浮动元素解决了块级元素在同一行内显示的问题。

**（2）语法**

属性：float:

取值：none 默认

left/right 当前元素脱离默认文档流；在当前行向左，或者向右对齐。

**（3）浮动元素引发的特殊情况**

当父元素显示不下所有浮动元素时，最后一个元素将换行。

元素一旦浮动起来，如果没有设置宽度，浮动元素宽度会以内容为准。（未浮动块级元素，没有设置宽高，则宽等于父级元素宽，高取决于内容高）

元素一旦浮动起来，会变为块级元素（可以设置宽高，上下外边距有效）。

练习：把a标签写成按钮

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  a{  /\*1.大体位置和尺寸\*/  width:120px;height:40px;  float:left; /\*等同display:block; block-块级\*/  /\*2.背景和边框\*/  background:#2196f3;  border-radius:2px;  /\*3.字体，文本\*/  color:#fff;  text-decoration:none;  text-align:center;  line-height:40px;  }  </style>  </head>  <body>  <a href="#">查看详情</a>  </body>  </html> |

**（4）浮动元素占位**

浮动元素一行显示不下，会折行显示，这个折行，会从高度上离原来行最近的一个位置。

如果之前的浮动元素高度太大，会发生浮动元素占位的问题：

即：浮动元素往哪边浮动，那一边的位置，就不能使用。

**（5）浮动元素周围有文字或者行内元素**

文本，行内元素，行内块，在与浮动元素相遇时，有特殊的算法，上述元素不会被浮动元素压在下方，而是巧妙的避开，环绕着浮动元素显示。

**（6）浮动元素对其它元素的影响**

**对后续元素的影响**

元素一旦浮动，就会对后续元素带来一定的影响（后续元素要上前补位）。如果后续元素不想上前补位，需要给后续元素添加清除浮动，让后续元素不被之前浮动元素的影响（自己不上前补位）。

清除浮动影响属性：clear:

取值：left; 清除当前元素前面左浮动元素的影响。

right; 清除当前元素前面右浮动元素的影响。

both; 清除当前元素前面所有浮动元素的影响。

none; 默认值，不清除任何浮动。

**对父元素的影响（高度坍塌）**

父元素是块级，不写高，那么高度是以内部内容高度为准，

但是，浮动元素脱离文档流，不在页面占据高度，

父元素的高度就为0了——高度坍塌

解决方案:

a.给父元素添加高度（弊端：动态内容，不知道高）；

b.给父元素添加浮动（弊端：会给其他非浮动元素带来影响）；

c.为父元素设置overflow

取值：auto/hidden;（弊端：如果父元素想溢出显示，冲突，overflow和float不相关）

d.在父元素中最后追加空的块级元素，并设置clear:both;

这个子元素不会被之前的浮动元素覆盖，

这个子元素默认宽高是父级100%，由于内部无内容，默认高是0，所以不会在页面上看到，但是会占据该在的位置，让父元素找到高。

**5.相对/绝对/固定-定位**

属性：position:

取值：static; 默认值，静态，默认文档流

relative; 相对定位

absolute; 绝对定位

fixed; 固定定位

名词解释：已定位元素

将元素的position设置为relative/absolute/fixed，任意一个值，这个元素就叫做已定位元素。

**（1）偏移属性**

属性：top/right/bottom/left:

取值：px为单位的数字 距离哪个方向有多远

**（2）相对定位**

语法：position:relative;

配合偏移属性，才能移动位置。

特点：

相对定位元素，没有脱离默认文档流，页面上还有占位；后面元素不补位（如果相对定位不使用偏移属性，效果跟没有设置定位一样，只是该元素变成了已定位元素）；

相对谁做偏移？

元素相对于他原来的位置做偏移。

使用时机：

在做元素微调的时候使用；

为内部的子元素做绝对定位时，提供目标（以我定位）。

**（3）绝对定位**

语法：position:absolute;

配合偏移属性使用。

特点：

绝对定位脱离默认文档流，页面不占位置，后面元素上前补位；

绝对定位的位移参照物？

绝对定位的元素会相对离他"最近的""已定位""祖先元素"去实现定位的初始化；

如果，没有已定位的祖先元素，相对body去实现位置的初始化。

使用场合：

有堆叠效果的布局；

弹窗，弹菜单。

注意：绝对定位让元素脱离默认文档流，会使元素变为块级。

**（4）固定定位**

将元素固定在页面的某个位置，位置不会随着页面的滚动发生变化，一直固定在可视区域。

语法：position:fixed;

配合偏移属性使用。

特点：

固定定位始终相对于body去实现位置的初始化；

固定定位元素变成块级；

固定定位元素脱离文档流。

**（5）堆叠顺序**

一旦元素变为已定位元素，元素们有可能出现堆叠效果。

默认堆叠顺序，后写的元素在上面。

改变堆叠顺序的属性：z-index:

取值：无单位数字，数字越大元素越靠上。

注意：

只有已定位元素才能使用z-index；

父子元素间，z-index无效，永远都是子元素压在父元素上。

**二．显示**

**1.显示方式**

显示方式决定了元素在网页中的表现形式（块级，行内，行内块…）

语法：display:

取值：none; 不显示元素，会脱离文档流，不占位置隐藏。

block; 让元素表现得和块级元素一致。

独占一行，可设置宽高和垂直外边距

inline; 让元素表现得和行级元素一致。

多个元素在一行中显示，不能修改尺寸，垂直外边距无效

inline-block;让元素表现得和行内块一致。

多个元素在上一行中显示，可以设置宽高和垂直外边距

table; 让元素表现得和table一致。

尺寸以内容为准，独占一行，可以设置宽高

注意：display一般进行行内元素修改显示方式为块级或者行内块，便于设置宽高尺寸。

**2.显示效果**

属性：visibility:

取值：visible; 默认值，可见的

hidden; 隐藏的，元素不可见

|  |  |
| --- | --- |
| display:none;和visibility:hidden;区别 | |
| display:none; | visibility:hidden; |
| 隐藏，会脱离文档流，不占页面空间 | 不脱离文档流，占据页面空间 |

**3.透明度**

属性：opacity:

取值：0~1 值越小，越透明

|  |  |
| --- | --- |
| opacity和rgba()区别 | |
| opacity | rgba() |
| 作用于元素，只要跟元素相关的颜色都会跟着透明。 | 只是设置当前颜色透明。 |

**4.垂直对齐方式**

属性：vertical-align:

使用场合：

表格中使用，控制表格中的文本垂直对齐方式；

取值：top/middel/bottom;

img中使用

取值：top/middel/bottom/baseline(默认值，基线);

控制图片旁边文字的对齐方式。

一般情况下：编写网页的时候，会把所有图片的垂直对齐方式改变为非基线对齐。

**5.光标样式**

改变鼠标悬停在元素上时的状态。

属性：cursor:

取值：default 箭头

pointer 小手

text I

wait 等待

crosshair +

help 问号

示例：img:hover{cursor:help}

**三．列表**

1.列表标识项

属性：list-style-type:

取值：none

disc

circle

square

**2.列表表示图像**

属性：list-style-image:

取值：url(wechat.png);

**3.列表标识项的位置**

属性：list-style-position:

取值：outside 默认值，标识在li外，在ul/ol的padding-left里面。

inside 标识在li内部。

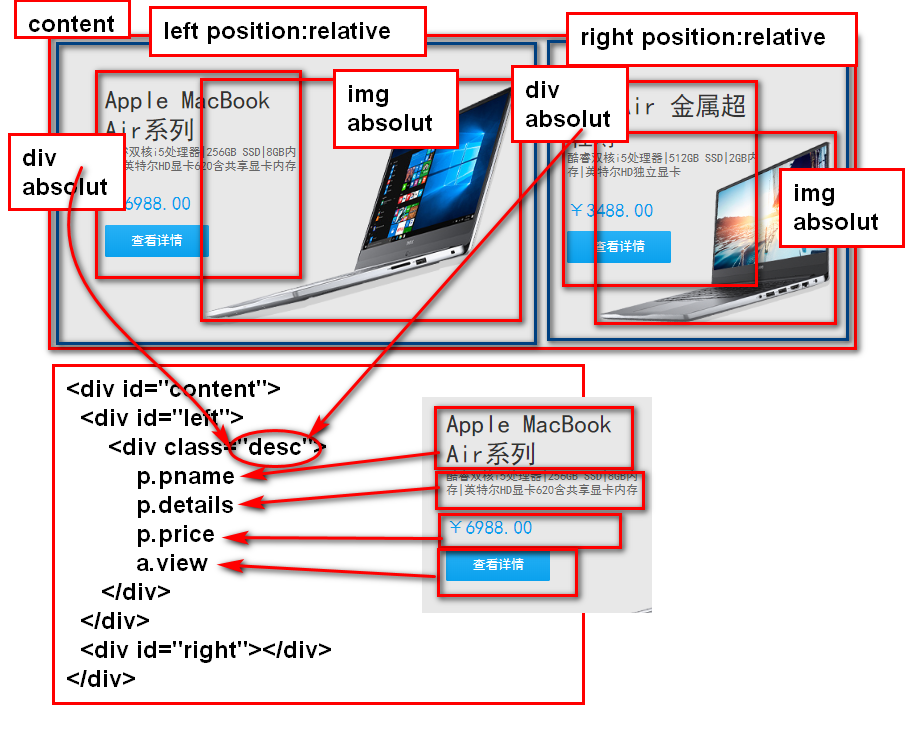
|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  ul{  list-style-image:url(wechat.png);  list-style-position:inside;  }  </style>  </head>  <body>  <ul>  <li>流浪地球</li>  <li>飞驰人生</li>  <li>肖申克的救赎</li>  <li>当幸福来敲门</li>  </ul>  </body>  </html> |

