# 第一阶段

## day01 server

### 一.网络 network

1. IP地址

用来区分网络中不同的设备（电脑、手机、打印机.....）;

分为两种：互联网中的IP地址、局域网中的IP地址

2. 查看IP地址

互联网：打开百度 --> 搜索IP

局域网：开始位置（单击右键）->运行 ->cmd ->ipconfig ->以太网适配器 以太网 -> IPv4地址

3. 查看设备之间能否通信 ping IP地址

4. 关闭防火墙

此电脑（左侧） ->网络->网络选项（顶部菜单）->网络和共享中心->Windows防火墙

依此关闭三个防火墙

### 二.服务器 server

提供多种服务的计算机

访问服务器：域名/IP地址

使用不同服务：端口

每种服务器还要有对应的：协议

1. 例子：浏览百度的网页

访问百度的服务器：www.baidu.com

网页服务的端口：443

使用网页服务协议：http/https(安全性高)

完整(URL)： http://www.baidu.com:443

访问学子商城： http://172.163.0.224:80

### 三、xampp

服务器端软件(套装)：包含有多款软件。

下载地址：https://www.apachefriends.org/download.html

Apache 提供网页服务的服务器(php用它)

MySQL 提供数据服务的服务器

FileZilla 提供文件服务的服务器

Mercury

Tomcat

### 四、FTP服务

FTP是一种协议，用于文件传输。

1.启动服务

点击xampp中FileZilla的start,启动服务;

2.进入服务面板

点击xampp中FileZilla的Admin;

3.添加用户并设置目录

点击菜单栏Edit->Users,进入用户面板

点击右侧Users下的add,添加用户名，可以自定义

点击添加的用户->点击Shared Folders->点击中间位置的add，来设置下载的目录

点击添加的用户->General->勾选password,设置密码。

4.连接FTP

ftp://172.163.6.71 (我自己的域名)

### 五、MySQL

是一款数据库软件，用于存储项目中的数据

1.启动：点击xampp中mysql后的start

2.使用界面化管理mysql数据库，需要启动Apache服务：点击xampp中Apache后的start

3.点击mysql后的Admin,进入到界面化管理

4.数据库的逻辑结构

server(服务器软件)->多个数据仓库->每个库中有多个数据表(用于保存不同类型的数据)->每个表中保存数据

\*\*英文翻译\*\*

server:服务器 network:网络 cmd:命令 全称command ipconfig:查询本机IP地址

ping 访问IP地址 Port(s) 端口 Admin 管理员 Edit 编辑 Users 用户们

add 添加 General 通用的 Remove 删除用户 Shared Folders 添加分享的文件

product 商品 price 价格

### 1.前端课程体系

(1)服务器端开发：项目导入/数据库/JS基础服务器开发

(2)前端核心技术：HTML/AJAX/CSS/bootstrap

(3)前端的进阶技术：JS高级/BOM&DOM/jQuery/VueJS框架

(4)H5移动端开发：H5/微信开发/HybridApp

(5)框架技术：Angular/React/Python

### 2、Windows中常用的快捷键

Window+E：打开资源管理器

Backspace：退格

Window+D：显示和隐藏桌面

Tab+Alt：向右切换窗口

Tab+Alt+Shift：向左切换窗口

Alt+F4：关闭窗口

Window+R：打开运行窗口 cmd(命令)/mspaint(画图)/calc(计算器)

F2：文件重命名

### 3、软件工程

IBM->dos->bill gates->microsotf->windows

程序设计阶段->软件设计阶段->软件工程

软件工程：研究和应用如何系统性的、规范化的、可定量的过程化方法开发和维护软件。

目标是在给定成本、进度的前提下、开发出具有有效性、可维护性、可重用性、可移植性的软件产品。

### 4、软件过程/软件生命周期

(1)软件的定义期

1>可行性研究阶段

技术、经济(资金)、时间、设备、政策法规、风俗、回报率

2>需求分析阶段

功能性需求、非功能性需求

(2)软件开发期

3>概要设计阶段

子系统、模块、各自的功能、模块间的接口

4>详细设计阶段

模块实现细节、属性、方法、算法

5>编码实现阶段

后端工程师：为前端提供页面中所需要的数据

前端工程师：把效果图转换成页面（html\css\js）

UI设计师：出效果图

6>测试阶段

(3) 软件维护期

7>项目部署阶段

部署到服务器

8>项目维护阶段

### 5、学子商城功能性需求

www.codeboy.com 前台 www.codeboy.com/admin/login.html 后台

前台/APP：

产品模块：首页、列表、详情

用户模块：注册、登录、用户中心、收藏夹

购物车模块：添加购物车、修改、删除、下单

后台：

产品模块：添加、修改、删除，列表

用户模块：列表、修改、删除、

订单模块：列表、修改

6、web项目中的服务器

服务器Server: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) 服务器的地址：域名/IP地址

(2) 所对应服务的端口号

(3) 提供该服务所使用的协议

以访问学子商城为例

http://www.codeboy.com:80

协议 域名/IP地址 端口

7、访问自己电脑创建的wed服务

启动web服务器 /webserver/start.cmd

使用浏览器去访问该服务

http://172.0.0.1:80 通过服务器的IP地址

http://localhost:80 通过服务器的域名

使用局域网的IP地址访问

ipconfig 查看电脑在局域网的IP地址

以太网适配器 以太网 IPv4 172.163.6.54

http://172.163.6.54:80

\*\*注释内容\*\*

服务器命令：

ping 使用域名找IP地址

## day01 数据库

### 1、项目中的数据

内存(临时存储数据不能长久保留)

特定的文件(exel、word)

第三方服务器(百度云、阿里云等)

数据库服务器(在浏览器中输入www.jd.com 发出请求 到服务器机房中的web服务器

web服务向文件服务器和数据库服务器查找所需求的数据，然后 响应给浏览器。)

### 2、数据库

数据库是按照一定的形式来组织、存储数据，目的是为了对数据操作————增删改查。

关系型数据库的逻辑结构server->databases->table->row->column

### 3、MySQL数据库

oracle分支：Mysql AB

martin分支：MariaDB

|  |
| --- |
| 服务器套装，包含多款服务器软件mysql,apache.....  下载：http://www.apachefriends.org/download.html  本地：E/ software/xampp-win32-5.6.32-0-VC11-installer.exe |

XAMPP软件

1）mysql 部署结构

服务器端：服务存储/维护数据——银行服务器机房

c:/xampp/mysql/bin/mysqld.exe 启动服务

确保3306端口不被占用

客户端：负责向服务器端发起增删改查——ATM机

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

2）使用客户端连接服务器端:

mysql.exe -h127.0.0.1 -P3306 -uroot -p

-h host 主机（IP地址/域名）

-P port 端口

-u user 用户

-p password 密码

简写形式： mysql -uroot

注意事项：连接的时候，不能在结尾加分号

### 4、mysql 常用管理命令

quit; 退出服务器连接

show databases; 显示服务器上当前所有的数据库

use 数据库名; 进入指定的数据库

show tables; 显示当前数据库中所有的数据表

desc 表名; 描述表中都有哪些列（表头）describe 描述

### 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，对数据执行增删改查等操作。

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，服务器执行一行。适用于临时性的查看数据。简写形式： mysql -uroot

（2）脚本模式：客户端把要执行的多行命令写在一个文本文件中，一次性的提交给服务器。适用于批量的操作数据。

操作方式： mysql -uroot < c:/xampp/../01.sql 回车

### 6、SQL语法规范

（1）每条SQL命令可以跨越多行，遇到英文分号作为结束。

（2）假如某一条命令出现语法错误，则此条语句以及后边所有的语句不会再执行。

（3）SQL命令不区分大小写，习惯上关键字用大写，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（/\*...\*/）,注释的内容不会被服务器所执行。

### 7、常用的SQL命令

（1）丢弃指定的数据库，如果存在的话 DROP DATABASE IF EXISTS xuezi;

（2）创建新的数据库 CREATE DATABASE xuezi;

（3）进入创建的数据库 USE xuezi;

（4）创建保存数据的表

CREATE TABLE student(

sid INT,

name VARCHAR(8),

sex VARCHAR(1),

score INT

);

注意最后一行不用再逗号。

(5)向数据表中插入数据 INSERT INTO student VALUES('1','tom','m','88')

(6)查询表中所有的数据 SELECT \* FROM student;

(7)修改数据 UPDATE student SET name='lucy',score='100' WHERE sid='2';

(8)删除数据 DELETE FROM student WHERE sid='3';

## day02 SQL

### 标准SQL命令分类

|  |
| --- |
| **DDL: Data Define Language 定义数据结构**  **CREATE/DROP/ALTER 创建 /丢弃/修改表**  **DML：Data Manipulate Languge 操作数据**  **INSERT/UPDATE/DELETE 插入/修改/删除**  **DQL:Data Query Language 查询数据**  **SELECT 查询**  **DCL:Data Control Language 控制用户的权限**  **GRANT（授权）/REVOKE（收权）** |

### 1、计算机存储字符

**解决mqsql存储中文乱码，使用utf-8编码形式**

**1.sql脚本文件另存为的编码**

**2.客户端连接服务器的编码（SET NAMES UTF8）**

**3.服务器端创建数据库使用的编码（CHARSET=UTF8）**

### 2、mysql中的列类型

**创建数据表的时候，指定的列可以存储的数据类型。**

**CREATE TABLE book(bid 列类型)**

**(1)数值类型——引号可加可不加**

**1> TINYINT 微整型，占1个字节，范围 -128～127**

**2> SMALLINT 小整型，占2个字节，范围 -32768～32767**

**3> INT 整型，占4个字节，范围 -2147483648～2147483647**

**4> BIGINT 大整型，占8个字节，范围太大了算不了**

**5> FLOAT（M、D） 单精度浮点型，占4个字节，范围3.4E38，范围比INT大的多，可能产生计算误 差。**

**6> DOUBLE（M、D） 双精度浮点型，占8个字节，范围比BIGINT大的多（天文运算中使用）。**

**7> 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。**

**工资6666.66 decimal(6，2)**

**8> BOOL布尔型，只有两个结果TRUE、FALSE（使用时不能加引号）,真正存储数据的时候，会自动变成1和0；也可能直接使用1和0；数据库的列类型自动变成TINYINT**

**TB -1024->GB -1024->MB -1024->KB -1024->BYTE(字节)-8->BIT(位)**

**(2)日期时间类型——必须加引号**

**1> DATE 日期型 '2018-12-31'**

**2> TIME 时间型 '14:37:30'**

**3> DATETIME 日期时间型 '2018-12-31 14:37:30'**

**(3)字符串类型——必须加引号**

**1> VARCHAR(M) 变长字符串,不会产生空间浪费，操作速度相对慢。M的最大值是65535.**

**2> CHAR(M) 定长字符串，可能产生空间浪费，操作速度快，用于存储手机号码，身份证号等固定长度字符。M的最大值是255.**

**3> TEXT(M) 大型变长字符串，M最多存2G。**

**CREATE TABLE ex(**

**id INT, /\*编号 \*/**

**age TINYINT, /\*年龄\*/**

**commentCount INT,/\*回复量\*/**

**price DECIMAL(6,2), #9999.99/\*价格\*/**

**phone CHAR(11),/\*手机\*/**

**article VARCHAR(3000),/\*详情\*/**

**sex BOOL, # 1 男 0 女/\*性别\*/**

**pubTime DATE /\*发布时间\*/**

**);**

### 3、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0-100.**

**（1）主键约束——PRIMARY KEY 以后用的最多的**

**声明了主健约束的列上不能出现重复的值，表中查询的记录会按照主键从小到大排序——加快查找速度，通常主键添加到编号列中。**

**注意：一个表中只能出现一个主键，主键列上不允许插入NULL。**

**NULL表示空，在插入数据时，无法确实要保存的值；例如：无法确定员工的工资，姓名，声明.....**

## day03 约束/查询

### 1.列约束

**（1）主键约束——PRIMARY KEY**

**（2）非空约束——NOT NULL 声明了非空约束的列上不允许插入NULL值**

**（3）唯一约束——UNIQUE**

**声明了唯一约束的列，不能插入重复的值 ，允许插入NULL，甚至多个NULL;一个表中可以出现多个唯一约束。**

**说明：NULL这个值比较特殊，它和任何值都不等，甚至和自身都不等。**

**练习：给笔记本家族表中，分类名称添加唯一约束，插入数据测试**

**（4）检查约束——CHECK**

**检查约束可以对插入的数据进行自定义验证**

**ex:CREATE TABLE student( score TINYINT CHECK(score>=0 AND score<=100) );**

**mysql不支持检查约束，会降低数据的插入速度。**

**（5）默认值约束——DEFAULT**

**可以使用DEFAULT关键字来声明默认值，有两种方式可以使用默认值**

**ex; INSERT INTO laptop\_family VALUES(50,'华硕',DEFAULT );**

**INSERT INTO laptop\_family (fid,fame )VALUES(60,'神州');**

**（6）外键约束——FOREIGN KEY**

**声明了外键约束的列上，取值必须在另一个表中的主键列上出现过，两者的列类型要保持一致，允许使用UNLL。**

**FOREIGN KEY(外键列) REFERENCES<引用>另一数据表（主键列）**

**ex: FOREIGN KEY (family Id ) REFERENCES （引用） laptop\_family**

### 2.mysql中的自增列

**AUTO\_INCREMENT:自动增长，假如一个列声明了自增列，无需手动赋值，直接赋值为NULL，会获取当前的最大值，然后加1**

**注意：1.自增列只适用于整型的列上**

**2.自增列允许手动赋值**

**练习：编写脚本文件01\_tedu.sql,创建数据库tedu，设置编码为utf-8；**

**创建保存部门数据的表dept,包含（did,dname）**

**10研发部 20市场部 30运营部 40 测试部**

**创建保存员工数据在表emp ，包含（eid,ename, sex, birthday, salary , deptld ）**

**使用恰当的列约束和自增**

**插入15条记录**

### 3.简单查询

#### (1)查询特定的列

**示例：查询所有员工的姓名，生日SELECT ename,birthday FROM emp;**

**练习：查询所有员工的编号，姓名，性别，工资SELECT eid,ename,sex,salary FROM emp;**

#### (2)查询所有的列 （有两种方式)

**1> SELECT \* FROM emp;**

**2>SELECT eid,ename,sex,birthday,salary,deptld FROM emp;**

#### (3)给列起别名

在起别名的时候，AS关键字可以省略

**示例：查询所有员工的姓名和工资，使用中文别名**

**SELECT ename AS 姓名, salary AS 工资 FROM emp;**

**练习：查询所有员工的编号，姓名，性别，生日，使用中文别名**

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

**练习：查询所有员工的编号的姓名，使用一个字母作为别名**

**select eid AS a,ename AS b from emp;**

**省略AS关键**

**select eid a,ename b from emp;**

#### (4)显示不同的记录/合并相同的记录

**练习：查询出都有哪些性别的员工 DISTINCT 有区别的，明显的，独特的**

**SELECT DISTINCT sex FROM emp;**

**练习：查询出员工都在哪些部门**

**SELECT DISTINCT deptId FROM emp;**

#### (5)查询时执行计算

**示例：计算2+3-4+5\*6/3**

**SELECT 2+3-4+5\*6/3 ;**

**练习：查询所有员工的姓名及其年薪salary工资\*12**

**SELECT ename,salary\*12 from emp;**

**练习：假如每个员工工资加500，年终奖加5000，查询所有员工的姓名及其年薪，给列起中文别名**

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

**SELECT ename 姓名,(salary+500)\*12+5000 年薪 from emp; （省略写法）**

#### (6)查询的结果集排序

**示例：查询所有部门表数据，结果集按照编号从小到大排序**

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

**示例：查询所有部门表数据，结果集按照编号从大到小排序**

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

**练习：查询员工所有的列，结果集按照工资降序排列**

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

**练习：查询员工所有的列，结果集按照年龄从小到大排列**

**SELECT \*FROM emp ORDER BY birthday DESC;**

**练习：查询员工所有的列，结果集按照姓名的升序排列(姓名会默认按升序排)**

**SELECT \* FROM emp ORDER BY ename ASC;**

**练习：查询员工所有的列，结果集按照工资降序排序，如果工资相同，按照姓名升序排列**

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

**练习：查询员工所有的列，要求女员工显示在前，如果性别相同按照生日降序排序**

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

|  |
| --- |
| **ORDER BY 可以按照数值，字符串，日期时间排序，默认是按照升序排列（ASC）** |

#### (7)条件查询

**示例：查询出编号为5的员工的所有列**

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

**练习：查询出姓名的king的员工的编号，工资，姓名，生日**

**SELECT eid,salary,ename,birthday FROM emp WHERE ename='king';**

**练习：查询出20号部门下员工所有的列**

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

**练习：查询出工资在6000以上员工所有列**

**SELECT \* FROM emp WHERE salary>=6000;**

|  |
| --- |
| **比较运算符：> < = !=（不等于） >= <=** |
|  |

**练习：查询出1993-1-1后出生的员工所有列**

**SELECT \* FROM emp WHERE birthday>='1993-1-1';**

**练习：查询出不在10号部门的员工所有列**

**SELECT \* FROM emp WHERE deptId!=10;**

**练习：查询出没有明确部门的员工所有列(NULL!=NULL)**

**SELECT \* FROM emp WHERE deptId IS NULL;（值为NULL）**

**练习：查询出有明确部门的员工所有列**

**SELECT \* FROM emp WHERE deptId IS NOT NULL;(不是NULL的值)**

**练习：查询出工资在6000以上的男员工所有列**

**SELECT \* FROM emp WHERE salary>=6000 AND sex=1;**

**练习：查询出工资在7000~10000之间员工所有列 （有两种写法)**

**1.** **SELECT \* FROM emp WHERE salary>=7000 AND salary<=10000;**

**2. SELECT \* FROM emp WHERE salary BETWEEN 7000 AND 10000;**

**BETWEEN....AND..... 在..和...之间**

**练习：查询出工资不在7000~10000之间员工所有的列s**

**SELECT \* FROM emp WHERE salary NOT BETWEEN 7000 AND 10000;**

**SELECT \* FROM emp WHERE salary<7000 OR salary>10000;**

**练习：查询出1990年之前和1995年之后出生的员工所有列**

**SELECT \* FROM emp WHERE birthday<'1990-1-1' OR birthday>'1995-12-31';**

**练习：查询出20号和30号部门员工所有的列（两种写法效果一样）**

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

**2. SELECT \* FROM emp WHERE deptId IN(20,30);**

**练习：查询出不在20号和30号部门员工所有的列**

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

|  |
| --- |
| **IS NULL / IS NOT NULL 是NULL /不是NULL**  **AND / OR 和/和**  **BETWEEN...AND.. / NOT BETWEEN...AND..... 在...之间/不在...之间**  **IN( ) / ONT IN( ) 在（ ）里面/不在（ ）里面** |

#### (8)分页查询

**假如查询的结果集有太多的数据，一次显示不完，可以使用分页显示。**

**分页查询需要两个条件：当前的页码，每页的数据量。**

**每页的开始=（当前的页码-1）\*每面数据量**

|  |
| --- |
| **SELECT \* FROM emp LIMIT srart,count;**  **start:每页的开始**  **count:每页数据量**  **LIMIT:限制的意思** |

**假设每页显示5条数据 算法：每页的开始=（当前的页码-1）\*每面数据量**

**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：假设每页显示6条记录，查询前3页**

**第一页： SELECT \*FROM emp LIMIT 0,6; 1-1\*6=0**

**第二页：SELECT \* FROM emp LIMIT 6,6; 2-1\*6=6**

**第三页： SELECT \* FROM emp LIMIT 12,6; 3-1\*6=12**

## day04 查询

### 1.简单查询——模糊条件查询

示例：查询出姓名中含有字母e的员工所有列（%任意位置)

SELECT \* FROM emp WHERE ename LIKE'%e%';

练习：查询出姓名中倒数第2个字符为e的员工所有列

SELECT \* FROM emp WHERE ename LIKE'%e\_';

练习：查询出姓名中以e结尾的员工所有列

SELECT \* FROM emp WHERE ename LIKE'%e';

|  |
| --- |
| SQL中提供了两个模糊查询的字符 |
| % 可以匹配任意个字符 >=0  \_ 可以匹配任意1个字符 =1  注意：在模糊条件查询中必须使用关键字LIKE，不能使用= |

### 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;

SELECT COUNT(\*) FROM emp; #推荐写法 \*代表所有

练习：使用员工的姓名计算员工数量

SELECT COUNT(ename) FROM emp;

练习：使用员工部门编号计算员工数量

SELECT COUNT(deptId) FROM emp;

练习：查询所有男员工的数理

SELECT COUNT(\*) FROM emp WHERE sex=1;

|  |
| --- |
| 聚合函数  函数就是一个功能体，需要提供若干个数据，产出某个结果。——饺子机  COUNT() 总数量  SUM() 总和  MVG() 平均  MAX() 最大  MIN() 最小 |

练习：查询出所有员工的工资总和 SELECT SUM(salary) FROM emp;

练习：查询出所有员工的平均工资

SELECT AVG(salary) FROM emp;

select sum(salary)/count(\*) from emp;

练习：查询出男员工工资最高的 select MAX(salary) from emp where sex=1;

练习：查询出年龄最大的员工 select MIN(birthday) from emp;

|  |
| --- |
| 分组查询：员能查询分组条件和聚合函数 GROUP BY分组条件 |

练习：查询出男女员工的平均工资，最大工资

select sex,AVG(salary),MAX(salary) from emp GROUP BY sex;

练习:查询出每个部门的员工数量，最高工资，最低工资

select deptId, MAX(salary) ,MIN(salary) from emp GROUP BY deptId;

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

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

|  |
| --- |
| 函数补充  YEAR() 获取日期中的年份  MONTH() 获取日期中的月份 |

练习：查询出1993年出生的员工所有例

select \* from emp where year(birthday)=1993;

练习：查询出5月份出生员工所有例

select \* from emp where month(birthday)=5;

#### (2)子查询

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

示例：查询出研发部员工所有的列

步骤1：查询出研发部的部门编号

select did from dept where dname='研发部';

步骤2：根据研发部的部门编号查询员工

select \* from emp where deptId=10;

综合：

SELECT \* FROM emp WHERE deptId=( select did from dept where dname='研发部');

练习：查询出比tom工资高的员工有哪些

select \* from emp where ename='tom',salary;

步骤1：查询出tom的工资

SELECT salary FROM emp WHERE ename='tom';

步骤2：查询出比6000高的

select \* from emp where salary>6000;

综合：

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

1：select year(birthday) from emp where ename='tom';

2：select \* from emp where year(birthday)=1990;

综合：

select \* from emp where year(birthday)=(select year(birthday) from emp where ename='tom');

#### (3)多表查询

示例：查询所有的员工姓名及其部门名称

SELECT ename,dname FROM emp,dept;

上面写法出现错误现象：笛卡尔积

多表查询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;

(不能查询出没有部门的员工，共15个员工，查询出14个员工)

|  |
| --- |
| 上述多表查询语法是SQL-92中的，无法查询出没有部门的员工，也无法查询出没有员工的部门  SQL-99中提出了新的多表查询语法 |

(1)内连接 INNER JOIN.....ON和SQL92结果一致

SQL-99: SELECT ename,dname FROM emp INNER JOIN dept ON deptId=did;

SQL-92: select ename,dname from emp,dept where deptId=did;

(2)左外连接 LEFT OUTER JOIN....ON

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

查询结果是左侧所有的记录都显示;OUTER 可以省略

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

INNER 内部 OUTER外部

(3)右外连接 RIGHT OUTER JOIN....ON

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

查询结果是右侧表中所有的记录都显示，OUTER可以省略不写。

#### (4)全连接 FULL JOIN......ON

显示左侧和右侧所有的记录——mysql不支持

UNION 合并相同的项

UNION ALL 不合并相同的项

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

UNION ALL

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

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

UNION

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

## day01 JS

### 1.JS概述

1.历史

1995年，JS出现在Nntscape浏览器中

1997年，JS提交给了ECMA，出现了ECMASscrip标准规范

ActionScript已经式微

2009年，JS遵循CommonJS规范，开始向服务器端发展

2.现状

既可以运行在客户端浏览器，也可以运行在服务器端NodeJS

3.javascript的特点

解释型语言，编译一行执行一行。

弱类型语言(弱类型语言，声明变量的时候不需要指定数据类型，后期可以存储任意的类型。)

基于对象 跨平台

4.应用场景

制作浏览器端的交互效果||创建移动App

创建web服务器，操作数据库等服务器端应用||2D绘图 (3D web gl全景看车)

### 2.JS的执行环境

1.浏览器自带的JS解释器

2.NodeJS下的JS解释器

3.执行JS代码

浏览器下执行方式<script src="..js"></script>

NodeJS执行方式 运行cmd加回车，然后输入：node +路径+回车

### 3.JS语法规范

1.区分大小写

2.每行代码结束分号可加可不加，习惯都加

3.分为单行注释//和多行注释/\*\*/

### 4.变量

变量是用存储数据的容器:只保存一个值的存储空间！(内存地址)

1.声明变量

var x=1; 在内存中创建空间，名称叫x，把数据1存放到空间x中。

2.变量的命名规则

变量的名称由字母，数字，美元符号，下划线组成的，不能以数字开头。

多个边词之间命名

userName 驼峰写法

user\_name 下划线写法

不能使用的关键字和保留字作为变量名(有很多最体看笔记)

3.声明变量未赋值

var num;

变量声明后未赋值，结果为undefined 未定义

弱类型语言，声明变量的时候不需要指定数据类型，后期可以存储任意的类型。

4. 一次性声明多个变量，多个变量之间用逗号隔开。

var a=1,b=2,c;

### 5.常量

一旦声明就不能重新赋值

例如：圆周率 const PI=3.14;

## day02 数据类型

数据分为原始类型和引用类型

### 1.原始类型

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

分为整型和浮点型 calc在运行中打开计算器

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，例如013=11

十六进制以0X开关，A~F代表10~15 不区分大小写 例如：0XFF=255 例如0X=10

浮点型分为普通小通和指数型小数

3.14E6=3140000 小数点向右移动六位

3.14E-6=0.00000314 小数点向左移动六位

2.字符串型

数据被引号包含就是字符串类型，不区分单双引号

查看任意一个字符的Unicode码

示例：console.log('苳'.charCodeAt()); //33523

typeof 检测数据类型

示例： var str2='2'; console.log(typeof str2);

3.布尔型

true/false 在程序中表示真或者假

一般用于是否的结果，例如是否注册，是否登录

4.未定义型 undefined

声明了变量未赋值，结果就是undefined

5.空——null

常和引用类型数据一起使用

### 2.数据类型转换

#### 1.隐式转换

数字+字符串 数字转换成字符串 ex: 1+'2' //'12'

数字+布尔型 布尔型转成数字 true=1 false=0

ex: 3+true //4 3+false //3

字符串+布尔型 布尔型转换成字符串 ex:'5'+true // '5true'

JS中+加号的作用：

1.执行加法运算

2.执行字符串拼接(必须有字符串参与)

隐式转换：减法-，乘法\*，除法

将运算符两端的数据转为数值型(自动调用Number)，如果转换失败，返回NaN(not a number),不是一个数字，任何值和NaN执行减乘除运算符都会返回NaN.

ex:查看以下程序的运行结果

var num1=3,num2=true,num3='tedu';

console.log(num1+num2+num3); //'4tedu'

console.log(num2+num3+num1); //'truetedu3'

console.log(num3+num1+num2); //'tedu3true'

#### 2.强制转换

1.将任意的数据强制转换成数值型 Number()

2.将数据转为整型 parseInt()

常用于将字符串或数字转为整型，其它的数据返回nan，如果要转换成字符串以非数字开头，也返回nan.

3.将数据转为浮点型 parseFloat()

和parseInt的用基本一致，只是转的数据是浮点型。

4.数值型和布尔型转为字符串型 toString()

ex: var num=10; num.toString(); //'10'

### 3.运算符

表达式：由运算符连接的操作数据，所组成的形式。

运算符分为算术运算符，比较运算符，逻辑运算符，位运算符，赋值运算符，三目运算符

1)算术运算符 + - \* / % ++ -- % 取余 ++ 自增+1 -- 自减-1

示例：1.console.log(num++); 先打印num的值，再执行自增

2.console.log(++num); 先执行自增，再打印num的值

ex: var num=3; console.log(num-- + --num); //4

先取出num的值3，然后执行自减，num的值变成了2.

先执行自减，num的值变成了1，也就是1。

2)比较运算符

> < >= =< !=(不等于) ==(等于) ===(全等于) !==(全不等于)

比较运算符会返回一个布尔型的值

==只是比较两个值是否相等

===不公比较值，还会比较类型是否一样。

如果数据类型不同，会发生数据类型转换。

3>'10' //false 3<'10' //true 在这里字符串用隐形转换将'10'转成数字10

'3'>'10' //true 两个字符串比较Unicode码3=51 10=49

3>'10a' 3<'10' 3=='10a' 结果都是false 因为'10a'转成了NaN

NaN和任何值比较都返回false

nan==nan 返回false (第一次复习时的疑问点)

3)逻辑运算符

或者( || ) 关联的两个条件只需要满足其一，结果是true,否则false

并且( && ) 关联的两个条件都是true，结果是true,否则false

非( ! ) 反向 !false=true !true=false

4)逻辑短路

&&并且 当第一个条件为false的时候，就不需要再执行第二个条件

||或者 当第一个条件为true的时候，就不需要再执行第二个条件

注意：逻辑短路无需关注最终结果是true还是false,重点是看是否会执行第二个条件(表达式)。

&& 第一个条件是true,才会执行第二个条件

var age=21;

age>=18 && console.log('成年人'); //第一个条件true，执行第二个条件，打印'成年人'

age<=18 && console.log('未成年'); //第一个条件false，因为不满足条件，则为空。

4.位运算符 calc在nodejs中打开计算器

在执行运算的时候，计算机会把数据转成二进制进行运算。

二进制遇二进一

1 10 11 100 101 110 111 1000 1001 1010

1 2 3 4 5 6 7 8 9 10

100110=100000+100+10=38

25=16+8+1=11001

16=10000 8=1000 1=1

按位与(&) 上下两位都是1，结果是1，否则是0.

按拉或(|) 上下两位含有1，结果是1，否则是0.

按位异或(^) 上下两位不同为1，相同为0.

按位右移(>>) 删除二进制的最后一位或者多位，每次缩小到原来的一半或者更多。

按位左移(<<) 在二进制的最后添加0，成倍增加。

用法是将数字转换成二进制，然后做位运算 示例：console.log(5^8);//13 在08\_binary.js

## day03 if-else

5.赋值运算符 = += -= \*= /= %=

6.三目运算符

由三个表达式或者数据组成的形式

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

### 三.浏览器端函数--对话框

alert() 弹出警示框也就是消息框

prompt() 弹出提示框也就是输入框，需要使用变量来接收输入的值；值的类型是字符串型。

### 四.流程控制

程序=数据+算法

程序分为顺序执行，选择执行，循环执行

#### 1. if语句 05\_if.js

语法：if(条件表达式){

语句;

}

如果条件表达式为true，会执行后边的语句

在条件表达式中，有一些值默认转成false, 例如 0,NaN,'',undefined,null

#### 2.if-else语句

07\_exercise.html 07\_exercise.js

语法：if(条件表达式){

语句1;

}else{

语句2;

}

如果条件表达式为true，执行语句1，否则条件表达式为false，执行语句2

如果if或者else后只用一行语句，则后边的大括号可以省略。

#### 3.if-else嵌套：用于判断多种情况

if(条件表达式1){

语句1;

}else....if(条件表达式n){

语句n;

}else{

语句n+1; //以上所有的条件表达式都是false,才会执行这一句

}

#### 4.switch-case语句

是一种特殊的分支语句，可以根据一个表达式的值，来选择执行不同的程序

switch(表达式){

case 值1： //如果表达式的值是值1

语句1;

break;

......

case 值n:

语句n:

break;

default:

语句n+1; //以上所有的表达式和值比较都是false

}

注意：在case中，表达式和值比较实用的是全等于比较，要求值和类型都满足才是true.

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既可以判断相等，也可以判断不等的情况，使用范围更广；

switch-case只能用于全等于的比较，结构上更为清晰，执行效率相对高。

## day04 循环

### 一.循环

循环：就是一遍又一遍执行相同或者相似的代码

循环的要素2个：1.循环条件：重复的次数 2.循环体：执行的代码

### 1.while循环

while(循环条件){ 循环体；}

break 可以结束任何形式的循环，break执行后，后边所有的循环体和循环条件都不再执行。

### 2.do-while循环

do{ 循环体; }while(循环条件);

不管循环条件是否为true,都会先执行一次循环体

### 3.for循环

for(初始值;循环条件;增量){

循环体

}

1/执行初始值

2/判断循环条件

3/如果循环条件是true执行循环体，是false结束循环

4/如果执行完循环体，执行增量

5/重新执行第二步

### 二、break和continue

break:结束循环，后续不会再执行其它的循环

continue:跳过本次循环，继续执行下一次循环

### 三、循环嵌套

while,do-while,for循环三者之间可以相互嵌套

## day05 函数

### 1.函数

parseInt()/parseFloat().....

分为系统函数和自定义函数

自定义函数

function:功能体，函数，用于封闭反复执行的代码，可以接受若干个数据，返回处理的结果。

1/创建普通的函数

function函数名称(){函数体; //封装的反复执行的代码}

调用:函数名称()

2/创建带有参数的函数

function函数名称(参数列表){ //形参=形式上的参数

函数体;

}

调用：函数名称(参数列表); //实参=传递的数据

参数列表：可以是0个或者多个，之间用逗号隔开，创建时参数称为形参，调用时的参数称为实参，实参会赋值给形参。

形参可以理解为是一个变量，只是未赋值，默认是undefined

3/带有返回值的函数

function函数名称(参数列表){

函数体;

return值;

}

调用：函数名称(参数列表)

return表示函数执行后，返回的结果;可以保存下来。

注意事项：

1.如果没有return和return后没有值，结果都是undefined

2.return后的所有代码都不会执行

### 2.变量作用域

#### 1.作用域

变量和函数的可访问范围，分为两种

函数作用域：在函数中使用var关键字声明的变量，只能在函数的内部访问。

全局作用域：在函数外部使用var声明的变量，可以在任意位置访问

注意：在函数内使用var声明的变量是局部变量，不使用var声明的变量是全局变量。

#### 2.变量声明提升

JS程序执行前，会将使用var声明的变量提升到所在作用域的最前边，赋值还是在原来的位置。

### 3.函数的作用域

函数的可访问范围，也分为全局作用域和函数作用域

(1)函数声明提升

和变量声明提升一样，JS程序执行前，会将function创建的函数提升到所在作用域的最前边，调用可以在任意合法的位置。

### 4.递归调用

在函数内部调用自身

递归要有结束的条件，结合return来使用。

## day06 函数

### 1.匿名函数

没有名称的函数 function(){}

(1)创建函数

函数声明方式：

|  |
| --- |
| function fn( ){ } |

函数表达式方式

|  |
| --- |
| var fun=function(参数列表){  函数体；  }  此时的变量fun就是函数名称 |

对比fun 和fun()的区别

fun()是函数的调用，返回函数的调用结果

fun是函数的名称，对应的是函数的整体结构

**函数声明和函数表达式创建函数的区别**

函数声明存在函数提升，调用可以在任意合法位置。

函数表达式不存在函数提升，必须先创建，再调用

练习：使用函数表达式，创建函数计算任意两个数字之间所有整数的和，并返回结果

(2)匿名函数自调用

目的：创建局部作用域，防止污染全局。

|  |
| --- |
| (function(形参列表){  函数体中的变量和创建的函数只能在内部使用。  })(实参列表); |

(3)回调函数

将匿名函数以实参形式传递，这个匿名函数就称为‘回调函数’；此时的形参名称，如果要调用匿名函数，只需要 形参名称()

|  |
| --- |
| 语法：  function fn(a){  //a就是传递的匿名函数的名称  a(); //执行传递的匿名函数函数体中的代码  }  fn(function(){......}); //回调函数 |

练习：创建函数，在函数中传递两个参数，实参都是以匿名函数的形式传递，在匿名函数中分别返回数字；计算两个数字相加的和，并打印出来。

### 2.全局函数

1)parseInt() 将数据转为整形

2)parseFloat() 将数据转为浮点型

3)encodeURI() 对一个URL中的中文进行编码

4)decodeURI() 对已经编码的URL进行解码

5)isNaN() 检测一个值是否为NaN 是->true 不是-> false

6)isFinite() 检测一个值是否为有限值 是----true 不是-----false

1/0 infinity 无穷，无限值

7)eval() 执行字符串表达式，

练习：使用弹出提示框输入一组运算，使用eval来执行运算，并打印出来 06\_eval.js

### 3.对象

对象是引用类型的数据

对象：是一组属性和方法(功能)的集合

哪些是对象？

一个手机：属性：颜色，品牌，尺寸大小....；方法有：打电话，玩游戏，看视频，发短信

一个汽车：属性：颜色，品牌，类型；方法：代步，拉货，载客，撞人

一个人：属性：性别，姓名，年龄；方法：说话，工作，写代码....