**4．简写方式**

语法：list-style:type url() psition;

最常使用的方式list-style:none; 去掉标识符

课后练习：



|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  \*{  margin:0;padding:0;  }  #container{  margin:0 auto;  width:1000px;height:377px;  box-sizing:border-box;  }  #container>div{  height:377px;  float:left;  background:#eee;  position:relative;  }  #container #left{  width:609px;height:377px;  }  #container #right{  width:381px;height:377px;  margin-left:10px;  }  .desc{  width:245px;height:243px;  position:absolute;  top:55px;  left:65px;  }  .pname{  font:bold 20px 宋体;  }  .details{  font:12px 宋体;  }  .price{  margin:20px 0px;  color:#03a9f4;  font:20px 宋体;  }  .view{  float:left;  width:130px;height:40px;  background:#2196f3;  color:#fff;  font:12px 宋体;  text-decoration:none;  text-align:center;  line-height:40px;  border-radius:2px;  }  .image:hover{  position:relative;  top:60px;  left:150px;  }  .image{  position:absolute;  top:56px;  right:30px;  }  </style>  </head>  <body>  <div id="container">  <div id="left">  <div class="desc">  <p class="pname">Apple MacBook Air系列</p>  <p class="details">酷睿双核i5处理器|256GB|8G内存|英特尔HD显卡620含共享显卡内存</p>  <p class="price">￥6988</p>  <a href="#" class="view">查看详情</a>  </div>  <img src="study\_computer\_img1.png" alt="" class="image">  </div>  <div id="right">  <div class="desc">  <p class="pname">小米Air 金属超轻薄</p>  <p class="details">酷睿双核i5处理器|512GB SSD|2GB内存|英特尔HD独立显卡</p>  <p class="price">￥3488</p>  <a href="#" class="view">查看详情</a>  </div>  <img src="study\_computer\_img2.png" alt="" class="image">  </div>  </div>  </body>  </html> |

**一．CSS3 core（核心）**

**1.复杂选择器**

（1）兄弟选择器

兄弟元素，具备相同父级元素的平级元素之间成为兄弟元素。

注意：兄弟选择器，只能往后找(兄弟)，不能往前找。

相邻兄弟选择器

获取紧紧挨在某个元素后面的兄弟元素。

语法：选择器1+选择器2{}（符合此条件的后面的一个兄弟都会选择）

通用兄弟选择器

获取某元素后面，所有的满足选择器2条件的兄弟元素。

语法：选择器1 ~选择器2{}

示例：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  p+span{background:#f00;}/\*选择：老三和老六\*/  #p1~p{background:#0ff;} /\*选择：老二和老五\*/  </style>  </head>  <body>  <div>  <p id="p1">老大</p>  <p>老二</p>  <span>老三</span>  <b>老四</b>  <p>老五</p>  <span>老六</span>  <div>老七</div>  </div>  </body>  </html> |

**（2）属性选择器**

attr：id class name type value style title

允许通过元素所有附带的属性及其值，来匹配页面元素。

任意属性

[attr]{}

匹配页面中所有附带attr属性的元素

示例：[id]{} [class]{} [style]{ background:#ff0;}

某一个元素的属性

elem[attr]{}

匹配页面中所有附带attr属性的elem元素

示例：div[id]{} p[class]{ background:#00f;}

匹配多个属性的元素

[attr][attr2]…{}

elem[attr][attr2]…{}

示例：p[class][title]{ background:#0ff;}

匹配attr属性值为value的元素

[attr=value]{}

示例：[id="one"]{background:#f00;}

属性值的匹配

[attr^=value]{} 匹配属性以指定值开头的元素

[attr$=value]{} 匹配属性以指定值结尾的元素

[attr~=value]{} 匹配属性值中包含指定单词的元素

[attr\*=value]{} 匹配属性值中包含指定值的元素

示例：[title^="t"]{background:#ff0;}

**（3）伪类选择器**

已经学过的伪类

链接伪类：:link :visited

动态伪类：:hover :active :focus

目标伪类

突出显示获得html锚点元素，匹配被激活的锚点。

语法：elem:target{}

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  div img{  display:none;  width:200px;  border:2px solid #f00;  }  div img:target{/\*点击a锚点，会显示对应的图片\*/  display:block;  }  </style>  </head>  <body>  <div>  <a href="#i1">海鲜面</a>  <a href="#i2">春卷</a>  <a href="#i3">八大碗</a>  <a href="#i4">排骨</a>  </div>  <div>  <img id="i1" src="index\_hot\_food\_img1.png" alt="">  <img id="i2" src="index\_hot\_food\_img2.png" alt="">  <img id="i3" src="index\_hot\_food\_img3.png" alt="">  <img id="i4" src="index\_Sbanner\_img3.png" alt="">  </div>  </body>  </html> |

结构伪类

通过元素结构关系匹配元素。

语法：elem:first-child{} 匹配元素是属于其父元素的第一个子元素。（找第一个elem）

语法：elem:last-child{} 匹配元素是属于其父元素的最后一个子元素。（找最后一个elem）

语法：elem:nth-child(n){}匹配元素是属于其父元素的第n个子元素。（找第n个elem）

n是从1开始的。

示例：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  table{width:400px;height:400px;}  table td{border:2px solid #000;}  tr:first-child{background:#00f;}/\*第一行\*/  tr:last-child{background:#ff0;} /\*最后一行\*/  tr:nth-child(2) td:nth-child(2){background:#f00;} /\*第二行，第二列\*/  table td:not(:first-child){background:#f0f;} /\*除了第一列的所有元素\*/  </style>  </head>  <body>  <table>  <tr>  <td>3</td>  <td>4</td>  </tr>  <tr>  <td>5</td>  <td>6</td>  </tr>  </table>  </body>  </html> |

elem:empty{} 匹配没有子元素的元素。

elem:only-child{} 匹配属于其父元素只有一个子元素的子元素。

elem:not(selector){} 否定伪类。

排除满足指定选择器(selector)的元素(elem)。

示例：p:not(.p1){background:#off;}排除class为p1的p标签

**（4）伪元素选择器**——只修饰内容、文本，不是修饰整个标签

匹配某个元素首字符

::first-letter{} 或者:first-letter

匹配某个元素的首行

::first-line{} 或者:first-line

注意：首行和首字符冲突，优先使用首字符样式。

匹配用户选中的内容

::selection{}

注意：只能设置两种样式，其它无效（字体颜色和背景颜色）。

**2.内容生成**

**（1）什么是内容生成？**

使用css动态的向元素内插入一段内容（文本、元素）。

**（2）伪元素选择器**

在匹配的元素内容区域之前，添加内容

::before 或者:before

在匹配的元素内容区域之后，添加内容

::after 或者:after

**（3）内容属性**

属性：content: 想匹配的位置添加内容

取值：""括起来的字符串

url(图片)

示例：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  #p1::before{/\*在p1标签内前面添加内容\*/  content:"子曾经曰过:";  display:block;  color:#f00;  }  #p2::after{/\*在p2标签内后面添加内容\*/  content:"----摘自《论语》";  display:block;  text-align:center;  color:#f00;  }  </style>  </head>  <body>  <p id="p1">学而时习之，不亦说乎。</p>  <p id="p2">有朋自远方来，不亦乐乎</p>  </body>  </html> |

**（4）可以解决的问题**

**高度坍塌**

父元素不写高，所有子元素都浮动，父元素的高度就为0。

结局方案：在父元素最后添加空的块级元素，并设置clear:both;

示例：

|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  #parent{border:2px solid #f00;}  div.d1{  width:200px;height:100px;  background:#0ff;  float:left;  }  div+.d1{margin-left:3px;}  #parent::after{  content:"";  display:block;  clear:both;  }  </style>  </head>  <body>  <div id="parent">  <div class="d1"></div>  <div class="d1"></div>  <div class="d1"></div>  </div>  </body>  </html> |