总结：万物皆对像

(1)JS中的对象 也有属性和方法

分为三大类：

1.内置对象：由JS提供的 (有17个) 明天学

2.宿主对象：根据JS不同的执行环境来划分，以后学

3.自定义对象：自己创建的对象 今天学

(2)自定义对象

1.对象字面量 今天重点

2.内置构造函数 今天重点

3.自定义构造函数 JS高级后期讲

### 4.使用对象字面量创建对象 07

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，最后一组属性可以不逗号，加上也不是错误的

属性名中引号可加可不加，如果含的特殊字符必须加引号

|  |
| --- |
| {属性名:属性值,......} |

练习：创建一个人对象，添加姓名，性别，年龄

练习：创建一个员工对象，包含编号，姓名，性别，生日，工资，所属部门编号

### 5.访问对象的属性

对象.属性名

对象['属性名']

可以添加之前不存在的属性;

如果属性不存在，属性值为undefined

练习：创建一个电脑对象，包含的属性有屏幕尺寸，品牌，颜色，内在大小；获取品牌的属性值，修改颜色的属性值，添加产地属性

### 6.使用内置构造函数创建对象 08

new Object(); 创建一个空的对象 //括号里留空就可以

需要单独添加每一个属性

对象['属性名']='属性值'；

对象.属性名='属性值';

练习：使用内置构造函数创建用户对象，包含编号，用户名，密码，邮箱，手机。

### 7.遍历对象的属性 09

访问所有的属性(一遍又一遍的把所有的属性都找完)

|  |
| --- |
| for(var key in 对象){ //key 意思是对象中所有的属性名  key 对象中所有属性名  对象[key] 通过属性名找对应的属性值  } |

练习：创建对象，包含若干成绩属性，遍历对象中属性获取总成绩。

提前声明sum用于保存总和。

### 8.对象中的方法

方法也称为成员方法，对应的是一个匿名函数

|  |
| --- |
| var person={  name:'tom',  say:function(){  this.name //this指代当前所在的对象  }  };  person.say(); //调用方法 |

课后任务：

1.复习今天内容，整理思维导图

2.练习

创建一个圆对象，包含半径，圆周率属性；添加获取面积和获取周长的方法，分别返回对应的值。

尝试可以构造函数创建对象，并添加属性和方法。

3.预习JS中的数组(创建，遍历，API);

## day07 数组

### 1.检测对象中是否含有某个属性 02

var person={

name:'tom',

sex:'男',

age:18

};

比较一个属性的值是否为undefined

1)对象.属性名===undefined true=不存在 false=存在

ex: console.log(person.phoen===undefined);//true

console.log(person.age===undefined); //false

查看是否含有某个属性

2)对象.hasOwnProperty('属性名'); true=存在 false=不存在

ex:console.log(person.hasOwnProperty('name')); //true存在

console.log(person.hasOwnProperty('phoen')); //false 不存在

查看某个属性是否存在于对象中

3)‘属性名’ in 对象 true=存在 false=不存在

ex: console.log('phone' in person); //false 不存在

console.log('sex' in person); //true 存在

对比了栈内存和堆内存数据调用时的变化 03

var book={

id:1005,

titile:'javascript高级程序设计',

price:89

};

把book变量中存储的数据赋给了book2变量

把book存储地地址赋给了book2

现在book和book2指向同一个对象

var book2=book;

使用不同的变量去修改同一个对象

修改book中的属性

book.price=86;

console.log(book2); 打印结果 { id: 1005, titile: 'javascript高级程序设计', price: 86 }

修改book2的属性

book2.title='javascript权威指南';

console.log(book);

打印结果 { id: 1005,

titile: 'javascript高级程序设计',

price: 86,

title: 'javascript权威指南' }

原始类型存储 是为了做对比 栈内存和堆内存 数据调用时的变化

var a=1;

var b=2;

var c=a;

c=3;

console.log(a); //1

数组

数组是由多个元素组成的集合，每一个元素就是一个数据

### 2.创建数组 04

(1)数组字面量

[元素1，元素2，元素3....]

例如：创建数组，保存若干个员工的姓名

var emps=['tom','jerry','lily','kate','sasum'];

console.log(emps);

打印结果 //[ 'tom', 'jerry', 'lily', 'kate', 'sasum' ]

练习：创建数组，包含多个商品的名称；

练习：创建数组，包含多个城市名称

(2)访问数组中的元素

数组[下标] 下标是从0开始，第一个元素的下标就是0

练习：创建数组，包含多个商品的名称；

练习：创建数组，包含多个城市名称

var pen=['毛笔','铅笔','铅笔','圆珠笔'];

var laptop=['小米','Apple pro','联想e450','戴尔灵越'];

var city=['巴黎','伦敦','罗马','马德里'];

下标 0 1 2 3

console.log(pen,laptop,city); //结果全部显示

通过下标来获取元素,下标就是[],想获取什么就在中括号里写相对应的数

console.log(city[2]); //罗马

修改数组，重新赋值

city[2]='米兰';

console.log(city); //打印结果：[ '巴黎', '伦敦', '米兰', '马德里' ]

添加数组

city[4]='布鲁塞尔';

console.log(city); //打印结果：[ '巴黎', '伦敦', '米兰', '马德里', '布鲁塞尔' ]

(3)内置构造函数 04

new Arrary(元素1，元素2....);

new Arrary(3); 初始化元素个数为3，也可以添加更多个元素

练习：创建数组，保存若干个课程名称；

练习：创建数组，初始化长度为5，添加篮球声上的五个位置

(4)数组的长度 04

数组.length 获取数组中元素的个数

作用：1.可以在数组的末尾添加新的元素

数组[数组.length]=值;

练习：创建一个空数组，要求使用数组长度添加若干个国家名称

2.数组的分类 05

数组分为索引数组和关联数组

索引数组以0以上的整数作为下标

关联数组以字符串作为下标，只能单独的添加元素

### 3.遍历数组 05

(1)for-in

|  |
| --- |
| for(var key in 数组){  key 要遍历的下标  数组[key] 下标对应的元素  }  既可能遍历关联数组，也可以遍历索引数组。 |

练习：创建数组，包含有多个学生的成绩，获取总成绩

var sum=0;

var score=[81,93,95,78,66];

for(var key in score){

console.log(key,score[key]); //可以得到属性名+值

sum+=score[key];

}

console.log(sum);

(2)循环 06

|  |
| --- |
| for(var i=0;i<数组.length;i++){  i代表下标 0~数组长度-1  数组[i] 代表下标对应的元素  }  只能遍历索引数组 |

练习：创建数组，包含有多个姓名，把tom全部改成‘汤姆’。06

1.创建数组

2.遍历得到每一个元素

3.判断每个元素是否为tom，如果是，重新赋值为‘汤姆’

4.在遍历之后，打印数组

var names=['lily','lucy','lee','tom','mei','tom'];

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

{

if(names[i]=='tom'){

names[i]='汤姆';

}

}

console.log(names); //打印结果：[ 'lily', 'lucy', 'lee', '汤姆', 'mei', '汤姆' ]

练习：创建数组，包含多个姓名，查询tom出现的次数 06

1.创建数组

2.遍历数组

3.初始化变量的次数为0

4.判断每个元素是否为tom，如果是，次数加1

5.打印变量

var names=['tom','lucy','tom','king','tom'] //遍历数组得到每个元素

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

{

if (names[i]=='tom')

{

count++;

}

}

console.log(count); //3

练习：创建数组，包含多个数字，获取这组数字的最大值 06

1.创建数组 23 85 74 46 95 100

2.声明一个变中的元素量max用于保存最大值，默认把第一个元素放进去

3.遍历数组

4.使用之前声明的变量和每个元素比较，如果变量小于任何一个元素，则把该元素放入到变量中。

5.打印声明的变量

练习：创建函数getAvg，传递一个参数(一组工资)，返回工资的平均值 06

1.实参就是一个数组，包含多个工资；

2.函数体中，计算元素的和，求平均(平均=总和/元素长度)

### 4.数组API(方法) 07

( Application Programming Interface )

API 应用程序编程接口，预定义好的一些方法或者函数

例如：var arr=['tom','jerrys','king'];

var arr2=['ling','jek','lee'];

var arr3=['luky','jeo','mei'];

1\*toString() 将数组中的元素按照逗号分组合成字符串

console.log(arr.toString()); //tom,jerrys,king

2\*join() 将数组中的元素按照指定字符组合成字符串，默认是逗号，也可以是-，|,''空字符等

console.log(arr.join('-')); //tom-jerrys-king

3\*concat(arr2,arr3) 拼接多个数组

console.log(arr.concat(arr2,arr3));

//[ 'tom', 'jerrys', 'king', 'ling', 'jek', 'lee', 'luky', 'jeo', 'mei' ]

4\*slice(start,end) 截取数组中的元素，start开始的下标，end结束的下标，不包含end本身，如果是负数表示倒数，返回一个数组。

var arr=['javascript','html','css','ajax','vue','react','angular'];

console.log(arr.slice(1,4)); //[ 'html', 'css', 'ajax' ]

console.log(arr.slice(-3,-1)); //[ 'vue', 'react' ] 倒数第三个到倒数第二

练习：创建数组a~g，每个字母是一个元素，分别截取c,d,f,b把截取的结果拼接成一个新的数组。

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

var arr1=arr.slice(2,4); //cd 不包含4

var arr2=arr.slice(-2,-1); //f 不包含-1本身

var arr3=arr.slice(1,2); //b 不包含2

console.log(arr1.concat(arr2,arr3)); //[ 'c', 'd', 'f', 'b' ]

5\*splice(start,count,value1,value2....) 删除数组中的元素，start开始的下标，count删除的数量，value删除后插入的元素；返回删除的元素，原数组会发生变化。

练习：创建数组a~h，每个字母是一个元素，删除d,e,替换f为m,在下标为1的位置插入字母z.

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

console.log( arr.splice(3,2) );

console.log( arr.splice(3,1,'m') );

console.log( arr.splice(1,0,'z') ); //删除的数字为0，插入z

console.log(arr);

6\*reverse() 翻转数组中的元素 ()里什么都不用写

var arr=['a','b','c','d'];

console.log( arr.reverse() ); //[ 'd', 'c', 'b', 'a' ]

7\*sort() 对数组中的元素进行排序，默认按照Unicode码，从小到大排序

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

练习：

var arr=[7,23,59,8,46];

console.log( arr.sort() ); //[ 23, 46, 59, 7, 8 ] 默认按照Unicode码排序

console.log( arr.sort( function(a,b){ 回调函数——匿名函数以实参传递,

return a-b; //[ 7, 8, 23, 46, 59 ] 从小到大

return b-a; //[ 59, 46, 23, 8, 7 ] 从大到小

} ) );

课后任务

1.复习今天内容，整理思维导图

2.练习：使用遍历数组翻转数组中的元素

[a,b,c,d]->[d,c,b,a]

使用冒泡排序将数组中的数字从小到大排序

循环嵌套

3.明天学习内容：字符串中的API

## day08 API

### 1.数组API方法 02 array api.js

pust() 往数组末尾添加元素，返回数组的长度

pop() 删除数组末尾的元素，返回删除的元素

unshift() 往数组的开关添加元素，返回数组的长度

shift() 删除数组开头的元素，返回删除的元素

### 2.二维数组 03 array.js

数组中的每一个元素也是数组

var arr=[ [],[],[].... ]

访问二维数组中的元素， arr[下标][下标]

### 3.字符串操作 04 string.js

包装对象：目的是让原始类型的数据也可以像引用类型一样，具有属性和方法。

JS提供了3种包装类型：String(字符串),Number(数字),Boolean(布尔 )

将任意数据包装成字符串

new String() 返回对象object，使用和字符串没有区别

String() 返回字符串 ‘true’

(1)转义字符——\ 04

作用：转换字符的意义

用法：在要转义的字符前加\

\' 将引号转义成普通的字符

console.log('It\'s a dog'); //It's a dog

\n 将字符n转义成换行符

console.log('hello \nworld'); //hello 换行 world

\t 将字符t转义成制表符(tab键效果)

console.log('a\tb'); //a b

练习：打印出现 welcome to chi\na console.log('welcome to chi\\na');

(2)字符串API 05

ex: var str='javascript'; //字面量

1)length 获取字符串的长度

console.log( str.length ); //10

2)charAt() 获取下标对应的字符，也可以使用数组型式

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

*字符串[下标]*

*console.log( str[2] ); //v*

练习:遍历字符串，打印出每一个字符。

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

{

//console.log(str[i],str.charAt(i)); //合起来打

console.log(str[i]); //第一种

console.log(str.charAt(i)); //第二种

}

3)charCodeAt() 获取某个字符的Unicode码

console.log( 'a'.charCodeAt() ); //97

4)indexOf( value,start ) 查找某个字符串的下标，value要查找的字符串， start开始的下标，默认是0，如果找不到返回-1

5)lastIndexOf( value ) 查找某个字符串最后一次出现的下标，找不到返回-1

练习：声明变量保存字符串，检测该字符串中是否含有@,如果有打印‘合法邮箱’，否则打印‘不合法邮箱’。

6)toUpperCase() 将英文字母转为大写

7)toLowerCase() 将英文字母转为小写

练习：声明变量保存4个英文字符，无限循环弹出提示框，输入验证码(不区分大小写)，如果输入正确结束循环。 06\_code.js

8)slice( start,end ) 05截取字符串，start开始的下标，end结束的下标，不包含end本身，如果为负数表示倒数，end为空表示截取到最后。

练习：声明变量保存邮箱，分别截取邮箱的用户名和域名tom123@163.com

步骤：查找@的下标

0~下标 用户名

下标的下一位~最后 域名

9)substr( start,count ) 05 截取字符串，start开始的下标，count截取的长度，如果count为空截取到最后，start为负数表示倒数。

练习：声明变量保存身份证号，截取出生年月日和性别，打印‘1998年06月20日 性别女’

110236199806202589

10)substring( start,end ) 05 截取字符串，start开始的下标，end结束的下标，如果end为空截取到最后，如果下标为负数，自动转成0.(感觉这个更加智能)

对比slice和 substring

slice中下标允许使用负数，substring会自动将负数转为0，slice中下标的顺序start小于end，substring中下标大小不分顺序

练习:将一个英文单词的首字母转大写，其余的字母转小写 heLLo -> Hello 05

11)split(sep) 05将字符串按照指定的字符分隔成数组，sep是分隔时指定的字符

练习：使用split分隔邮箱，获取用户名和域名 tom123@163.com

### 4.匹配模式(目前只需要掌握，后期还要学) 07

作用：用于查找、替换字符串

match(value) 用于查找匹配的字符串，返回所有满足条件的元素，组成数组/china/ig

i-> ignore 忽略大小写

g->global 全局查找

search(value) 用于查找满足条件的第一个字符串的下标，如果找不到返回-1

replace(value1,value2) 查找并替换，value1 要查找的字符串；value2是要替换的字符串，也可以使用正则表达式 /china/ig \*\*屏敝

### 5.Math对象 08

(没有写完还有很多，可以自己在网上查找)

Math 对象不需要使用new创建，可以直接使用

PI 获取圆周率

abs() 取绝对值

floor() 向下取整

ceil() 向上取整

round() 四舍五入取整

max() 取一组数字的最大值

min() 取一组数字的最小值

pow(x,y) 求x的y次幂

random() 取随机 0~1 >=0 <1

练习：随机产生0~9之间的一个整数

课后任务

1.复习今天内容，整理思维导图

2.练习：

创建数组，包含有若干个元素，每次随机取出1个元素；数组有10元素 随机取0-9，作为下标。

创建数组，包含所有的英文字母(大写，小写)和数字(0-9)，随机取4个元素(循环4个)，放入一个新数组push。如何不重复？

3.预习Date对象

## day09 对象

### 1.Date对象 02

作用：用于对日期时间进行存储和计算

(1)创建日期时间

1)new Date(‘2019/4/18 10:30:40’);

2)new Date(2019,3,18,10,30,40); 第二个参数月份范围0~11 2019/4/18

3)new Date(); 存储当前的系统时间

4)new Date(15000000000); 存储的是距离计算机元年的毫秒数，

对应的时期时间2017-07-14T02:40:00.000Z

(2)获取Date对象中的日期时间 02

1)getFullYear()

2)getMonth 范围0~11

3)getDate

4)getHoures

5)getMinutes

6)getSeconds

7)getMillise conds

8)getDay 范围0~6

9)getTime 获取计算时元年毫秒数

练习：创建Date对象保存当前系统的日期时间；根据对象打印 2019年4月18日星期四

提示思路：星期是用数组对象做出来的[日，一，二，三，四，五，六]

(3)将对象转为本地字符串 03

1)toLocaleDateString() //年-月-日

2)toLocaleTimeString() //时：分：秒

3)toLocaleString() //年-月-日 时：分：秒

(4)修改Date对象中的日期时间 03

1)setFullYear

2)setMonth

3)setDate

4)setHours

5)setMinutes

6)setMilliseconds

7)setTime 修改后，可能会影响其它的日期时间

练习：创建Date对象，保存员工的入职时间‘2019-4-18’；3年和合同到期，计算到期时间，合同到期前一个月续签合同，如果是周末，提前到周五，要求计算续签时间。

入职时间：2019-4-18

到期时间：2022-4-18

续签时间：2022-3-18

### 2.Number 对象 04

将数据转为一个数值型，并且返回一个数值

1)new Number() 将数据转为数值型，返回对象

2)Number() 将 数据转为数值型，返回数值

3)Number.MAX\_VALUE 获取Number的最大值

4)Number.MIN\_VALUE 获取Number的最小值

5)toFixed(n) 保留小数点后n位

6)toString(n) 将数值转为字符串，n转换的进制

### 3.Boolean 对象 (布尔) 05

将任意的数据转为布尔型

new Booleadn() 将数据转为布尔型，返回对象

Booleadn() 将数据转为布尔型，返回布尔型

！！值 隐式将数据转为布尔型

toString() 将布尔型数据转为字符串

### 4.错误处理 06

1)Syntax Error:语法错误，错误的使用了中文，缺少括号等，出现后所有的代码都不执行

2)ReferencesError: 引用错误，使用了未声明的变量，属于运行时的错误，会影响后续代码的执行

3)TypeError:类型错误，错误的使用了数据，例如把变量或数组当函数使用，属于运行时的错误，会影响后续代码的执行。

4)RangeError:范围错误，参数的使用超出了范围；属于运行时的错误，影响后续代码的执行。

5)尝试执行代码

|  |
| --- |
| try{  尝试执行的代码，可能产生错误 }catch(err){  err捕获到的错误  具体处理错误的内容 } |

练习：声明一个变量add未赋值；尝试调用add函数；如果执行错误，给add赋值一个匿名函数，然后再调用add

### 5.ES6 07

ECMAScript6 ECMA脚本6

ES6标准入门 作者：阮一锋 http://es6.ruanyifeng.com

#### (1)块级作用域

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

块级作用域：{ } for while do if

目的：防止污染全局

#### (2)箭头函数

是回调函数的另一种写法，和之前的匿名函数不完全一样

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

如果箭头函数的函数体中只有一行代码，并且是return形式的，可以简化为sort( (a,b)=>a-b )

练习：创建函数add，传递两个参数，每个参数都是回调函数，在回调函数中返回一个数字，在函数add中计算两个数字相加的和。

===================2019/4/19 (nodejs day01笔记)=====================

1.ES6 08

#### (1)函数中的参数

ES6允许为参数设置默认值，如果没有传递实参，会自动调用形参的默认值。

#### (2)模板字符串

|  |
| --- |
| ` 一组字符串，可以直接写Js ${js表达式}` |

练习：创建一个对象，保存一个员工的对象(姓名，性别0/1，工资，生日)，使用模板字符串打印员工的信息。性别0显示女，1显示男

课后任务：

1.复习今天内容，整理思维导图

2.练习：代码在09

计算2019年5月1日10点30分0秒距离2019年12月25日相差的天数，小时，分钟，秒

3.复习什么是服务器，预习nodejs中的模块

## **day01 nodejs**

2019/4/19

### 1.nodejs概述

nodejs基于谷歌v8引擎，(速度和效率高)，运行在服务器端的语言，基于js

1.对比js和nodejs

(1)js运行在客户端浏览器，存在多个浏览器，容易产生兼容性的问题;nodejs在服务器端只有一个环境，不存在兼容性问题。

(2)两者都有内置(ES)对象，自定义对象，宿主对象(根据执行环境的不同划分)

(3)js主要是用于网页中的交互效果，nodejs用于服务器的操作，例如web服务器创建，数据库操作，文件操作等。

nodejs.org 英文官网

nodejs.cn 中文网，不是官网所以更新没有官网快

2.node.js的特点

单线程逻辑处理 好处：处理速度快，效率高，节省内存资源

非阻塞的异步I/O处理 操作速度快

支持数万个并发连接 浏览承载量是PHP的两倍

事件驱动编程简单，避免过度设计

简单，避免过度设计

无锁机制，不会产生死锁 目前不讲因为我们理解不了

3.nodejs的应用场景

基于社交网络的大规模web应用

nodejs不适合CPU密集型的应用

例如：递归，数据加密解密，数据挖掘和数据分析

4.nodejs的执行方式

脚本模式 node+空格+把文件拖拽过来 c:/......

交互模式 node 回车 直接进入交互模式

两次ctrl+c 或者 .exit 退出交互模式

### 2.全局对象

全局作用域下的变量就是全局对象下的属性，全局对象下的函数就是全局对象下的方法，可以使用全局对象来访问

nodejs:global nodejs下全局对象 03

在交互模式下，声明的变量和创建的函数都属于全局下的，可以使用global来访问；

在脚本模式下，声明的变量和创建的函数都不属于全局下的，一个文件默认会创建一个独立的作用域，也可以叫做文件作用域或者叫模块作用域，好处是可以防止污染全局。

js:window 浏览器下全局对象 04

在浏览器下，文件中声明的变量或者创建的函数都属于是全局作用域下的，会污染全局。

(1)console对象——控制台 05

global.console.log() 打印日志

global.console.info() 打印消息

global.console.warn() 打印自定义的警告

global.console.error() 打印自定义的错误

global.console.time('自定义字符串') 开始计时

global.console.timeEnd('自定义字符串') 结束计时

\*\*自定义字符串前后要保持一致

练习：计时查看for,while,dowhile循环一万次的耗时

(2)process对象——进程

process.arch 查看当前CPU的架构

process.platform 查看当前的操作系统

process.env 查看当前系统的环境变量

process.version 查看当前nodejs的版本号;(在脚本模式下查看web>node -v)

process.pid 查看当前的进程编号

process.kill() 结束某个编号的进程

(3)Buffer对象——缓冲区 06

一块用于临时存储数据的内存区域，(一旦关机或者关闭应用就会消失)可以存储文件数据，网络上存储的资源(视频，在线直播......)

创建Buffer

Buffer.alloc(5,'abcde') 5

将buffer数据转为字符串

buf.toString()

示例：

创建buffer空间 5表示可存储的字符数

var buf=global.Buffer.alloc(5,'abcde');

console.log(buf); //<Buffer 61 62 63 64 65> 转成uincode码

console.log(buf.toString()); //abcde 转成字符串型

3.模块

模块就是一个独立的封闭的功能体

在nodejs下模块分为三种：自定义模块、核心模块(官方提供的)、第三方模块

npmjs.com 模块网站，全世界都可以往上面发模块

(1)自定义模块 07\_1.js 07\_2.js

在nodejs下，任意一个文件都是一个模块，文件中的代码默认被一个构造函数所包含。

|  |
| --- |
| (function(exports,require,module,\_filename,\_dirname){  //程序员编写的代码  }) |
| \_\_filename 当前模块的完整路径和文件名称(两个下划线)  \_\_dirname 当前模块的完整路径(两个下划线)  require() 引入一个模式  module 指代当前的模块对象  module.exports 当前模块导出的对象，包含供其它模块使用的属性和方法 ,建议用它  exports ===module.exports 但是它们没有任何关系，老师不建于用它 |

练习：在07\_2.js声明变量b，创建函数fn,导出变量和函数

课后任务：

1.复习今天内容，整理思维导图

2.练习：

创建两个模块main.js(主模块)，circle.js(功能模块)，在功能模块中创建两个函数，传递1个参数半径，计算圆的周长和面积，导出这两个函数；在主模块中引入功能模块，并调用两个方法。

3.预习：核心模块下的querystring url fs

## day02 模块

2019/4/22

### 1.模块分类

|  |  |  |
| --- | --- | --- |
|  | 以路径开头 | 不以路径开头 |
| 文件模块 | require('./circle.js')  常用于用户自定义的模块，如果后缀名为js，则可以省略， | require('url')  用于引入官方提供的核心模块 |
| 目录模块 | requitr('./02\_2');  在02\_2目录下寻找  package.json中寻找main属性对应的文件，找不到才会去引入index.js; | require('04\_2')  自动到当前目录下的node\_modules中寻找目录模块04\_2，如果找不到会到上一级目录下寻找，直到顶级目录。常用于第三方模块 |

练习1：创建模块03\_1.js，引入当前目录下的03\_2目录模块；在03\_2下创建test.js，导出一个函数fn(计算两个数字相加)，在03\_1.js调用。

练习2：在05目录下创建模块05\_1.js,引入不带路径模块05\_2在05\_2下含有hello.js文件(打印一句话)

https://www.npmjs.com 可以免费下载模块的网站

C:\Users\web>node -v

v8.11.1

C:\Users\web>npm -v

5.6.0

C:\Users\web>npm tall mysql 连网才可以连接的命令，

C:\Users\web>cd 切换目录

### 2.npm和包

包(package)：是一个目录模块，里边包含多个文件，其中有一个文件命名为package.json，是包说明文件，含有包的信息

npm：是一个包管理工具，安装nodejs的时候会附带安装，用于管理包，包括下载，上传，升级等。

npm官网：https://www.npmjs.com 可以免费下载模块的网站

(1)切换目录

cd 完整路径

如果要进入其它的盘符

d：回车

进入指定的目录下，按住shift键，在空白区域单击鼠标右键->在此处打开powershell

(2)使用npm下载安装第三方包

npm install 包名称

C:\Users\web>npm init -y //初始化文件

Wrote to C:\Users\web\package.json:

{

"name": "web",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC"

}

(3)生成package.json

npm init -y 自动生成一个package.json文件，后期使用npm安装的包都会记录到这个文件中。

npm install 自动安装package.js文件中记录的包

### 3.全局函数

06 global.js

parseInt/parseFloat/encodeURI/decodeURI/isNaN/inFinite/eval(JS中学过的)

(1)一次性定时器 setTimeout(参1，参2)

set 设置 Timeout 超时 clear 清除

|  |
| --- |
| 开启  var timer=setTimeout(回调函数，间隔的时间);  当间隔的时间到了，执行回调函数，单位是毫秒  清除  clearTimeout(timer)； |

(2)周期性定时器 | interval 间隔，幕间休息

|  |
| --- |
| 开启  var timer=setInterval(回调函数，间隔时间)；  每隔一段时间，执行一次回调函数  清除  clearInterval(timer) |

练习：使用周期性定时器，每隔3秒钟，打印一次hello，打印三次以后，清除定时器

(3)立即执行定时器 immediate 立即的，直接的

|  |
| --- |
| 开启  var timer=setImmediate(回调函数)  回调函数会放到队列中  当主线程程序执行完，才会执行队列中的内容  清除  clearImmediate(timer) |

nsxt Tick滴答声

|  |
| --- |
| process.nsxtTick(回调函数)  在主程序的末尾执行 |

### 4.querystring模块——查询字符串

07 query查询 string字符串

属于核心模块，是nodejs官方提供的，可以直接引入，不需要创建

www.codeboy.con/detail.html?lid=5&name=dell

浏览器向服务器发请求，传递数据的一种方式

parse() 将查询字符串格式化为对象

stringify() 将对象转换成查询字符串

练习：获取百度查询中查询字符串中的关键字e=utf-8& &tn=baidu&wd=电脑

https://www.codeboy.com:80/product(目录)/detail.html?lid=5&name=dell

?后面的是字符串

### 5.url模块

08 有打印的有结果

1)parse() 将url格式化为对象，

需要关注的重点：

1.protocol 协议

2.hostname 服务器域名/IP地址

3.port 端口号

4.pathname 文件在服务器上的路径

5.query 查询字符串

2)format() 将对象转成url

query属性对应的是对象 写法：query:{uname:'root',upwd:'123456'}

练习：获取浏览器中传递的数据 09\_exercise.js

https://www.wmooc.cn:3000/web/1903.html?sid=10&course=javascript

获取url中的查询字符串

将查询字符串格式化为对象

### 6.fs模块——文件系统模块

用于文件的操作，目录的创建，删除，读取；文件的创建，读取，写入删除.......

(1)查看文件的状态

fs.stat( path,callback )

=

fs.stat(‘05’,function(err,stats){ifFile/isDirectory})

path 要查看的文件的路径

callback 回调函数，用于获取文件的状态，包含两个参数：

err: 如果查看失败的错误信息

stats 文件的状态信息，包含是否为目录和是否为文件

isDirectory() 是否为目录

isFile() 是否为文件

课后任务：

1.复习今天内容，整理思维导图

2.练习：

创建模块01\_1.js 引入02\_2不带路径的目录，要求在02\_2中创建文件，file.js,在file.js中导出一个查询字符串，在01\_1.js中获取该字符串，获取其中的数据。

3.预习文件系统模块中的mkdir,rmdir,readdir,writefile,appendFile,readFile

预习http协议

\*单词\*

require：需要，要求，命令

package：包，套装软件，[计]程序包

json：基于JavaScript语言的轻量级的数据交换格式（JavaScript Object Notiation）

object：目标，物体

node：节点

modules：模块

index：做引索

test：试验，测试

fn：功能

## day03 同步和异步

### 1.同步和异步 02

同步：(sync)方法调用后，必须等待方法调用结束，才能继续后边的任务；通过方法的返回值获取结果，把方法的返回值放到一个变量里。

异步：(回调函数)方法调用后，无需等待结束，直接去执行后续的代码，整个过程，不会阻碍后边的任务，通过回调函数获取结果。

### 2.fs模块——文件系统

(1)查看文件状态 fs.stat(path,callback) / statSync(path) //昨天学的

(2)创建目录 fs.mkdir(path,callback) / mkdirSync(path)

(3)移除目录 fs.rmdir(path,callback) / rmdirSync(path)

(4)读取目录 fs.readdir(path,callback) / readdirSync(path)

callback有两个参数：

err 可能产生的错误信息

files 读取的结果，格式为数组

(5)清空写入 fs.writeFile(path,data,callback) / writeFileSync()

path 要写入的文件的路径

data 要写入的数据

callback 回调函数

如果文件不存在会创建文件，如果文件已经存在，会清空文件中的内容，然后写入。

(6)追加写入 appendFilse

fs.appendFile(path,data,callback) / appendFileSync()

如果文件存在，在文件的末尾写入数据

练习：创建数组，包含多个姓名，遍历数组，获取每个元素，要求把每个元素使用同步方法写入到文件data.txt

(7)读取文件 readFile read阅读

fs.readFile(path,callback) / readFileSync()

callback:

err 可能产生的错误信息，

data 读取的文件的数据，格式为buffer (要加toString转成字符串，示例里有写)

(8)删除文件 unlink()

unlink(path,callback) / unlinkSync()

已经删除的目录，再次删除会报错。

(9)判断文件是否存在 异步方法被deprecated废弃

fs.existsSync(path)

如果存在返回 true 不存在false

练习：1.判断文件2.txt是否存在，如果存在就删除

2.决断目录mydir2是否存在，如果不存在创建

### 2.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) general一般的，普通的

Request URL：请求的url，对应浏览器地址栏的内容，表示要向服务器获取哪些内容。

Request Method：请求的方法，GET/POST.....获取内容的方式 method方法 request请求

Status Code：响应的状态码，可以查看响应的结果。 status 状态

1\*\*：正在请求，没有结束

2\*\*：成功的响应

3\*\*：响应的重定向，跳转到另一个url通常结合响应头信息中Location一起使用

4\*\*：客户端错误

5\*\*：服务器端错误(例如双十一同时访问同一个购物网站造成服务器端的死机)

Remote Address：请求的服务器的IP地址和端口号 address地址 remote远程

(2)响应头信息(ResponseHeaders)

Connection：连接的方式，keep-alive 持续连接

Content-Type：响应的文件类型

Content-Length：响应的文件长度

Transfer-Encoding：传输方式chuncked分段传输

Content-Encoding：压缩模式 gzip

Location：当响应重定向的时候，跳转的URL，结合状态码3\*\*来用。

(3)请求头信息(RequestHeaders)

Accept：客户端接受的文件类型有哪些

Accept-Encoding：客户端接受的压缩形式

User-Agent：客户端发请求使用的浏览器

(4)请求主体(可能是字符串也可能是其它的形式，不定)

可有可无的，表示客户端向服务器端传递的数据

### 3.http模块 03

作用：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,callback)

1.get 请求的方法 2.url 请求的路径

3.callback 回调函数，用于获取服务器端的响应

res 响应的对象

statusCode 响应的状态码

4.res.on(‘data’,function(buf){})

通过事件获取服务器端的响应，当有数据传输的时候自动触发：通过回调函数来接收数据，buf就是传输的数据，格式为buffer。

语法：

http.get(url,function(res){

res.on('data',function(buf){

});

});

练习：使用http模块下的get方法向http://www.tmooc.cn发送请求，并获取响应的数据

(2)创建web服务器(比较重要的) 04 listen听

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

2)设置web服务器端口号，作用是监听端口变化：

语法：server.listen(8080)

3)接收浏览器的请求，是一个事件，当有请求发生会自动触发

语法：server.on(‘request’,function(req,res){})

1.req 请求的对象/内容

语法：req.url/method/headers

url 请求的url,显示端口后的部分

method 请求的方法

2.res 响应的对象 语法：res.writeHead/wirte/end

writeHead(code.obj) 设置响应的状态码和头信息code状态obj是一个头信息的对象

write() 设置响应的内容/文本

end() 结束响应，发送响应的内容到浏览器

练习：当请求的url为/login.响应的一句话‘login’,当请求的url为/study,跳转到http://www.tmooc.cn

课后任务：

1.复习今天内容，整理思维导图

2.练习：使用http模块创建web服务器，设置端口8080，接收浏览器的请求

/login 响应 this is login page

/ member 响应 welcome to home

/ 重定向 /login

以上都没有 响应内容 404 not found

3.预习 express

callback 回收、在这里是回调函数的意思

path 路径

sync 同步，使同步

make 获得 +directory 目录 =fs.mkdir(' ')创建目录

method 方法，条理，类函数

## day04 框架/路由

### 1.express框架

好处：节省时间，提高开发效率

基于Node.js平台，快速，开放，极简的web开发框架

www.expressjs.com.cn

安装命令： npm install express

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

### (1)路由

浏览器向web服务器发送请求，web服务器根据请求的方法和请求的url来做出响应。

三要素：请求方法、请求的URL、响应(回调函数)

响应的对象(res)

send() 响应内容，只能响应一次；第二次会报错

sendFile() 响应文件，必须使用绝对路径(\_\_dirname)

redirect() 响应的重定向

请求的对象(req)

req.method 请求的方法在服务器端看F12

req.url 请求的URL，在服务器端看F12

req.headers 请求的头信息F12

req.query 获取请求时查询字符串传递的数据，并格式化为对象

练习：创建文件03\_express.js,使用express创建web服务器，添加以下路由

get /index '这是首页'

get /login 响应文件login.html

post /reg '欢迎注册'

get / 跳转到 /index

练习:创建文件04\_post.js，使用express创建web服务器，添加路由

get/ /reg 响应文件 reg.html

### (2) post 和get传递数据

post请求是通过表单提交(暂时)传递数据，服务器端是通过事件的形式获取数据(后期会有简单的方法，后期会学到)

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,结果是对象。

查询字符串传递数据容易被浏览器所缓存,而post传递数据不会出现在地址栏。

|  |
| --- |
| req.on('data',function(buf){  buf就是获取的数据，格式化为buffer,转为字符串后格式为查询字符串，需要借助查询字符串模式格式化为对象(示例在04\_post.js)  }) |

练习：添加路由，请求的方法get,请求的url 为 /login 响应login.html,要求在html文件中创建登录的表单(04\_post.js)

### (3)使用路由传递数据——路由传参

设置路由中接收的名称 :lid

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

浏览器传递

http://127.0.0.1:8080/detail/5

说明：5就是传递的数据，使用lid来接收

练习：创建购物车的路由，请求的方法get,请求的URL /shopping,传递商品的名称(pname)和价格(price)

### (4)路由器 05

路由的使用过程中，不同的模块可能出现相同的URL，把同一个模块下的所有路由放到一个容器，这个容器就是路由器。路由器最终要引入web服务器下才能使用。

1.创建一个路由器模块(自定义模块)

|  |
| --- |
| const express=reqire('express');  var router=express.Router(); //创建路由器对象  router.get('/list',function(req,res){ }); //添加路由  module.exports=router; //导出路由器对象 |

2.在web服务器下使用路由器

|  |
| --- |
| const productRouter=require('./product.js');  server.use('/product',productRouter); //把路由器挂载到/product(自定义的)下  访问形式 /product/list |

练习：创建用户模块的路由器(user.js),添加路由用户列表(/list,get) ,在web服务器下引入并使用，挂载到/user下

课后任务

1.复习今天内容，整理思维导图

2.练习：创建web服务器，添加路由(get /page),响应一个html文件，输入身份证号，点击提交(post,/mypage)，响应出生的年月日以及性别。

3.复习mysql中sql语句

\*单词\*params:参数

\*单词\*cannot：不能，无法

\*单词\*post：公布，邮递

\*单词\*submit：提交

\*单词\*Buffer:缓冲区

\*单词\*parse：解析

## day05 中间件

### 1.中间件

02\_middleware.js

中间件也可能理解为是一个过滤器，作用是为主要的业务逻辑服务。

中间件分为五大类：应用级/路由级/内置/第三方/错误处理

1)应用级(自定义)中间件

每个中间件都是一个函数

server.use(path,function(req,res,next){ }) 过滤路由中url为path的路由。

server.use(function(req,res,next){ }) 过滤所有的路由

练习1：给url为/reg的路由添加前置的中间件，验证用户名是否可用

练习2：创建路由（get,/view）响应当前的浏览次数，每次浏览次数加1.在中间外部先声明一个变量被始化为0，在中间件中加1，在路由中把该变量响应到浏览器。

2)路由级中间件

路由器的使用 \*单词\*path路径

server.use(path,路由器);

3)内置中间件 03\_static.js

在express4下只保留了一个内置的中间件

express.static('目录')

把静态资源托管到指定的目录，如果浏览器请求静态资源，自动到该目录下查找。

静态资源：html,css,浏览器js,img.....

4)第三方中间件

npm install 中间件名称

使用中间件

|  |
| --- |
| server.use(bodyParser.urlencoded({  extended:false }))  urlencoded：将post请求的数据格式化为对象  extended：不是用第三方qs模块，而是使用核心模块  querystring 将查询的字符串格式化为对象 |

在路由中获取post请求数据

req.body 格式化为对象

练习3：创建04\_post.js,创建web服务器，托管静态资源到public下，在public下，创建login.html.使用浏览器请求该文件，点击按钮，向服务器发请求。

5)错误处理中间件(没有学)

\*\*

[**query language**](javascript:void(0);)查询语言

[**query builder**](javascript:void(0);)查询生成器

[**structured query language**](javascript:void(0);)结构化查询语言

[**query the database**](javascript:void(0);)查询数据库

**querystring** 将查询字符串格式化为对象

### 2.mysql模块

连接 mysql.exe -h127.0.0.1 -P3306 -uroot -p(简写：mysql -uroot)

增 INSERT INTO emp VALUES(NULL,'tom'....)

删 delete from emp where uid=5;

改 update emp set uname='jerry',salary=8000 where uid=5;

查 select\*from emp;

(1)连接mysql

var connection=mysql.createConnection(); //创建连接对象，传递连接数据库需要的服务器，

端口，用户名，密码，要使用的数据库

connection.connect() 建立连接

connection.query(sql,callback); sql是要执行的sql语句，callback是回调函数，用于sql语

句的结果。

connection.end(); 执行完所有的sql语句后，关闭连接

练习4：使用mysql模块删除员工表emp中编号为5的数据，打印结果。

(2)连接池

var pool=mysql.createPool(); 创建连接池对象，传递连接需要的服务器地址，端口，用户

名，密码，要使用的数据库，设置连接池的大小connectionLimit,默认是15

pool.query(sql,callback) 执行SQL语句，sql要执行的sql语句，callback回调函数，用于

获取结果。

练习5：要求往dept部门表里添加数据 did /dname

练习6：修改数据将dept部门表中编号为40的数据，部门名称改为‘测试1部’

routes 文件里存放的是路由器

课后任务：

1.复习今天内容，整理思维导图

2.练习：

创建web服务器，托管静态资源到public下，在该目录下创建add.html,点击提交按钮，向服务器发请求(get/add),服务器获取数据，并将数据插入到tedu下的dept,如果插入成功，响应‘部门添加成功’。

## day06 学子商城

学子商城创建步骤

xz：一级文件夹

1.app.js： 服务器 5个步骤

4.pool.js：连接池 3个步骤

routes：专门放路由器的文件夹

2.user.js：路由器 6个步骤 每添加一个html文件都要在此文件里添加一个router和执行SQL语句

public：专门放公共资源的文件夹

3.user\_reg.html 用户注册

5. user\_login.html 登录用户

6. user\_update.html 修改用户

7. user\_detail.html 用户检索

8. user\_list.html 用户列表

9. user\_delete.html 删除列表

Error: Can't set headers after they are sent. 多次send调用了，

1.注册表单 user\_reg.html页面 托管到public下

用户： 注意name="名字要对应数据库设置的名字"

密码：

手机：

邮箱：

提交

2.用户登录 user\_login.html public下 post提交 /user/login  
用户：

密码：

提交

3.修改用户 user\_update.html get /user/update

编号： uid

手机： phone

邮箱： email

姓名： uname

性别： gender

4.用户检索 user\_detail.html get /user/detail

编号： uid 不用换行

提交

路由 先获取数据 验证是否为空 执行SQL语句(查询编号对应的用户)，把result响应到浏览器

5. 用户列表 user\_list.html get /user/list

获取get数据 如果页码为空设置默认值为1 如果大小为空设置默认值为3

页码： pno

大小： count

提交

分页数据量 2

start=(页码-1)\*数据量

select \* from xz\_user limit 0,2;

select \* from xz\_user limit 2,2;

6.删除用户 user\_delete.html get /delete

编号 提交

课后任务

商品模块 xz\_laptop

添加、修改、删除、检索、列表

创建商品路由器 product.js

1.添加商品product\_add.html get /product/add

2.修改商品product\_update.html get /product/update

3.商品检索product\_detail.html get /product/detail

4.商品列表product\_list.html get /product /list

5.删除列表product\_delete.html get /product/delete

# 第二阶段

## day01 HTML

### **一.课程安排**

**1.Html5 Basic 2天 搭建网页结构**

**2.Ajax异步数据交换(3天) 异步完成前后台数据的交互**

**3.项目1（2天）MySQL nodejs html ajax 从前端页面实现对用户表中数据的增删改查**

**4.css3 （4+2天）美化页面**

**5.Bootstrap框架 （4天）响应式的框架，css的框架**

**6.项目2 （3天）boot完成商城的商品部分所有页面**

**7.这20天课程遇到的问题**

**1.知识点细碎，没条理。2.单词量大，关键词比较长，初期没有提示。3.键速200**

### **二.web基础知识**

|  |  |
| --- | --- |
| **HTML:泛指前端网页技术**  **2014年9，html4.0升级html5.，简称h5**  **Html5 大前端技术**   |  | | --- | | **html4.01在1999.12月发布**  **xhtml1.0 在2000.1月发布，语法更严谨**  **<img/> <img>** | |

**1.web与internet的关系**

|  |
| --- |
| **internet 全球性计算机互联网络**  **俗称 互联网，因特网，交换网，交际网**  **www服务world wide web 万维网**  **ftp:文件的上传下载的服务**  **Email:电子邮件的服务**  **telnet：远程登陆的服务**  **BBS：电子公告板，俗称论坛**  **以上都是运行在internet应用程序** |

**2. internet上的应用程序都会分成两类**

|  |
| --- |
| **1. C/S client 客户端 server服务器**  **王者荣耀 QQ.exe** |
| **2. B/S browser浏览器 server服务器**  **所有的网站 网页版微信** |
| **3.C/S结构和B/S结构的区别**  **C/S需要升级 B/S不需要升级** |

**3.web运行原理**

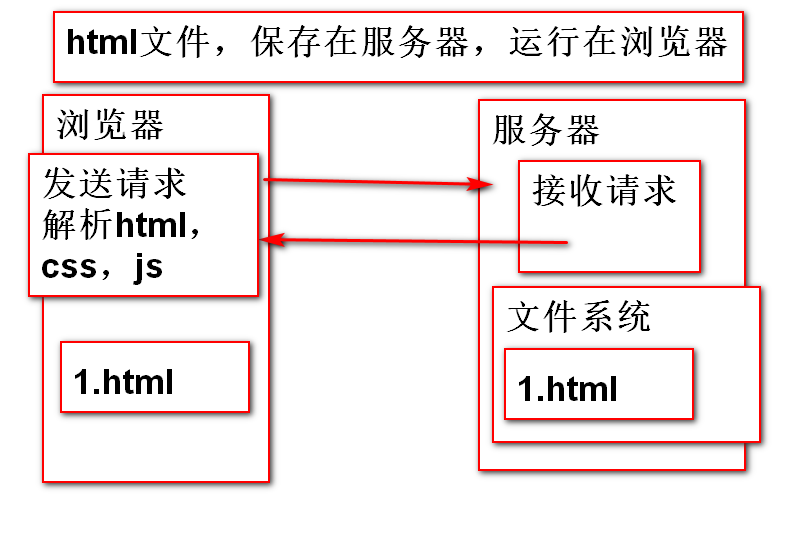
|  |
| --- |
| **web：运行在internet上的一种B/S结构的应用程序**  **俗称网站**  **internet:为web运行提供了网络环境**  **web运行的原理：基于浏览器和服务器，以及通信协议来实现数据的传输和展示**  **通信协议：规范了数据是如何打包和传递的**  **网页的通信协议 http:// https://** |
| **服务器**  **1.功能**  **存储数据**  **接收用户的请求，并且给出响应**  **提供程序的运行环境**  **具备一定的安全功能**  **2.服务器产品**  **Apache—php**  **TOMCAT--java**  **IIS .net**  **3.服务器端技术**  **nodejs java php .net python** |
| **3.浏览器**  **1.功能**  **代表用户发送请求**  **作为html，css和js的解析器**  **以图形化的界面展示给用户看**  **2.浏览器产品**  **safari chrome firefox opera ie**  **3.浏览器端的技术**  **html5 css3 javascript** |

### **三.html快速入门**

**1.什么是html**

|  |
| --- |
| **HyperTexts Markup Language**  **超文本标记语言**  **标记：超文本组成形式，<关键字>，具有独特的功能** |

**2.html的特点**

****

|  |
| --- |
| **1.以.html或者.htm为后缀**  **2.由浏览器解析执行**  **3.用<>来定义标记**  **4.可以嵌套js脚本** |

**3.html语法**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1.标记**   |  |  |  | | --- | --- | --- | | **标记，元素，标签，节点**  **<关键字>**  **标签具备一些功能**  **学习html第一步，背关键字，熟悉对应的功能**  **标记分为两类**   |  | | --- | | **1.双标记（封闭类型标记）**  **<关键字></关键字>**  **双标记成对出现，有开有关**  **ex:<a></a>** | | **2.单标记(非封闭类型标记，空标记)**  **<关键字> 或者 <关键字/>**  **ex：<input/> <br/> <img/>** |   **练习：**  **新建一个text文档，改名为first.html**  **写一对html标记**  **一对head标记**  **一对title标记**  **一对body标记** |   **2.标签的嵌套**   |  | | --- | | **在一对标签内部，出现另一对或者另一个标签，组合形成功能的叠加**  **学习html第二步，背嵌套顺序**  **语法：缩进，嵌套顺序**  **<标签1>**  **<标签2>**  **<标签3></标签3>**  **</标签2>**  **</标签1>** |   **3.属性和值**   |  | | --- | | **通过属性和值，对标签进行修饰**  **<标签 属性1="值1" 属性2="值2" ...></标签>** | | **1.标准属性，通用属性。所有标签都有的属性**  **id:定义元素在页面中唯一的标识**  **title：鼠标悬停在元素上显示的文本**  **style：css中，定义内联样式**  **class：css中，引用类选择器时使用** | | **2.专有属性，只针对一个标签起作用的属性** |   **学习html第三步，背专有属性，及效果**  **4.注释**   |  | | --- | | **不会被浏览器解析运行的部分**  **一般编写代码说明**  **<!-- -->**  **注释不能嵌套注释**  **注释不能写在标签内部** | |
| **学习html就三步**  **1.学习固定标签关键字，及功能**  **2.学习固定的嵌套关系。**  **3.学习固定的属性及其值** |

### 四.HTML的文档结构

**1.html文档的构成**

|  |
| --- |
| **1.文档类型声明**  **<!doctype html>**  **告诉浏览器，解析运行本篇代码，使用h5的规则解析** |
| **2.网页的结构**  **<html></html>**  **表示网页的开始和结束。一个html文件中只能有一对html标签**  **<!doctype html>**  **<html>**  **<head></head> 网页头部，定义网页的信息**  **<body></body> 网页主体，展示内容**  **</html>** |
| **3.head标签，定义网页的信息**   |  | | --- | | **1.<meta> 元数据标签**  **<meta charset="utf-8">**  **<meta name="description" content="描述内容">**  **<meta name="Keywords" content="关键字">** | | **2.head中其它标签**  **<title></title> 网页标题**  **<script></script> 编写js或者引用js文件**  **<style></style> css中定义内部样式**  **<link>引入外部css样式** |   **4.body元素**   |  | | --- | | **指定网页的主体**  **<body></body>**  **属性：bgcolor body的背景颜色，取值颜色合法值**  **text body的文本颜色，取值颜色的合法值** |   **练习：01\_ex.html**  **编写完整的html页面结构** |

### 五.文本标记(重点\*\*\*)

**1.标题元素**

|  |
| --- |
| **在页面中以醒目的方式显示文本**  **<h1></h1>**  **....**  **<h6></h6>**  **特点：1.字体大小有变化 h1最大，h6最小**  **2.字体加粗**  **3.单独成行，上下有垂直间距 属性：align 设置标记内容水平对齐方式**  **取值：left/center/right** |

**2.段落标记**

|  |
| --- |
| **<p></p> 以突出的形式表示一段文字**  **特点：1单独成行**  **2.上下有垂直间距**  **属性：align:left/cener/right** |

**3.换行标记**

|  |
| --- |
| **空格折叠现象：在html中，文本中不管有多少个空格和回车，都会被浏览器解析成一个空格显示**  **<br> 或者 <br/>** |

**4.特殊字符(实体)**

|  |
| --- |
| **&nbsp; 空格**  **&lt; <**  **&gt; >**  **&times； X**  **&reg; 商标注册 ®**  **&copy；版权 ©**  **&yen;** |

**5.分割线，水平线**

|  |
| --- |
| **<hr> 或者 <hr/>**  **属性**  **size="5px" 取值px为单位的数字，水平线的粗细**  **width="50%" 取值px或者%,水平线的宽度**  **align="left" left/center/right**  **color="blue" 取值为合法颜色值** |

**6.预格式化**

|  |
| --- |
| **<pre></pre>**  **保留html代码中的回车和空格效果** |

**7.文本样式标签**

|  |
| --- |
| **<em></em> <i></i> 斜体**  **<strong></strong> <b></b> 加粗**  **<del></del> <s></s> 删除线**  **<u></u> 下划线**  **<sup></sup> 上标**  **<sub></sub> 下标** |

**8.分区元素--不写样式的分区元素，是看不见的。**

|  |
| --- |
| **1.块分区<div></div>**  **用于页面布局** |
| **2.行分区<span></span>**  **为文本添加样式的时候用span** |

**9.元素的显示方式**

|  |
| --- |
| **1.块级元素**  **元素独立成行，在页面中从上往下排列**  **ex:h1~h6 p div pre** |
| **2.行内元素(行级元素)**  **多个行内元素在一行中显示，从左往右排列**  **ex:span i em strong b u sub sup del s** |
| **3.行内块**  **表现方式是行内元素(多个行内块可以共用一行)**  **但是具有块级元素的其它特征(宽高，边距)还没学**  **ex:input** |
| **4.table** |

### 六.图形和链接

**1.图形图像**

|  |
| --- |
| **<img/> 或者 <img>**  **属性：src 引用图片资源 url**  **\*.html和图片是兄弟关系，直接在src中写图片名称** |

**练习**

****

**EditPlus快捷键设置总结**

|  |
| --- |
| **加载1.key。1.key eclipse的开发快捷键**  **1.设置代码补全和lorem的补全**  **在zc下，把第二个和第三个勾勾上**  **2.设置tab空两个格**    **把默认的4变成2**  **3.导入1.key**    **选择1.key就可以使用了**  **4.常用快捷键**  **ctrl+d 删除当前行**  **alt+↓/↑ 移动当前行下移/上移**  **ctrl+alt+↓ 复制当前行**  **5.打开浏览器运行html的快捷方式**  **ctrl+b 打开浏览器**    **6.添加注释快捷方式**  **ctrl+/** |

**作业:**

**1.完成nodejs6个接口的编写**

**2. web第二阶段练习效果图示\day01\_html\_01作业**

****

## day02 表格

### 六.图像和链接

1.图像的使用

|  |
| --- |
| <img>  属性  src="url/图片资源路径"  ex：<img src="图片资源路径"> |

2.URL

|  |
| --- |
| 1.URL是什么  Uniform Resource Locator 统一资源定位器  就是路径 |
| 2.绝对路径  完整的路径  协议+主机名称+文件目录结构+文件具体的名称  使用场合：使用网络资源的时候，使用绝对路径  使用网络资源图片  优点：节省服务器本地存储空间  缺点：资源不稳定  绝对路径使用本地的资源，是从最高盘符开始查找  c:\img\08.png  但是本服务器资源，在项目中，不能用绝对路径 |
| 3.相对路径    使用本服务器资源，用相对路径  1.同级目录资源  直接使用资源名称 src="08.png"  2.兄弟文件夹的子元素  直接使用兄弟文件夹的名称，在用/调用资源名称  src="img/08.png"  3.父级目录中资源  先使用../返回上一级，在引用资源名称  src="../08.png" |

3.属性

|  |
| --- |
| src：source 源，资源路径  alt： 图片错误时，显示的文本  title：鼠标悬停时显示的文本  width：设置图片的宽度  height：设置图片的高度  如果设置宽高比，不符合图片原始宽高比。  图片会产生失真效果  解决图片失真效果：  一般情况下，width和height只设置一个，让另外一个自动适应 |

练习

|  |
| --- |
| 01\_ex.html  页面中引入6张图片  1.图片与页面是兄弟关系  2.图片是页面的兄弟的儿子  3.图片是页面的兄弟的儿子的儿子  4.图片是页面父亲的兄弟  5.图片是页面的父亲的兄弟的儿子的儿子  6.图片是页面父亲的父亲的兄弟 |

4.链接

|  |
| --- |
| 1.语法  <a href=""></a>  属性：href 链接路径  target 指定打开链接的方式  取值 \_self 默认值，在当前页面打开新网页  \_blank 在新的页面打开网页 |
| 2.a标签的其它表现形式  1.新建邮件，配合windows的邮件软件使用  <a href="mailto:cheng@t.com">发邮件</a>  2.执行js  <a href="javascript:show()">执行js</a>  3.下载  <a href="1.zip">下载</a>  4.返回页面顶部  <a href="#">返回页面顶部</a> |
| 3.锚点  什么是锚点  锚点是网页中的一个记号  可以通过超链接的方式，链接到锚点，让页面跳转到锚点位置显示  使用锚点   |  | | --- | | 1.定义锚点  第一种方式：任何标签的id中写锚点名称  <Any id="锚点名称"> </ Any >  第二种方式：a标签的name属性中写锚点名称  <a name="锚点名称"></a> | | 2.链接到锚点  <a href="#锚点名称"></a> | | 3.链接到其它页面的锚点  <a href="其它页面的url#锚点名称"></a>  <a href="02\_a.html#hl">葫芦娃</a> | |

### 七.表格

1.语法

|  |
| --- |
| 早期table用于布局  由于效率低下，后来被淘汰，使用div+css的布局  表格现在就一个单纯的作用---显示数据  <table> 表格  <tr> 行 table row  <td></td> 列 table data  ....  </tr>  ....  </table> |
| 属性   |  | | --- | | table的属性  width="300px" 表格的宽度  height="300px" 表格的高度  border="1px" 表格的边框  align="left/center/right" 表格的水平对齐方式  bordercolor="red" 边框颜色  bgcolor="yellow" 背景颜色  cellpadding 单元格内边距，内容到边框的距离  cellspacing 单元格的外边距，边框与边框的距离 |   练习：   |  | | --- | | 03\_ex.html  3行3列的表格，宽高400px\*400px  背景颜色pink，边框为2px 表格居中显示  表格内边距5px 外边距10px | | tr的属性  bgcolor="yellow" 背景颜色  align="right" 表格这一行的内容水平对齐方式  valign表格这一行的内容垂直对齐方式  取值 top/middle/bottom | | td的属性  width="200px"  height="200px"  align="right" 内容的水平对齐方式  valign="bottom" 内容的垂直对齐方式  bgcolor="red"  colspan 跨列  rowspan 跨行 | |

2. 不规则的表格

|  |
| --- |
| 1.跨列：colspan  从指定单元格位置处开始，横向向右合并n个单元格(n包括自己)，被合并的单元格要删除 |
| 2.跨行  从指定单元格位置处开始，纵向向下合并n个单元格(n包含自己)，被合并的单元格要删除 |

3.可选标记

|  |
| --- |
| 1.表格的标题  <caption></caption>  如果要设置标题，caption必须放在<table>之后  2.行/列的标题  <th></th>,td的属性，th都可以用  th的特点：文字加粗，水平居中 |

4.表格的复杂应用

|  |
| --- |
| 我们制作的表格，浏览器解析的时候会自动添加上<tbody></tbody>标签  行分组：可以将连续的几行，划分到同一个组中，进行统一的管理。  1.表头<thead></thead>  2.表主体<tbody></tbody>  3.表脚<tfoot></tfoot>  <table>  <thead>  <tr><th></th></tr>  <tr><th></th></tr>  </thead>  <tbody>  <tr>...</tr>  <tr>...</tr>  <tr>...</tr>  </tbody>  <tfoot>  <tr></tr>  </tfoot>  </table> |
| 2.表格的嵌套  表格中所有的嵌套，都只能放在td中完成 |

### 八.列表（重点\*\*\*\*\*\*\*）

1.列表的作用

|  |
| --- |
| 按照从上到下，或者从左到右来显示所有的数据  并且可以在数据前增加标识 |

2.列表的组成

|  |
| --- |
| 列表都是由列表类型和列表项来组成的  1.列表类型  有序列表 <ol></ol> order list  无序列表 <ul></ul> unorder list  2.列表项  <li></li> list item |

3.有序列表

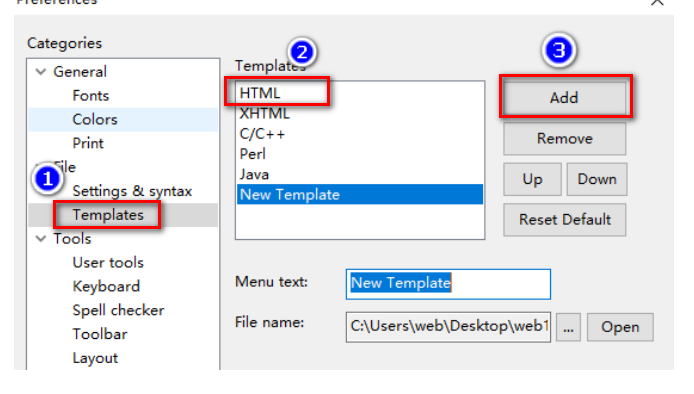
|  |
| --- |
| <ol>  <li></li>  <li></li>  ...  </ol>  属性：type 指定标识项的类型  取值 1 默认值，数字  a/A 字母  i/I 罗马数字  start 指定起始编号 取值：无单位数字 |

4.无序列表---经常用于页面布局

多个li的样式大体一致的时候，使用ul>li布局

|  |
| --- |
| <ul>  <li></li>  <li></li>  </ul>  属性：type 设置列表标识类型  取值 1.disc 默认值 实心圆  2.circle 空心圆  3.square 实心方块  4.none 不显示 |

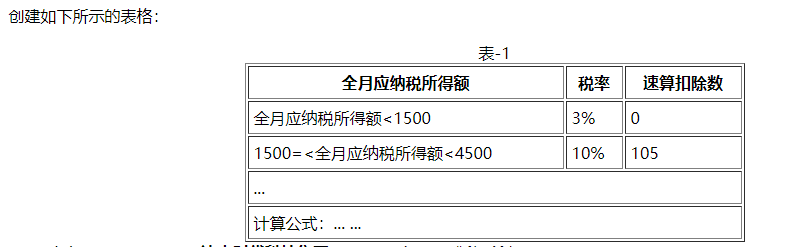
设置html模板



作业

1.nodejs 6个接口

2.表格练习





## day03 列表

### 八.列表

1.有序

2.无序

3.列表的嵌套

|  |
| --- |
| 1.列表中嵌套其他元素 |
| 2.列表中嵌套列表 |
| 原则，嵌套只能写在li中。不管是ol还是ul，直接子元素只能是li. |

4.定义列表

|  |
| --- |
| h5新出的，一个名词，以及对这个名词的解释和定义  <dl> 定义列表  <dt> </dt> 要解释说明的名词  <dd></dd> 对名词解释的内容  </dl> |

### 九.结构标签

|  |
| --- |
| h5新出的标签，专门用于网页布局  为了让代码有可读性，h5推出一堆与div功能一模一样的标签，专门用作网页布局  1.<header></header>  定义网页的头部，或者某个区域的顶部  2.<footer></footer>  定义网页的脚部，或者某个区域的底部  3.<nav></nav>  定义网页的导航栏  4.<section></section>  定义网页的主体内容  5.<aside></aside>  定义网页的侧边栏  6.<article></article>  定义与文字相关的内容  比如，论坛，回帖，用户评论.... |

### 十.表单，\*\*\*\*重点\*\*\*\*\*

1.作用

|  |
| --- |
| 提供可视化的输入控件  收集用户输入的信息，并提交请求给服务器  总结：form自带提交请求的功能  ajax提交不需要form的支持  在同一个功能中，使用form就不用ajax,使用ajax就不用form |

2.form的组成

|  |
| --- |
| 前端部分  提供表单控件，与用户交互的可视化控件  后端部分  后台接口对提交的数据进行处理 nodejs+mysql |

3.form表单详解

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| <form></form>  属性：   |  | | --- | | 1.action="" 定义表单提交时，发送请求的动作(url)  如果不写值，默认提交给本页 | | 2.method="" 定义表单的提交方法  取值   |  | | --- | | 1.get 默认值  特点：明文提交(参数在地址栏中显示)，不安全  提交的数据有大小限制，最大为2kb  使用场合：向服务器要数据的时候，使用get | | 2.post  特点：隐式提交，参数不在地址栏中显示  提交数据没有大小限制  使用场合：要传递数据给服务器的时候，使用post | | 3.其它的提交方法  delete(删除18k)  put(添加18k)  option(预请求20k)....... | | | 3.enctype="" 指定表单数据的编码方式  设置允许将什么类型的属性传递给服务器  取值：  1.application/x-www-form-urlencoded  默认值，允许将任意的字符提交给服务器  (不能提交文件)  2.text/plain 允许提交普通字符给服务器，不能包含特殊符号  3.multipart/form-data 允许提交文件给服务器 |   <form action="url" method="get"  enctype="application/x-www-form-urlencoded">  </form>  收集的数据，发送给url这个接口，使用get提交  要求提交的数据是任意字符，不是文件 |

4.表单控件

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 在form标签中，能够与用户交互的可视化控件  1.表单控件的分类   |  | | --- | | input元素 基础9种  textarea 多行文本域  select+option下拉选择器  其它元素  h5新出的input元素 10种 |   2.input元素   |  | | --- | | <input type="">  在页面中，input提供了各式各样的输入控件  文本框，密码框，单选框，多选等....  共同的属性  1.type="" 指定input元素的类型  2.name="" 为控件定义名称，此名称是给服务器用的，如果想往服务器传递参数，此名称必须写  3.value 是控件的值，真正传给服务器的值  例外,在按钮控件中,value是设置按钮上显示的文字  4.disabled 禁用，不能修改值，也不能提交  取值：disabled="disabled"  disabled="true" --js使用  disabled  这种属性，称之为无值属性 |   3.input详解9种   |  | | --- | | 1.文本框和密码框  文本框 type="text" input默认是text  密码框 type="password"  属性：  maxlength 设置输入的最大值  readonly 无值属性，不能修改，允许提交  placeholder 占位提示符 | | 2.按钮  提交按钮 type="submit"  将表单中的数据收集整理，发送给服务器  普通按钮 type="button"  没有功能，配合事件，调用js代码  重置按钮 type="reset"  将当前表单内容恢复到初始化状态  按钮中value，设置按钮上的文本，此value不提交 | | 关于按钮的题外话  h5给我们推出了新的按钮标签  <button></button> 自带submit功能 | | 3.单选和复选按钮  单选按钮 type="radio"  多选按钮 type="checkbox"  属性name，除了定义控件名称之外，还起到分组的作用，分了组才会有单选效果  必须有value属性，不然传递的参数永远是on  checked 无值属性 设置默认被选中的项 | | 4.隐藏域  type="hidden"  想把数据提交给服务器，但是又不想用户看到  这种数据放到隐藏域中  用户看不见，但是可以提交 | | 5.文件选择框，上传文件  type="file"  前提：提交方法，使用post  enctype属性，设置为multipart/form-data  属性multiple，可以选择多个文件上传 |   4 textarea多行文本域   |  | | --- | | <textarea></textarea>  允许录入多行文字  cols 设置输入的列数  rows 设置输入的行数  根据计算硬件不同，不准确 |   5.下拉选   |  | | --- | | <select>  <option></option>  <option></option>  ....  </select>  注意：  1.如果option没有value，那么select的value是option的内容  2.如果option有value,那么select的value是选中option的value值  属性 name value  selected 默认选中  size 默认值为1，  值为1的时候，为下拉选择框  值>1的时候，为滚动选择框  multiple 无值属性，多选 |   6.其它元素   |  | | --- | | 1.label 关联文本与表单的控件  <label for=""></label>  属性for绑定要关联的表单控件的ID |   7.为控件分组   |  | | --- | | <fieldset> 分组  <legend>标题</legend>  form表单控件......  </fieldset> | |

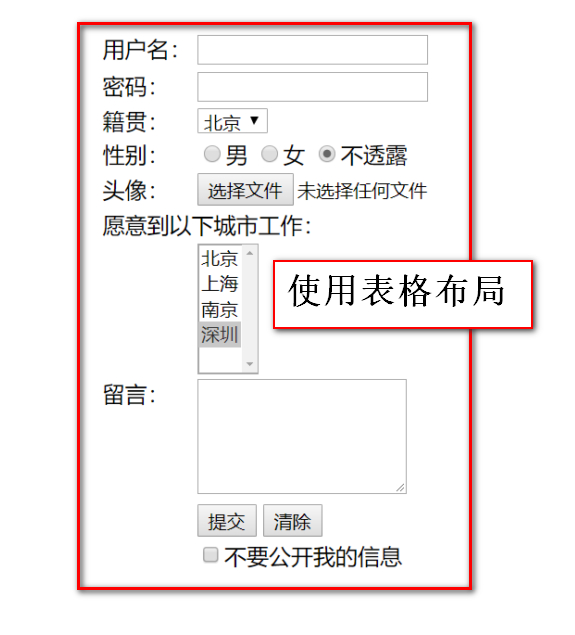
### 十一.浮动框架

|  |
| --- |
| <iframe style="width:100%" src="01\_ex.html"  frameborder="0" scrolling="no"></iframe>  语法 <iframe></iframe>  属性 src 引用的资源路径  frameborder="0" 清除边框  scrolling="no" 清除滚动条  width/height |

|  |
| --- |
| colspan 跨列，删除当前tr中td  rowspan 跨行，删除其他tr中td |

作业：

1.表单练习



2.完成下图布局



3.nodejs 6个接口，配合今天学习form

## day04 form

### 一.form表单

1.h5新表单元素

①邮箱，提交的时候，验证@以及@前后至少有一个字符

|  |
| --- |
| <input type="email"> |

②搜索类型，自带快速清除功能

|  |
| --- |
| <input type="search"> |

③url类型，提交时，验证以http://开头

|  |
| --- |
| <input type="url"> |

④电话号码，移动端使用时，会弹出虚拟键盘

|  |
| --- |
| <input type="tel"> |

⑤数字类型

|  |
| --- |
| <input type="number"  max="10" min="2" step="3">  max：能接收的最大值  min：能接收的最小值  step：步长，按一次按钮，变化几个数字 |

⑥范围类型

|  |
| --- |
| <input type="range" max="22"  min="10" step="4">  max：能接收的最大值  min：能接收的最小值  step：步长，按一次按钮，变化几个数字 |

⑦颜色类型

|  |
| --- |
| <input type="color"> |

⑧日期类型，提供了一个日期选择的控件

|  |
| --- |
| <input type="date"> |

⑨月份类型

|  |
| --- |
| <input type="month"> |

⑩周类型

|  |
| --- |
| <input type="week"> |

超纲

|  |
| --- |
| <button></button>  替代<input type="submit"> 增加可读性 |

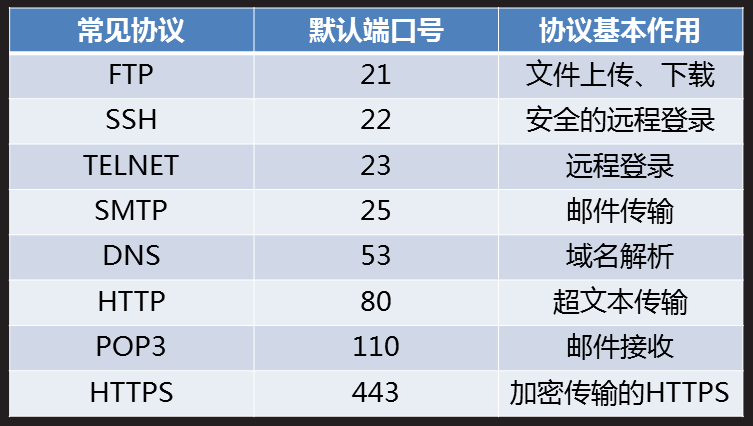
### 二.HTTP协议

1.URL

|  |
| --- |
| 结构：协议+主机名称+目录结构+文件名称  URL完整的结构：  <scheme>://<user>:<pwd>@<host>:<port>/  <path>;<params>?<query>#<frag> |

①scheme：方案，协议。设置以哪种方式获取服务器资源

协议是不区分大小写的，常用的协议http:/ftp：...