**子元素外边溢出**

修改子元素上边外边距，父级元素跟着受影响。

解决方案：在父元素第一个子元素位置，添加一个空table。

示例：

|  |
| --- |
| #parent::before{content:"";display:table;} |

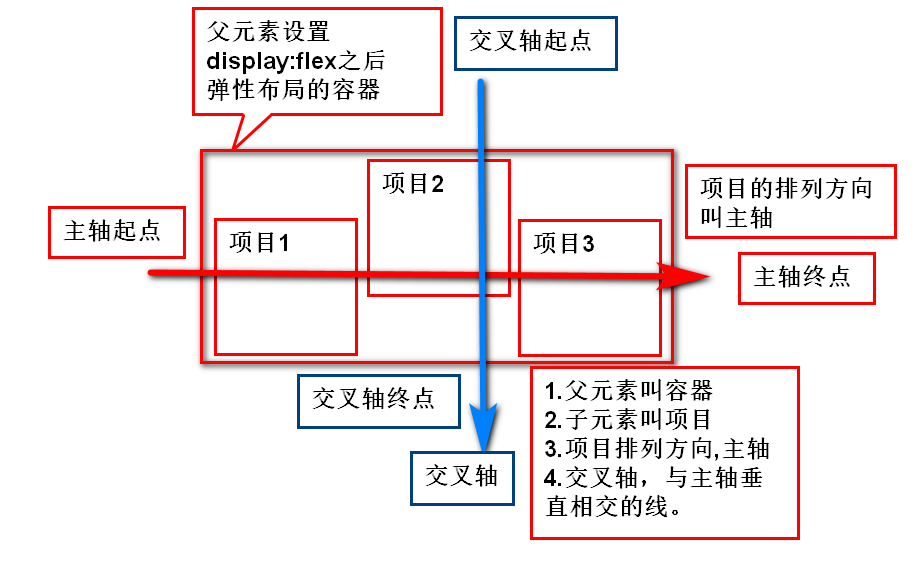
**一．弹性布局**

**1．什么是弹性布局？**

弹性布局是一种灵活的布局方式，主要解决某元素中子元素的布局方式，

为布局提供了最大的灵活性。

**2.弹性布局相关概念**

****

弹性布局容器

要弹性布局的子元素的父元素称之为容器。

弹性布局的项目

要弹性布局的子元素称为项目。

主轴

项目们排列方向的一根轴，称之为主轴。

如果项目们按照横向排列，水平轴就是主轴。

如果项目们按照纵向排列，垂直轴就是主轴。

交叉轴

与主轴垂直相交的轴，称之为交叉轴。

**3.语法**

将元素变为弹性容器后，他的所有子元素将变为弹性项目，都允许按照弹性布局的方式排列。

属性：display:

取值：flex; 将块级元素变为容器

inline-flex; 将行内元素变为容器

注意：

给弹性项目添加float/clear/vertical-align无效；

弹性项目如果是行内元素，也可以设置尺寸；

弹性容器设置text-align无效。

**4.容器的属性**

**（1）主轴的方向**

设置容器的主轴以及项目排列的方向。

属性：flex-direction:

取值：row; 默认值，主轴就是x轴，起点在左端

row-reverse; 主轴是x轴，起点在右端

column; 主轴是y轴，起点在顶端

column-reverse; 主轴是y轴，起点在底端

**（2）设置压缩/换行**

作用：当一个主轴排列不下所有项目，可以设置压缩显示或者换行。

属性：flex-wrap:

取值：nowrap; 默认值，空间不够也不换行，项目自动缩小

wrap; 换行，不压缩

wrap-reverse; 换行并反转

**（3）简写**

属性：flex-flow: 使用flex-direction和flex-wrap的缩写。

取值：row nowrap; 默认值

direction wrap; 主轴的方向和压缩/换行

示例：flex-flow:row-reverse wrap;

**（4）项目在主轴上的对齐方式**

属性：justify-content:

取值：flex-start; 在主轴的起点对齐

flex-end; 在主轴的终点对齐

center; 在主轴居中对齐

space-between; 两端对齐

space-around; 每个项目左右外边距相同

**（5）项目们在交叉轴上的对齐方式**

属性：align-items:

取值：flex-start; 在交叉轴起点对齐

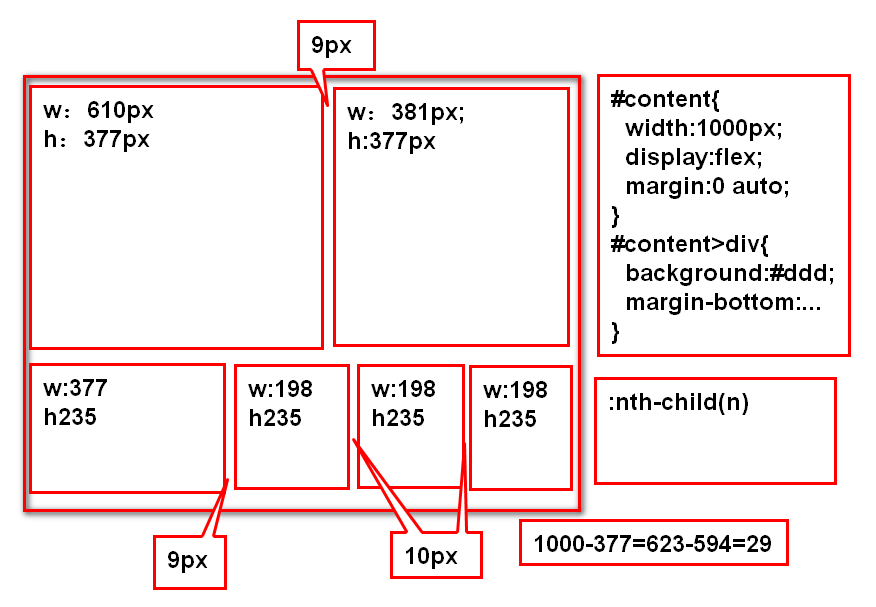
flex-end; 在交叉轴终点对齐

center; 在交叉轴居中对齐

baseline; 在交叉轴基线对齐

stretch; 如果项目未设置尺寸，在交叉轴上占满空间

示例：（自动计算内容外边距大小）



|  |
| --- |
| <!doctype html>  <html>  <head>  <title></title>  <meta charset="utf-8">  <style>  \*{margin:0;padding:0}  #content{  margin:0 auto;  width:1000px;  display:flex;  flex-wrap:wrap;  justify-content:space-between;  }  #content div{background:#eee;margin-top:5px;}  #content div:first-child{width:610px;height:377px;}  #content div:nth-child(2){width:381px;height:377px;  /\* margin-left:9px;\*/ }  #content div:nth-child(3){width:377px;height:235px;  /\* margin-right:9px;\*/ }  #content>div.d4{width:198px;height:235px;}  /\* div.d4+div.d4{margin-left:10px;}\*/  </style>  </head>  <body>  <div id="content">  <div></div>  <div></div>  <div></div>  <div class="d4"></div>  <div class="d4"></div>  <div class="d4"></div>  </div>  </body>  </html> |

**5.项目属性**

该组属性只能设置在某一个项目上，不影响容器和其他项目显示效果。

（1）order:

定义项目的排列顺序，值越小越靠近起点，默认值为0。

取值：无单位数字

（2）flex-grow:

定义项目的放大比例。

如果容器有足够大的剩余空间，项目将按照比例放大。

取值：整数数字，无单位

默认值0，不放大

注意：比例并不精确，取值越大，占据剩余空间越多

（3）flex-shrink:

定义项目的缩小比例，空间不足时，项目如何缩小。

取值：默认值1，空间不足，所有项目等比缩小

取值越大，占据空间越小

值为0时，不压缩

（4）align-self:

定义某一个项目在交叉轴对齐方式。

取值：flex-start; 在交叉轴起点对齐

flex-end; 在交叉轴终点对齐

center; 在交叉轴居中对齐

baseline; 在交叉轴基线对齐

stretch; 如果项目未设置尺寸，在交叉轴上占满空间

auto; 继承父元素的align-items效果

**二．CSS hack**

由于不同的浏览器内核。

internet explorer6，IE7 Mozilla Firefox

对css的解析和认识不一样，会导致生成的页面效果不同。

得不到我们需要的效果。

这种时候，我们需要针对不同的浏览器，些不同的css，

让他能够兼容不同的浏览器，

能在不同的浏览器中，得到我们想要的效果。

这种针对不同浏览器写不同的css code的过程，

就叫做css hack，也写作css hack。

已经过时，css hack编写，在tmooc有拓展视频。

**三．转换**（重点\*\*\*\*\*\*\*）

**1．什么是转换？**

改变元素在页面中的位置，大小，角度，形状的一种方式。

2D转换，在x轴和y轴上，发生的转换效果。

3D转换，增加了z轴上的转换效果。

**2.转换的属性和值**

**（1）转换**

属性：transform:

取值：none 默认值，无任何的转换效果

transform-function——转换函数，如果有多个转换函数，用空格隔开。

**（2）转换原点**

属性：transform-origin:

取值：带px的数字/%/关键字(bottom right)

2个值，表示原点在x轴和y轴上的位置

3个值，表示原点在x轴，y轴和z轴上的位置

默认值，原点在元素的中心（center center）

**3.2D转换函数**

**（1）位移（改变位置）**

改变元素在页面中的位置。

语法：transform:translate();

取值：translate(v1); 指定元素在x轴上的位移距离

+ 元素右移/- 元素左移

translate(x,y); 指定元素在x轴和y轴上的位移距离

x：+ 元素右移/- 元素左移

y：+ 元素下移/- 元素上移

translateX(x) 只设置x轴上的位移

translateY(y) 只设置y轴上的位移

**（2）缩放**

改变元素在页面中的尺寸。

语法：transform:scale();

取值：scale(v1); v1：横向和纵向的放缩比，默认值为1

v1>1 放大

0<v1<1 缩小

-1<v1<0 缩小并反转

v1<-1 放大并反转

scale(x,y);

x：横向缩放比

y：纵向缩放比

scaleX(x); 单独设置水平缩放

scaleY(y); 单独设置垂直缩放

**（3）旋转**

改变元素在页面中的角度。

语法：transform:rotate();

取值：rotate(ndeg)

n：+ 顺时针/- 逆时针

注意：旋转原点会影响最后的旋转效果；

旋转是连同坐标轴一起旋转的，旋转后再位移会，影响位移效果。

**（4）倾斜**

改变元素的形状。

语法：transform:skew();

取值：skew(v1);

v1：让元素向着x轴发生倾斜效果，实际上是改变y轴的角度。

v1：+ y轴逆时针倾斜/- y轴顺时针倾斜

skewx(x); 同skew(v1)

skewY(y); 让元素向着y轴发生倾斜效果，实际上是改变x轴的角度

y：+ x轴顺时针倾斜/- x轴逆时针倾斜

skew(x,y);

**4.3D转换**

**（1）透视距离**

模拟人的眼睛到元素之间的距离，根据距离不同，元素的3D转换效果也会不同。

属性：perspective:

取值：px为单位的数字

注意：该属性要加载转换元素的父元素上。

**（2）3D旋转**

语法：transform:rotate3D(1,1,1,20deg)

取值：rotateX(xdeg); 以x轴为中心，旋转元素

rotateY(ydeg); 以y轴为中心，旋转元素

rotateZ(zdeg); 以z轴为中心，旋转元素

rotate3D(x,y,z,ndeg);

x,y,z取值大于0的数字的时候，表示该轴参与旋转；

取值为0，表示该轴不参与旋转。

**一．过渡**（重点\*\*\*\*\*\*\*\*\*\*\*\*\*）

**1.什么是过渡？**

让css的值，在一段时间内平缓变化的效果。

哪些属性可以使用过渡？

过渡时间的设置？

平缓（变化速度的设置）

**2.语法**

**（1）设置过渡属性**

属性：transition-property:

取值：all; 支持所有过渡的属性

具体哪种属性的属性名（属性之间用空格隔开）

|  |
| --- |
| 能使用过渡效果的属性：  颜色属性  取值为数字的属  所有的转换属性  所有的阴影属性  visibility |

**（2）设置过渡的时长**——指定多长时间内完成过渡的操作

属性：transition-duration:

取值：以s或者ms为单位的数字

**（3）过渡时间曲线函数**

属性：transition-timing-function:

取值：ease; 默认值，慢速开始，加速，慢速结束

linear; 匀速

ease-in; 慢速开始，快速结束

ease-out; 快速开始，慢速结束

ease-in-out; 慢速开始，加速，减速，慢速结束

**（4）设置过渡延时时间**

属性：transition-delay:

取值：以s或者ms为单位的数字

**（5）过渡属性的编写位置**

要发生过渡的元素，本身的样式中编写；

管去管回

写在过渡的触发操作里面。

只去不回

**（6）过渡的简写方式**

语法：transition: property duration timing-function delay;

属性 过渡时长 时间曲线函数 延时

最简化的写法：transition:duration;

|  |
| --- |
| 示例：  <style>  div{  width:200px;height:200px;  background:#0ff;  border-radius:50%;  /\*1.设置过渡属性\*/  transition-property:all;  /\*2.设置过渡时间\*/  transition-duration:5s;  /\*3.过渡时间曲线函数（过渡的速度）\*/  transition-timing-function:ease-in-out;  /\*4.延迟时间\*/  transition-delay:2s;  }  div:hover{  border-radius:0%;  transform:translate(1000px,1000px);  }  </style> |

**二．动画**

**1.什么是动画？**

使元素从一中样式逐渐变为另一种样式，就是将多个过渡效果放在一起。

（直接使用网站animate.css）

**2.关键帧**

动画是通过关键帧，来控制每一步播放效果的

执行的时间点

该时间点要应用的样式

**3.动画的使用步骤**

**（1）声明动画**

语法：@keyframes 动画名称{

0%{元素动画开始时，该元素的样式}

…

50%{元素动画进行一半时，该元素的样式}

…

100%{元素动画完成时，该元素的样式}

}

**（2）调用动画**

属性：animation-name: 动画名称

animation-duration: 过渡时长

animation-timing-function: 时间曲线函数

animation-delay: 延迟时间

设置动画的重复次数

属性：animation-iteration-count:

取值：具体的整数

infinite; 无限次

设置动画的播放方向

属性：animation-direction:

取值：normal; 默认值，正常播放0%~100%

reverse; 逆向播放，100%~0%

alternate; 轮流播放，基数次正向，偶数次逆向

动画的简写方式

语法：animation:name duration timing-function delay count direction;

动画名 过渡时长 时间曲线函数 延时 次数 方向

最简写方式：animation:name duration;

设置动画在播放前和播放后的状态

属性：animation-fill-mode:

取值：none; 默认值，前后无填充

forwards; 动画播放完成后，将保持在最后一帧的样式上

backwords; 动画播放之前的延迟状态，保持在第一帧的样式上

both; forwards和backwords都生效

设置动画的播放或者暂停

属性：animation-play-state:

取值：paused; 暂停

running; 播放

|  |
| --- |
| 示例：  <style>  @keyframes change{  0%{background:#00f;border-radius:0%;}  50%{background:#f0f;border-radius:0%;}  100%{background:#f00;border-radius:0%;}  }  div{  width:200px;height:200px;  border:2px solid #f00;  }  div:hover{  animation:change 5s linear 1s;  animation-iteration-count:infinite;  animation-direction:alternate;  }  </style> |

**4.动画的兼容性**

如果要兼容不同的额浏览器

chrome&Safari：-webkit-keyframes-

IE： -ms-keyframes-

Firefox： -moz-keyframes-

Oprah： -o-keyframes-

**三．CSS的优化**

**1.优化的目的**

减少服务器压力；

提升用户体验。

**2.优化的原则**

尽量减少HTTP请求的个数；

页面的顶部引入css文件；

将css和js文件放在独立的文件中。

**3.代码优化**

合并样式；

缩小样式文件；

减少样式的重用；

避免出现空的src、href；

选择更优的样式属性；

代码压缩。

**四．响应式**（重点\*\*\*\*\*\*\*\*\*\*\*）

**1.什么事响应式？**

Responsive Web Page自适应网页

可以根据浏览器设备不用(pc，pad，phone)，不更改布局，图片，文字效果，不影响用户的体验。

**2.响应式网页必须做到**

（1）布局：不能固定宽度，必须使用流式布局；

（2）文字和图片大小随着容器的大小而改变；

（3）媒体查询技术（根据不同设备显示不同效果）。

响应式网页存在的问题：代码复杂程度，极大的增加。

适用于：内容不太多的页面（企业的官网，用户网站）。

**3.如何测试响应网页**

**（1）方法1：使用真实设备测试**

好处：真实可靠；

坏处：成本高，测试任务量大。

**（2）使用第三方模拟器**

好处：测试极其方便；

坏处：测试效果有限，有待进一步验证。

**（3）使用chrome等自带的模拟器**

好处：简单方便；

坏处：测试效果十分有限。

**4.编写响应式网页**

**（1）在meta中声明viewport元标签（手机适配）**

width=device-width 设置视口宽度为设备宽度

initial-scale=1.0 表示视口宽度是否能缩放，1.0代表不缩放

maximum-scale=1.0 允许缩放的最大倍率

user-scale=0 是否允许用户手动缩放

取值：yes/1 no/0

设置视口最精简的写法，boot官网的写法

<meta name="viewport" content="width=device-width, initial-scale=1">

**（2）所有内容，文字，图片，使用相对尺寸(%,rem)，不要使用绝对值数字。**

练习中，可是使用绝对值，项目中不要出现。

**（3）流式布局+弹性布局，在配合媒体查询技术**

默认文档流/float+flex

**（4）CSS3的媒体查询技术**

media：专制设备

比如：screen(pc/pad/phone)，tv，print…

Media Query：媒体查询，可以根据当前浏览设备不同，或者屏幕尺寸，方向，解析度不同。

有选择的执行部分css样式，忽略其他部分css样式。

使用media query，根据媒体查询结果，使用不同的外部css文件

示例：

|  |
| --- |
| <meta name="viewport" content="width=decive-width,initial-scale=1">  <link rel="stylesheet" href="pc.css" media="screen and (min-width:992px)">  <link rel="stylesheet" href="pad.css" media="screen and (min-width:768px) and (max-width:991px)">  <link rel="stylesheet" href="phone.css" media="screen and (max-width:767px)"> |