②<user>:<pwd> 用户名：密码。此种写法已过时。

③<host> 主机名 (域名、ip地址)

④<port> 端口号

⑤<path>资源，在服务器上具体存放的位置

⑥<params> 参数，保存跟踪状态的参数(session/cookie)

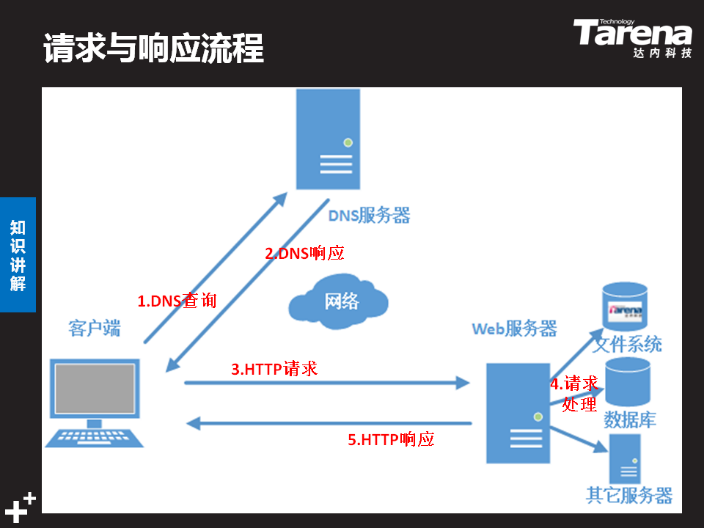
⑦<query>get方法提交请求时的查询字符串

⑧#<frag>锚点

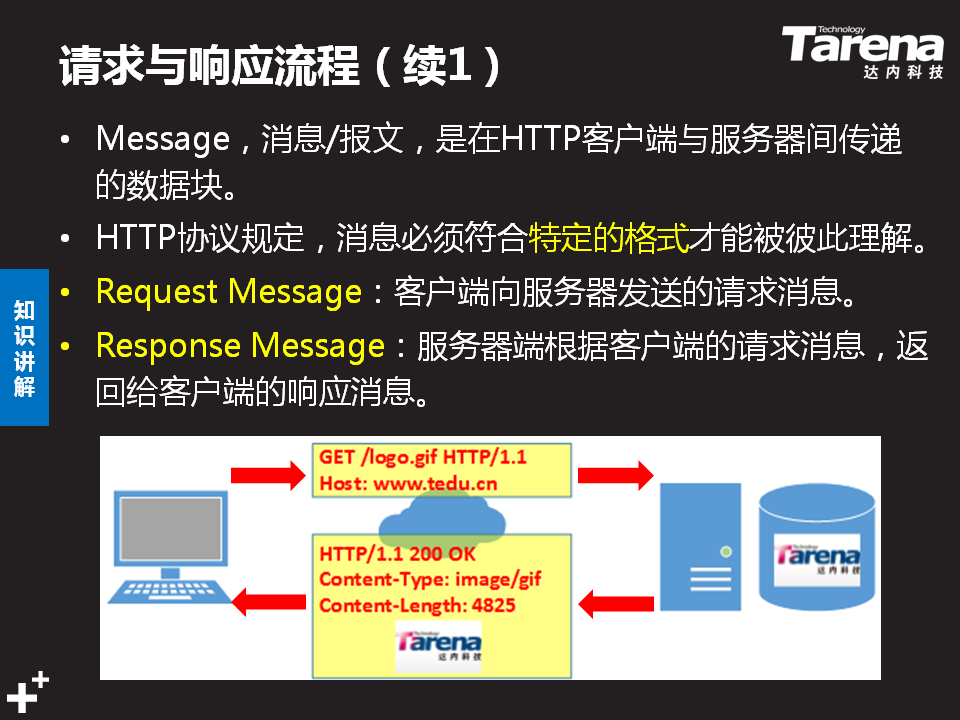
2.HTTP协议

|  |
| --- |
| HTTP:HyperText Transfer Protocol 超文本传输协议  规范了数据是如何打包以及传递的  (专门用于传输html文件的协议，专门浏览网页的协议)  HTTP协议的历史  C:\Users\web\Desktop\web1903\web第二阶段练习效果图示\HTTP协议历史.png |

3.web请求原理详解



4.消息/报文 Message



①请求消息 Request Message(请求起始行，请求头，请求主体)

②响应消息 Response Message(响应起始行，响应头，响应主体)

5.Request Message

请求消息，客户端发送给服务器的数据块

由三部分组成，请求起始行，请求头，请求主体

①请求起始行

|  |
| --- |
| 1.请求方法  get:明文提交，不安全，有大小限制 2kb  客户端向服务器要数据的时候使用get  靠地址栏传递查询字符串的方式传数据，  无请求主体  post:隐式提交，无大小限制  客户端要传递数据给服务器时使用。  有请求主体，靠请求主体传递数据  delete:客户端想要删除服务器上的内容(一般禁用)  put：客户端想往服务器上放文件内容(一般禁用)  connect:测试连接  trace：追踪请求路径  option 预请求  head：表示客户端只获取响应消息头  2.协议版本号 HTTP/1.1  3.请求的url |

②请求头

|  |
| --- |
| 1.host：www.tmooc.cn  告诉服务器请求的是哪一个主机  2.Connection:keep-alive  告诉服务器开启持久连接  3.User-Agent：告诉服务器，浏览器的类型和浏览器版本号  4.Accept-Encoding:gzip  告诉服务器，我这个浏览器可以接收的压缩文件的格式  5.Accept-Language:zh-CN  告诉服务器，我这个浏览器可以接收的自然语言的类型  6. Referer: http://www.tmooc.cn/free/  告诉服务器，当前请求来自于哪个网页 |

③请求主体

|  |
| --- |
| form data  get请求没有请求主体  post有请求主体 |

6. Response Message

响应消息，服务器发送给客户端的数据块

由三部分组成：响应起始行，响应头，响应主体

①响应起始行

|  |
| --- |
| 1.协议版本 http/1.1  2.响应状态码  告诉浏览器，服务器的响应状态是什么  1XX 正在请求，提示信息  2XX 200 响应成功  3XX 重定向 301 永久重定向  302 临时重定向  304 同一个请求，没有任何变  命中缓存  4XX 404 请求资源不存在  403 权限不够  405 请求方法不被允许  5XX 500服务器代码错误  3.原因短句(对响应状态码的简短解释) |

②响应头

|  |
| --- |
| 1.Date:Sun, 05 May 2019 08:19:02 GMT  告诉浏览器，服务器响应的时间，格林威治时间  2.Connection:keep-alive  告诉浏览器，服务器已经开启了持久连接  3.Content-Type:  告诉浏览器，响应主体的类型是什么  取值 text/html 响应回来的数据是html文本  text/css 响应回来的数据是css文本  application/JavaScript 是js文本  image/jpg gif png 响应回来的是图片  text/plain 响应回来的是普通文本  application/json响应回来的是json字符串  application/xml 响应回来的是xml字符串 |

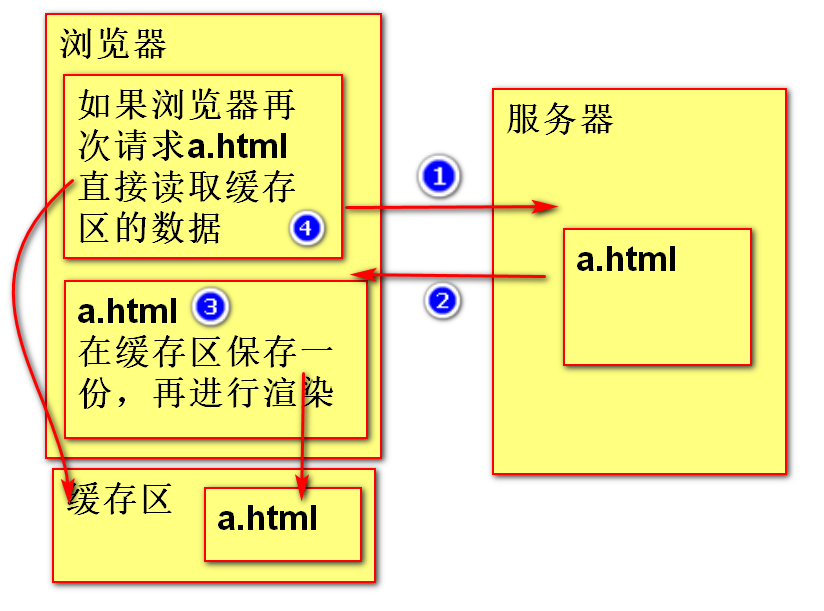
③响应主体

|  |
| --- |
| 服务器传递给浏览器的数据 |

7.缓存

客户端经服务器响应回来的数据进行自动的保存

当再次访问的时候(请求不改变)，直接使用保存的数据



①缓存的优点：

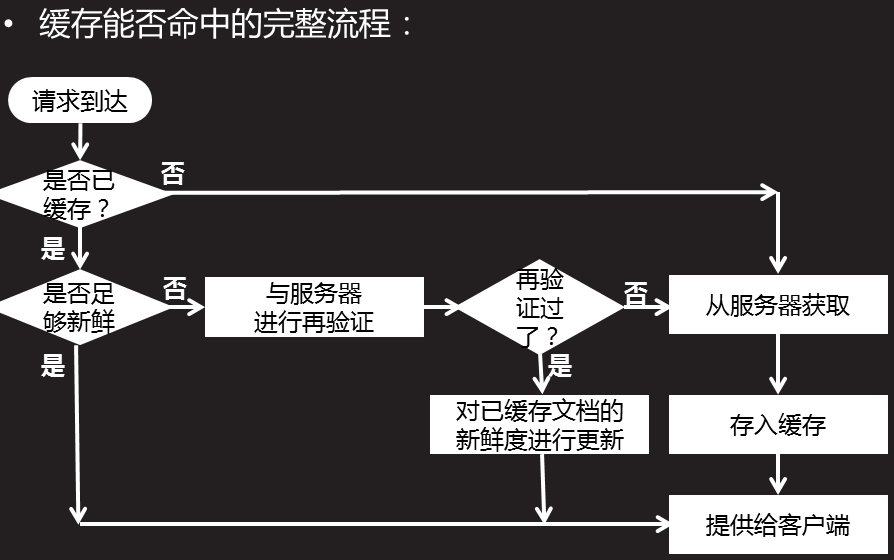
1.减少冗rong余的数据传输，节省客户端流量

2.节省服务器带宽

3.降低了对服务器资源的消耗和运行的要求

4.降低了由于远距离传输而造成加载延迟

②缓存的新鲜度和过期



|  |
| --- |
| 1.请求--无缓存--连接服务器--存缓存--客户端得到数据  2.请求--有缓存--够新鲜--使用缓存--客户端得到  3.请求--有缓存--不新鲜--连接服务器确认是否过期--没过期--更新缓存的新鲜度--客户端得到数据  4.请求--有缓存--不新鲜--连接服务器确认是否过期--已过期--连接服务器拿数据--存缓存--客户端得到数据 |

③与缓存相关的消息头

|  |
| --- |
| Cache-Control:max-age=0  从服务器将响应传到客户端之时起，此数据处于新鲜的秒数，这是一个相对时间  语法：Cache-Control:max-age=3600 |

④在网页中添加缓存，需要修改消息头

|  |
| --- |
| <meta http-equiv="消息头属性" content="值">  <meta http-equiv="Cache-Control"  content="max-age=3600"> |

8.HTTP性能优化

①HTTP的连接过程

|  |
| --- |
| 发起请求-->建立连接-->服务器处理请求-->访问资源-->构建响应-->发送响应-->记录日志 |

②HTTP连接性能优化

|  |
| --- |
| 1.减少连接的创建次数(开启持久连接)  2.减少请求的次数（整合代码，不要让文件过多） 3.提高服务器端运行速度  4.尽可能的减少响应数据的长度 |

9.安全HTTP协议。https

|  |
| --- |
| HTTPS：安全版本的http协议  SSL：为数据通信特供安全支持  ①客户端发送请求--->ssl层加密-->服务器接收到加密文件-->ssl层解密，得到明文  ②服务器发送响应--->ssl层加密-->客户端得到加密文件-->ssl层解密，得到响应内容 |

### ajax常见错误

|  |
| --- |
| 404 请求资源不存在  url是否写错  接口的路径是否写错  路由是否写错  文件位置是否放错 |
| 500服务器代码错误  一定是服务器中代码出错 |

作业

1.熟练完成nodejs接口，路由，静态资源

2.使用table完成下面布局



## day05 dom

### 一.Dom操作(简单dom操作)

|  |
| --- |
| ajax提交请求，不需要使用form表单  但是，form表单自带收集数据的功能  不使用form标签，就没有自动收集数据的功能了  我们需要自己手敲代码，收集数据  使用dom技术 |

1.完整的javascript的组成

①js核心，ECMA Script 6

②Dom:Document Object Model 文档对象模型

③Bom:Browser Object Model 浏览器对象模型

2.使用js的Dom获取页面某个元素

①获取元素对象

|  |
| --- |
| 1.使用元素的id获取对象  var elem=document.getElementById("id"); |

②通过对象，获取元素的内容/值

|  |
| --- |
| 双标签中间的东西，叫内容  单标签的value，叫值 |
| 获取内容，使用dom对象的innerHTML属性  获取：var res=elem.innerHTML;  设置：elem.innerHTML="abc"; |
| 获取值，使用dom对象的value属性 |

③使用dom操作的进阶

|  |
| --- |
| 获取对象的方案进阶  1.不使用id就能获取对象(后期xml中要使用)  通过标签名称获取当前页面中所有的该标签的元素，通过下标，获取某一个元素  var elem=document.getElementsByTagName("div")[0];  2.使用id获取对象的简写方式，ES6支持的写法  直接使用id代表对象  <div id="d1">测试文本1</div>  var res=d1.innerHTML; |

练习：01\_ex.html 一个div，一个input-text,button

需求，点击按钮，把input-text中填入的文本

显示到div上

|  |
| --- |
| 对于innerHTML的进阶  innerHTML是可以改变html的结构的  <div id="d1">  <a href="">123</a>  </div>  <script>  console.log(d1.innerHTML);  //innerHTML是可以改变html的结构的  d1.innerHTML+="<h1>abc</h1>"  </script> |

④事件

|  |
| --- |
| 通过用户的行为来激发的操作，就是事件  用户发生一个行为，这个行为绑定的方法就会被调用  button的事件  onclick 单击事件  文本框和密码框  onblur 焦点移除事件  onfocus 获取焦点事件  body标签的事件  onload 页面加载完成后，马上调用js方法 |

练习

|  |
| --- |
| 03\_ex.html  1.页面中创建如下元素  文本框 uname  密码框 upwd  重复验证密码框 cpwd  在上面三个元素后面各创建一个span（3个span）  2.需求  1.文本框获取焦点时，span提示用户名6~10位  失去焦点时，非空验证，显示span中  2.密码框同上  3.cpwd失去焦点时，判断与upwd的值是否一致，提示写在span中 |

### 二.Ajax

1.同步Synchronous

|  |
| --- |
| 在一个任务进行的过程中，不能开启其它任务  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响应，不能做其他事  同步访问的出现场合：  1.地址栏输入url，是同步访问  2.a标签跳转  3.form提交 |

2.异步Asynchronous

|  |
| --- |
| 在一个任务开启时，可以进行其它的任务  异步访问：浏览器在向服务器发送请求时，用户可以在页面上做其它的操作  使用异步的场合：  1.用户名重复的验证  2.聊天室  3.百度搜索建议  4.股票走势图 |

3.什么是ajax

|  |
| --- |
| Asynchronous JavaScript And XML  异步的 js 和 xml  本质：使用js提供的异步对象，  异步的向服务器发送请求，  并接收响应回来的数据  异步对象 XMLHttpRequest |

4.使用ajax

|  |
| --- |
| 1.创建异步对象  2.创建请求  3.发送请求  4.接收响应数据 |

①创建异步对象

|  |
| --- |
| var xhr=new XMLHttpRequest();  这种创建方式，不兼容IE8以下的版本  下面是兼容ie8以下版本的创建方式  if(window.XMLHttpRequest){  //如果有这个属性，说明是IE8以上的浏览器  var xhr=new XMLHttpRequest();  }else{  //如果没有这个属性,说明是IE8以下的浏览器  var xhr=new  ActiveXObject("Microsoft.XMLHttp");  } |

②创建请求，打开连接

|  |
| --- |
| xhr.open(method,url,isAsyn);  method：string类型，请求的方法  url:string类型，请求的url  isAsyn:boolean，是不是要采用异步的方式提交请求 |

③发送请求

|  |
| --- |
| xhr.send(formdata);  注意：只有post请求，才有请求主体formdata  get方法不需要请求主体。  所以使用get请求的时候，有两种发送请求的写法  xhr.send(),xhr.send(null) |

④接收响应

|  |
| --- |
| 1.xhr.readyState属性  用于表示xhr对象的 请求状态  一共有5个状态  0：请求尚未初始化  1：已经打开连接，请求正在发送中  2：接收响应头  3：接收响应主体  4：接收响应数据成功 |
| 2.xhr.onreadystatechange监听  监听xhr.readyState值的改变，每改变一次，方法会调用一次，一共会调用4次。  注意：只有当xhr.readyState=4的时候，是我们要接收响应正确时刻 |
| 3.xhr.status响应状态码  只有响应状态码为200的，我们才接收响应  4.响应数据放在了xhr.responseText属性中 |
| xhr.onreadystatechange=function(){ if(xhr.readyState==4&&xhr.status==200){ var result=xhr.responseText;  console.log(result);  }  } |

5.带参数的get请求

### 注意(错误总汇)：

|  |
| --- |
| 1.服务器中的get接口，可以使用浏览器地址栏验证 |
| 2.跨域错误    把ajax的html文件放入node服务器的静态资源中托管，使用访问服务器的方式打开页面 |
| 3.大括号缺失 |
| 4.没有错误信息的错误  使用console.log在不同位置打桩观察 |
| 5.错误的符号 |

作业1：

|  |
| --- |
| 使用get请求，完成完整的登录模块  input输入---ajax异步发送请求---服务器接收数据--查询数据库--发送响应--页面接收响应 |

作业2：

|  |
| --- |
| 使用get请求，获取商品列表，把响应得到的数据在div中显示 |

提高作业：

|  |
| --- |
| 注册验证，验证新用户名是否可用  模仿学子商城注册页面，使用onblur事件 |

## day06 ajax

### 作业1：

|  |
| --- |
| 使用get请求，完成完整的登录模块  input输入---ajax异步发送请求---服务器接收数据--查询数据库--发送响应--页面接收响应 |
| 1.完成接口  2.完成ajax  C:\Users\web\Desktop\web1903\day06\day06_getlogin_html.pngC:\Users\web\Desktop\web1903\day06\day06_getlogin_node.png |

### 一.ajax的post请求

1.任务，使用get请求再完成一遍登录模块 10:45我写

接口名称 login\_post

2.使用post方法发送请求

①xhr.open，需要改成post方法

②xhr.open中的url，不需要？和参数了

③添加创建请求主体的代码，把数据放到请求主体中

④修改请求消息头，可以发送特殊符号

⑤xhr.send(请求主体)

|  |
| --- |
| //url不要?和后面参数  var url="http://127.0.0.1:8080/ajax/login\_post";  xhr.open("post",url,true);  //创建请求主体  var formdata="uname="+u\_name  +"&upwd="+u\_pwd;  //由于ajax默认传输是text/plain  //无法传递特殊符号，我们需要更改消息头  //让ajax请求可以传递特殊符号  xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  xhr.send(formdata); |

### 二.Json的解析

### 作业2：

|  |
| --- |
| 使用get请求，获取商品列表，把响应得到的数据在div中显示  1.创建一个接口userlist  2.在html使用ajax |
| 前台，ajax内部，xhr.responseText得到数据，是string类型。string中放着js对象组成的数组  1.js对象的数据格式  var obj={name:"abc",age:18}  语法：var 对象名称={属性1：值1，属性2：值2.....}  注意：js对象中，属性名没有双/单引号  2.JSON数据格式  json是一个string字符串，亲切的称之为json串  JavaScript object Notation  js 对象 表示法  以js对象的格式表现出来的字符串  3.json字符串的格式  ①用一对{}表示一个对象  ②json中对象的属性名，必须使用""引起来(使用单引号也是正确的，但是不推荐)  ③属性的值如果是字符串，也要带双引号  ④json的表现是一个字符串，所有最外层加引号(推荐使用单引号)  ⑤请求访问服务器，调用数据库代码，得到的结果，默认是一个json串  4.json解析，把json字符串转换成js对象数组  var arr=JSON.parse(result); |

练习

|  |
| --- |
| 04\_list.html 重写一遍get方法获取userlist01接口  要求，数据最后变成下图样式 15:05~15:20休息  C:\Users\web\Desktop\web1903\web第二阶段练习效果图示\ajax用户列表效果.png |

### 三.XML的解析

1.什么是XML

|  |
| --- |
| eXtensible Markup Language  可拓展 标记 语言  是html的一个变种，专门负责承载数据用的  xml中标签，关键字/属性都需要自己定义  语法  1.第一行写版本声明  <?xml version="1.0" encoding="utf-8"?>  2.xml的标记只有双标记，必须成对出现  3.xml的标签严格区分大小写，开始标签和结束标记必须一致  4.xml标记，允许嵌套，但是要注意嵌套顺序  5.每个xml文档，有且只有一个根元素 |

2.xml解析

|  |
| --- |
| 使用dom解析  ??.getElementsByTagName("student")  ①不能使用xhr.responseText获取xml。这样获取的xml不能解析。要使用var result=xhr.responseXML;  ② |

### 四.项目阶段

### 常见错误

|  |
| --- |
| 数据库出问题，有可能是sql语句错误 |
| 设置请求消息头，必须在xhr.opne和xhr.send之间写 |
| xml文件中，有结构错误 |

ajax项目

### 一.搭建项目结构

|  |
| --- |
| router文件夹中创建路由pro.js---------接口  挂载路由  创建静态文件夹mypro，与public平级-------html  静态文件夹资源的托管 |

1.登陆模块

|  |
| --- |
| 1.在pro.js中定义接口login  使用post方法，接收前端传递过来的用户名和密码  查询数据库，给出响应  2.在mypro文件夹下，新建网页01\_login.html  添加文本框和密码，点击按钮完成ajax异步请求  登陆响应结果使用弹窗显示17:13完成 |

2.显示用户列表的模块

|  |
| --- |
| 1.在pro.js中创建接口list get  查询用户所有数据，并响应给前端  2.在mypro文件夹中，创建网页02\_list.html  页面加载时，发起ajax异步请求。把数据格式显示成下图模式  C:\Users\web\Desktop\web1903\web第二阶段练习效果图示\ajax用户列表效果.png |

作业：

1.使用get方法获取userlist接口数据，显示成表格



2.删除

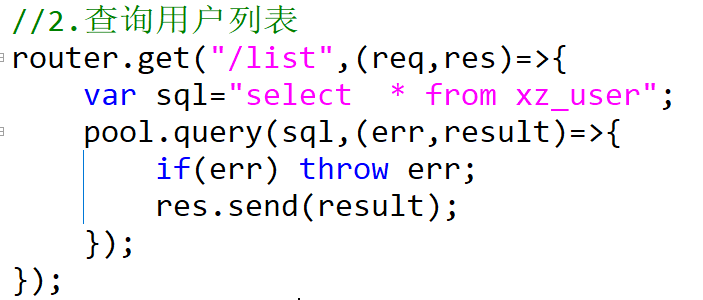
|  |
| --- |
| input-text输入uid,点击按钮，删除数据库中数据 |

## day07 模块

### 一.登录模块

### 二.查询所有的用户信息9:35我写

|  |
| --- |
| 1.在pro.js中创建接口list get  查询用户所有数据，并响应给前端  2.在mypro文件夹中，创建网页02\_list.html  页面加载时，发起ajax异步请求。把数据格式显示成下图模式  ajax用户列表效果 |



### 三.删除

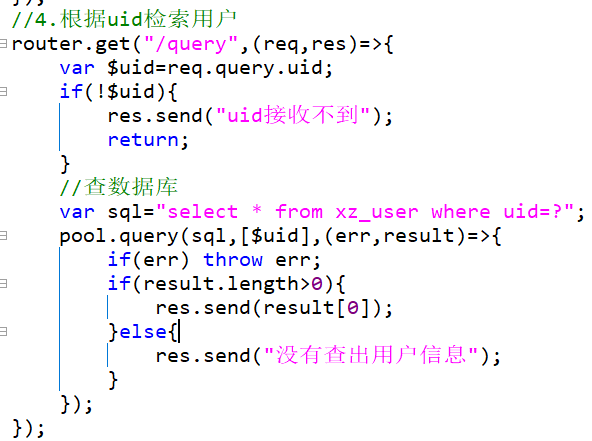
|  |
| --- |
| input-text输入uid,点击按钮，删除数据库中数据  get方法 |





### 四.修改模块

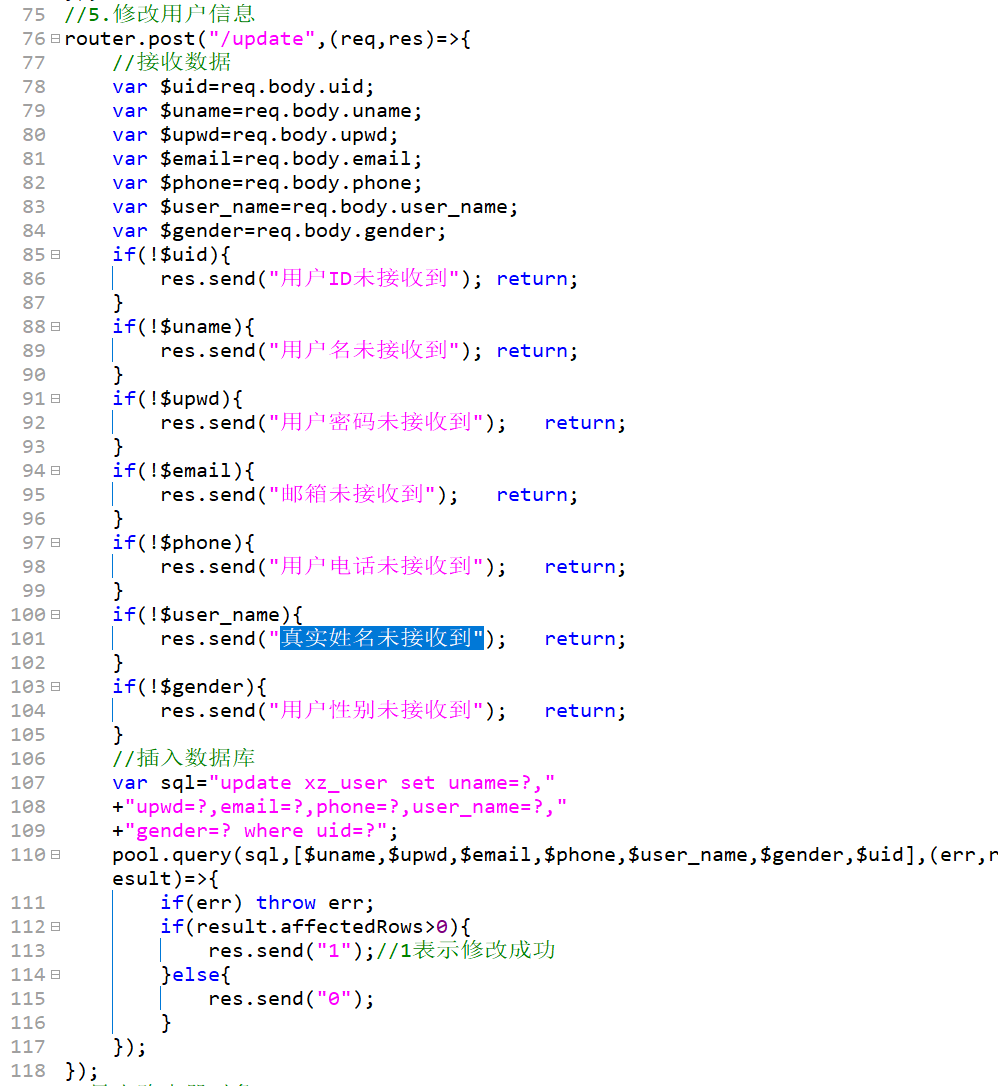
1.根据uid检索用户





2.修改模块

接口post方法 /update



作业：

1.注册

|  |
| --- |
| 1.注册名重复的验证  2.账户注册 |

常见错误

|  |
| --- |
| 服务器没有接受到参数，把提示当成响应传递给前台  前台把提示按照json格式解析了 |

## day08 CSS

### 一.CSS概述

1.什么是css

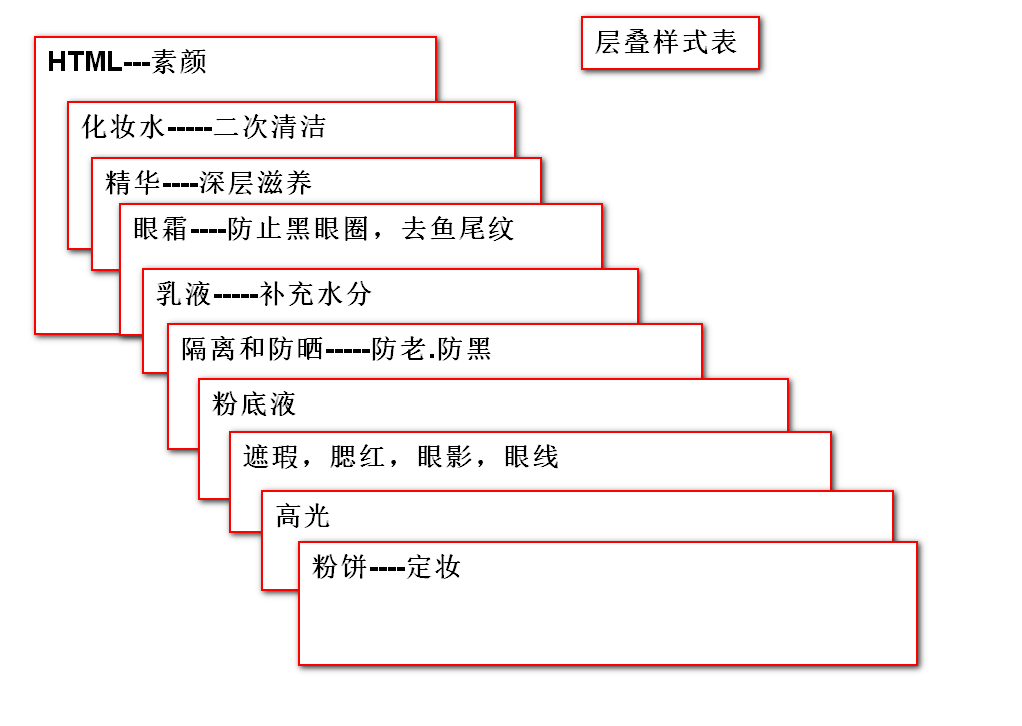
|  |
| --- |
| Cascading style sheets 层叠样式表，级联样式表  简称样式表 |

2.css的作用

|  |
| --- |
| 设置html网页中元素的样式，美化页面 |

3.HTML与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的属性，使用原则

|  |
| --- |
| W3C规定我们尽量的使用css的方式来取代HTML属性  css样式  1.样式代码可以高度重用，html属性不能重用  2.方便后期维护，提高可维护性。 |

### 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)

|  |
| --- |
| 在html标签的style属性中，编写样式  语法：<any style="样式声明"></any> any:任意标签  样式声明：样式属性 : 值  有多个样式声明：样式属性1：值1 ；样式属性2：值2  ex:  <div style="color:red;background-color:yellow">  内联样式在项目中，很少使用。  学习和测试的时候用  原因：1.内联样式不能重用  2.内联样式优先级最高 |

目前常用的样式

|  |
| --- |
| color:orange; 字体颜色  background-color:yellow; 背景颜色  font-size:36px； 字号大小 |

②内部样式

|  |
| --- |
| 在网页头部中，添加一对<style></style>标签  在style标签定义此网页所有的样式  <style>  选择器{样式属性：值；样式属性：值；........}  </style>  选择器：就是一个条件，符合这个条件元素，可以应用这个样式  项目中使用内部样式较少，学习和测试使用较多  内部样式的重用不能在其他html页面中生效 |

③外部样式

|  |
| --- |
| 单独创建一个css文件，编写样式  在html页面的head中使用link，引入这个外部样式  <link rel="stylesheet" href="css文件的url">  注意：rel="stylesheet"必须写，不然无效  此种方式，是开发中使用最多方式。 |

练习

|  |
| --- |
| 01\_ex.html  h1 内容随意，内联，背景为粉色pink，字体为yellow  h2 内容随意，内部，背景为粉色yellow，字体为red  h3 内容随意，外部，背景为粉色red，字体为blue |

2.CSS的特性

①继承性

|  |
| --- |
| 大部分的css效果，是可以被子元素继承的  必须是父子结构，子继承父 |

②层叠性

|  |
| --- |
| 可以为一个元素定义多个样式规则  多个规则中样式属性不冲突时，可以同时应用到当前元素上 |

③优先级

|  |
| --- |
| 如果对一个元素的多个样式声明，发生冲突时，按照样式规则的优先级去应用  默认优先级：由高到低  1.内联样式(行内样式)  2.内部样式和外部样式，就近原则  3.浏览器默认样式 |

④调整优先级

|  |
| --- |
| !important规则  放在属性值后面，与值之间有一个空格  作用，调整样式显示的优先级  内联样式不可以写!important  !important优先级，比内联样式高 |

练习

|  |
| --- |
| 02\_ex.html中  一个p标签，内容假文lorem  用内部样式设置文本颜色为蓝色，字号24px  用外部样式设置文本颜色为 红色，字号40px  让外部样式引入，F12查看页面效果  然后对调内部样式和link的位置，F12观察结果  尝试使用!important调整样式优先级 |

### 三.基础选择器(重点\*\*\*\*\*\*\*\*\*)

1.选择器的作用

|  |
| --- |
| 规范了页面中哪些元素能够使用定义好的样式  选择器就是为了匹配元素用  (选择器是一个条件，符合这个条件的元素就可以使用这个样式) |

2.选择器详解

①通用选择器

|  |
| --- |
| \*{样式声明}  由于\*的效率低下，项目中很少使用通用选择器  唯一使用的场合\*{margin:0;padding:0}所有元素内外边距清0 |

②元素选择器(标签选择器)

|  |
| --- |
| 页面中所有对应的元素，都应用这个样式  设置页面中某种元素的默认样式  元素名称{样式声明}  ex:p{} div{} |

③ID选择器，专属定制

|  |
| --- |
| <any id="id值"></any>  #id值{样式声明} 这种写法仅仅对页面上一个标签生效  一般id选择器在项目中很少单独使用。  通常会作为子代选择器或者后代选择器的一部分 |

练习

|  |
| --- |
| 03\_ex.html h2标签，内容随意(假文)，这个h2标签id为text1  使用id选择器设置文本为purple，背景yellow，  字体为斜体font-style:italic；查看页面观察效果  再使用元素选择器，设置文本为红色，背景pink。  这里有坑，会对就近原则产生怀疑，选择器权值问题 |

④类选择器

|  |
| --- |
| <any class="类名"></any>  .类名{样式声明}  作用：定义页面上某个或者某类元素的样式  (公共样式，谁想用，谁就可以用)  class属性来引用类名  类名的规则：1.类名之前的点不能省略，引用的时候没有点  2.类名不能以数字开头  3.类名只能包含 - \_ 符号  4.类名尽量的见名知意  类选择器的特殊用法  1.多类选择器  让元素引用多个类名，这些类的样式都会作用到当前元素上  <div class="text-danger bg-warning font-24">d</div>  2.分类选择器  元素选择器+类选择器{}  div.text-danger{ }div元素有text-danger类,才能应用这个样式  类选择器+类选择器{}  .font-24.text-danger{ }  元素，必须有font-24类和text-danger，才能应用此样式  作用：1.更精确的确定使用样式的元素  2.增加选择器的权值 |

练习

|  |
| --- |
| 04\_ex.html 页面中有div和p元素，内容假文随意  用类选择器设置字体颜色为红色，注意类名的命名规范  用分类选择器为p元素设置背景色为green |

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,......{公共样式声明}  ex:  #content,div,.mycolor,p.text{color:red}  相当于  #content{color:red}  div {color:red}  p.text {color:red}  .mycolor {color:red} |

⑥后代选择器

|  |
| --- |
| 通过元素的后代关系匹配元素  后代：一级嵌套或者多级嵌套  语法：选择器1 选择器2 选择器3........{}  #content p span{color:red;}  id我content的元素，内部不管隔着多少代，有一个p元素  p元素内部，不管隔着多少代，有一个span.这个span就符合要求 |

⑦子代选择器

|  |
| --- |
| 通过元素的子代关系匹配元素  子代：一级嵌套，直接的儿子  选择器1>选择器2>....{}  子代选择器和后代选择器可以混写  #content p>span{background-color:yellow;} |

练习

|  |
| --- |
| 05\_ex.html  ul#content>li\*4>span>a  a标签字体36px，字体颜色red，背景颜色yellow  使用后代或者子代选择器，至少3种选择器写法 |

⑧伪类选择器

|  |
| --- |
| 匹配同一个元素，不同的状态下的样式  所有的伪类选择器都是这样开头的 选择器:  1.链接伪类  a标签，没有访问的状态  a:link{color:red;}  a标签，被访问过之后的状态  a:visited{color:black;}  2.动态伪类  :hover 鼠标悬停在元素上时，元素的样式  :active 匹配元素被激活时的状态  :focus 匹配元素获取焦点时的状态 |

练习

|  |
| --- |
| 1.06\_ex.html  一个a标签，内容随意，href随意-------------坑  1.访问后，文本颜色orange  2.被激活时，文本颜色绿色  3.鼠标悬停时，文本颜色红色  4.未被访问时，文本pink  2.页面在红添加一个input--text  默认字体为gray，font-style:italic；  1.被激活时，字体不斜：font-style:normal;  2.获取焦点时：文本为红色 |

⑨选择器的权值

|  |
| --- |
| 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  类选择器 10  元素选择器 1  \*通用选择器 0  继承的样式 无 |
| 权值的特点  1.当一个选择器中含有多个选择器时，需要将所有的选择器权值进行相加，然后比较，权值大的优先显示  2.权值相同，就近原则  3.群组选择器的权值，单独各算各的，不能相加  4.样式后面追加!important直接获取最高优先级  内联样式不能加!important  5.选择器的权值计算，结果不会超过自己的最大数量级  (1000个1相加，也不会大于10) |

作业1：css使用方法3个

css特性3个

选择器8个

权值

上述知识点的demo重做一遍

作业2：重新完成学子注册模块

## day09 框模型

### 四.尺寸和边框

1.尺寸属性

|  |
| --- |
| 作用，设置元素的宽高  属性：  width: 设置宽  height:设置高  ----------------------  max-width 最大宽度  min-width 最小宽度  max-height 最大高度  min-height 最小高度  使用场合，响应式布局  取值：以px为单位的数字  父元素尺寸的%百分比 |

附加知识点：单位

|  |
| --- |
| 尺寸的单位  1.px像素  2.in英寸 1in=2.54cm  3.pt 磅值，多用于修饰字体大小粗细 1pt=1/72in  4.cm  5.mm  ---------相对单位-------------  6. % 父元素的百分之多少  7.em 是相对于父元素数值的单位  8.rem 是相对html标签数值的单位 |

2.页面中允许设置尺寸的元素

①块级元素

|  |
| --- |
| 所有的块级元素都可以设置尺寸  块级元素不设置宽，宽度占父元素100%。  块级元素不设置高，高度靠内容撑开，没有内容，就没有高 |

②行内元素

|  |
| --- |
| 行内元素设置宽高无效，  行内元素宽高，是靠内容撑开  但是，自带宽高属性的行内元素，可以设置尺寸 |

③table

|  |
| --- |
| table自带宽高属性，可以设置宽高 |

④行内块

|  |
| --- |
| input  行内块可以设置宽高 |

3.溢出处理

|  |
| --- |
| 当内容较大，元素区域较小，就会发生溢出效果  默认是纵向溢出  属性 overflow  取值 1.visible 默认值，溢出部分可见  2.hidden 溢出部分隐藏  3.scroll，不管是否溢出，都添加滚动条。  不溢出的时候，滚动条不能拖动  4.auto 自动，溢出的时候，溢出的方向有滚动条。  不溢出的时候，没有  控制滚动条的方向  overflow-x/overflow-y  如何让内容横向溢出  需要在宽度比较小的容器内部，添加一个宽度较大的元素盛放内容  在父容器上写overflow:auto。就可以做到横向溢出 |

附加知识点---颜色

|  |
| --- |
| 合法颜色值  1.颜色的英文单词(red,blue,yellow,pink,purple......)  2.#rrggbb  #000000---黑 #ffffff---白 #ff0000---红  #00ff00---绿 #0000ff---蓝  3.#aabbcc---->#abc  #000 #fff #f00 #0f0 #00f #666  4.rgb(0~255,0~255,0~255)  rgb(255,0,0)  5.rgba(r,g,b,alpha) alpha透明度 0~1 1不透明，0透明  #f00 #0f0 #00f #ff0 #0ff #f0f  #faa #afa #aaf #ffa #aff #faf |

4.边框

①边框属性---简写方式---4个方向边框一起设置

|  |
| --- |
| border:width style color;  width:边框的宽度，取值以px为单位的数字  style:边框的样式 取值：solid实线  dotted 点点虚线  dashed 线状虚线  double 双实线  color：边框的颜色 取值 合法的颜色值/transparent(透明)  最简方式 border:style;  取消边框 border:0; 或者 border:none; |

练习

|  |
| --- |
| 03\_ex.html  3个div,给每个div设置边框，每个div 10px\*10px  第一个 1px 实线 红色  第二个 5px 点虚线 蓝色  第三个 10px 线虚线 黄色 |

②单边定义边框

|  |
| --- |
| 只设置某一条边的3个属性  border-top：width style color;  border-right  border-bottom  border-left |

练习

|  |
| --- |
| 03\_ex中，div#d4 200px\*200px  上边框，3px实线 红色  右边框 5px 点 蓝色  下边框 10px 线虚线 黄色  左边框 15px 双实线 #f0f |

③单属性定义

|  |
| --- |
| border-width:  border-style:  border-color:  ex:  border-width:3px;  border-style:solid;  border-color:#f00;  border:3px solid #f00; |

④单边单属性

|  |
| --- |
| border-top-width:  border-top-style:  border-top-color:  border-right-width:  border-right-style:  border-right-color:  border-bottom-width:  border-bottom-style:  border-bottom-color:  border-left-width:  border-left-style:  border-left-color: |

5.边框的倒角(圆角)

|  |
| --- |
| 将直角设置成倒角，圆角  border-radius：  取值：以px为单位数字  %  50%就是个圆  单角设置  border-top-left-radius  border-top-right-radius  border-bottom-left-radius  border-bottom-right-radius |

练习

|  |
| --- |
| 画柠檬/芒果 |

6.边框阴影

|  |
| --- |
| box-shadow  取值：h-shadow v-shadow blur spread color inset;  h-shadow:水平方向的阴影偏移量  +：往右移动， -：往左移动  v-shadow:垂直方向的阴影偏移量  +：往下 -：往上  blur：阴影模糊距离，越大越模糊  无负值  spread：阴影尺寸，在基础阴影上扩出来的大小  负值，尺寸变小  color:阴影颜色  inset:向内扩撒阴影 |

练习

|  |
| --- |
| 圆形发光体 太阳，吸顶灯，日食，月食 |

7.轮廓

|  |
| --- |
| 轮廓指的是边框的边框，绘制于边框外的线条  outline:width style color;  outline:none;或者outline:0 去除 轮廓 |

### 五.框模型---盒子模型

|  |
| --- |
| 框模型--元素在页面上实际占地空间的计算方式  默认情况，一个元素在页面的实际占地宽度  左外边距+左边框宽度+左内边距+内容区域宽度+右内边距+右边框+右外边距  实际占地高度  上外边距+上边框宽度+上内边距+内容区域高度+下内边距+下边框+下外边距 |

### 1.外边距margin

|  |
| --- |
| 元素边框以外的距离，改变margin，元素有位移的效果 |

①语法

|  |
| --- |
| margin:v1; 设置4个方向的外边距  margin:v1 v2; v1上下 v2左右  margin:0 auto; 块级元素水平居中  auto对垂直外边距无效  margin:auto;  auto只对设置了宽的块级元素生效  margin:v1 v2 v3; v1上 v2左右 v3下  maring:v1 v2 v3 v4; 上右下左  单方向外边距设置  margin-top:  margin-right:  margin-bottom:  margin-left:  取值：1.以px为单位数字  2.%  3.+ margin-top ↓ margin-left →  - margin-top ↑ margin-left ←  4.auto 自动计算块级元素外边距，让块级元素水平居中  auto只对设置了宽度的块级生效  auto对下上外边距无效 |

练习

|  |
| --- |
| 05\_ex  两个div，尺寸300px\*300px  背景色随意。  设置两个div之间的间距是50px，  两个div都水平居中 |

2.外边距的特殊效果

①外边距合并

|  |
| --- |
| 两个垂直外边距相遇时，他们将合并成一个，值以大的为准  只能在布局设计的时候，尽量避免发生 |

作业

1：坑

|  |
| --- |
| 页面中两个div，宽高200px,分别设置背景颜色  为两个div设置4个方向外边距，观察效果  与div平级，添加两个span，内容随意，分别设置背景颜色  为两个span添加4个方向外边距，观察效果  与div平级，添加两个input-text，  为两个input添加4个方向外边距，观察效果 |

2.坑

|  |
| --- |
| div#d1>div#d2  #d1宽高300px  #d2宽高100px  分别设置背景颜色  给#d2设置上外边距，观察效果 |

3.今天所有知识点的demo

4.ajax 登录模块/注册模块/查询用户列表/删除/修改

## day10 渐变

### 五.框模型

#### 2.外边距的特殊效果

①外边距合并

|  |
| --- |
| 两个垂直外边距相遇时，他们将合并成一个。值以最大的为准。 |

练习

|  |
| --- |
| 01\_ex  1.两个div 200px\*200px 背景颜色不同  为两个div单独设置4个方向外边距 margin-方向  2.两个span，内容随意，背景颜色不同  为两个span单独设置4个方向外边距  3.两个文本框，  为两个input单独设置4个方向外边距 |

②关于块级元素，行内元素，行内块的外边距总结

|  |
| --- |
| 行内元素的特点：  设置宽高无效，宽高根据内容自动撑开  上下垂直外边距无效。可以与其他的行内元素和行内块元素共用一行 |
| 块级元素特点：  设置宽高有效。如果不设置宽高，高按内容撑开，宽占父元素宽度的100%。上下外边距有效，独占一行. |
| 行内块的特点：input  设置宽高有效，不设置宽高，自带默认宽高。  上下外边距有效，但是同一行修改一个行内块的垂直外边距，整行都会跟着一起改变位置。可以与其他行内元素和行内块共用一行 |

③自带外边距的元素

|  |
| --- |
| body h1~h6 p ol ul dl pre  由于不同浏览器对默认的外边距解析可能有差别  所以一般情况下，开发之前，需要把内外边距清空  \*{margin:0;padding:0}  或者通过群组选择器来写  blockquote,body,button,dd,dl,dt,fieldset,form,  h1,h2,h3,h4,h5,h6,hr,input,legend,li,ol,  p,pre,td,textarea,th,ul{margin:0;padding:0} |

练习

|  |
| --- |
| 01\_ex  div#d3 尺寸300px\*300px  #d3中有div#d4 100px\*100px  两个div不同背景颜色。  给#d4设置上外边距，在f12中改变上外边距的数值  观察效果 |

④外边距溢出

|  |
| --- |
| 在特殊情况下，为子元素设置外边距，会作用到父元素上  特殊情况：  1.父元素没有上边框  2.子元素内容区域上边与父元素内容区域上边重合的时候  (为父元素中第一个子元素设置上外边距时，这种说法不严谨)  解决方案：  1.给父元素添加上边框，弊端，影响父元素实际占地高度  2.给父元素添加上内边距，弊端，影响父元素实际占地高度  3.给父元素的一个子元素的位置添加空table元素 |

#### 3.内边距

|  |
| --- |
| 内边距的改变效果，感觉是改变了元素的大小  不会影响其它元素，但是真正改变的是本元素的占地尺寸  内边距颜色和元素背景相同  语法：  padding:v1; 设置4个方向的内边距  padding:v1 v2; 上下 左右  padding无auto  padding:v1 v2 v3; 上 左右 下  padding:v1 v2 v3 v4;上右下左  单方向设置  padding-top  padding-right  padding-bottom  padding-left  取值，以px为单位的数字 %  padding无auto值 |

#### 4.box-sizing属性(设置元素实际占地尺寸的公式)

|  |
| --- |
| box-sizing:  取值：1.content-box.默认值，按照之前的盒子模型计算元素占地大小(左外+左边+左内+内容区域宽度+右内+右边+右外)  2.border-box。设置的width和height是包含border+padding+内容区域的 整体宽高  元素实际占地大小  左外+设置的width+右外  上外+设置的height+下外  给复杂元素关系做布局时，经常使用border-box |

### 六.背景

1.背景颜色

|  |
| --- |
| background-color:合法颜色值 |

2.背景图片

|  |
| --- |
| background-image:url(08.png);  使用背景图片，可以让该元素的子元素，堆叠显示在背景图片上。  而使用img标签，默认不会有堆叠效果 |

3.背景图片的平铺

|  |
| --- |
| background-repeat:  取值：repeat 默认值，平铺  no-repeat 不平铺  repeat-x 水平方向平铺  repeat-y 垂直方向平铺 |

4.背景图片的定位

|  |
| --- |
| background-position:  取值：1. x y 以px为单位的数字  2.x% y% 百分比  3.关键字 x:left/center/right  y:top/center/bottom |

5.背景图片的尺寸

|  |
| --- |
| background-size:  取值：1. x y 以px为单位设置具体宽高  2.x% y% 按父元素宽高比设置  以元素的角度考虑这两个单词  3.cover 充满，只要求元素被背景图充满，背景图显示不全，也没关系.  4.contain 包含，只要求元素要把背景图整个包含住，元素有空白，也没关系 |

6.背景图片的固定

|  |
| --- |
| background-attachment  取值： scroll 默认值，背景图会随着窗口滚动条滚动  fixed 固定，背景图相对于窗口固定，窗口滚动时，背景图位置不变，但是只会在原容器内显示 |

7.背景的简写方式

|  |
| --- |
| background:  取值 color image repeat attachment position  最精简方式 background:color/image; |

### 七.渐变 gradient

1.什么是渐变

|  |
| --- |
| 渐变是指多种颜色，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，以及这种颜色出现的位置  一个渐变，最少两个色标 |

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，来填充渐变色  3.重复渐变，将线性，径向渐变重复几次显示 |

4.线性渐变

|  |
| --- |
| background-image: linear-gradient(angle,color-point1, color-point2......);  angle:表示渐变的方向  取值 1.关键字 to top 从下往上  to right 从左往右  to bottom 从上往下  to left 从右往左  2.角度 0deg to top  90deg to right  180deg to bottom  270deg to left  可以取负值  color-point:色标 颜色 %/px |

5.径向渐变

|  |
| --- |
| background-image:  radial-gradient(半径 at 圆心x 圆心y,color-point1,  color-point2.................)  半径 以px为单位的数字  圆心 1 x y 以px为单位的数字  2. x% y%  3关键字 x left/center/right  y top/center/bottom  color-point: 位置取值 1. % 半径的百分比%  2.px为单位的数字，半径失效 |

6.重复渐变

|  |
| --- |
| 重复的线性渐变  background-image:  repeating-linear-gradient(to left,#000 0px,  #ff0 25px,#000 50px);  重复的径向渐变  background-image:  repeating-radial-gradient(50px at center center,  #000 0%,#0ff 25%,#000 50%); 17:07~17:23休息 |

7.浏览器兼容问题

|  |
| --- |
| 如果有低版本(ie8.0以下)，想要使用渐变  chrome/safari -webkit-  firefox -moz-  opera -o-  IE -ms-  要低版本兼容渐变，需要在linear-gradient之前添加浏览器内核  background-image:  -webkit-linear-gradient()  background-image:  -moz-linear-gradient()  ......  兼容低版本浏览器，线性渐变的方向，要改变写法  不写to top/to left/to bottom/to right  改成初始点 top/left/bottom/right |

### 八.文本格式化(重点\*\*\*\*\*\*\*\*)

1.字体属性

①指定字号大小

|  |
| --- |
| font-size:  取值 px/pt/rem/em为单位的数字 |

②字体的类型

|  |
| --- |
| font-family |

一个页面中，编写css的思路

|  |
| --- |
| 从上往下，从左往右，从外往里  1.尺寸，大体位置  2.边框，背景相关  3.文本相关  4.微调 |

作业1：

margin:0 auto和text-align:center区别

作业2：完成登陆页的样式(table布局)



作业3.渐变的所有demo

## day11 定位

|  |
| --- |
| 项目顺序：  1.尺寸，大体位置 2.边框，背景 3.文本 4.微调 |

### 三.文本格式化(重点\*\*\*\*\*\*\*\*)

1.字体属性

①字号大小

|  |
| --- |
| font-size:  取值：px/pt/em/rem |

②设置字体类型

|  |
| --- |
| font-family:"mv boli",华文彩云,黑体;  在当前设备的字体库，查找字体，如果有就使用，如果没有，就查找下一个。  如果字体名称中间有空格，必须加双引号 |

③粗体

|  |
| --- |
| font-weight:  取值 1.关键字 lighter normal bold bolder  2.无单位的100的整倍数 一般400~1000 |

④字体样式

|  |
| --- |
| font-style:normal/italic; 正常和斜体 |

⑤小型大写字母

|  |
| --- |
| font-variant:small-caps; |

⑥简写方式

|  |
| --- |
| font:style variant weight size family;  font:italic small-caps bolder 40px chiller;  最精简的方式 font:size family; |

2.文本属性

①字体颜色

|  |
| --- |
| color |

②文本水平对齐方式

|  |
| --- |
| text-align:  取值 left/center/right/justify两端对齐 |
| margin:0 auto和text-align:center的区别  margin:0 auto，控制的是当前元素本身在页面中居中(让自己居中)  text-align:center 控制的是当前元素内部的内容，在元素内部居中(让我的儿子居中) |

③行高

|  |
| --- |
| 定义一行数据的高度  特性：如果行高的高度大于字体本身的大小  那么该行文本将在指定的行高内，呈垂直居中方式显示  line-height  取值：1.以px为单位的数字，一般，行高的值与容器高度相同。  这样就可以让文字在容器垂直中间显示了  2.无单位的数字(整数小数都可以)，行高为字号大小的倍数  注意:文字如果有多行，不建议使用行高，文字会溢出，每一行的行高都是你设置的大小 |

④文本线条修饰

|  |
| --- |
| text-decoration  取值：overline 上划线  underline 下划线  line-through 删除线  none 去掉所有的修饰线条  a{text-decoration:none}去除a标签下划线 |

⑤首行缩进

|  |
| --- |
| text-indent:  取值：px为单位的数字 |

⑥文本阴影

|  |
| --- |
| text-shadow:h-shadow v-shadow blur color;  h-shadow:水平偏移  v-shadow:垂直偏移  blur：阴影模糊度  color：阴影颜色 |

练习

|  |
| --- |
| 01\_ex  定义div，内容随意，假文，有中文  1.设置字体系列 2.设置字号 大小  3.加粗并斜体 4.所有小写字母转为小型大写字母  5.两端对齐 6.文本有下划线  7.首行缩进30px 8.文本垂直居中 |

### 四.表格的样式

1.表格的常用样式属性table>tr>td

|  |
| --- |
| 1.table:之前学习过的样式，基本都可以使用  尺寸，边框，背景，字体，文本，内外边距  给table设置border,只设置最外面的大边框 |
| 2.td/th的样式  尺寸，边框，背景，字体，文本，内边距  外边距无效  vertical-align指定单元格数据的垂直对齐方式  取值：top/middle/bottom |

练习

|  |
| --- |
| 02\_ex  创建4\*4表格，宽高400px。内容随意  1.设置每个单元格的尺寸为100px\*100px  2.设置每个单元格边框1px solid #000  3.尝试给每个单元格设置上外边距20px.上内边距20px； |

表格是一种特殊的表现方式

|  |
| --- |
| 表格实际尺寸是根据内容数据的多少而决定的  单元格小，内容多，自动撑开  内容少，单元格就按照设置的尺寸来展示 |

2.表格特有的样式属性

①边框合并

|  |
| --- |
| border-collapse:  取值 ：1.separate 默认值，边框分离模式  2.collapse;边框合并模式 |

②边框边距

|  |
| --- |
| 向设置边框边距，必须保证边框是分离状态  border-collapse:separate ;  属性border-spacing:  取值：1.只有一个值 设置水平和垂直边框的 外边距  2.两个值 第一个值设置水平 第二个值设置垂直 |

③标题位置

|  |
| --- |
| caption-side:  取值：top 默认值  bottom |

④设置表格的显示规则

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 用来确定如何布局一张表格  table-layout  取值 1.auto 默认值 自动布局表格  列的尺寸实际使用内容决定的，内容比尺寸大，按照  内容大小显示。内容比尺寸小，按照尺寸显示  2.fixed 固定表格布局，列的尺寸以设置的为准  自动布局与固定布局   |  |  | | --- | --- | | 自动布局 | 固定布局 | | 单元格的大小自动适应内容 | 单元格大小取决于设置的值 | | 表格复杂时，加载速度较慢 | 任何情况下，都会加速加载表格(优点) | | 比较灵活(优点) | 固定布局不够灵活 | | 适用于不确定每列大小的，并且不复杂的表格 | 适用于能够确定每列尺寸大小的表格 | |

练习



### 五.定位(重点\*\*\*\*\*\*\*\*\*)

1.什么是定位

|  |
| --- |
| 改变元素在页面中的位置 |

2.定位的分类

|  |
| --- |
| 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 |

①普通流定位

|  |
| --- |
| 又称为默认文档流定位  1.每个元素在页面上都有自己的空间  2.每个元素都是从父元素的左上角开始渲染（显示）  3.块级元素按照从上到下逐个排列，每个元素单独成行  4.行内元素是多个元素在一行中显示，从左往右顺序排列，显示不下换行 |

②浮动定位

|  |
| --- |
| 让块级元素横向显示  float：  取值：left 让元素浮动后，在当前行停靠在父元素的左侧，或者挨着左侧已浮动元素  right让元素浮动后，在当前行停靠在父元素的右侧，或者挨着右侧已浮动元素  none 默认值，不浮动  浮动的特点  1.元素一旦浮动，脱离文档流(不占页面空间，后面未浮动元素会上前补位)  2.浮动元素会在当前行，停靠在父元素的左/右边，或者停靠在其他已浮动元素的边缘  3.当父元素横向显示不下所有浮动元素时，最后显示不下的，会自动换行  4.浮动用于解决，多个块级元素在同一行显示的问题 |

③浮动定位引发的特殊情况

|  |
| --- |
| 1.浮动元素占位的问题  当元素显示不下所有浮动元素时，最后显示不下的元素会换行  但是，已浮动元素会根据自己的浮动方向占据位置，  导致被挤下去的浮动元素，需要让开位置，在更下面的地方显示 |
| 2.元素一旦浮动，如果元素未定义宽度，那么元素浮动之后的宽度将以内容为准 |
| 3.元素一旦浮动，都会变成块级元素  允许设置尺寸，可以设置垂直外边距 |
| 4.文本，行内元素，是不会被浮动元素压在下面的。而是巧妙的避开，环绕着浮动元素显示 |

④清除浮动

|  |
| --- |
| 清除之前的浮动元素给自己带来的影响  由于元素浮动之后，会脱离文档流，会让后续不浮动的元素上前补位.  如果后续元素不想上前补位，需要对此元素设置清除浮动  clear:  取值：1.left 清除左浮动对我的影响  2.right 清除右浮动对我的影响  3.both 清除左右浮动对我的影响  4.none 不清除影响 |

⑤高度坍塌

|  |
| --- |
| 什么叫高度坍塌  1.父元素不写高，靠子元素撑起高度  2.所有子元素都浮动  那么所有子元素都脱离文档流，父元素认为自己内部没有元素了  所以父元素就没有高度了  解决方案：  1.父元素也浮动，弊端，影响父元素后面的非浮动元素  2.直接给父元素写高度 弊端，不是每次都能知道具体高是多少  3.overflow：hidden 弊端，会让真正要溢出不能显示  4.在父元素中追加一个块级元素，这个块级元素，没有内容，不写高。只写clear:both。就可以让父元素在文档流中找到内容的高度  解决，高度坍塌问题 |

作业

1.掌握css编写思路



|  |
| --- |
| 1.宽高，大体位置  2.边框，背景  3.文字所有  4.微调 |

2.今天关于table和浮动的所有demo

3.回家安装WebStorm10.zip，并且保证注册成功

## day12 定位

### 一.元素的显示方式

|  |
| --- |
| display  取值： 1.block 让元素的表现和块级一致  2.inline 与行内元素一致  3.inline-block 与行内块元素一致  4.table 与table一致  5.none 不显示元素，隐藏  块级：独占一行，可以设置尺寸，上下外边距有效  行内：共用一行，尺寸无效，上下外边距无效  行内块：共用一行，可以设置尺寸，上下外边距有效  table：独占一行，可以设置尺寸，尺寸以内容为主 |

### 二.显示效果

|  |
| --- |
| visibility  取值：1.visible 默认值，可见  2.hidden 隐藏不可见  问题：visibility:hidden和display:none区别  visibility:hidden；隐藏，元素不脱离文档流，在当前页面不可见，但是占据位置  display:none；隐藏，元素脱离文档流，隐藏后不占位置，后面元素上前补位 |

### 三.透明度

|  |
| --- |
| opacity  取值：0~1 值越小越透明  问题：opacity和rgba的区别  rgba只会改变当前颜色的透明度  opacity，元素内部只要元素相关的颜色都会跟着透明 |

### 四.垂直对齐方式

|  |
| --- |
| vertical-align  使用场合  1.表格中 td/th  取值 top/middle/bottom  2.img与文字的排版  改变的是img与前后文本的对齐方式  取值 top/middle/bottom/baseline基线  通常，会将所有的图片与文字的对齐方式，改为非基线对齐的方式 |

### 五.光标的设置

|  |
| --- |
| cursor  取值：1.default 箭头  2.pointer 小手  3.crosshair 十字  4.text 文本输入的I  5.wait 等待  6.help 帮助 |

### 六.列表的样式

|  |
| --- |
| ul的样式  1.列表项标识  list-style-type:  取值 none/disc/circle/square  2.列表项标识，设置为图片(图片要小)  list-style-image:url(路径);  3.列表项标识的位置  list-style-position:inside/outside(默认值)  设置列表项是在li的内部还是外部  4.简写方式  list-style:type image position;  项目中最常用的写法list-style:none;  去掉列表标识 |

### 七.定位---相对，绝对，固定定位（\*\*\*\*\*\*\*\*\*）

|  |
| --- |
| position  取值：static 默认，静态(默认文档流定位)  relative 相对定位  absolute 绝对定位  fixed 固定定位  当一个元素，被position修饰，并且取值为relative/absolute/fixed其中一种时，这个元素被称为已定位元素  已定位元素，解锁了4个偏移属性  top + 往下 - 往上  right + 往左 -往右  bottom + 往上 -往下  left +往右 -往左  偏移属性，定义了元素距离某一个方向移动了多少距离 |

1.相对位移

|  |
| --- |
| 相对定位，相对自己原来位置偏移某个距离  position:relative;配合4个偏移属性使用  特点：1.不脱离文档流。后面元素不会上前补位  2.相对定位，如果不写偏移量，元素效果与没写定位是一样的.  使用场合  1.自身位置的微调  2.作为绝对定位的祖先级元素 |

2.绝对定位

|  |
| --- |
| position:absolute；配合偏移量使用  特点：  1.绝对定位，脱离文档流，元素不占页面空间，后面元素上前补位。  2.绝对定位的元素，会相对于  "离自己最近的""已定位的""祖先元素"实现位置的初始化  如果没有 "已定位的""祖先元素"，相对于body实现位置初始化  3.绝对定位元素会变成块级元素  4.绝对定位元素，如果不写宽，定义之后，宽靠内容撑开 |

3.固定定位

|  |
| --- |
| position:fixed;配合偏移量使用  将元素固定在页面的某个位置，不会随着滚动条发生位移变化  特点：1.脱离文档流  2.元素变为块级元素  3.不写宽的会被内容撑开  4.相对body做位置的初始化 |

4.堆叠顺序

|  |
| --- |
| 特点：  1.默认堆叠顺序，后定位的元素，堆叠顺序高  2.定位的脱离文档流，和浮动的脱离文档流，不是一个体系  3.使用z-index设置堆叠顺序  z-index:无单位数字 一般情况 1~1000  4.堆叠顺序，只对已定位元素有效  5.堆叠顺序，对父子级无效。子元素永远在父元素上面显示 |

CSS3 CORE

一.复杂选择器

1.兄弟选择器

|  |
| --- |
| 兄弟元素：具有相同父元素的平级元素  兄弟选择器，只能找弟弟，不能找哥哥 |

①相邻兄弟选择器

|  |
| --- |
| 获取紧紧挨在某元素后面的兄弟元素  选择器1+选择器2{} |

②通用兄弟选择器

|  |
| --- |
| 获取某元素后，所有符合要求的兄弟元素  选择器1~选择器2{} |

练习

|  |
| --- |
| <h3>小白兔</h3>  <div>小了白了兔</div>  <div>白了又了白</div>  <div>爱了吃了萝了卜了和了青了菜</div>  <div>蹦了蹦了跳了跳了真了可了爱</div>  标题与正文之间间距30px  正文每行之间间距10px |

2.属性选择器

|  |
| --- |
| id class style title name value width.......  允许通过元素所附带的属性，及其值来匹配页面元素，很精准 |

①[attr] attr代表任意属性

|  |
| --- |
| 匹配页面中所有带attr这个属性的元素  [id]{} [id][title]{} |

②elem[attr]

|  |
| --- |
| 匹配页面中所有带attr属性的elem元素  p[title]{} p[id][title][class].....{} |

③elem[attr=value]

|  |
| --- |
| 匹配页面中带attr属性，并且值为value的elem元素  div[title="woyouyeye"][class="c"]{} |

④模糊属性值匹配

|  |
| --- |
| [attr^=value]{} 匹配attr的值以value开头的元素  [attr$=value]{} 匹配attr的值以value结尾的元素  [attr\*=value]{} 匹配属性值中，有value的元素  [attr~=value]{} 匹配属性值中，有value这个独立单词的元素 |

3.伪类选择器

|  |
| --- |
| :link :visited :hover :active :focus 已经学过的伪类 |

①目标伪类(锚点)

|  |
| --- |
| 让被激活的锚点，应用样式  选择器：target{} |

练习

|  |
| --- |
| 09\_ex  点击菜名，显示图片  菜名放在ul>li>a  海鲜面 四季春饼 八大碗 |

作业：1.完成学子首页1楼



2.完成学子商城轮播图的静态页面



## day13 选择器/弹性/转换

### 一.复杂选择器

3.伪类选择器

①目标伪类target

②结构伪类

|  |
| --- |
| elem:first-child{} 代表两个条件  匹配elem的父元素的第一个儿子（elem的大哥）  这个大哥必须是elem元素  elem:last-child{}  匹配elem的父元素的最后一个儿子(elem的小弟)  这个最小的弟弟必须是elem元素  elem:nth-child(n)  匹配elem的父元素的第n个儿子(n从1开始)  这个儿子也必须是elem元素 |

练习

|  |
| --- |
| 01\_ex 4\*4的表格 200px\*200px. 有边框，内容随意  需求：表格第一行背景为红色  表格最后一行最后一列的背景为蓝色  表格第三行第二列背景为黄色 |

③匹配空元素

|  |
| --- |
| :empty{}  空元素  空元素：没有文本，没有空格，没有其他子元素的元素 |

④:only-child

|  |
| --- |
| 匹配，当前元素是其父元素的唯一的子元素 |

⑤否定伪类

|  |
| --- |
| :not(selector)  /\*除了第一个a标签，其它a的字体都变成黄色 \*/  a:not(:first-child){color:#ff0;}  /\*除了第三个a标签，其它a的字体都变成蓝\*/  a:not(:nth-child(3)){color:#f0f;} |

4.伪元素选择器

①内容伪元素

|  |
| --- |
| 1::first-letter{ } 或 :first-letter{}  匹配首字符的样式  2::first-line{ } 或 :first-line{}  匹配首行的样式  当首行和首字符冲突的时候，以首字符为准  3.::selection{} 必须是两个 ::  匹配选中部分的文字样式  注意：只能改变字体颜色和背景颜色 |

②伪元素选择器，内容生成

|  |
| --- |
| 使用css，添加httml元素，称之为伪元素内容生成  :before 或者 ::before  匹配到某个元素的内容区域的最前面，  添加一个内容，content：添加的文本或者图片  display：是设置这个添加的内容的显示规则  ::after 或者 :after  匹配到某个元素的内容区域的最后面，添加一个内容  注意：content中，只能添加文本或者图片  我们可以理解为在内容区域中，最前或者最后面，添加了一个元素。这个元素的显示方式，由display来设定 |

③伪元素内容添加，可以解决的问题

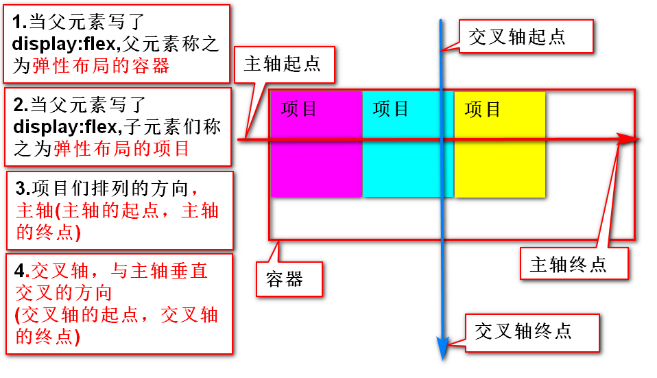
|  |
| --- |
| 1.外边距溢出  #parent::before{  content:"";  display:table;  }  2.思考，伪元素内容添加，解决高度坍塌？  #d1:after{  content:"";  display:block;  clear:both;  } |

### 二.弹性布局（重点\*\*\*\*\*\*\*\*\*）

1.什么是弹性布局

|  |
| --- |
| 是一种布局方式  主要解决某个元素中子元素的布局方式  为布局提供很大灵活性 |

2.弹性布局相关的概念和名词解释



|  |
| --- |
| 1.容器  要发生弹性布局的子元素，他们的父元素称之为容器  容器要设置属性display:flex;  2.项目  要发生弹性布局的子元素们，称之为项目  就是设置了display:flex那个元素的，子元素们  3.主轴  项目们在容器中排列的方向，就是主轴  如果项目横向排列，x轴就是主轴  如果项目纵向排列，y轴就是主轴  项目们的排列顺序，靠主轴的起点和主轴的终点来定义  4.交叉轴  与主轴垂直相交的一条轴，叫做交叉轴  项目们在交叉轴上的对齐方式，是交叉轴的起点和终点 |

3.语法

|  |
| --- |
| display，写在父元素中  取值：1.flex 将块级元素设置为容器  2.inline-flex 将行内元素设置为容器  特点：  1.弹性项目，默认x是主轴，主轴起点在左侧  (块级元素横向排列的第二个解决方案)  2.项目的float/clear/text-align/vertical-align属性失效  3.每个项目可以自由的设置尺寸 |

4.容器的属性

①主轴的方向

|  |
| --- |
| flex-direction:  取值： row默认值，主轴是x轴，主轴起点是左端  row-reverse,主轴是x轴，主轴起点是右端  column 主轴是y轴，主轴起点是顶端  column-reverse 主轴是y轴，主轴起点是底部 |

②设置项目换行

|  |
| --- |
| flex-wrap:  取值 nowrap 默认值，容器空间不够，也不换行，项目自动缩小  wrap 空间不够就换行  wrap-reverse 换行并反转 |

③主轴方向，项目换行的缩写

|  |
| --- |
| flex-flow:  取值 direction wrap |

④定义项目在主轴上的对齐方式

|  |
| --- |
| justify-content:  取值 1.flex-start 默认值，以主轴起点对齐  2.flex-end 以主轴终点对齐  3.center 在主轴上居中对齐  4.space-between 主轴两端对齐，两端无空白  5.space-around 每个间距大小相同 |

⑤项目在交叉轴上的对齐

|  |
| --- |
| align-items:  取值1.flex-start 交叉轴起点对齐  2.flex-end 交叉轴终点对齐  3.center 交叉轴居中对齐  4.baseline 交叉轴基线对齐  5.stretch 前提，项目不写高，占满交叉轴上所有的空间 |

练习

|  |
| --- |
| 06\_ex,ul>li 配合弹性布局完成 |

5.项目的属性

①order

|  |
| --- |
| 定义项目的排列顺序  取值，无单位整数，值越小，越靠近主轴起点 |

②flex-grow

|  |
| --- |
| 定义项目的放大比例  如果容器有足够大的剩余空间，项目将按比例方大  取值：无单位的数字 默认值0，不放大 |

③flex-shrink

|  |
| --- |
| 定义项目缩小的比例，容器空间不够时，项目该如何缩小  取值，无单位数字 默认值1. 取值越大，缩小比例越大。0，不缩 |

④align-self

|  |
| --- |
| 设置此项目在交叉轴上的对齐方式，不影响其它项目  取值：flex-start  flex-end  center  baseline  stretch  auto 使用容器定义的align-items的值 |