注意：这种方法使用较少。

所有css文件都会被加载，每个css文件中存在很多重复代码，

不符合css优化原则。

根据媒体查询结果，执行同一个css文件中的不同代码块。

语法：@media screen and (min-width:n1px) and (max-width:n2px){

选择器样式{}

}

示例：

|  |
| --- |
| html写入：  <head>  <meta name="viewport" content="width=decive-width,initial-scale=1">  <link rel="stylesheet" href="myMedia.css">  </head> |
| css写入：  body{  font:16px 黑体;  padding:0px;margin:0px;  }  /\*pc>=992\*/  @media screen and (min-width:992px){  .content{  color:#fff;  background:#0ff;  font-size:24px;  }  }  /\*768<=pad<=991\*/  @media screen and (min-width:768px) and (max-width:991px){  .content{  color:#f00;  background:#afa;  }  }  /\*phone<=767\*/  @media screen and (max-width:767px){  .content{  color:#fff;  background:#f00;  font-size:12px;  }  } |

**一．Bootstrap**

简洁、直观、强悍的前端开发框架，让web开发更迅速、简单。

|  |
| --- |
| 1.如何使用框架  2.全局css样式  3.组件  4.js插件  5.定制(sass/scss) |

**1.起步，如何使用bootstrap**

（1）下载

（2）创建基本模板ws使用

在html中依次引入：

<meta name="viewport" content="width=decive-width,initial-scale=1">

<link rel="stylesheet" href="css/bootstrap.css">

<script src="js/jquery.min.js"></script>

<script src="js/popper.min.js"></script>

<script src="js/bootstrap.min.js"></script>

**2.全局CSS样式**

**（1）按钮相关class**

基本类： .btn

危险色： .btn-danger

成功色： .btn-success

警告说： .btn-warning

信息色： .btn-info

主要色： .btn-primary

次要色： .btn-secondary

深色： .btn-dark

浅色： .btn-light

链接： .btn-link

|  |
| --- |
| 示例：  <input type="button" class="btn" value="按钮">  <button class="btn btn-danger">危险色</button> |

不同边框的按钮： btn-outline-danger/success…

大按钮： btn-lg

小按钮： btn-sm

块级按钮： btn-block

**（2）图片和边框**

响应式图片（图片可以缩放，但是不会超过原图大小）：.img-fluid

圆角图片： <img src="../day2.22/img/1.jpg" alt="" class="rounded">

椭圆图片： <img src="../day2.22/img/2.jpg" alt="" class="rounded-circle">

缩略图： <img src="../day2.22/img/3.jpg" alt="" class="img-thumbnail">

示例：

<img src="../day2.22/img/4.jpg" alt="" class="img-fluid">

**（3）文本**

控制字体大小和加粗： .h1~.h6

字体加粗： .font-wight-bold/normol/light

字体颜色： .text-danger/success…

文本对齐方式： .text-left/center/right/justify

媒体查询的水平对齐（大屏右对齐，中屏居中，小屏左对齐）：.text-lg/md/sm-right/center/left

字体大写/小写/首字母大写：.text-uppercase/lowercase/capitalize

**（4）列表**

ul：

列表组基础类： .list-group

去除列表项： .list-unstyled

list：

列表组基础类： .list-group-item

激活状态列： .list-group-item.active

禁用列表项： .list-group-item.disable

列表颜色： .list-group-item-success/danger/warning…

span:

文本显示不下，后面用省略号…： .text-truncate

|  |
| --- |
| 示例：  <ul class="list-group">  <lic class="list-group-item list-group-item.active">  <span class="text-truncate>危险，后面显示不下用…</span>  </li>  </ul> |

**（5）table**

基本类： .table

带边框的表格： .table-bordered

隔行变色表格： .table-striped

带悬停效果： .table-hover

表格颜色： .table-danger/success/…

响应式表格，要写在table标签的父元素中： .table-responsive-xl/lg/md/sm

**3.辅助类**

**（1）边框**

基本类/方向： .border/.border-top/right/bottom/left

边框颜色： .border-danger/warning…

去除边框： .border-0/border-top/right/bottom/left-0

**（2）倒角**

默认倒角/无倒角： .rounded/rounded-0

设置一侧的倒角： .rounded-top/right/left

设置圆： .rounded-circle

**（3）背景颜色**

bg-warning/danger/success…

**（4）显示、隐藏**

visible

invisible

**（5）浮动**

.float-\*-left/right/none

\*：xl/lg/md/sm

**（6）清除浮动**

写在父级，解决高度坍塌：clearfix

**（7）边距**

外边距：

m-0/1/2/3/4/5 (对应的大小 0/0.5rem/0.75rem/1rem/1.5rem/3rem)

上/右/下/左： mt-n/mr-n/mb-n/ml-n

响应式布局： m-\*-n

\*：xl/lg/md/sm

n：1,2,3,4,5

块级元素水平自动居中：m-auto

内边距：

p-0/1/2/3/4/5 (对应的大小 0/0.5rem/0.75rem/1rem/1.5rem/3rem)

上/右/下/左： pt-n/pr-n/pb-n/pl-n

响应式布局： p-\*-n

\*：xl/lg/md/sm

n：1,2,3,4,5

**（8）尺寸**

w-25/50/75/100

h-25/50/75/100

**（9）元素显示效果**

.d-none/block/inline/inline-block/table/flex

display:none/block/inline/inline-block/table/flex

**二．栅格布局**

**1.web页面的布局方法（三种）**

|  |  |  |
| --- | --- | --- |
| table布局 | DIV+CSS | boot的栅格布局 |
| 简单，容易控制 | 语义正确，渲染效率高 | 简单，容易控制，语义正确，渲染效率高，支持响应式 |
| 语义错误(表格是用于展示数据，这里却用于布局)  渲染效率低(最后一个数据加载完成后，才开始渲染) | 控制很繁琐 | 页面过于复杂，不适用 |

**2.使用boot栅格布局**

随着屏幕或视口尺寸的增减，系统自动将行分为12份，通过行row与列col的组合在创建页面布局。

**（1）做外层需要使用容器**

.container 定容器

.container-fluid 变宽容器

**（2）在容器中声明行**

.row 一行均分为12份

**（3）在行中声明列**

.col 每个列都需要制定占多少份 n/12

|  |
| --- |
| 示例：  <div class="container">  <div class="row">  <div class="col-lg-4 col-md-6 col-sm-12">1</div>  <div class="col-lg-4 col-md-6 col-sm-12">2</div>  <div class="col-lg-4 col-md-6 col-sm-12">3</div>  <div class="col-lg-4 col-md-6 col-sm-12">4</div>  <div class="col-lg-4 col-md-6 col-sm-12">5</div>  <div class="col-lg-4 col-md-6 col-sm-12">6</div>  </div>  </div> |

**（4）屏幕尺寸**

|  |
| --- |
| xl：Extra large 超大屏 w>=1200px  lg：large 大屏 w>=992px  md：medium 中屏 w>=768px  sm：small 小屏 w>=576px  xs：Extra small 超小屏 w<576（boot不支持超小屏） |

**（5）不同屏幕列的适应性**

|  |
| --- |
| col-xl-n 只能作用到xl(超大屏)  col-lg-n 在xl/lg下有效  col-md-n 在xl/lg/md下有效  col-sm-n 在xl/lg/md/sm下有效  可以精简代码，如：  <div class="col-lg-6 col-md-6 col-sm-6">1</div>  精简代码：<div class="col-sm-6">1</div> |