### 三.CSS hack

|  |
| --- |
| div{  background:-webkit-linear-gradient(....);  background:-o-linear-gradient(....);  background:-ms-linear-gradient(....);  background:-moz-linear-gradient(....);  }  由于不同的浏览器对css的解析认知不同，会导致同一份css在不同浏览器中生成页面效果不同  面对这种情况，开发人员需要针对不同浏览器写不同的css  这个行为，就叫做css hack  tmooc有css hack相关拓展视频 |

### 四.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，以及形状  2D转换。只在x轴和y轴发生的转换效果  3D转换。增加了z轴的转换效果。3D是模拟出来的 |

2.转换的属性

|  |
| --- |
| transform: 取值 1.none 默认值，无任何转换效果  2.transform-function 转换函数  表示1个或者多个转换函数  如果是多个转换函数，每个函数之间用空格分开  transform:translate(400px) rotate(90deg);  学习转换，就是学习转换函数 |

作业1.使用弹性布局完成学子商场首页1F



作业2：复杂选择器，弹性布局的所有属性，单词和效果

作业3：确保webstrom10 确保可以正确使用(注册码的问题)

## day14 转换/过渡/动画

### 四.转换

1.什么是转换

2.转化属性

|  |
| --- |
| transform：转换函数1 转换函数2 ........  学习转换，就是学习在转换函数 |

3.转换原点

|  |
| --- |
| transform-origin:  取值： 1.以px为单位的数字  2.%  3.关键字 x(left/center/right) y(top/center/bottom)  取值个数： 2个值，原点在x轴和y轴上的位置  3个值，原点在x轴，y轴和z轴上的位置  默认值 center center |

4.2D转换

①位移（改变元素的位置）

|  |
| --- |
| transform:translate(参数)  参数：1.translate(x) 等同于translateX(x)  指定元素在x轴上的位移距离  + 往右 - 往左  2.translate(x,y)指定元素在x轴和y轴上的位移距离  x: + 往右 - 往左  y:+ 往下 - 往上  3.translateX(x)  4.translateY(y)  取值：1.px为单位的数字  2.% |

### 面试题

|  |
| --- |
| 一个div200px\*200px。使用css操作，如何在浏览器窗口不断变化的情况下，让该div永远在窗口中心显示 |

②缩放(改变元素的尺寸)

|  |
| --- |
| transform:scale(value)  取值 一个值 value>=1 x轴和y轴都放大的倍数  0<value<1 x轴和y轴都缩小  -1<value<0 x轴和y轴都缩小，并反转180度  value<=-1 x轴和y轴放大，并反转180度  两个值 scale(x，y) 分别设置x轴和y轴的放大比例  scaleX(x) 单独设置x轴  scaleY(y) 单独设置y轴 |

③旋转(改变元素的角度)

|  |
| --- |
| transform:rotate(190deg)  取值： + 顺时针旋转 - 逆时针  练习：01\_ex  #d1 先旋转45deg,再位移300px  #d2 先位移300px，再旋转45deg  注意：旋转原点会影响效果  旋转是连同坐标轴一起旋转的，会影响旋转之后的位移方向 |

练习

|  |
| --- |
| 翻滚吧，亮亮  div200px\*200px,倒角50%，背景为亮亮照片  鼠标悬停时，该元素向右位移1000px，旋转1080deg |

④倾斜

|  |
| --- |
| 1.skew(x)等同于skewX(x)  让元素向着x轴发生倾斜，实际上改变的是y轴的角度  x:+ 逆时针倾斜  - 顺时针倾斜    2.skewY(y)  让元素向着y轴发生倾斜，实际上改变的是x轴的角度  y:+ x轴顺时针倾斜  - x轴逆时针倾斜 |

练习

|  |
| --- |
| 02\_ex 创建div 200\*200 背景，鼠标悬停时  该元素向右偏移200px,向下偏移200px,旋转135deg，用x轴倾斜45deng，放大1.5倍 |

5.3D转换---3d都是模拟

①透视距离

|  |
| --- |
| 模拟人的眼睛到3D转换元素之间的距离，透视距离不同，看到的效果不同  设置透视距离  perspective:距离，此属性要加载到3d转换元素的父元素上 |

②3D旋转

|  |
| --- |
| 属性：transform  取值：1.rotateX(xdeg)  以x轴为中心轴旋转，烤羊腿 老式爆米花机  2.rotateY(ydeg)  以y轴为中心轴旋转，旋转门，旋转木马，钢管舞  3.rotateZ(zdeg)  以z轴为中心轴旋转,风扇，风车，摩天轮  4.rotate3D(x,y,z,ndeg)  x,y,z取值0，代表这条轴不参与旋转  取值>0, 表示该轴参与旋转 |

### 五.过渡（重点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*）

1.什么是过渡

|  |
| --- |
| 让css属性的值，在一段时间内平缓变化的效果  哪些css属性，可以过渡  怎么设置一段时间是多少  变化速度 |

2.语法

①指定过渡的属性

|  |
| --- |
| transition-property  取值：直接写css的属性，多个属性值之间用空格分开  all 所有支持过渡的属性，都参与此次过渡效果  哪些属性支持过渡  1.颜色属性  2.大多数取值为具体的数字的属性  3.阴影  4.转换transform  5.visibility |

②指定过渡时长

|  |
| --- |
| transition-duration  指定用多长时间完成此次过渡操作  取值 ：s/ms为单位的数字 1s=1000ms |

③过渡时间曲线函数

|  |
| --- |
| transition-timing-function:  取值  1.ease 默认值，慢速开始，中间变快，慢速结束  2.linear 匀速  3.ease-in 慢慢开始，快速结束  4.ease-out 快速开始，慢慢结束  5.ease-in-out 慢速开始，慢速结束，中间先加速后减速 |

④过渡的延迟时间

|  |
| --- |
| transition-delay:  让过渡效果，延迟多少时间执行  取值 s/ms为单位的数字 |

⑤过渡代码的编写位置

|  |
| --- |
| 原始选择器中，过渡效果有去有回  :hover中，过渡效果有去无回 |

⑥简写方式

|  |
| --- |
| transition：property duration timing-function delay;  最简方式  transition:duration; |

练习：

|  |
| --- |
| 1.翻滚吧，亮亮  2.亮亮转圈圈  03\_ex 一个div,向右移动800px.并且旋转1080deg  14:58~15:13休息 |

### 六.动画

1.什么是动画

|  |
| --- |
| 元素从一种样式逐渐变为另一种样式  其实就是多个过渡效果放到一起 |

2.使用关键帧来控制动画的每一个状态

|  |
| --- |
| 关键帧  1.动画执行的时间点  2.在这个时间点上的样式 |

3.使用动画

①使用关键帧定义动画

|  |
| --- |
| @keyframes 动画名称{  /\*关键帧\*/  0%{样式}  ...  50%{样式}  ..  100%{样式}  }  ex:  @keyframes liangjump{  0%{transform:translate(0px,0px);}  20%{transform:translate(0px,200px);}  40%{transform:translate(0px,0px);}  60%{transform:translate(0px,150px);}  80%{transform:translate(0px,0px);}  100%{transform:translate(0px,110px);}  } |

②调用动画

|  |
| --- |
| 1.调用动画名称  animation-name:动画名称  2.设置动画执行时间  animation-duration: s/ms  3.设置动画的时间曲线函数  animation-timing-function  4.设置动画延迟播放  animation-delay:2s; |

练习

|  |
| --- |
| 05\_ex  div#d1  创建一个动画change  0% 背景色为蓝色，矩形  25% 背景色为黄色，圆形  50% 背景色为红色，矩形  75% 背景色为orange,圆形  100% pink 矩形 |

③动画的其它属性

|  |
| --- |
| 设置动画的播放次数  animation-iteration-count:  取值 1.具体次数，无单位数字  2. infinite 无限 |
| 设置动画的播放顺序  animation-direction:  取值：1.normal 默认 0%~100%  2.reverse 100%~0%  3.alternate 轮流播放，第一遍正向，第二遍逆向，... |
| 动画的简写方式  animation：name duration timing-function delay count direction  最简方式 animation:name duration; |
| 设置动画，播放前后的状态  animation-fill-mode  取值：1.backwards，动画播放之前的延迟时间内，显示第一帧  2.forwards，动画播放完成，保存在最后一帧  3.both,同时使用backwards和forwards  4.none 默认值，不填充 |

设置动画的播放状态

|  |
| --- |
| animation-play-state  取值：1.paused 暂停  2.running 播放 |

5.动画的兼容性

|  |
| --- |
| 如果要兼容低版本浏览器，需要在动画声明的时候加前缀  @keyframes 动画名称{}  @-webkit-keyframes 动画名称{}  @-o-keyframes 动画名称{}  @-moz-keyframes 动画名称{}  @-ms-keyframes 动画名称{} |

6.开发中使用的动画样式

|  |
| --- |
| 使用animate.css中预定义好动画 |

### 七.CSS优化

1.CSS优化的目的

|  |
| --- |
| 1.减少服务器压力  2.提升用户体验 |

2.CSS优化原则

|  |
| --- |
| 尽量减少http的请求个数 .css .js .jpg  在页面顶部，引入css文件  将css和js文件放到外部独立的文件中 |

3.CSS代码优化

|  |
| --- |
| 1.合并样式（能简写，就不分开写.能写群组，就不单写）  2.缩小样式文件的大小（能重用的样式，尽量重用）  3.减少样式重写  4.避免出现空的href和空的src  5.选择更优的样式属性  6.代码压缩 |

## day15 响应式/媒体查询/boot

### 准备webstrom10,见视频

### 一.响应式布局(css3)

1.什么是响应式网页

|  |
| --- |
| Responsive web page 响应式/自适应网页  可以根据浏览网页的设备不同(pc,pad,phone)  而自动改变布局，文字，图片效果。  不会影响用户体验 |

2.响应式网页必须保证几件事情

|  |
| --- |
| 1.布局，不能固定元素宽度.必须是流式布局(默认文档流+浮动)  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(css3的技术) |

3.如何测试响应式网页

|  |
| --- |
| 1.使用真实设备测试  好处：真是可靠  缺点：成本高，测试任务巨大  2.使用第三方测试软件  好处：不需要太多真实设备，测试方便  坏处：测试效果有限，有待进一步验证  3.使用chrome等浏览器自带的模拟软件  好处：简单方便  坏处：测试效果十分有限，需要进一步验证 |

4.编写响应式布局(重点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

①手机适配,设置视口

|  |
| --- |
| 只有移动端设备，需要设置视口  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">  viewport 视口  width=device-width 视口宽度等于设备宽度  initial-scale=1.0 设置视口宽度能否缩放 1.0不能缩放  maximum-scale=1.0 允许视口缩放的最大倍率 1.0不缩放  user-scalable=0 是否允许用户手动缩放 0不能  最简洁的写法  <meta name="viewport" content="width=device-width, initial-scale=1"> |

②所有内容/文字/图片都尽量使用相对尺寸，不要使用绝对数字

③流式布局+弹性布局+媒体查询，组成响应式布局

④媒体查询 CSS3 Media Query

|  |
| --- |
| 媒体查询是写响应式必须具备的技术  Media：媒体，不同的用于浏览网页的设备  设备：screen（pc/pad/phone）  TV  print  Media Query:可以自动的根据当前浏览设备不同(尺寸，方向，解析度。。。)，有选择性的，执行一部分代码，忽略其他代码 |

### 二.BootStrap

|  |
| --- |
| boot封装了很多常用样式，以类名的形式让我们调用  但是，真正项目，很多样式还是需要我们手动开发和媒体查询  学会查boot文档的能力  boot内容分为5部分  1.如何使用bootstrap  2.全局样式  3.组件  4.js插件  5.sass  6.boot项目 |

### 三.如何使用bootstrap

|  |
| --- |
| 导入boot文件  *<!--设置视口-->* <**meta name="viewport" content="width=device-width,initial-scale=1"**/> *<!-- 导入css-->* <**link rel="stylesheet" href="../css/bootstrap.css"**/> *<!-- 导入3个js,有顺序要求-->* <**script src="../js/jquery.min.js"**></**script**> <**script src="../js/popper.min.js"**></**script**> <**script src="../js/bootstrap.min.js"**></**script**>  制作boot模板  file--->new--->edit Template file |

### 四.全局样式

|  |
| --- |
| .container 定宽容器，在每种尺寸下，都定义了最大宽度  .container-fluid 变宽容器，每种尺寸都是100%  boot支持4种屏幕尺寸 xl--lg--md--sm  boot中rem，默认一个rem是16px |

1.按钮相关的class

|  |
| --- |
| 基本类 .btn 所有的按钮，都需要添加这个类 btn：按钮  按钮颜色  btn-success 绿色/成功的  btn-danger 红色/危险的  btn-warning 黄色/警告的  btn-info 青色/信息/情报  btn-primary 蓝色/原色/主要的/基础的  btn-secondary 灰色/第二的/中等的  btn-dark 深色/黑暗的/模糊的  btn-light 浅色/光/轻的  btn-link 链接 /关系/连结  不同边框的按钮  btn-outline-danger/success/warning......  不同大小的按钮  btn-lg  btn-sm  btn-block |

2.图片相关

|  |
| --- |
| rounded 圆角 0.25rem  rounded-circle 圆形  img-thumbnail 缩略图，有一圈白色的边(padding)  img-fluid 响应式图片，图片可以缩放，但是不会超过原始大小 |

3.文本相关的class

|  |
| --- |
| .h1~.h6 字号大小，同h1标签到h6标签  text-danger/primary.......文本颜色  text-uppercase/lowercase/capitalize首字母大写  font-weight-bold/light/normal 字体 粗细正常  text-left/right/center/justify 文本对齐  text-\*-left/right/center 响应式 \*:sm/md/lg/xl |

4.列表相关class

|  |
| --- |
| list-unstyled 取点，左内边距清0  ul.list-group>li.list-group-item 创建列表组和列表项  激活项 .list-group-item.active 两个类都引用  禁用项 .list-group-item.disabled 两个类都引用  列表项颜色 list-group-item-danger/warning...... |

5.表格相关的样式

|  |
| --- |
| .table 基本类，对table本身和table的后代布局  .table-bordered 一个有边框的表格，后代td、th也添加边框  .table-danger/warning....表格颜色  .table-striped 隔行变色，奇数行变色  .table-hover 带鼠标悬停效果  .table-responsive-xl/lg/md/sm 响应式表格，写在table的父元素上 |

6.辅助类

①边框

|  |
| --- |
| 基本类  border/border-0/ 定义边框/取消边框  border-left/top/bottom/right 单边定义  border-left-0 单边的取消  border-danger/warning/info.... 边框颜色 |

②浮动

|  |
| --- |
| float-left/right/none  float-sm/md/lg/xl-left响应式的浮动  clearfix 解决高度坍塌，写在父元素 |

③显示

|  |
| --- |
| visible 显示  invisible 隐藏 |

④背景颜色

|  |
| --- |
| bg-danger/warning/success/primary.... |

⑤圆角

|  |
| --- |
| rounded/rounded-0 设置圆角/圆角清0  一个方向的圆角设置  rounded-top/right/bottom/left  rounded-circle 50% |

⑥内外边距

|  |
| --- |
| m-n mt/mr/mb/ml-n mx-n左右外边距 my-n上下  p-n pt/pr/pb/pl-n px-n 左右内边距 py-n上下  n:0---取消,  1--0.25rem,  2---0.5rem  3--1rem，  4---1.5rem,  5---3rem  上面所有的都有响应式  m-\*-n  m:m/mt/mb/ml/mr/mx/my \*:xl/lg/md/sm n:0~5  p-\*-n  p:p/pt/pb/pl/pr/px/py \*:xl/lg/md/sm n:0~5 |

### 作业

1.使用媒体查询完成布局



2.完成下图



## day16 栅格/弹性/组件

### 四.全局css

6.辅助类

⑦尺寸

|  |
| --- |
| w-25/50/75/100  h-25/50/75/100  如果需要其他尺寸，需要手写css  mw-100 mh-100 |

### 7.栅格布局 （重点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*）

02\_grid

①web页面中的布局方式，一般有三种

|  |  |  |
| --- | --- | --- |
| table布局 | div+css | Boot的栅格布局 |
| 简单，易控制 | 语义正确 渲染效率高 | 简单，易控制。语义正确，渲染效率高，支持响应式 |
| 语言错误 渲染效率低 | 控制起来很麻烦 | 不适用于复杂页面 |

②栅格的使用

|  |
| --- |
| 1.栅格必须在container或者container-fluid中  2.每一行都是一个row(弹性，主轴是x轴，可以换行，  左右-15px外边距)  3.每一个col自带左右15内边距 |

③响应式栅格布局

|  |
| --- |
| col-\*-n  \*:lg/md/sm/xl  n:1~12  在不同的屏幕下，执行不同的列数  <**div class="row"**>  <**div class="col-lg-2 col-md-3 col-sm-12"**>111</**div**>  <**div class="col-lg-8 col-md-9 col-sm-12"**>222</**div**>  <**div class="col-lg-2 d-lg-block d-md-none col-sm-12"**>333</**div**> </**div**> |

④不同屏幕尺寸，向上兼容

|  |
| --- |
| 只写小屏，在小屏，中屏，大屏，超大屏，都会应用  只写中屏，在中屏，大屏，超大屏，都会应用  只写大屏，大屏，超大屏，都会应用  只写超大屏，只在超大屏应用 |

⑤使用.col这个类

|  |
| --- |
| 使用.col,不添加数字，自动平局分配占地宽度  每一个col平局分配空间，col的个数可以超过12个  使用场合：  当一行内，元素需要横向现实，元素个数不能被12整除  每个元素占地空间一样 |

⑥列偏移

|  |
| --- |
| offset-\*-n 源码就是在操作左外边距 margin-left  \*:xl/lg/md/sm  n:0~11 |

⑦栅格嵌套

|  |
| --- |
| 在一个.col中再写一个row |

### 五.弹性布局

|  |
| --- |
| d-block/inline/inline-block/none display的取值  d-\*-block.....响应式 \*：xl/lg/md/sm  想使用弹性布局，需要在父元素上添加d-flex/d-inline-flex  创建弹性容器 |

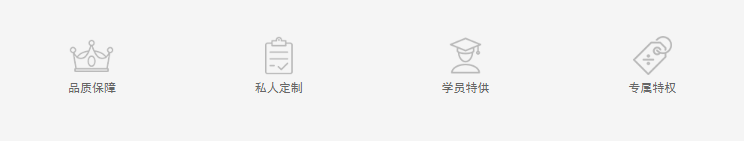
#### 1.主轴方向

|  |
| --- |
| flex-row/column/row-reverse/column-reverse  响应式设置  flex-\*-row/column/row-reverse/column-reverse  \*:xl/lg/md/sm |

#### 2.主轴对齐方式

|  |
| --- |
| justify-content-start/center/end/between/around  响应式设置  justify-content-\*-start/center/end/between/around  \*：xl/lg/md/sm |

练习 ul>li boot完成



### 六.表单相关的class

#### 1.表单元素的排列放向

|  |
| --- |
| form-group 堆叠表单，内部控件垂直显示  form-inline 内联表单，内部控件横向显示(弹性，x轴) |

#### 2.表单控件元素样式

|  |
| --- |
| form-control 对于text,password 基本类  块级，w100 字体 背景颜色 边框 圆角 过渡  col-form-label 设置文本框上下内边距  col-form-label-lg 上下内边距比较大  col-form-label-sm 上下内边距比较小  checkbox的基本类  form-check-input,绝对定位  要配合父元素的 form-check使用  form-text 块级，有上外边距 15:06~15:21休息 |

练习：



|  |
| --- |
| form-inline可以代替row作为栅格行  form-inline没有左右-15px的外边距 |

### 七.组件

|  |
| --- |
| 网页中有一些复杂的特殊效果，之前必须使用js,dom,jq  boot为我们封装好了很多常用的组件，我们可以直接使用。  方便，快捷。但是如果需要个性化的处理，比较繁琐。  boot中，大量使用元素的自定义属性，来调用方法 |

#### 1.下拉菜单

|  |
| --- |
| 基本结构  <**div class="dropdown"**><**button class=" dropdown-toggle"**  **data-toggle="dropdown"**>下拉菜单</**button**>  <**ul class="dropdown-menu"**> </**ul**> </**div**>  div.dropdown 父级，相对定位  button.dropdown-toggle 画向下小三角  ul.dropdown-menu display:none  **data-toggle="dropdown"** 自定义属性的作用  1.当button被点击的时候激活事件  2.自定义属性的值，决定激活哪一个事件 |

#### 2.按钮组

|  |
| --- |
| <div class="btn-group">  <button></button>  .....  </div>  外层div的class有以下几个值  btn-group 横向按钮组  btn-group-vertical 纵向按钮组  可以添加 btn-group-lg/sm 调整按钮组的大小 |

#### 3.信息提示框

|  |
| --- |
| <div class="alert alert-danger alert-dismissible">  <span class="close" data-dismiss="alert">  &times;</span>  提示的信息  </div>  alert 基本类  alert-danger/warning/.....提示框颜色  alert-dismissible 为了修饰内部span.close  .close 右浮动  data-dismiss="alert" 自定义属性，事件使用alert的方式执行 |

#### 4.导航

①水平导航

|  |
| --- |
| 基本结构 ul.nav>li.nav-item>a.nav-link  <ul class="nav nav-justified">  <li class="nav-item">  <a class="nav-link"></a>  </li>  </ul>  ul.nav 定义了弹性布局，主轴为x  那么之前学习的所有的弹性布局的样式都可以用  .nav-justified>.nav-item 要同时设置，让元素等宽显示  a.nav-link 块级，设置了hover，focus等样式 |

②选项卡导航

|  |
| --- |
| 基本结构  上层是选项卡  ul.nav.nav-tab>li.nav-item>a.nav-link  下层内容  div.tab-content>div.pane  ul.nav 变成选项卡添加类 nav-tab  nav-tab设置自己的孩子 nav-item nav-link  a标签添加自定义属性 data-toggle="tab"  以tab的方式激活事件  绑定目标元素 a的href属性指向下面内容的id值  div.tab-content>div.tab-pane配合使用  让div.tab-pane都是是display:none  然后div.tab-pane要写id，让a标签绑定 |

|  |
| --- |
| 一个事件激活，关注几件事情  1.激活这个事件的元素是啥 <a>  2.这个事件以什么方式激活 tab  3.事件激活后，改变的目标元素是谁 |

### 作业1.

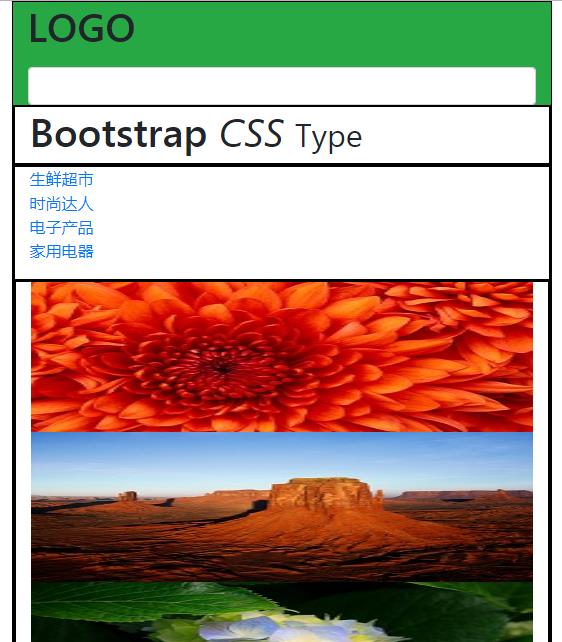
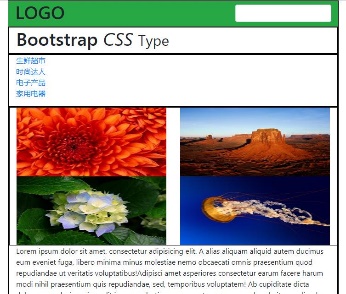
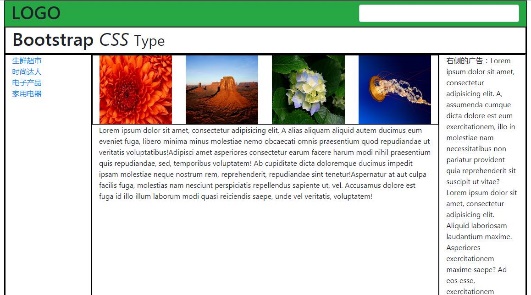
提高题，完成学子首页1L布局，使用栅格实现6个灰色框

允许手写宽高样式

坑：div灰块之间的空白怎么解决

### 作业2.

使用栅格完成



### 作业3：

组件所有demo都敲一遍

## day17 导航栏/

### 七.组件

#### 4.导航

③胶囊导航

|  |
| --- |
| <**ul class="nav nav-pills"**>  <**li class="nav-item"**>  <**a data-toggle="pill" href="#tab1" class="nav-link"**>口红</**a**>  </**li**>  </**ul**> <**div class="tab-content"**>  <**div id="tab1" class="tab-pane"**>  YSL,Dior,chanel,故宫  </**div**> </**div**>  ul的nav-tabs改成nav-pills  a的data-toggle="tab" 改成data-toggle="pill"  其他与选项卡导航一样 |

#### 5.导航栏

|  |
| --- |
| <**h1**>导航栏</**h1**> <**div class="navbar navbar-expand-lg"**>  <**ul class="navbar-nav"**>  <**li class="nav-item"**><**a class="nav-link" href=""**>长的丑</**a**></**li**>  </**ul**> </**div**>  最外层div.navbar.navbar-expand-xl/lg/md/sm  .navbar-expand-xl/lg/md/sm作用，作为子元素ul.navbar-nav的选择器条件，让ul.navbar-nav的主轴方向从y轴变成x轴(ul.navbar-nav原本主轴方向是y轴)    响应式的导航栏，以.navbar-expand-lg为例  超大屏，大屏，li横向显示  中屏，小屏，li纵向显示  div.navbar-expand-\* 配合 ul.navbar-nav可以让导航栏在不同屏幕下，横向或者纵向显示 |

#### 6.折叠效果

|  |
| --- |
| <**button data-target="#demo" data-toggle="collapse" class="btn btn-info"**>折叠</**button**> <**div id="demo" class="collapse"**>Lorem  折叠，需要折叠内容添加.collapse--->display:none  折叠事件 data-toggle="collapse"  折叠目标  如果使用a,就可以使用href=""  如果使用button,则使用data-target="" |

#### 7.卡片

|  |
| --- |
| <**div class="card"**>  <**div class="card-header"**></**div**>  <**div class="card-body"**></**div**>  <**div class="card-footer"**></**div**> </**div**> |

#### 8.卡片+折叠=手风琴

|  |
| --- |
| <div id="parent">  <div class="card">  <div class="card-header">  <a data-toggle="collapse"  href="#one" class="card-link">折叠</a>  </div>  <div id="one" data-parent="#parent"  class="collapse">  <div class="card-body">内容</div>  </div>  </div>  </div>  总结：  1. .collapse不能和.card-body在同一个标签上  不然会发生折叠/显示的卡顿  解决方案 div.collapse>div.card-body  2.多个折叠部分，可以同时打开，而不是开启一个，其他的关闭  解决方案，在所有的card外部，添加一个父元素div#parent  在所有的div.collapse上添加自定义属性  data-parent="#parent"  这样就可以保证，在外层div中，只存在一个折叠区域被打开 |

练习1：



练习2：响应式导航栏，中屏，小屏 垂直排列



#### 9.折叠导航栏

|  |
| --- |
| 1.div.navbar-expand-sm和.anvabar-toggler的关系和作用    2.在外层div中.navbar-dark/light的作用  这个类对本div没有任何修饰  修饰的是后代元素的字体颜色  3..navbar-collapse的作用  .navbar-expand-sm .navbar-collapse配合  让ul的父级div在sm/md/lg/xl下显示  在xs下，ul的父级div执行.collapse的样式，隐藏  4.小总结 .navbar-expand-sm决定了几件事？  1.按钮在sm以上隐藏，在sm以下显示  2.ul在sm以上横向显示，在sm以下垂直显示  3.折叠的div在sm以上显示，在sm以下隐藏 |

#### 10.媒体对象

|  |
| --- |
| boot提供了很多类，来处理媒体相关的对象  图片 影音  <div class="media">  <img>  <div class="media-body"><div>  </div> |

#### 11.焦点轮播图

|  |
| --- |
| 1.轮播图结构  div.carousel data-ride="carousel" 相对定位  >div.carousel-inner w100 溢出隐藏  >div.carousel-item display:none  >img  2.左右箭头  a.carousel-control-prev/next 宽度，背景，位置  >span. carousel-control-prev/next-icon 左右箭头  需要重写.carousel-control-prev/next中的宽高，位置  .**carousel-control-prev**, .**carousel-control-next**{  **width**:4%;**height**: 20%;  **background**:**#aaa**;  **border-radius**: 0.25**rem**;  **top**:40%; }  a标签要添加自定义属性 data-slide="prev/next"  事件的目标是最外层div#demo href="#demo"  3.轮播指示器  ul.carousel-indicators 绝对定位，横向弹性  >li 宽高，背景色，外边距  我们需要重写li的样式  .**carousel-indicators li**{  **width**: 0.75**rem**;**height**:0.75**rem**;  **background-color**:**#fff**;  **border-radius**: 50%;  **margin**:0 4**px**; }  如果让li有被选中的状态，需要第一个li添加.active  我们需要重写.active .**carousel-indicators** .**active**{  **background-color**:**#0aa1ed**; }  点击li激活事件  data-slide-to="图片下标" 下标从0开始  事件改变的目标  data-target="#demo" 整个轮播最外层div的id |

#### 12.模态框

|  |
| --- |
| 覆盖在父窗体上的一个子窗体，用于与用户做一些互动  div.modal>div.modal-dialog>div.modal-content  >div.modal-header+div.modal-body+div.modal-footer  关闭的事件  button data-dismiss="modal" |

### 八.其它组件

#### 1.徽章

|  |
| --- |
| 跟小按钮十分类似  基本类 badge  颜色 badge-danger/warning/info...........  徽章样式 badge-pill |

#### 2.巨幕

|  |
| --- |
| 巨大的内边距，有背景颜色，有圆角  div.jumbotron |

#### 3.分页条

|  |
| --- |
| ul.pagination>li.page-item>a.page-link  li的修饰类 .active激活  .disabled禁用 |

### 作业

1.重新完成手风琴

2.重新完成焦点轮播图

3.折叠导航栏



作业：

1.完成学子一楼响应式布局，使用栅格

注意外边距，不同的对齐

## day18 面包屑/CSS/函数

### 八.其他组件

#### 4.面包屑导航

|  |
| --- |
| 常用于有路径关系的导航  ul.breadcrumb>li.breadcrumb-item  每个li中间连接的符号，需要重写  .**breadcrumb-item** + .**breadcrumb-item**::**before**{  **content**: **'>'**; } |

#### 5.进度条

①普通进度条

|  |
| --- |
| div.progress>div.progress-bar.w-50.bg-danger  父div是进度条的槽  子div是进度条，子div的宽度，是进度条的进度  子div背景，是进度条的颜色 |

②带条纹的进度条

|  |
| --- |
| 在子div添加.**progress-bar-striped** |

③带动画的条纹进度条

|  |
| --- |
| <**div class="progress-bar w-100 bg-secondary progress-bar-striped progress-bar-animated"**></**div**> |

### 九、BOOT定制:scss

|  |
| --- |
| **bootStrap最重要：响应式+栅格布局+scss** |

### 1.CSS有很多缺点

|  |
| --- |
| 语法不够强大，没有变量，没有合理的样式复用机制  导致难以维护  我要使用动态样式语言，赋予css新的特性  提高样式语言的可维护性  常用的动态样式语言  1.SCSS/SASS （scss兼容sass,scss更接近css语法）  2.stylus  3.less |

### 十.SCSS详解

#### 1.什么是scss

|  |
| --- |
| scss是一款强化css的辅助工具  它和css语法很像。  它在css的语法基础上，添加了变量，嵌套，混合，导入，函数等高级功能。  这些拓展命令让scss更加强大和优雅  浏览器不能直接解析scss文件，需要在想项目中把scss转义成css文件，让浏览器解析  scss可以让css开发更高效 |

#### 2.scss的安装使用

|  |
| --- |
| scss在服务器端使用  nodejs v8.11以上，才可以使用scss  1.在线安装scss  在cmd中，输入命令npm install -g node-sass  2.无网络安装  找到sass的4个文件，  找到nodejs的安装路径,把sass4个文件放入node文件夹  在cmd中使用node-sass -v 验证版本 |

#### 3.SCSS文件转换成css文件

①单文件的转换

|  |
| --- |
| 1.创建scss/01.scss文件  2.在项目路径下，打开黑窗口  3.输入下面命令 node-sass scss文件路径 css文件路径 |

②多文件转换 文件夹--->文件夹

|  |
| --- |
| node-sass scss文件夹 -o css文件夹 |

③单文件监听转换

|  |
| --- |
| node-sass -w scss/01.scss css/01.css |

④多文件监听，监听一个文件夹中所有文件

|  |
| --- |
| node-sass -w scss文件夹 -o css文件夹 |

### 十一.SCSS基本语法

#### 1.变量

|  |
| --- |
| 使用$声明变量，变量名可以包含-，\_  命名规则基本与css选择器相同，尽量做到见名知意  变量声明在{}外，整个scss文件都可以使用。  变量声明在{}内，只有当前{}内可以使用  ！default规则，如果此变量在之前已经声明赋值了，那么使用之前的值。如果之前没有声明赋值，使用现在值 |

#### 2.嵌套

|  |
| --- |
| 1.选择器的嵌套  **#content**{  **width**:**$my\_width**;  **div**.**top**{  **margin**:**$my\_width**;  **h1**{**font-size**:46**px**;}  **p**{**padding**:12**px**;}  }  **div**.**bottom**{  **border**: **$my\_border**;} }  自动转换成css的后代选择器  **#content** { **width**: 521**px**; } **#content div**.**top** { **margin**: 521**px**; } **#content div**.**top h1** {**font-size**: 46**px**; } **#content div**.**top p** {**padding**: 12**px**; } **#content div**.**bottom** { **border**: 1**px solid #00f**; } |
| 2.伪类的嵌套  需要在伪类选择器之前添加&，如果不添加，会生成一个空格导致伪类失效  a{width:21px;  &:hover{width:20px}  } |

练习 使用scss完成

|  |
| --- |
| $myblue:#0aa1ed;  $fontcolor:#fff;  $myheight:2.5rem;  $fontsize:0.8rem; |



|  |
| --- |
| 3.属性的嵌套  **div**{  **border**:{**style**:**solid**;**width**:1**px**;**color**:**#fff**;}; } |
| 4.群组选择器的嵌套  **nav**,**div**,**header**,**footer**{  **a**{**width**:100**px**;} } |

练习

|  |
| --- |
| 一个div#main.200px\*200px;红色背景  鼠标悬停，过渡到黄色背景，并且变为圆形  使用scss编写样式 |

#### 3.导入

|  |
| --- |
| 在scss的语法中，如果一个scss文件以下划线开头，  那么这个scss文件就是一个局部scss文件  scss文件转换成css文件的时候，不会把局部scss文件进行转换  只转换，不以下划线开头的scss文件(全局scss文件)  导入的语法 @import "name";  真正导入的文件名称 \_name.scss 掐头去尾  导入时，不写\_,不写.scss后缀  局部文件被导入后，局部文件中的样式，会在全局文件转换的css中生成。  同时，局部文件中声明的变量，可以在全局文件中使用 |

#### 4.混合器

|  |
| --- |
| 把多个选择器都会使用的样式，封装进一个混合器  需要使用的这些样式选择器，可以调用这个混合器  实现代码的重用。  关键字 声明混合器 @mixin 混合器名称(参数1，参数2){样式}  调用混合器 @include 混合器名称(参数1，参数2)  混合器使用场合---css hack |

#### 5.继承

|  |
| --- |
| 一个选择器，可以使用另外一个选择器的所有样式  .my1{  width:100px;height:100px;  }  .my2{  background:#f00;  @extend .my1;  }  转换之后，继承的表现方式，是群组选择器  .my1, .my2 {  width: 100px;  height: 100px; }  .my2 {  background: #f00; } |