**（6）列偏移**

使用列偏移实现指定的列，及其后面的列向右移动。

.offset-\*-n

\*：xl/lg/md/xm

n：需要偏移的份数1~11

**（7）.col不加任何属性**

同一行的每个列，宽度相等，自动计算。

**（8）栅格嵌套**

在列中去嵌套其他行。

在一个列中，嵌套一个row

那么嵌套的row的宽度，是这个列的宽度。

分成12份，给内部的列使用

|  |
| --- |
| 示例：  <div class="row">  <div class="col-6">  <div class="row">  <div class="col-6">文本内容1</div>  <div class="col-6">文本内容2</div>  </div>  </div>  <div class="col-6">1</div>  </div> |

**3.弹性布局**

**（1）创建弹性容器**

d-\*-flex

d-\*-inline-flex

\*：xl/lg/md/sm

**（2）设置主轴方向**

flex-\*-row/column/row-reverse/column-reverse

项目在主轴方向的排列方式：justify-content-start/center/end/between/around

**4.表单**

**（1）表单元素排列方向**

.form-group 堆叠表单（垂直排列，全屏的宽度）

.form-inline 内联表单（水平方向排列）

**（2）样式类**

.form-control 块级，w-100

.form-check 相对定位，向右移动1.25rem

.form-check-input 绝对定位，向左移动1.25rem

.form-check-label 字体颜色，下外边距

.form-text 块级，上外边距0.25rem

**总结：**

|  |  |  |
| --- | --- | --- |
|  | 默认栅格 | 水平表单 |
| 行 | .row | form.form-inline |
| 列 | .col | .div.col |

示例：

|  |
| --- |
| <div class="container">  <form action="" class="form-inline mt-3">  <div class="col-md-2">用户名</div>  <div class="col-md-6">  <input type="text" class="form-contaol w-100">  </div>  <div class="col-md-4">用户名包含字母数字3~6</div>  </form>  <form action="" class="form-inline mt-3">  <div class="col-md-2">用户密码</div>  <div class="col-md-6">  <input type="password" class="form-contaol w-100">  </div>  <div class="col-md-4">用户密码包含字母数字3~6</div>  </form>  <form action="" class="form-inline m-3">  <div class="col-12 text-center">  <input type="submit" value="注册" class="btn btn-success">  <input type="reset" value="取消" class="btn btn-danger">  </div>  </form>  </div> |

**一.组件**

**1.下拉菜单**

|  |
| --- |
| 样式布局最小结构：——3层结构  <div class="dropdown">----------------1层  <button data-toggle="dropdown"></button>  <div class="dropdown-menu">---------2层  <a class="dropdown-item">1</a>----3层  <a class="dropdown-item">2</a>  </div>  </div> |
| 事件写法：  在button写自定义属性  data-toggle="dropdown"  转换数据的目标，式按钮的下一个元素 |

在按钮上画三角 .dropdown-toggle

给菜单加标题 .dropdown-header

给menu选项加分割线 .dropdown-divider

选项禁用状态 .disable

**2．按钮组**

在外包裹的div上添加类

创建案件组： .btn-group

其他修饰类

垂直按钮组： .btn-group-vertical

按钮的大小： .btn-group-lg/sm

**3.提示框**

提示框基本类： .alert

设置颜色： .alert-danger/success/…

后代选择器，设置内部元素.close的样式

.alert-dismissible

事件(点击span内容，alert提示的信息被驳回)

.data-dismiss="alert"

|  |
| --- |
| 示例：  <div class="alert alert-danger w-50 alert-dismissible">  请检查用户信息  <span class="close" data-dismiss="alert">&times;</span>  </div> |

**4.导航**

boot提供了3中导航，水平导航，选项卡导航，胶囊导航

**（1）水平导航**

最基本结构： ul.nav>li.nav-item>a.nav-link

ul让ul变成弹性布局，所有弹性布局属性都可以使用

justify-content-start/end/center/between/around

justified 让内部每个item等宽显示

flex-column/row/…

anav-link 设置块级和内边框，hover，focus，disable都设置了

|  |
| --- |
| 示例：  <ul class="nav">--------------1层  <li class="nav-item">-------2层  <a href="#" class="nav-link">秒杀</a>--3层  </li>  <li class="nav-item">  <a href="#" class="nav-link">优惠</a>  </li>  </ul> |

**（2）选项卡导航**

选项卡的部分：

在水平导航栏的基础上，ul增加了一个nav-tabs类

内容部分：

div.tab-content对父级div没有任何修饰

当子元素出现tab-pane 让该元素display:none;

当子元素出现active 让该元素display:block;

给a标签添加事件：

data-toggle="tab"

设置数据概变div的id

给第一个显示的a标签添加active(默认显示)

|  |
| --- |
| <!-- 选项卡导航栏 -->  <ul class="nav nav-tabs">  <li class="nav-item">  <a data-toggle="tab" href="#d1" class="nav-link">神奇四侠</a>  </li>  <li class="nav-item">  <a data-toggle="tab" href="#d2" class="nav-link">闪电侠</a>  </li>  </ul>  <!-- 内容显示区域 -->  <div class="tab-content">  <div id="d1" class="tab-pane active">  神奇四侠：Lorem ipsum dolor sit amet consectetur adipisicing.  </div>  <div id="d2" class="tab-pane">  闪电侠：Lorem ipsum dolor sit amet consectetur adipisicing.  </div>  </div> |

**（3）胶囊导航**

在选项卡导航的基础上，ul的类改为nav-pills类

在选项卡导航基础上将a标签添加的事件类改为pill

data-toggle="pill"

|  |
| --- |
| 示例：  <ul class="nav nav-pills">  <li class="nav-item">  <a data-toggle="pill" href="#d1" class="nav-link">神奇四侠</a>  </li>  <li class="nav-item">  <a data-toggle="pill" href="#d2" class="nav-link">闪电侠</a>  </li>  </ul> |

**5.导航栏**

导航外面加一个div。

结构： div.navbar>ul.navbar-nav>li.nav-item>a.nav-link

响应式导航栏： div.navbar-expend-\*

\*：xl/lg/md/sm 可以在不同屏幕下，控制导航是横向显示(默认纵向显示)

|  |
| --- |
| 示例：<div class="navbar navbar-expand-md">  中屏以上的(中屏+大屏+超大屏)都横向显示，小屏纵向显示。 |

可以给ul设置bg-color来控制导航栏背景颜色

**6.折叠**

结构：button折叠没有要求样式

div.collapse  让div隐藏：display:none;

事件：

在button写自定义属性： data-toggle="collapse"

改变数据的目标是div，所以给div设置id

在button中指向这个id： data-target="#id"

|  |
| --- |
| 示例：  <button class="btn btn-danger" data-toggle="collapse" data-target="#demo">折叠</button>  <div id="demo" class="collapse">  Lorem ipsum dolor sit amet consectetur adipisicing elit.  </div> |

另外，a标签指向目标<a href="#id"></a>

**7．折叠导航栏**

结构

<div>

//1.不隐藏的标题

<a>Bootstrap中文网</a>

//2.小屏幕显示的折叠按钮

<button>

//画三条线

<span></span>

</button>

//3.折叠的导航

<div>

ul.navbar-nav>li.nav-item>a.nav-link

</div>

</div>

完成折叠

<button data-toggle="collapse" data-target="#content">

<div id="content" class="collapse">

<ul class="navbar-nav"></ul>

</div>

导航栏中的导航ul

ul.navbar-nav>li.nav-item>a.nav-link

导航栏，最外层div(ul外的div已经用来折叠，所以导航栏只能放在最外层div)

<div class="navbar navbar-expand-md bg-dark">

在最外层div中写类名修饰内部样式 .navbar-dark/light

注意，这个类对当前div没有任何修饰，它的作用是让div内部的brand和按钮上的三条线变成浅色(外深里浅，外浅里深)。

完成不折叠的a标签brand(商标)

<a href="#" class="navbar-brand">Bootstrap中文网</a>

完成按钮三条线

<span class="navbar-toggler-icon"></span>

按钮在md以上的屏幕不显示(.navbar-expand-md .navbar-togglerdisplay:none;)

<button class="navbar-toggler">

折叠的导航正常显示

<div id="content" class="collapse navbar-collapse">

|  |
| --- |
| 示例：  <div class="container">  <div class="navbar navbar-expand-md bg-dark navbar-dark">  <a href="#" class="navbar-brand">Bootstrap中文网</a>  <button data-toggle="collapse" data-target="#content" class="navbar-toggler">  <span class="navbar-toggler-icon"></span>  </button>  <div id="content" class="collapse navbar-collapse">  <ul class="navbar-nav">  <li class="nav-item">  <a href="#" class="nav-link">bootstrap3中文文档</a>  </li>  <li class="nav-item">  <a href="#" class="nav-link">bootstrap4中文文档</a>  </li>  <li class="nav-item">  <a href="#" class="nav-link">Less教程</a>  </li>  <li class="nav-item">  <a href="#" class="nav-link">jQuery API</a>  </li>  <li class="nav-item">  <a href="#" class="nav-link">网站实例</a>  </li>  </ul>  </div>  </div>  </div> |