练习

|  |
| --- |
| 在选择器.my-border定义1px solid transparent  设置宽度400px 左外边距20px  在选择器.my-bottom-border中，继承.my-border  并设置底边框为红色，当鼠标悬停时，底边框为蓝色 |

#### 6.运算

|  |
| --- |
| 加减乘除余  如果必要，会在不同单位间转换(前提是scss能转) |

①加法

|  |
| --- |
| + 除了做加法，还做字符串拼接  字符串拼接的时候  如果用有引号的字符串拼接无引号的，结果带引号的  如果用无引号的字符串拼接有引号的，结果不带引号 |

②减法

|  |
| --- |
| 由于变量声明的时候可以使用-  系统分不清楚 - 是不是属于变量名称  所以我们写减法的时候，要在 - 前后添加空格  width:$my-w - $my-h; |

③除法

|  |
| --- |
| 在scss中， / 的作用是分隔符  只有下面几种情况，我们判定为除法  1.运算式的两边，有变量，或者函数返回值的时候，是除法  width: $w/2;  2.运算式被()包裹的时候，是除法  height:(500px/2);  3.运算式是其他算术运算式的一部分的时候，是除法  margin-left:5px+8px/2px; |

④字符串的插值操作

|  |
| --- |
| 使用#{}在字符串中做插值  content: "liangliang ate #{50+32} baozis"; |

⑤颜色的运算

|  |
| --- |
| #112233+#445566=#557799  #rrggb rgb(r,g,b) 都是分段计算，红+红 绿+绿 蓝+蓝  rgba的元素，需要透明度相同，才允许计算  background:rgba(11,22,33,0.6)+rgba(22,33,44,0.6); |

### 十二.函数

1.scss预定义很多函数，有些函数直接可以在css中使用

|  |
| --- |
| rgba(r,g,b,alpha)  hsl(h,s,l)  hue:色调 取值0~360 3个色段 0~120 120~240 240~360  saturation：饱和度 0.0%~100%  lightness:亮度 0.0%~100% |

2.数学函数

|  |
| --- |
| round($v) 四舍五入  ceil($v) 向上取整  floor($v) 向下取整  min($v1,$v2....)  max($v1,$v2....)  random 随机数 |

3.字符串

|  |
| --- |
| unquote($v) 去掉双引号  quote($v) 加双引号  to\_upper\_case($str) 把$str转成大写  to\_lower\_case($str) 把$str转成小写 |

4.自定义函数

|  |
| --- |
| @function get\_msg($a,$b){  @return $a\*$b+($a/$b);  }  div{  width:get\_msg(3,2)+px;  } |

练习

|  |
| --- |
| 定义一个带参数的函数get-width($n)  函数内定义两个变量，分别是60px和70px  函数的返回值为，$n乘以两个数的最大值，加上$n乘以两个数的最小值。把返回值赋给div的width属性  min() max() |

### 十三.指令

|  |
| --- |
| @if(){}  @else if(){}  @else{}  bool的小括号可以去掉 |

项目阶段

|  |
| --- |
| 学子首页，产品列表页，产品详情页  注意：整个学子不是使用响应式布局设计，也没有使用boot  有一些细节，不去实现  在项目中创建一个文件夹，pro  在pro中，把css文件夹创建好，把图片资源导入 |

### 作业

1.使用面包屑导航完成



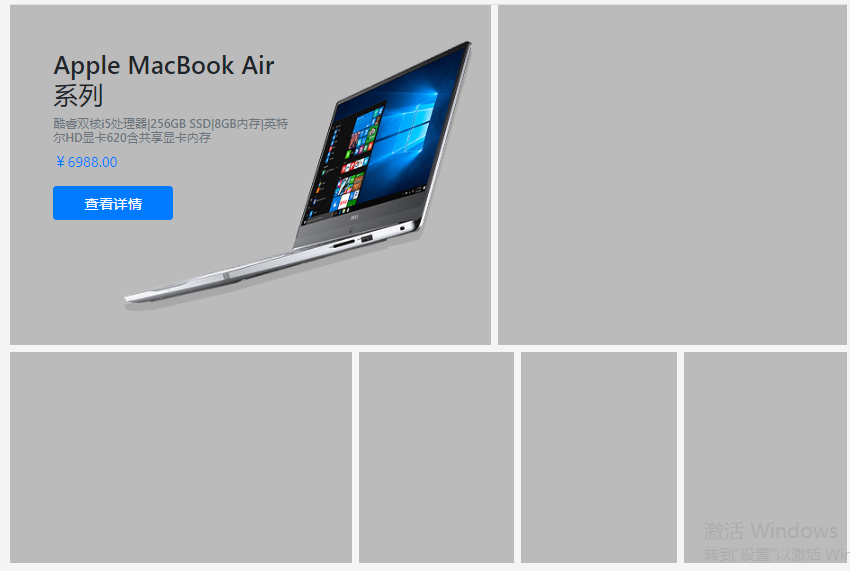
2.作业2，boot项目 头部



## day19 项目

作业

1.完成学子1L其它5个部分



2.完整这个学子首页

3.尝试完成商品列表页

## day20 项目

完成学子商城

商品详情页

# 第三阶段

day01正课:1. 正则表达式: 2. String类型中的正则函数:

张东 微信:18301092802 规矩: 不允许记笔记！

准备:

1. 书: 犀牛书——JAVASCRIPT权威指南

2. 公众号: 前端大全

3. 上届的笔记，打印出来，上课跟着看

内容:

1.JavaScript核心编程: 闭包，面向对象，ES5，ES6，异步编程

2.DOM: 为网页添加交互行为的一套程序——所有场合使用！

3.jQuery: DOM的简化版——PC（待定）

4.Vue: 自动化的框架

每个案例或小视频，至少写/看三遍

1. 代码+注释抄一遍

2. 保留注释，删掉代码，试着看着注释把代码填回来

3. 自己试着将注释+代码一起写回来——才是你自己会

肯定不能都记住，但是用到什么就复习什么，用到什么保证可以找到。——思维导图

## 1. 正则表达式: PPT

## 2. String类型中的正则函数:

在字符串类型中，支持3种正则表达式相关的操作:查找、替换、切割

### 1. 查找敏感词：4种场景:

1. 查找一个固定的敏感词出现的位置 var i=str.indexOf("敏感词")

比如: 查找一个字符串中是否包含"我草"

var i=str.indexOf("我草"[,starti]) //笔记中[]表示可选

在str中从starti位置开始找下一个"我草"出现的位置

返回: 如果找到，就返回我草的中我字的下标位置i。如果找不到，返回-1

第二个形参starti，可省略。默认从0位置(开头)开始找

问题: 关键词是固定的！只要用户稍微换一个字，就无法匹配了！

2. 用正则表达式模糊查找一个关键词的位置: var i=str.search(/正则/i)

在str中从头开始查找符合正则要求的敏感词的位置

返回值: 同indexOf的返回值完全相同！

问题: 正则表达式默认区分大小写的！

解决: 在第二个/后加i ignore 忽略(大小写)

问题: 只能返回敏感词出现的位置，不能返回敏感词的内容

3. 用正则查找敏感词的内容和位置2种:

1. 只查找第一个敏感词的内容和位置: var arr=str.match(/正则/i)

返回值: 即返回找到的敏感词的内容，又返回敏感词出现的位置。

其实返回的是一个数组: arr:[ 0:"敏感词内容", index:敏感词位置 ] arr: [ "0": "我草", "index": 9 ]

如果找不到，返回null

问题: 只能找一个敏感词！

2. 查找字符串中所有符合条件的敏感词的内容: var arr=str.match(/正则/ig) global

例如: arr: [ "我艹", "我槽" ].length = 2

强调: 只包含关键词的内容，不再包含关键词的位置。如果找不到，返回null

强调: 如果一个函数可能返回null，就应该先验证返回值不是null，再使用！

4. 查找所有关键词的内容和位置: reg.exec()

作业:

1. 微信 正则表达式: (微|w(ei)?) \s\*(信|x(in)?)

第一个字: (微|w(ei)?) 微 或 wei 其中ei可有可无

中间: \s\*

第二个字: (信|x(in)?) 信 或 xin 其中in可有可无

### 2. 替换: 2种:

1. 简单替换: 将所有关键词都替换为一致的新值

str=str.replace(/正则/ig, "新值");

将str中所有符合正则要求的关键词都替换为指定的"新值"

强调: 所有字符串都是不可变类型, 所有字符串的函数都无权直接修改原字符串，只能返回新字符串

所以，如果想获得新字符串，必须自己用变量接一下。

比如: str=str.replace(/小[\u4e00-\u9fa5]/,"\*\*")

翻译：将str中 所有 小字开头的人名 都 替换成 \*\*，并将替换后的结果，手动保存回原变量中。

2. 高级替换: 根据搜到的不同敏感词替换成不同的新值

比如: 鄙视题: 将每个单词首字母改为大写！

//找到每个单词首字母: /\b[a-z]/g

//用replace将字符串中的所有首字母，替换为其对应的大写字母:

str="you can you up";

str=str.replace(//2个实参

/\b[a-z]/g, //第一个实参是要查找字符的正则表达式

//第二个实参是一个没有名字的函数

//每找到一个敏感词，就自动调用一次

//每次调用时都自动传入本次找到的敏感词给kword，比如: 本次找到“y”，那么kword=“y”

function(kword){

//1. 必须有一个形参

//2.根据kword接到的值不同，返回不同新的替换值

return y.toUpperCase() -> Y

//return出的新值，会被自动替换到当前找到的敏感词位置。

}

)

衍生: 删除关键词: 其实就是替换为空字符串

str=str.replace(/正则/ig, "");

作业: 鄙视: 删除字符串开头和结尾的空字符

### 3. 切割：2种

将一个字符串按指定的分隔符，切割为多段子字符串

1. 分隔符是固定的: var arr=str.split("分隔符")

将str按指定的分隔符切割为多段子字符串，保存在一个字符串数组中。

比如: var email="zhangdong@tedu.cn";

将email按中间的@切割: var arr=email.split("@");

["zhangdong","tedu.cn"]

0 1

var uname=arr[0];

var domain=arr[1];

2. 分隔符不确定，但有规律:var arr=str.split(/正则/i)

比如: //再按</li>空字符<li>切割 var arr=html.split(/<\/li>\s\*<li>/i);

## 3. RegExp对象:

什么是: 保存一条正则表达式，并包含用正则表达式执行验证和查找功能 的对象

何时: 只要在js中使用正则表达式，都要先创建正则表达式对象，再使用正则。

如何使用:

### 1. 创建正则表达式对象: 2种

1. 创建一个固定的正则表达式对象:

var reg= /正则/ig

问题: //之间不允许写js语句动态生成正则表达式

2. 正则表达式需要动态生成

var reg=new RegExp("正则","ig")

说明: 因为RegExp的第一个形参要求接受一个字符串格式的正则表达式。所以，我们有无数种办法，拼接出我们想要的任何字符串。

比如: var str=kwords.join("|"); "明月|白鹭"

var reg=new RegExp(str); /明月|白鹭/

### 2. RegExp对象中包含两个函数:

1. 验证字符串是否符合格式要求 test检查

验证: var bool=reg.test(str)

用正则表达式reg，检查str字符串是否符合格式要求

返回值: true/false

坑: test()默认只要能找到符合条件的部分内容，就返回true

解决: 今后凡是验证，都必须前加^，同时后加$

比如: 定义手机号的规则表达式reg

var reg=/^1[3-9]\d{9}$/; //用规则表达式去验证手机号是否符合格式要求

var result=reg.test(phone);

//如果符合要求

if(result==true){

... ...

\*\*程序执行过程拍照

如果看不见函数的执行过程，该怎么办: 在函数内添加console.log()输出函数的执行和参数，以及返回值

比如:

str=str.replace(/\b[a-z]/,function(kw){

console.log("===内层函数被调用一次===");

console.log("=====自动传入kw=${kw}");

console.log("=====本次调用返回${kw.toUpperCase()}")

return kw.toUpperCase()   
});

再比如:

var arr=[12,3,123,2,1,23];

arr.sort(function(a,b){

console.log("\*\*\*内层函数被调用了一次\*\*\*");

console.log(`\*\*\*\*\*本次用${a}和${b}做比较`);

console.log(`\*\*\*\*\*本次调用返回${a-b}`)

return a-b

})

提示: 在浏览器控制台中，按上下键可找回曾经执行过的代码。

console.log的输出问题：

var arr=[12,3,123,2,1,23];

console.log(arr); //排序前输出数组的内容

arr.sort(function(a,b){ return a-b })

console.log(arr); //排序后再输出数组的内容

执行结果:

▶ arr

1

2

3

12

23

123

▶ arr

1

2

3

12

23

123

问题: 两次输出的结果都是排序后的。

原因: console.log在首次输出对象时，仅是输出的对象引用地址。当我们点旁边的三角时，才自动去内存中找到对象现在的样子，展开对象。但此时，程序执行完了。

—— console.log()不是拍照！无法留住对象的瞬时内容

解决: 将对象的内容转为字符串才是拍照/打印

console.log(String(arr)) [ 1, 2, 3, 12, 23, 123 ]

|| "1,2,3,12,23,123"

arr.toString()

转 字符串

基础好的同学: <http://code.tarena.com.cn/WEBCode/wfd1812/>

2. 查找关键词: 查找敏感词的内容和位置

var arr=reg.exec(str)

在字符串str中查找一个符合正则表达式reg要求的关键词

返回值: 跟str.match(reg)不加g，返回的值完全相同

arr [ 0: 关键词内容, index: 关键词的位置i ]

问题：每次只返回一个敏感词的内容和位置

解决：只要反复调用，reg.exec()会自动跳到下一个继续查找

找所有：用循环:

var reg=/小[\u4e00-\u9fa5]/g;

do{//反复do

//查找敏感词

var arr=reg.exec(str);

//如果找到敏感词，才输出

if(arr!=null){

console.log("找到敏感词!");

console.log(arr);

}else{//否则(找不到)，就退出循环

console.log("找不到了，就退出！")

break;

}

}while(true);

//true 不使用循环条件控制退出，因为我也不知道循环几次！

//而是在循环体内根据条件，用break随时可能退出循环。

## 4.Function对象:(重点)

什么是: 内存中保存一段可重用的代码段的对象，再起一个名字。

为什么: 重用！

何时: 如果一段代码段需要反复使用，就应该保存在一个函数中，反复调用函数。

如何:

创建: 3种:

1. 声明方式:

function 函数名(形参列表){ 函数体;

return 返回值;

}

问题: 声明提前: 在程序开始执行前，会自动将所有var声明的变量和function声明的函数，都提前到当前作用域的顶部，集中创建。赋值留在原地

2. 用赋值的方式:

var函数名=function (形参列表){

函数体;

return 返回值;

}

优点: 仅函数名提前，函数体留在原地。

揭示了函数的重要秘密:

1. 函数名本质就是一个普通的变量

2. 而函数定义其实是一个对象(独立的存储空间)

3. 用new创建:

var fun=new Function("形参","形参",...,"...")

调用: var 返回值=函数名(实参值列表)

调用函数，等效于执行函数中的代码段。调用几次，就重复执行几次。如果不调用，是不会执行函数的内容的

### 重载:overload

什么是: 相同函数名不同形参列表的多个函数，在调用时，可根据传入实参的不同，自动选择匹配的函数调用。

为什么: 调用不方便！

何时: 同一件事，根据传入实参的不同，自动选择不同的执行过程执行时。

如何:

问题: js语言不支持多个同名函数同时存在，所以默认和不支持重载。

解决: 借助: arguments

什么是arguments: 每个函数中自动包含的可直接使用的接收所有传入函数实参值列表的类数组对象

类数组对象是: 长得像 数组的 对象

vs 数组: 相同: 1. 下标, 2. length, 3. for遍历

不同: 根本不同 类型不同 导致arguments不能使用数组家的任何函数。

### 匿名函数: 2种

什么是: 创建函数时，不指定函数名

为什么: 2大作用:

1. 节约内存 2. 避免全局污染

何时: 2种:

1. 只要一个函数只使用一次 2. 划分临时作用域的时候，避免全局污染

如何: 2种:

1. 回调函数: 虽然我们自己定义的，但是不是我们自己调用执行的函数。

比如: arr.sort(function(a,b){return a-b})

str.replace(/\b[a-z]/g, function(kw){ return ... })

都是为了节约内存！

2. 匿名函数自调: 创建函数后，立刻调用函数

(function(){

var t=new Date();

document.write("<h1>页面加载完成，at: "+t.toLocaleString()+"</h1>")

})();

创建函数后，自己调用自己

优点: 不但自己调用后释放，且函数内的局部变量调用后，也会自动释放。

总结: 1. 今后所有js代码都要写在匿名函数自调中

2. 今后禁止使用全局变量：因为1. 浪费内存; 2. 全局污染

今日英语单词:

1. sort 排序 将多个数值，按从大到小或从小到大的顺序排列。

2. while 只要...的时候 while(条件) 只要 满足条件 的时候 就做什么事儿

3. break 中断 用于中途停止循环。

4. function 功能 程序中所有反复使用的功能都定义在function 功能名(参数){ —— 像极了数学里的函数}

5. declare 声明 其实就是"创建"

var a 创建一个变量a

function fun() 创建一个fun函数对象

6. return 返回 用于从函数内，向函数外返回一个结果值

7. hoist 吊车、起重机 将物体从低处，提高到高处

8. argument 参数 parameter 也叫参数

调用函数时()中定义的和传入的，都成为参数变量

只不过，定义时的变量成为 形参

调用时的变量称为 实参

### 1. 作用域(scope)和作用域链day04

什么是作用域: scope 范围 chain 锁链

用法: 一个变量的可用范围

本质: 保存多个变量的内存空间

为什么: 为了保护不同范围的变量间不互相干扰

包括: 2种:

1. 全局作用域: window对象

凡是不在函数内创建的变量和函数都自动加载全局作用域(window)中保存

2. 函数作用域: 也叫函数临时作用域

函数作用域仅在调用函数时临时创建

所有函数内定义的变量和内层函数，都自动保存在临时的函数作用域中

一旦函数调用后，临时的函数作用域及其内部的局部变量一起释放！

vs Java: 有三级作用域: 全局 函数作用域 局部作用域:

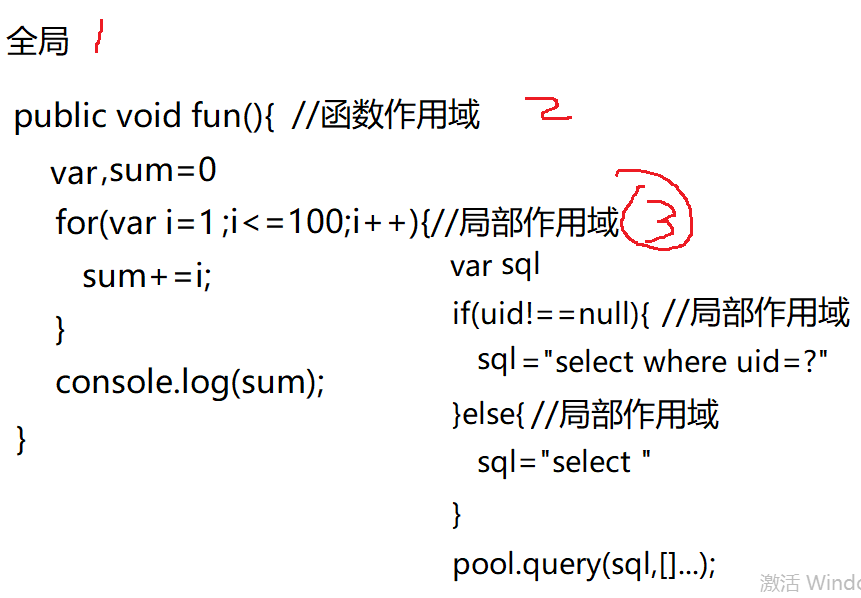
for() { ... }

if(){ ... } else { ... }

while(){ ... }

如果在大括号内声明的变量出了{}，都不能用。

比如:



但js中，出了{}还能用

作用域链: scope chain

什么是: 由多级作用域组成的链式内存结构

原理:

每个函数都有一个隐藏的属性scopes

不调用函数时scopes只包含window的引用，指向全局作用域。

调用函数时，创建临时作用域对象，并将临时函数作用域对象的引用地址添加到scopes中，window之前。

调用时优先使用临时函数作用域中的变量，如果函数作用域中没有，才去全局window中找。如果都找不到，才会报错。

调用后: 函数作用域会被释放，函数作用域中的局部变量也就释放！

### 2. 闭包:Closure

什么是: 即重用一个变量，又保护变量不被篡改

为什么: 全局变量: 优: 可重复使用。缺: 易被污染。

局部变量: 优: 仅函数内可用。缺: 不可重用。

何时: 当一个函数需要拥有一个可重用的专属的变量时

如何: 给函数和变量找一个共同的父母函数3步:

1. 定义外层函数包裹内层函数和要保护的变量

问题: 原来全局的函数，现在变成局部的函数了，外边用不了了

2. 外层函数将内层函数抛出到外部！

问题: 外层函数如果不调用，是不执行的

3. 使用者调用外层函数，获得外层函数返回的内层函数保存在变量中。

结果: 使用者可反复调用保存在变量中的内层函数

function parent(){

//包 红包

var total=1000;//要保护的变量

//2. 外层函数将内层函数返回到外部

//生 一个孩子

return function(money){//内层函数

total-=money;

document.write(`本次花了${money}元，还剩${total}元<br>`)

}

}

//3. 使用者调用外层函数，返回内层函数的对象，使用者用自定义的变量接住返回的内层函数对象。

var child=parent();

/\*child:function(money){

total-=money;

document.write(`本次花了${money}元，还剩${total}元<br>`)

}\*/

total=0;

child(100);//剩900

child(100);//剩800

问题: 全局没有total=1000，内层函数也没有total=1000，total保存在哪儿了？又是怎么被重复使用的？

鄙视: 闭包如何形成:

外层函数parent调用后，外层函数的作用域被内层函数引用着，无法释放。

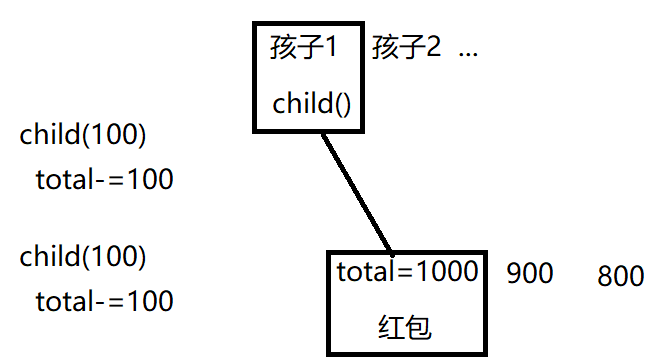
外层函数的作用域，因为是保存在内层函数的私密的scopes通讯录里，所以外人都不知道！且通过全局，绝对无法修改。

鄙视时: 画简图: 找两样东西: scope 范围

1. 外层函数共向外返回了几个内层函数

父母 共 生了 孩子

2. 那个变量是受保护的变量（给孩子包的红包）



再比如:

function fun(){

//包 红包

var n=999;

//任何位置，给不存在的变量强行赋值，结果自动在全局创建！

//生 孩子1

nAdd=function(){n++};

//生 孩子2

return function(){

console.log(n)

}

}

var getN=fun();

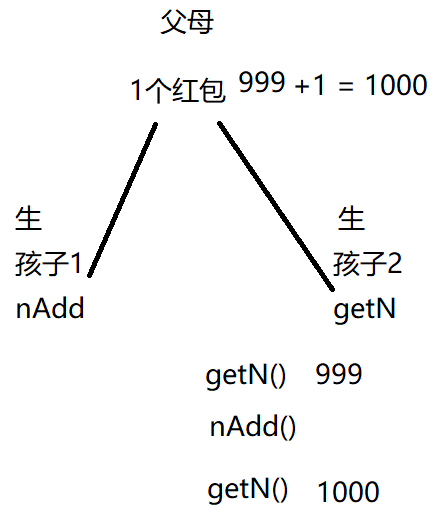
//生了两个孩子: getN和nAdd

//包了一个红包，两个孩子共用

getN();//获取红包金额 999

nAdd();//将红包+1

getN();//获取红包金额 1000



闭包的缺点: 闭包比普通的函数多占用一块内存！多占用外层函数的作用域对象，不能释放

解决: 释放闭包: 2种: chain 锁链

1. 给child变量赋值为null 。变量child依然在内存中

2. 删除child: delete window.child

删除 window中的child变量

前提: child变量使用window.添加的。

鄙视: var a=10; window.a=10; window["a"]=10;差别

相同: 都是向全局添加一个全局变量a=10

不同: var a不能用delete删除

window.a和window["a"]可用delete删除

总结: 将来就算被迫用全局变量，也必须用window.强行添加！

补: js中向数组中一个不存在的位置强行赋值:不会报错，而是自动添加该新元素

比如: var arr=["亮","然"]

arr[6]="东" //不报错

结果: arr中6号位置，会自动添加东: arr:[ 0:"亮",1:"然",6:"东"]

因为js中对象底层都是关联数组，所以，强行向对象中一个不存在的属性赋值，结果，会自动添加该属性。

比如: window.child=xxx; 结果: 会在window中自动添加一个变量child

且child可被delete删除

## 5. 面向对象: 封装 继承 多态

什么是对象:

用法: 描述现实中一个具体事物的属性和功能的一种程序结构

本质: 内存中可存储多个数据或函数的的存储空间，再起一个名字。

为什么: 为了大量数据的维护和使用方便。

何时: 今后都是用面向对象方式开发程序

如何使用:

三步，也是面向对象三大特点: 封装 继承 多态

### 1. 封装3种:

什么是封装: 创建一个对象结构，集中保存一个事物的属性和功能。

为什么: 便于大量数据的维护和使用

何时: 只要用面向对象方式开发，都必须先封装对象。

如何: 3种:

1. 用{}创建:

var 对象名={

属性名 : 属性值 ,

... : ... ,

方法名 : function(){

可重用的功能

}

}

访问对象成员: 成员=属性+方法

访问属性: 对象名.属性名

属性其实就是保存在对象内部的变量而已

访问方法: 对象名.方法名()

方法其实就是保存在对象内部的函数而已

问题: 对象自己的方法，想使用自己的属性，也不能直接写属性名。

原因: 任何函数在执行时，只要没有加.前缀的变量，被限制只能在全局window中查找。不允许擅自进入对象。

不好的解决: 在属性名之前写死对象名:

console.log(`I'm ${lilei.sname},I'm ${lilei.sage}`)

问题: 紧耦合: 别人改，我被迫要跟着改

程序中追求的是松耦合: 别人变，我可不变，或自动跟着变。

好的解决方法: this.属性名

什么是this: 自动获得正在调用当前函数的.前的对象主语。——.前是谁？this自动就变成谁！

何时: 只要对象自己的方法，要使用自己的属性时，必须加this.前缀！

比如:

console.log(`I'm ${this.sname},I'm ${this.sage}`)

day05

2. 用new: 2步:

1. 先创建一个空对象:

var obj=new Object(); 等效于 {}

2. 用强行赋值的方式添加新属性:

obj.新属性=值

obj.新方法=function(){ ... }

揭示了一个js最底层的原理:

js中一切对象底层都是关联数组。

vs 关联数组:

1. 访问关联数组成员: 2种:

对象["属性名"] 比如: ym["math"]

↑翻译为

对象.属性名 比如: ym.math

固定用法: 如果属性名是动态获得，不是写死的，就必须用[]方式获得属性值，且不能加""

比如： for(var name in obj){

console.log(obj[name])

}

2. 访问不存在的成员: 不报错，而是返回undefined

3. 强行给不存在的成员赋值: 不报错，自动添加新成员。所以，今后向对象中，临时添加新成员，都用强行赋值的方式。

4. 遍历: 关联数组可用for in循环遍历。对象中的成员也可以用for in遍历

鄙视: 克隆对象:

什么是克隆对象: 照着原对象的成员结构，再创建一个新的一模一样的对象。

何时: 不希望别人直接使用原对象时，保护原对象

如何: 3步:

1. 先创建一个空对象

2. 遍历原对象中的每个属性，每遍历一个原属性，就向新对象中添加同名属性，值也和原属性值相同。

3. 返回新创建的对象.

前两种方式的问题: 一次只能创建一个对象。如果反复创建多个相同结构的对象时，代码会很繁琐

3. 用构造函数，反复创建多个相同结构的对象

什么是构造函数: 专门描述一类对象的统一成员结构的函数

为什么: 重用结构

何时: 当需要反复创建相同类型的多个对象时

如何: 2步:

1. 先定义构造函数，规定一类对象的统一结构

function 类型名(属性参数列表){ //有几个属性，就要同时定义几个形参

this.属性名=xx;

this.属性名=xx;

this.方法=function(){

... this.属性名

}

}

2. 用new调用构造函数，照图施工，创建新对象

var obj=new 类型名(属性值列表)

↓ ↓ ↓ 传入

function 类型名(属性形参列表)

原理: new做了4件事:

1. 创建一个新对象:

2. 设置新对象的爹/原型继承构造函数的原型对象

3. 用新对象，调用构造函数，将构造函数中的this都换为新对象。通过给新对象强行赋值新属性的方式，添加新属性。

4. 返回新对象的结果

问题: 如果将方法的定义放在构造函数内，那么，每创建一个新对象，都会创建方法的副本——浪费内存

解决: 继承

### 继承:

什么是: 父对象的成员，子对象无需重复创建，就可直接使用。

为什么: 节约内存，代码重用

何时: 只要多个子对象，拥有相同的属性值或方法定义时，都应该在父对象中定义一次。所有子对象共用！

如何: js中都是通过原型对象的机制来实现继承

什么是原型对象: 集中存储多个子对象，共有成员的父级对象。

何时: 只要为多个孩子定义统一的方法定义或属性值时，都要定义在原型对象中。

如何:

1. 原型对象不用自己创建: 买一赠一

每当创建一个构造函数时，都会附赠一个空的原型对象

可通过构造函数的prototype属性找到，当前构造函数对应的原型对象

比如: Student.prototype

2. 如何让孩子继承爹: 自动的

new的第二步是设置新对象的爹/原型继承构造函数的原型对象。

所有子对象都有一个\_ \_proto\_ \_属性指向构造函数的原型对象

凡是从\_ \_proto\_ \_指出的关系，都是继承关系。

比如: lilei.\_ \_proto\_ \_=Student.prototype

3. 如何利用继承关系，优化代码: 需要自己做

凡是多个子对象共用的成员方法或属性值，都要定义在构造函数的原型对象中——通过强行赋值的方式

比如: Student.prototype.intr=function(){ ... }

结果：所有孩子可以共用！

总结:

将来要定义构造函数时:

1. 所有的属性，都定义在构造函数中

所有定义在构造函数中的属性，将来都会创建到子对象中。

子对象中有什么属性，取决于构造函数中有什么属性

2. 所有的方法，都要强行添加到构造函数的原型对象中

构造函数中有多少个方法，子对象就能使用多少功能。

内置类型也都有构造函数和原型对象:

内置类型: Array Date RegExp Function ... ...

比如: Array家:

构造函数 + 原型对象

function Array(){ Array.prototype原型对象

... ... sort() join() ... ...

}

才能 new Array()

反例: Math不能new，因为不是构造函数

var arr1=[]; arr1.sort(); arr1.join()

var arr2=[]; arr2.sort(); arr2.join()

如果内置类型中的原型对象中的方法不够用，可自己向原型对象中添加共有方法

比如: 希望给所有数组都添加一个共有的方法sum()，可对当前数组中的所有元素求和:

Array.prototype.sum=function(){

... this->将来调用sum()的.前的数组对象 ...

}

比如，将来:

var arr1=[1,2,3];

var arr2=[1,2,3,4,5];

console.log(

arr1.sum(), //6 this->.前的arr1

arr2.sum() //15 this->.前的arr2

);

### 原型链:

什么是: 由多级原型对象组成的链式结构

原理:

1. 所有对象都有\_ \_proto\_ \_属性指向自己的爹

2. Object类型(function Object(){}和Object.prototype是所有对象的顶级父类型)

3. 只要在自己的原型链上的方法，自己都能用。只要不在自己的原型链上的方法，自己都不能用

强调: 因为爸爸和妈妈之间没有继承关系，所以子对象用不了姥爷中的方法

### 多态:

什么是多态: 同一个函数在不同情况下表现出不同的状态

包括2种情况:

1. 重载:

2. 重写(override)

什么是: 子对象中定义和父对象同名的成员

何时: 只要爹的不好用，就自己定义自己的

如何: 只要在子对象中，定义和父对象同名的成员

后果: 使用时，先用自己的。不再用父对象继承来的

建议: 所有自定义类型: Student或定义对象var lilei={}，都应该重写toString()方法便于调试

### 总结:

三大特点： 封装，继承，多态

封装: 创建对象: 2种

1. 如果只创建一个对象，不会反复创建:

首选: var obj={ ... ... }

2. 如果反复创建多个相同结构的对象时:

首选构造函数: 2步:

1. 定义构造函数:

function Student(sname,sage){

this.sname=sname;

this.sage=sage;

//不能将方法定义放在构造函数中

}

2. 用new 调用构造函数：

var lilei=new Student("Li Lei",11);

继承: 所有子对象共有的方法，应该添加到构造函数的原型对象中

用子对象调用方法时，先在子对象本地查找。如果自己没有，才延原型链向父级对象查找。直到找到为止。

强调: 原型对象中的方法中的this指将来调用该方法的.前的某个子对象。

多态: 重写: 如果从父对象继承来的方法不好用，可在子对象本地定义同名方法，覆盖父对象中的不好用的方法。

补: 按值传递，没有按~~引用传递~~

什么是: 当两个变量间赋值时，只是将原变量中的值赋值一个副本给对方。

两个后果:

1. 基本类型/原始类型: number string boolean ...

赋值是将原变量中的值本身复制一个副本给对方

修改新变量，原变量不受影响

2. 引用类型: 对象，数组，。。。

赋值只是将变量中保存的对象地址，复制一份给新变量。对象并没有创建一个新的。还是只有一个对象。只不过两个变量用相同的对象地址，指向同一个对象而已。任何一方修改对象，另一方，同时受影响。

今日英文:

1. DRY Don't repeat yourself

不要 重复 你自己的(代码)

2. constructor 构造函数

3. prototype 原型对象

\_ \_proto\_ \_ 原型

4. override: 重写 推翻/遮挡

day06

### 总结: this

1. obj.fun() fun()中的this->.前的obj

2. fun() fun()中的this->默认指window =window.fun()

3. new fun() 中的this->新对象

4. Array.prototype.sum=function(){ this-> 将来调用sum的.前的任意数组对象！}

### 自定义继承: 2种:

1. 只修改一个对象的父对象:

child.\_\_proto\_\_=father

Object.setPrototypeOf(child,father)

设置 child 的 prototype为father

2. 批量修改多个对象的父对象:

构造函数.prototype=father

时机: 必须在创建子对象之前换，才能保证之后创建的所有孩子，自动继承新爹！

## 6. ES5

ECMAScript: javascript语言的国际标准,仅定义了js语言的基本语法,没有定义如何操作网页内容.

问题: JavaScript的语言诞生极其草率，有很多广受诟病的缺陷。

ES5: 是ECMAScript制定第五版js语言的标准

### 1. 严格模式:

什么是: 比普通js运行机制要求更严格的模式

为什么: 为了改善js中很多广受诟病的缺陷

何时: 所有js程序都要运行在严格模式下

如何: 只要在当前代码的顶部添加: "use strict" 使用严格模式 strict 严格

新的要求:

1. 禁止给未声明的变量赋值:

旧js中: 给未声明的变量赋值，不会报错，而是在全局创建该变量——造成全局污染。

严格模式: 给未声明的变量赋值，报错！xxx is not defined!