**8.卡片**

结构：

<**div class="card"**>  
 <**div class="card-header"**></**div**>  
 <**div class="card-body"**></**div**>  
 <**div class="card-footer"**></**div**>  
</**div**>

|  |
| --- |
| 示例：  <div class="card">  <div class="card-header">  <h2>卡片标题</h2>  </div>  <div class="card-body">  <p>内容</p>  </div>  <div class="card-footer">  <p>版权所有</p>  </div>  </div> |

**9.手风琴(卡片和折叠)**

|  |
| --- |
| 结构：  <div id="parent"> ----->这个div内部只有一个card能够打开  //card1  <div class="card">  <div class="card-header">  <a href="#cols1" class="card-link" data-toggle="collapse"></a>  </div>  //在card-body外部多包裹一层，单写折叠的原因  //如果不写，效果会卡顿  <div id="cols1" class="collapse">  <div class="card-body"></div>  </div>  </div>  </div> |

一个#parent内部有多个card，只希望一个card显示，其它关闭；

需要在div.collapse中添加自定义属性data-parent="#parent"。

|  |
| --- |
| 示例：  <div id="parent">  <div class="card">  <div class="card-header">  <a href="#d1" class="card-link" data-toggle="collapse">路飞</a>  </div>  <div data-parent="#parent" class="collapse" id="d1">  <div class="card-body">  我是要成为海贼羊的男人  </div>  </div>  </div>  <div class="card">  <div class="card-header">  <a href="#d2" class="card-link" data-toggle="collapse">柯南</a>  </div>  <div data-parent="#parent" class="collapse" id="d2">  <div class="card-body">  真相只有一个  </div>  </div>  </div>  <div class="card">  <div class="card-header">  <a href="#d3" class="card-link" data-toggle="collapse">灰太狼</a>  </div>  <div data-parent="#parent" class="collapse" id="d3">  <div class="card-body">  我还会回来的  </div>  </div>  </div>  </div> |

**10.媒体对象**

结构：div.media>img+div.media-body

|  |
| --- |
| 示例：  <div class="media border p-3">  <img class="mr-3 rounded" src="../img/1.jpg" alt="">  <div class="media-body">  <h4>多媒体对象</h4>  <p>台湾歌手：张惠妹</p>  </div>  </div> |

**11.轮播图**

|  |
| --- |
| 第一部分，轮播图片  结构  1.最外层大包裹 div.carousel  2.所有图片放在一个div中 div.carousel-inner  3.每一张图片放在div.carousel-item  div.carousel> div.carousel-inner>div.carousel-item>img.w-100  事件  在外包裹中添加事件 data-ride="carousel"  在默认要显示的图片上一级的div中添加active  div.carousel-item.active |
| 第二部分，左右箭头  结构  a.carousel-control-prev>span. carousel-control-prev-icon  a.carousel-control-next>span. carousel-control-next-icon  事件  在a标签添加自定属性 data-slide="prev/next"  数据改变的目标是最外层的div.carousel #demo  <**a data-slide="prev" href="#demo" class="carousel-control-prev"**>  <**a data-slide="next" href="#demo" class="carousel-control-next"**>  重写boot的类，自定义样式  1.找到重写的类名  2.找到类名中需要重写的样式属性  3.重写  */\*重写左右箭头的宽高/背景色/位置\*/*  .**carousel-control-prev**,  .**carousel-control-next**{  **width**:4%;**height**:20%;  **background**:**rgba**(0,0,0,0.3);  **top**:40%;  **border-radius**:0.2**rem**;  }  .**carousel-control-prev**{**left**:0.2**rem**;}  .**carousel-control-next**{**right**:0.2**rem**;} |
| 第三部分，轮播指示器  结构  ul.carousel-indicators>li 默认显示的li加.active  事件  li中data-slide-to="图片的下标" 0/1/2/3  li中写data-target="#demo" 指定目标  重写boot样式  */\*重新指示器的宽高，圆角，背景\*/* .**carousel-indicators li**{  **width**:10**px**;**height**:10**px**;  **border-radius**:50%;  **background**:**#fff**; } .**carousel-indicators** .**active** {  **background-color**: **#0aa1ed**; } |

|  |
| --- |
| 示例：  <style>  /\* 重写左右箭头宽高 \*/  .carousel-control-prev,.carousel-control-next{  width:4%;height:20%;  background:rgba(0,0,0,0.3);  top:40%;  border-radius:2px;  }  .carousel-control-prev{left:0.2rem;}  .carousel-control-next{right:0.2rem;}  /\* 重写指示器的宽高，圆角，背景 \*/  .carousel-indicators li{  width:10px;height:10px;  border-radius:50%;  background:#fff;  }  .carousel-indicators .active{  background:#0aa1ed;  }  </style> |
| <div id="demo" class="carousel" data-ride="carousel">  <!-- 轮播图片 -->  <!-- carousel-inner溢出隐藏，只显示一张图片 -->  <div class="carousel-inner">  <!-- 每张图片都放在item中，carousel-item →display:none -->  <div class="carousel-item active">  <img class="w-100" src="../img/banner1.png" alt="">  </div>  <div class="carousel-item">  <img class="w-100" src="../img/banner2.png" alt="">  </div>  <div class="carousel-item">  <img class="w-100" src="../img/banner3.png" alt="">  </div>  <div class="carousel-item">  <img class="w-100" src="../img/banner4.png" alt="">  </div>  </div>  <!-- 左右箭头 -->  <!-- carousel-control-prev控制箭头位置 -->  <a data-slide="prev" href="#demo" class="carousel-control-prev">  <!-- 箭头样式 -->  <span class="carousel-control-prev-icon"></span>  </a>  <a data-slide="next" href="#demo" class="carousel-control-next">  <span class="carousel-control-next-icon"></span>  </a>  <!-- 轮播指示器 -->  <ul class="carousel-indicators">  <li class="active" data-slide-to="0" data-target="#demo"></li>  <li data-slide-to="1" data-target="#demo"></li>  <li data-slide-to="2" data-target="#demo"></li>  <li data-slide-to="3" data-target="#demo"></li>  </ul>  </div> |

**12.模态框**

模态框modal是覆盖在父窗体上的子窗体，

模态窗可以在不离开父窗体的情况下和用户有一些互动，提供一些交互的信息。

结构

<**button**>触发模态框</**button**>  
<**div id="demo" class="modal"**>  
 <**div class="modal-dialog"**>  
 <**div class="modal-content"**>  
 <**div class="modal-header"**></**div**>  
 <**div class="modal-body"**></**div**>  
 <**div class="modal-footer"**></**div**>  
 </**div**>  
 </**div**>  
</**div**>

事件

1.点击button，弹出模态框，在button中写

**data-toggle="modal" data-target="#demo"**

2.在modal-footer中创建button，点击关闭模态框

<**button data-dismiss="modal" class="btn btn-warning"**>关闭</**button**>

<**button class="close" data-dismiss="modal"**>**&times;**</**button**>

|  |
| --- |
| 示例：  <button data-toggle="modal" data-target="#demo" class="btn btn-danger">触发模态框</button>  <div id="demo" class="modal">  <div class="modal-dialog">  <div class="modal-content">  <div class="modal-header">  <h4>模态框的标题</h4>  <button data-dismiss="modal">&times;</button>  </div>  <div class="modal-body">  模态框的提示信息  </div>  <div class="modal-footer">  <button class="btn btn-warning" data-dismiss="modal">关闭</button>  </div>  </div>  </div>  </div> |

**一．其他组件**

**1.徽章**

基本类：badge

设置类型：badge-pill

设置颜色：badge-danger/warning/…

**2.巨幕**

div.jumbotron——巨大的内边距

**3.分页**

结构：ul.pagination>li.page-item>a.page-link

<**ul class="pagination"**>

<**li class="page-item disabled"**>

<**a class="page-link" href="#"**>上一页</**a**>

</**li**>

<**li class="page-item active"**>

<**a class="page-link " href="#"**>1</**a**>

</**li**>

</**ul**>

默认选中： li.active

禁用： li.disabled

分页条尺寸：ul.pagination-lg/sm

**4.面包屑导航**

按路径导航，使用面包屑导航。

div .breadcrumb>li .breadcrumb-item

|  |
| --- |
| 示图：首页 > 学习用品 > 笔记本 > 华为  示例：  <style>  .breadcrumb-item + .breadcrumb-item::before {  display: inline-block;  padding-right: 0.5rem;  padding-left: 0.5rem;  color: #6c757d;  content: ">";  }  </style> |
| <ul class="breadcrumb">  <li class="breadcrumb-item">  <a href="">首页</a>  </li>  <li class="breadcrumb-item">  <a href="">学习用品</a>  </li>  <li class="breadcrumb-item">  <a href="">笔记本</a>  </li>  <li class="breadcrumb-item">  <a href="">华为</a>  </li>  </ul> |

**5.进度条**

普通进度条：div.progress>div.progress-bar

通过设置内部的div宽度，来设置进度

带条纹进度条:progress-bar-striped

为进度条设置动画：progress-bar-animated

一个进度条的槽中，可以写多个进度

|  |
| --- |
| 示例：  <div class="progress">  <div class="progress-bar bg-success w-89"></div>  <div class="progress-bar progress-bar-striped bg-info w-50"></div>  <div class="progress-bar bg-dark progress-bar-animated progress-bar-striped w-50"></div>  </div> |

**二．SCSS 动态样式语言**

**1.CSS有几个缺点**

语法不够强大，没有变量和合理的样式复制机制，导致难以维护。

动态样式语言：为css赋予了动态语言的特性，极大的提高了样式语言的可重用性，可维护性。

常用的动态样式语言：

SCSS/SASS（scss兼容sass，scss更接近css的语法）

stylus

Less---boot3

SCSS/SASS/STYLUS/LESS都需要css的预处理器,

用一种专门的编程语言进行web页面的样式设计，在通过编译器转化为正常的css文件，以供项目使用。

**2.SCSS**

scss是一款强化的css辅助工具，

在css语法的基础上添加了变量，嵌套，混合，导入，函数等高级功能，

这些拓展使css更加强大且优雅。

**3.SCSS的作用**

有助于更好的管理样式文件，更高效的开发项目。

**4.SCSS的使用**

**（1）在服务器端使用**

必须安装nodejs解析器

安装scss

a.在线安装

npm install -g node sass

b.线下安装

使用下载好的安装包（把scss包中4个文件复制粘贴到nodejs文件夹下）

验证：在ws中alt+f12(vscode中ctrl+`)或者桌面win+r输入cmd

命令行输入node-sass -v，查看scss版本（nodejsv8.11及以上）

**（2）使用SCSS**

scss文件转换为css

单文件转换，命令行中：

node-sass scss 文件夹路径/01.scss css 文件夹路径/01.css

多文件转化：

node-sass scss 文件夹路径 -o css 文件夹路径

监听

单文件监听：（更改scss的同时，css自动相同更改）

node-sass -w scss 文件夹路径/01.scss css 文件夹路径/01.css

多文件监听：

node-sass -w scss 文件夹路径 -o css 文件夹路径

**5.SCSS的基本语法**

**（1）变量**

使用$符号标识变量。

变量的命名规范遵循css中选择器命名规范：

可以使用- \_ 数字，尽量使用见名知意。

变量值可以包含：颜色，数字，字符串，样式，其他变量

注意：

变量声明时，变量值可以引用其他变量；

变量定义在大括号外面，整个样式文件都可以使用；

定义在大括号内部，只能在当前规格块中使用。

声明重复变量，后声明的值会覆盖前面的值，尽量不要重复声明变量。

!default规则：

如果变量之前已经声明赋值了，就使用之前已经有的值；

如果该变量之前没有声明，使用当前值。

比如：

$color: #000;——使用此值。

$color: #fff !default;

**（2）嵌套规则**

注意：如果是a标签和伪类的嵌套，需要占位符 &

|  |  |
| --- | --- |
| 示例：  scss：  #contant{  div.top{  h1{color:#333;}  p{padding:10px;}  }  div.middle{background:#fff;}  } | scss:  a{  &:hover{color:#321;}  color:#123;  } |
| 生成的css：  #contant div.top h1 {  color: #333; }  #contant div.top p {  padding: 10px; }  #contant div.middle {  background: #fff; } | 生成的css：  a {  color: #123; }  a:hover {  color: #321; } |

**（3）群组嵌套**

|  |
| --- |
| scss写入：  nav,aside,div,footer{  a{color:#f00;}  } |
| css生成：  nav a, aside a, div a, footer a {  color: #f00; } |

**（4）属性嵌套**

|  |
| --- |
| scss写入：  div{  border{width:2px;style:solid;color:#000;}  } |
| css生成：  div border {  width: 2px;  style: solid;  color: #000; } |

练习：div#main，200\*200，bg为红色，鼠标悬停时，背景过渡为黄色，变为圆形

|  |
| --- |
| scss写入：  div#main{  width:200px;height:200px;  background:#f00;  transition:all 2s ease;  &:hover{  background:#ff0;  border-radius:50%;  }  } |
| css生成：  div#main {  width: 200px;  height: 200px;  background: #f00;  transition: all 2s ease; }  div#main:hover {  background: #ff0;  border-radius: 50%; } |
| html写入：  <body>  <div id="main"></div>  </body> |

**（5）导入scss文件**

局部scss文件：以下划线开头，不会直接被编译成css文件，需要在全局scss文件中导入。

全局scss文件：没有下划线开头的scss文件，会被解析成css文件。

导入语法：

比如：局部文件名为 \_abc.scss

全局导入： @import "abc";

在全局文件中可以导入多个局部文件。

|  |
| --- |
| \_close.scss写入：  $bg-color:#f00;  $fontSize:10px; |
| scss写入：  @import "close";  div{  background:$bg-color;  font-size:$fontSize;  } |
| css生成：  div {  background: #f00;  font-size: 10px; } |

**（6）混合器**

把一段scss代码放入混合器，在任何样式中调用，都可以实现这段代码，实现了代码的重用。

定义混合器：

@mixin 混合器名称{

scss代码

}

使用混合器：

p{@include 混合器名称}

带参数的混合器：

@mixin 混合器名称($a,$b,$c){

scss代码

}

使用混合器：

p{@include 混合器名称(10px,solid,#f00)}

**（7）继承**

一个选择器可以继承另外一个选择器定义的所有样式。

实现继承：是把多个选择器，拼成群组选择器来实现的继承。

@extend 类名;

|  |
| --- |
| scss写入：  .c1{  border:1px solid #f00;  border-radius:50%;  }  .c2{  @extend .c1;  width:100px;  height:200px;  } |
| css生成：  .c1, .c2 {  border: 1px solid #f00;  border-radius: 50%; }  .c2 {  width: 100px;  height: 200px; } |

**SCSS**

**一．运算**

1.数学

加减乘除模 + - \* / %

如果必要的情况下，会在不同的单位间转换（前提是scss支持转换，相对单位不支持转换）

**（1）加法**

+ 还可以用于字符串连接

如果用有引号字符串去连接无引号字符串，结果是有引号（"a"+b"ab"）

如果用无引号字符串去连接有引号字符串，结果是无引号（a+"b"ab）

**（2）减法**

由于变量名称会含有- 如：$width-100，用减法的时候，- 左右用空格，才能区分

**（3）除法**

在scss中，/ 通常起到分割数字的用途

在以下情况会被视为除法：

如果除号两边的值，或者值的一部分是变量，函数的返回值

如果表达式被小括号包裹

如果是其他运算表达式的一部分（含有其他运算）

**（4）运算表达式与其他值连接时**

用空格做连接。

示例：

margin:5px+10px/2px auto;

**（5）在有引号字符串中**

可以使用，#{}插值语句添加动态值。

示例：

content:"I ate #{100+2} baozi";

**2.颜色**

颜色值的元素加法时，分段计算的：红+红 绿+绿 蓝+蓝

rgba的运算，alpha的值必须一致才能运算。

如果计算的结果超过了ff，按ff取值。

**二．函数**

1.scss定义了好多函数，有些函数可以直接用css调用。

**（1）颜色函数**

rgba(r,g,b,a);

hsl(hue,saturation,lightness);

hue： 色调， 取值0~360， 3个色段，每120一个

saturation：饱和度， 取值0.0%~100%

lightness： 亮度， 取值0.0%~100%

**（2）数字函数**

round($value) 四舍五入

ceil($v) 向上取整

floor($v) 向下取整

min($v1,$v2,…)找出几个值中最小值

max($v1,$v2,…)找出几个值中最大值

random() 随机数

**（3）字符串函数**

to\_upper\_case() 将小写字母转换为大写

to\_low\_case() 将大写字母转换为小写

unquote($v) 删除字符串的引号

quote($v) 给字符串添加引号

**2.自定义函数**

@function 函数名($v1,$v2,…){

函数体;

@return 值;

}

|  |
| --- |
| 示例：  @function get-width($n1,$n2){  @return $n1 - $n2;  }  p{  width:get-width(50px,20px);  } |

**三．控制指令**(if-else if-else)

@if($n>=90){

函数体;

}@else if($n>=60){

函数体;

}@else{

函数体;

}

可以把条件的括号去掉

@if $n>=90 {}

@else if $n>=60{}

@else{}