2. 静默失败升级为错误:

什么是静默失败? 执行不成功还不报错！

ES5严格模式中: 将所有静默失败都变为报错！

3. 普通函数调用或匿名函数自调中的this，不再指向window，而是undefined。

4. 禁止使用arguments.callee

arguments.callee() 在当前函数执行内部，再次调用当前函数自己！——专门用于做递归调用。

递归2个问题:

1. 紧耦合: 在函数内写死函数名，将来函数名变化，函数内部被迫跟着修改。

解决: arguments.callee-> 自动获得当前函数本身，而不用管函数名

2. 致命问题: 重复计算量太大！递归效率极低！

严格模式禁用arguments.callee，就意味着，禁用了递归算法！

解决: 绝大多数递归，都可用循环代替！

### 2. 保护对象:

什么是: 防止对对象的成员或结构做一些无意义的篡改。

为什么: js中的对象，毫无自保能力

如何: 2大块内容:

1. 保护单个属性的值: 2种:

1. 保护普通数据属性的值:

数据属性: 值直接保存在属性的属性

如何: 其实ES5中，对象的每个数据属性，都变成了一个缩微的小对象，每个缩微的小属性对象中都包含一个属性值+3个开关属性: {

value: 实际存储属性值,

writable: 控制是否可修改当前属性值,

enumerable: 控制是否可被for in 遍历到

问题: 虽然可防住for in，但用对象.的方式依然可强行访问。

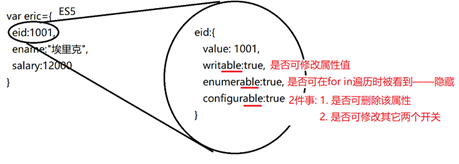
configurable: 控制是否可删除该属性

控制是否可修改前两个开关

一旦改为false，不可逆

所以当设置其他两个开关时，都带上configurable:false，作为双保险。

}



如果修改属性的开关:

1. 只修改一个属性的开关:

例如：(重新)定义 obj的指定属性

Object.defineProperty(obj, "属性名", {

开关: true/false,

开关:true/false,

})

2. 同时修改多个属性的开关:

Object.defineProperties(obj,{

属性名: {

开关: true/false,

... : ... ,

},

属性名: {

开关: true/false,

... : ... ,

},

})

问题: 使用开关保护属性，只能提供基本的单调的保护。无法根据自定义规则保护属性。

比如: 年龄必须介于18~65之间

2. 用自定义规则保护属性值:

访问器属性:

什么是: 保镖，不实际存储属性值，专门提供对另一个数据属性的自定义保护。

何时: 只要使用灵活的自定义规则保护属性时，都要请保镖。

如何: 3步:

1. 前提: 先定义一个半隐藏的受保护的数据属性，实际存储属性值。且数据属性名不要用正式属性名，最好\_开头，以示区分。

Object.defineProperty(eric,"\_age",{

value: 26,

writable:true, //可以改

enumerable:false, //半隐藏

configurable:false //双保险

})

define: 定义 un未 define定义 d过的 property: 内存中所有用对象.xxx都可访问的属性统称为property

2. 请保镖: 一请，就是一对儿

一个保镖叫get函数, 专门负责从受保护的数据属性中取值

另一个保镖叫set函数，专门负责将要修改的值，保存回数据属性中。但是，set函数通常都会对传入的新值进行验证！只有验证通过，才保存回去。如果验证不通过，不保存，且报错！

Object.defineProperty(eric,"age",{

get:function(){ return this.\_age },

set:function(value){

//value会自动接住要赋的新值

if(value>=18&&value<=65){

this.\_age=value;

}else{

throw Error("年龄必须介于18~65之间")

}

}

})

3. 保镖何时发挥作用:

当试图用保镖获取属性值时，自动调用get()

比如: console.log(eric.age)

自动调用age:{ get:function(){ ... } }

当试图用保镖给属性赋值时，自动调用set()，且自动将要赋的新值，传给value形参，先验证  
比如: eric.age=27 27

↓ 自动

会自动调用age:{ set:function(value){ ... } }

总结: 单从使用上来看，访问器属性的用法和普通属性完全一样！只不过，运行机制，不同。

### 3. 保护对象结构: 3个级别:

保护对象结构: 防止别人对对象添加新属性或删除现有属性

#### 1. 防扩展: 禁止给对象添加新属性

阻止对obj对象扩展任何新属性: Object.preventExtensions(obj) prevent阻止 extensions扩展

问题: 只能防止添加新属性,不能防止删除现有属性

保护对象结构: 防止别人对对象添加新属性或删除现有属性

原理: 其实每个对象中都有一个隐藏的内部属性:

extensible: true 所有对象默认都是可扩展的

Object.preventExtensions(obj); 将extensible改为false

#### 2. 密封: 在兼具防扩展同时，禁止删除obj中现有属性

属性值依然可以改

Object.seal(obj) 禁止删除obj中任何属性 seal密封，打包

原理: 1. 自动调用preventExtensions()阻止扩展

2. 自动将所有属性的configurable开关都改为false!

将来一个对象的结构，保护到seal密封程度就够了。

#### 3. 冻结: 在兼具密封的基础上，又进一步禁止修改所有属性值。

Object.freeze(obj) 禁止修改obj中的所有属性的值 freeze 冻结，冷冻

原理：1. 自动调用(执行)seal()密封了obj

2. 自动将所有属性的writable：false

### 4.Object.create():

Object.create(): 如果没有构造函数，也想创建子对象继承父对象时 create创建的意思

什么是: 基于一个现有的父对象，创建子对象，自动设置继承关系

何时: 没有构造函数，不能new时，也想创建子对象

如何:

var child=Object.create(father,{

属性1:{ value + 三个开关},

... : ...

})

1. 创建一个新的空对象child

2. 自动设置child继承father

3. 为child添加自有属性

### Object.create() VS new

相同: 前三步做的事儿都是相同的

不同: new必须构造函数才能执行; create只要有father就能执行;

### 5.call()/apply()/bind()

相同: 都是用来替换函数中不正确的this

何时: 只要函数中的this不是想要的，都可随意替换

如何: 2种:

1. 在调用函数时，临时换一次: call 打电话，程序中: 调用函数

函数.call(替换this的对象, 实参值列表...)

比如: calc.call(lilei, 10000,1000,2000)

1. call是调用的意思，先调用calc函数

2. call将第一个参数对象替换calc中的this

3. call会将后续实参值列表注射入calc中传给形参

问题: 如果不确定要注入的实参值个数，可用数组向函数中注入实参值列表。

如果传入的实参值是放在一个数组中的，可将call()换成apply()，

2.apply()可先打散数组为单个值，再分别传入要调用的函数中。

3. bind:

什么是: 基于原函数，创建一个和原函数一模一样的新函数，并永久绑定this指定的对象：

何时: 如果函数中的this，需要反复被替换，就用bind创建一个新的专属的函数。永久绑定this。

如何: var 新函数=原函数.bind(对象, 要绑死的实参值,...)

var newFun=原fun.bind(替换this的对象)

原理: 1. bind会创建一个和原函数一幕一样的新函数副本——只创建，并未调用

2. 将新函数中的this，永久绑定为指定的对象

结果: 从此，调用新函数时，不用再每次都临时替换this

总结: 只要函数调用时，其中的this不是想要的，就可用call/apply/bind替换

1. 如果只在本次调用时，临时替换一次，就用call()

如果给定的实参值是放在数组中的，则用apply()先打散，再调用函数，并传入。

2. 如果希望永久绑定this时，就用bind()，创建一个专属的新函数，永久绑定this为指定的对象，同时可绑定部分固定的参数值。

强调: 用bind创建的新函数中的this，不可能再被别人用call替换。因为bind后的函数中的this已经被替换成固定的对象了。所以再用call/apply已无this可换

### 6.数组家的新函数=数组API:

ES5中为数组扩展了很多新的API

#### 1. 查找元素:

数组中也有indexOf方法，查找一个指定元素的位置

var i=arr.indexOf("元素值",starti)

在arr中，从starti位置开始，找下一个匹配的元素的位置i

如果找到，返回元素的位置，如果找不到，返回-1

#### 2. 判断: 判断数组中的元素是否符合要求

arr.every(条件) 判断数组中是否所有元素都符合条件 every每个

arr.some(条件) 判断数组中是否包含符合条件的元素 some一些

1. 判断数组中的元素是否都符合要求:

var bool=arr.every(function(elem, i, arr){

//回调函数: 自动在每个元素上执行一次

//每次调用时，都会拿到三个值:

// elem 自动获得当前正在遍历的元素值

// i 自动获得当前正在遍历的元素位置

// arr 自动获得当前正在遍历的整个数组对象

return 判断条件

})

原理: 如果将回调函数去每个元素上都调用一次，都返回true，整个every才返回true

如果调用过程中，只要有一个元素返回false，则整个every返回false

2. 判断数组中是否包含符合要求的元素:

var bool=arr.some(function(elem,i,arr){

return 判断条件

})

原理: 只要有一个元素判断返回true，整个some()就返回true。

除非所有元素判断后都返回false，整个some()才返回false

【执行原理: every会自动遍历arr中每个元素，每遍历一个元素，就用回调函数在当前元素上执行一次。每次调用是都会自动给回调函数传入当前元素值、当前位置和当前数组对象。中途every会记录每个元素判断后得到的结果。最后，只有所有元素判断的结果都为true，整体才返回true。只要有一个元素的判断结果返回false，整体就返回false

arr.some()和arr.every()用法完全一样，只不过不要求所有元素都满足条件才返回true。而是只要有一个元素满足条件，就返回true。】

#### 3. 遍历: 对数组中每个元素执行相同的操作。2种：

1. arr.forEach(): 仅遍历，对原数组中每个元素执行相同操作

何时: 如果仅遍历数组没有返回值，或仅对原数组中的元素执行修改操作，不需要返回新数组时。

如何：arr.forEach(

//for(var i=0;i<arr.length;i++)

function(elem,i,arr){

//不需要返回值 并对elem、i、arr执行操作!

} )

2. arr.map(): 遍历取出原数组中每个元素，执行相同操作后，放入一个新数组中返回。原数组保持不变

何时: 如果希望保护原数组不变，而返回新数组时

如何:

var newArr=arr.map(

//1.先创建一个新的空数组 var newArr=[] 2.再自动遍历原数组中每个元素 3.每遍历一个元素，就调用一次回调函数

//for(var i=0;i<arr.length;i++)

function(elem,i,arr){

return 处理后的新值

}

//每次调用回调函数后，都会将返回的新值，自动放入新数组中对应位置

//newArr[i]=返回新数组

//return newArr; 当遍历完成后，将新数组返回！

)

#### 4. 过滤和汇总:

##### 过滤: filter

什么是过滤: 复制出原数组中符合条件的元素，组成新的子数组返回。原数组不变。

为什么: 因为未经允许不能擅自修改原数组。

何时: 只要希望仅获得数组中符合条件的元素时

如何:

var subArr=arr.filter(

var subArr=[]; //filter会拿着回调函数去每个元素上检查一次

function(elem,i,arr){

return 判断条件

}

//如果当前元素检查结果返回true，则复制一份，追加到subArr中。不保证是原数组中相同的位置

if(return true) subArr.push(elem) //遍历结束后

return subArr

)

示例：

var newArr=arr.filter(

var newArr=[];

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

function(elem,i,arr){

//自动在每个元素上执行一次

return 判断条件

}//只有返回true的元素，才被复制到新数组中保存

return newArr;

)

比如:

var arr=[1,2,3,4,5];

var evens=arr.filter( //filter会拿着回调函数去每个元素上判断一次

var subArr=[] //先创建空数组等着接符合条件的元素

function(elem){

return elem%2==0// 只要当前元素是偶数,就放入新数组

}

if(return true) subArr.push(elem) //如果当前元素判断结果为true，就追加到新数组

//遍历结束，返回subArr

)

console.log(arr);

console.log(evens);

##### 汇总: 对数组中的元素内容，进行统计，返回统计结果

什么是：对数组中元素内容，进行统计，返回统计结果 result统计，结果

如何: var result=arr.reduce(function(prev,elem,i,arr){ reduce减少，降低？？ reduce 统计、汇总

//prev: 截止到当前元素之前临时的汇总值

//elem、i、arr同上

return 将当前元素值，汇总到prev中，形成新的汇总结果

}, 起始值 )

比如:

var class1=[1,3,5,7,9];

var result=class1.reduce(

function(prev,elem){

return prev+elem;//将当前值汇总到临时汇总值中

},

0//从0开始累加

);

比如:

var sum=arr.reduce(

var prev=0;

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

function(prev,elem,i,arr){

//prev接住的是，截止到目前的临时汇总值

return prev+elem; //将当前元素值累加到临时汇总值

}//将本次累加的值再放回prev中

//return prev

)

## 7. ES6 day08

ECMAScript的第6个版本: 在不改变原理的基础上，尽量简化了代码。

### 1. let: 专门代替var，声明变量

什么是: 专门代替var，用于声明变量用的关键词

何时: 今后所有的var，都用let代替

为什么: var有两个广受诟病的缺陷:

1. 声明提前——打乱了正常的执行顺序

2. 没有块级作用域——块内和块外的变量就会互相干扰, 块内的变量很有可能提前到块外部，影响原本正确的外部的程序。

块: if(){ ... }else if(){ ... }else{ ... }

for(){ ... }

while(){ ... } do{ ... }while();

都是块，但都不是作用域

其内部的变量，都会被提前到块外对外部造成不可预期的影响。

let的特点:

1. 用let声明的变量不会被提前，也就不能提前使用

如果提前使用报错:

Cannot access 'a' before initialization initialization初始化 n. let a=10 initial()初始化: v. 简写: init

能 不 访问 a 之前 初始化 //不能在a初始化之前，访问a

在函数内:

let a=xx之前，到function(){内部，这篇区域是变量a的禁区。决不允许使用变量a。即使有全局变量a，也会报错！所以，如果局部变量和全局变量重名时，必须有一个变量要改名以示区分。

2. 添加了块级作用域:

块级作用域: if(){} while(){} for(){} ...

凡是这些块级作用域中的变量，都自成体系，不会影响外部！

原理: let相当于匿名函数自调！

### 2. 箭头函数:

代替一切函数定义：普通函数定义，匿名函数自调，回调函数

口诀：

1. 去function 变为 ()=>{ ... }

2. 如果只有一个形参，可省略()

3. 如果函数体只有一句话，则可省略{}。但如果仅有的一句话是return，则必须去掉return

箭头函数特点: 让内部this和外部this保持一致！

今后，只要希望回调函数内的this，和外部的this保持一致时，就可将回调函数改为箭头函数。

但是，如果就是希望函数内的this与函数外的this不同时，就不能改成箭头函数！

比如: 对象中的方法不能改成箭头函数

### 3. for of: 专门简化for循环，遍历数组

遍历索引数组: 3种:

1. 普通for: for(var i=0;i<arr.length;i++)

场景: 什么都可以做: 正序 倒序

依次递增，也可以2,4,6 也可以5 10 15 20

2. forEach: arr.forEach((elem,i,arr){ ... })

场景: 即获得元素的内容，又需要使用元素的位置时

局限: 只能从头到尾，依次遍历！

3. for of:

场景: 仅关心每个元素的内容，不关心位置时

for(var elem of arr){

of会依次取出arr中每个元素的内容

局限: 无法获得当前正在遍历的位置

}

以上三个专门遍历索引数组

遍历关联数组或对象，必须用for in

for in只能遍历关联数组，不能遍历索引数组：

in的本意是找属性的意思，且不仅找自有属性，还找原型链上的共有属性。

### 4. 参数增强:

1. 默认值: default default: 默认

ES6中可以给最后一个形参，定义默认值

如何：调用函数时，没有给定最后一个实参值，则自动使用默认值代替

如何:

在函数定义时: function fun(形参1,...,形参n=默认值){ }

调用时: fun(实参1,...缺实参n) 自动用默认值代替

2. 剩余参数: rest rest 剩余

专门代替arguments。将来想用arguments的地方，都可用剩余参数代替

为什么: arguments的缺点：

1. arguments不是纯正的数组，是类数组对象，数组家好的函数，arguments一个都不能用

2. arguments只能获得全部，不能有选择的获得部分实参值

如何: 定义函数时:

function fun(固定形参1,固定形参2,...数组名){

//数组中会接住除前两个固定形参之外的所有剩余实参值。——是纯正的数组

}

优势: rest语法:

1. 是纯正的数组，数组家的函数，都可使用 2. 可有选择的获取部分想要的实参值。

3. 打散数组: spread spread 散播

传统js中: 可用apply()打散数组为当值，再传入函数。

问题: apply()的本职工作是替换this，然后顺便打散数组。而实际需求中，多数情况仅打散数组，而与this无关。

比如: 本意从数组中找出最大值

Math.max.apply(null, arr) //null可换成任何东西,因为本需求和this无关,但是必须写！

解决: 调用函数时: Math.max(...arr) //拆

### 5. 解构: destruct

什么是: 将一个大的对象/数组中的成员，提取出来，单独使用。destruct 解构/拆解

何时: 只要仅使用一个大的对象/数组中的一小部分成员时

如何: 3种:

1. 数组解构: 从数组中取出个别元素，单独使用

如何: 下标对下标

var date=[2019,6,7,8,30,0];

0

↓原数组中相同位置上的值被提取出阿里，赋值给了相同位置的变量

0

var [y]=date; y=2019;

var [下标, , ,h,m,s]=date; //h=8, m=30, s=0

2. 对象解构: 从一个大的对象中提取出想要的个别成员单独使用:

如何: 属性对属性:

var 对象={

属性1: 值1,

... : ... ,

方法1: function(){ ... },

方法2: function(){ ... }

}

//比如只想用属性1的值和方法1函数

var {属性1: 变量1, 方法1: 变量2}=对象

结果: 变量1=值1,

变量2=方法1function(){ ... }

如果属性名和要用的变量名一致，可简写为:

var {属性1, 方法1}=对象:

结果: 属性名即当做配对，又作为提取出来之后的变量名使用。

3. 参数解构: 将所有参数放在一个对象中传入:

问题: 如果多个形参都不确定有没有时，既不能用默认值解决(只能用在最后一个参数)，又不能用rest解决(rest只能用在末尾的多个形参)

解决: 必须用参数解构： 2步

1. 定义函数时，就将所有的形参都以名值对的方式定义在一个对象中

2. 调用函数时，所有实参也必须以名值对的方式，放在一个对象中整体传入。

## 8.DOM

### 1. 什么是DOM: Document Object Model

什么是: 专门操作网页内容的一套对象和函数的集合

JS=ECMAScript核心语法 + DOM操作网页 + BOM操作浏览器软件

何时: 今后只要操作网页的内容，都用DOM

DOM其实是一套对象和函数的标准——W3C制定

为什么: 统一所有浏览器开发网页内容的标准，几乎所有浏览器100%兼容DOM标准

如何使用: 增删改查元素 事件处理

### 2. DOM 树

什么是: 内存中，保存当前网页中所有内容的树形结构

为什么: 树形结构是最直观的存储上下级包含关系的结构。而网页中的所有元素内容，都是上下级包含关系的。

DOM树长什么样？

1. 当浏览器读到网页时，先在内存中创建网页的根节点对象: documen

2. 浏览器按从上到下的顺序读取网页内容，每读到一项网页内容，就创建一个节点对象，并挂到指定父节点下。

### 3. 查找元素: 4种:

#### 1. 不需要查找就可直接获得的元素

document 根节点对象

<html> 根元素 document.documentElement

<head>元素 document.head

<body>元素 document.body

element元素: 2个意思: 1. 指数组中的每一项 2. 页面上一组<ANY></ANY>，称为一个页面元素

#### 2. 按节点间关系查找

何时: 如果已经获得一个节点对象，找周围附近的节点对象时

节点树: 包含所有网页内容的完整树结构

包括: 2大类关系

1. 父子关系: 4个属性

节点.parentNode 父节点

节点.childNodes 子节点们(直接)

返回多个直接子节点组成的类数组对象，可用下标访问指定位置的子节点

节点.firstChild 第一个子节点(直接)

节点.lastChild 最后一个子节点(直接)

2. 兄弟关系: 2个属性

节点.previousSibling 前一个兄弟

节点.nextSibling 后一个兄弟

问题:节点树包含所有节点，包括看不见的空字符节点。给查找元素造成了极大的障碍。程序员只关心元素节点，不关心文字节点。

解决: 元素树: 仅包含元素节点的树结构

包括: 2大类关系

1. 父子关系: 4个属性

元素.parentElement 父元素

元素.children 孩子们(直接)

返回多个直接子元素组成的类数组对象，可用下标访问指定位置的子元素

元素.firstElementChild 第一个 元素 子 (直接)

元素.lastElementChild 最后一个 元素 子

2. 兄弟关系: 2个属性

元素.previousElementSibling 前一个 元素 兄弟

元素.nextElementSibling 后一个 元素 兄弟

元素树的优点: 不包含看不见的空字符。不会受到空字符的干扰。

鄙视题: 遍历一个指定元素下的所有后代元素，找到符合条件的元素

递归遍历: 2步:

1. 仅遍历指定父元素下的直接子元素，执行替换操作

2. 对每个直接子元素，调用和父元素完全相同的操作

#### 3. 按HTML特征查找

问题: 按节点间关系查找，必须先有一个节点，才能找周围的关系。

解决: 今后，在还未获得元素的情况下，执行首次查找，都用按HTML特征查找

如何: 4个函数：

1. 按id查找一个元素 var elem=document.getElementById("id值");

强调: 1. 必须用document调用

2. 只能找到一个符合条件的元素。万一有两个id重了，只能找第一个。

2. 按标签名查找多个元素 var elems=任意父元素.getElementsByTagName("标签名")

强调: 1. 可以用任意父元素调用，用哪个父元素调用，就只在哪个父元素下查找

2. 返回多个元素组成的类数组对象

3. 不仅查找直接子元素，且查找所有后代

3. 按class名查找多个元素 var elems=任意父元素.getElementsByClassName("class名")

强调: 1. 用任意父元素调用

2. 返回多个元素组成的类数组对象

3. 不仅查找直接子元素，且查找所有后代

4. 仅靠其中一个class，就可以找到元素

4. 按name属性值查找多个元素 var elems=document.getElementsByName("name值")

专门用于找表单中的有name属性的表单元素

强调: 1. 只能用document调用

2. 返回多个元素组成的类数组对象

问题: 一次只能按一个条件查找！当查找条件复杂（需要多个条件反复查找才能找到元素）时，代码会很繁琐，步骤会很多。

#### 4. 按任意选择器查找元素:2个函数:

1. 仅查找一个符合条件的元素: var elem=任意父元素.querySelector("选择器")

强调: 1. 用任意父元素调用

2. 只能返回1个元素

3. 不仅查找直接子元素，且在所有后代中查找

强调:如果采用指定父元素，调用函数时必须是相对于父元素内的相对选择器

比如:

document.querySelector("table>tbody td:last-child ")

table.querySelector("tbody td:last-child")

~~table.querySelector("table>tbody td:last-child ")~~

2. 查找所有符合条件的元素: var elems=任意父元素.querySelectorAll("任意选择器")

强调: 1. 用任意父元素调用，缩小范围

2. 可返回所有符合条件的元素的类数组对象

3. 不仅查找直接子元素，且在所有后代中查找

缺点: 用选择器查找，效率不如用HTML特征查找高

总结: 1. 如果只要一个条件，就能找到想要的元素时，首选按HTML特征查找

2. 如果查找条件复杂时，需要多个条件才能找到想要的元素时，就首选按选择器查找——简单！

### 4. 修改: 3种:

#### 1. 内容:

1. 获取或修改元素开始标签到结束标签之间的原始HTML代码片段: 元素.innerHTML

2. 获取或修改元素开始标签到结束标签之间的纯文本内容: 元素.textContent

注意:textContent 比 innerHTML 多干了两件事:

1. 去掉了内嵌的标签 2. 转义符号翻译为正文

3. 获取或修改表单元素的值:表单元素.value

因为表单元素是单标签，氢必须用.value才能获得

#### 2. 属性: 3大类

1. HTML标准属性: HTML标准中规定的属性; 用.访问(2种方式修改:)

标准属性: title, class, href, src, target, id，value,name.....

1. 用最初的核心DOM函数: 4个

1. 获取一个元素指定属性的值: var 属性值=元素.getAttribute("属性名");

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

a.getAttribute("href") -> "http://tmooc.cn"

2. 修改一个元素指定属性的值: 元素.setAttribute("属性名","新属性值")

比如: a.setAttribute("target","\_self") 结果: <a href="http://tmooc.cn" target="\_self">

3. 判断是否包含指定属性: var bool=元素.hasAttribute("属性名")

比如: a.hasAttribute("target") => true

a.hasAttribute("title") => false

4. 移除属性: 元素.removeAttribute("属性名")

比如: a.removeAttribute("target") 结果: <a href="http://tmooc.cn">

问题: 太长！

2.使用HTML DOM提供的属性——核心DOM函数的简化版

HTML DOM在核心DOM基础上，对常用的属性和元素对象提供了部分简写的代码

福利: HTML DOM已经将所有标准属性提前定义在了内存中的元素对象上！可直接用元素.属性名获取或修改。

所有的HTML标准属性都提前定义在了元素对象上，属性值暂时都是""。

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

其实: a在内存中:

a:{

id:"",

href:" http://tmooc.cn ",

target:" \_blank ",

title:"",

name:""

}

所以:

获取a的href: a.href

修改a的target: a.target="\_self"

判断a是否包含title: a.title!=""

移除a的target: a.target="";

总结: 今后只要是HTML标准属性，都可用.访问

特例: class属性，虽然是标准属性，但不能直接用.访问，因为class是js的关键词。

所以DOM标准规定，凡是使用class属性，一律更名为className

访问元素的className属性，其实就是在访问<标签 class="">

ECMAScript中规定: 所有js对象内部都有一个隐藏的class属性，记录当前对象的类型名: "[object Object]"

如果ECMAScript标准中已经提前规定过了class属性另有用途。则DOM标准中不能重复定义class属性

总结： 1. 标准属性: 2种:

1. 核心DOM: 4个函数:

.getAttribute() .setAttribute() .hasAttribute() .removeAttribute();

坑: 核心DOM 仅对字符串类型的属性起作用

2. HTML DOM: 元素.属性名

2. 三大状态属性: disabled checked selected

特殊: 他们的值都是bool类型，不能用核心DOM函数，

而核心DOM的4个函数只能操作字符串类型的标准属性。所以，状态属性只能用.访问！

比如1: 想启用按钮 ~~btn.setAttribute("disabled",false)~~  无效

应该: btn.disabled=false;

比如2: 想判断一个checkbox复选框是否选中:

应该: if(chb.checked==true) 可简写为 if(chb.checked)

补充: CSS中有三大状态伪类: :checked :selected :disabled

专门选择处于某种状态的元素

3. 自定义扩展属性:

什么是: HTML标准中没有规定的，程序员自发添加的属性

何时: 2个典型场景:

1. 在客户端临时缓存数据，减少向服务器端发送请求的次数

2. 代替其它选择器，用于选择触发事件的元素，绑定事件处理函数

如何使用自定义扩展属性:

1. 在html开始标签中手动添加自定义扩展属性:

在html页面中写死: <ANY data-自定义属性名="值">

比如: <img src="1.png" data-m="m.png" data-l="l.png">

用程序动态设置: 只能用setAttribute，不能用.

比如: img.setAttribute("data-m","m.png")

结果:<img data-m="m.png">

2. 用js为元素添加/修改自定义扩展属性: 只能用setAttribute()

不能用.添加: 因为自定义扩展属性不是标准属性，内存中的元素对象上没有！

只能用核心DOM函数添加  
a.setAttribute("data-trigger","tab") 等效于: <a href="#" data-trigger="tab">

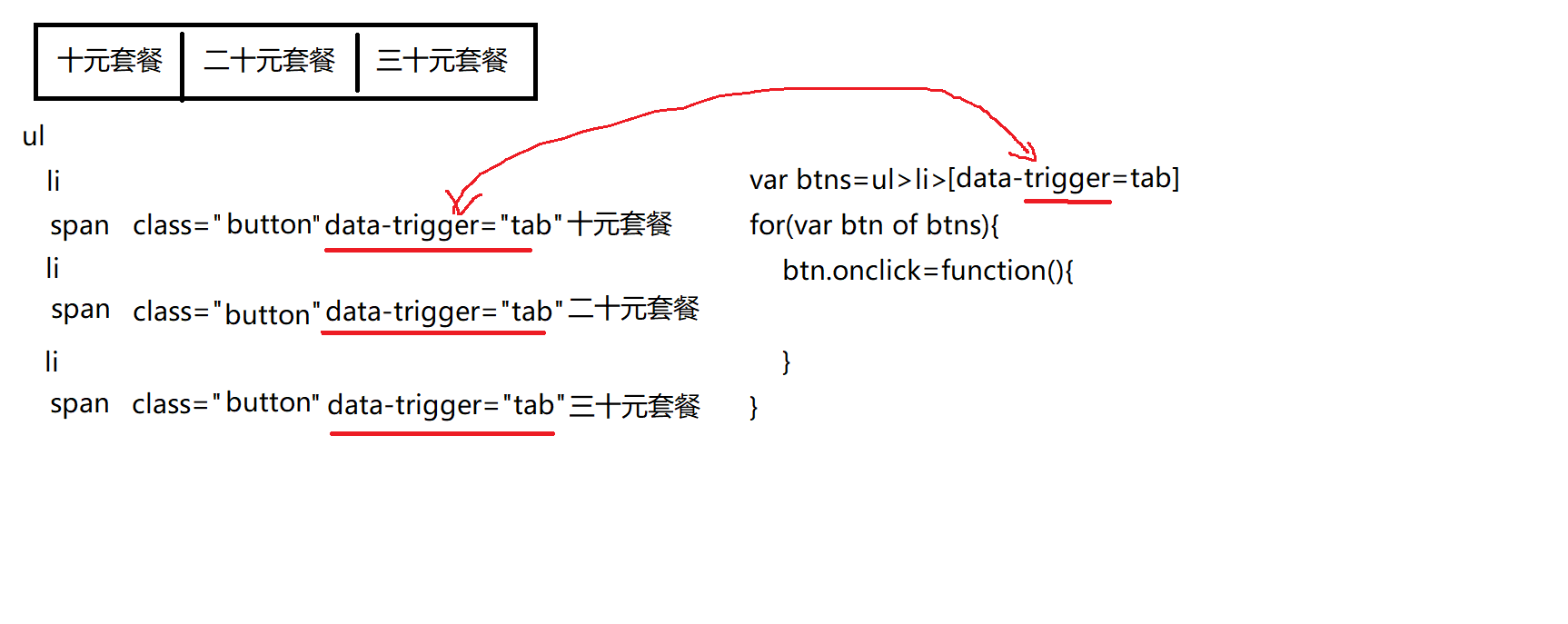
3. 读取自定义扩展属性的值: 只能用getAttibute("data-自定义属性")

比如: var m=img.getAttribute("data-m") m="m.png"

强调: 因为自定义扩展属性，只出现在页面中，不在内存中，所以，只能用getAttribute() setAttribute()...来访问，不能用.访问.

补充： 2个典型的场景

1. 代替id, class和元素选择器，用于选择触发事件的元素



2. 在网页中临时缓存业务数据



#### 3.样式:

1. 修改内联样式: 元素.style.css属性="属性值"

比如: div.style.display="none"; 隐藏 相当于: <div style="display:none">

div.style.display="block"; 显示 相当于:<div style="display:block">

修改元素的背景图片: div.style.backgroundImage="url(...)" 相当于: <div style="background-image:url(...)">

强调: 1.如果css属性名包含-，则必须去横线，变驼峰。首字母小写，之后每个单词首字母大写

比如: z-index -> zIndex background-position -> backgroundPosition

比如:修改div的z-index为11 ~~div.style.z-index=11;~~  应该: div.style.zIndex=11;

2. 如果css属性值带单位，则必须加单位！

div.style.width="400px" 必须加单位 相当于: <div style="width:400px">

修改 ul的高: ul.style.height="106px"

问题: 如果批量修改同一个元素的多个css属性时，使用style一次只能改一个css属性。代码会很繁琐。

2. 如果批量修改样式，最好用 .className="xxx"

今后，只要批量修改元素的样式都要先将多个css属性定义为一个class，再用.className="class名"一句话批量应用。

### 5. 添加/删除

添加一个新元素: 3步:

1. 先创建一个空元素:

var elem=document.createElement("标签名")

例如：var a=document.createElement("a") 结果: <a></a>

2. 设置空元素的关键属性:

a.href="http://tmooc.cn" a.innerHTML="go to tmooc"; 结果: <a href="http://tmooc.cn"> go to tmooc </a>

3. 将新元素挂载到DOM树上的指定位置上:

浏览器显示网页内容是根据DOM树上的节点对象画出来的。如果DOM树上的节点不变化，则浏览器不回重绘页面。

要想让浏览器将新元素画出来，只能将新元素添加到DOM树的指定位置，改变DOM树，才会触发浏览器的重绘操作。

如何: 3个函数:

1. 在指定父元素末尾追加子元素

父元素.appendChild(新元素) 比如: body.appendChild(a) append追加 child孩子

将a元素追加到body元素下子元素的末尾

2. 插入到现有一个元素之前

父元素.insertBefore(新元素, 现有元素)

比如: body.insertBefore(a, h1) 将a插入到h1之前

父元素.insertBefore(a,child) 将a插入到父元素下的child元素之前

3. 替换现有的一个元素

父元素.replaceChild(新元素, 现有元素)

比如: body.replaceChild(a, h1) 用a替换body下的h1

父元素.replaceChild(a, child) 用a替换父元素下的child元素

优化: 尽量减少操作DOM树的次数，避免重绘

为什么: 每操作一次DOM树，浏览器都要重绘整个页面，效率很低！会造成闪屏

解决: 2种情况:

1. 如果同时添加父元素和子元素到DOM树，应该先在内存中将子元素全部添加到父元素中，最后，再一次性将整个父元素加入DOM树。

结果: 浏览器只重绘一次

2. 如果父元素已经在页面上了，要添加多个平级子元素。应先将平级子元素添加到文档片段对象中，最后，再一次性将文档片段对象添加到DOM树

借助文档片段:

什么是文档片段: 内存中临时保存多个子元素的虚拟父元素对象

何时使用: 如果父元素已经在页面上了，要添加多个平级子元素

如何: 3步: create创建 document文档 fragment片段

1. 临时创建文档片段托盘对象: var frag=document.createDocumentFragment();

2. 将子元素临时添加到文档片段对象中 frag.appendChild(子元素)

3. 将文档片段添加到页面中指定的父元素下 父元素.appendChild(frag)

结果: frag将子元素都送到父元素指定位置后，就释放了！不占用页面空间。浏览器也只重绘一次

删除元素: 父元素.removeChild(child) 等效于 孩子.parentNode.removeChild(孩子)

### 6. HTML DOM常用对象

HTML DOM是对原有DOM函数和对象的简化

HTML DOM 对一些常用的元素对象也提供了简化的函数和属性

比如: <img> <select><option> <table>...</table> <form>...

#### Image对象: 就代表页面上一个<img>元素

唯一的简化，就是在创建<img>元素时:

var img=new Image(); 等效于: var img=document.createElement("img") 结果: <img>

强调: 通常只有两个元素可以用new简写: <img> <option>。其它元素依然需要document.createElement()创建

#### Select对象: 就是一个<select>元素的简化:

属性:

.selectedIndex 获得select选中的option的下标位置

select.options: 获得select下所有的option元素对象组成的集合。

select.options.length 获得select下所有option元素对象的个数

.length: 等效于 .options.length

比如: 强行select.length=0; 会清空select

select.value: 获得select中当前选中项的内容/值

如果选中的option有value，则优先返回value，如果选中的option没有value，则返回innerHTML内容代替

方法: .add(option) <==> .appendChild(option) 追加一个option

问题: .add()不支持frag .appendChild()支持frag的

.remove(i): 移除i位置的一个option

#### Option对象: 就是一个option元素

Option:代表select元素下的每个option元素对象

唯一的简化，就是在创建时: var opt=new Option(innerHTML, value)

比如: var option=new Option("-请选择-",0) 结果: <option value="0">-请选择-</option>

vs原DOM函数需要三步:

var option=document.createElement("option")

option.value=0;

option.innerHTML="-请选择-"

#### Table对象: 代表页面上一个<table>元素

Table是个大家族，采用逐级管理的方式:

Table管着行分组：

创建行分组: table.createTHead()

table.createTBody()

table.createTFoot()

例如：Table可添加行分组: var thead=table.createTHead();

等效于: var thead=document.createElement("thead") 和 table.appendChild(thead);

删除行分组: table.deleteTHead()

table.deleteTFoot()

获得行分组: table.tHead

table.tFoot

table.tBodies[i]

行分组管着行: 1步

创建行: var tr=tbody.insertRow(i)

结果：在tbody中第i行位置插入一个新行，并返回刚添加的新行对象，便于后续对新行中添加格和数据。原位置的行不是替换，而是向后顺移一位。

强调: insertRow()不但创建新行，而且立刻将新行添加到tbody中。

vs 原DOM: 2步

var tr=document.createElement("tr")

tbody.appendChild(tr)

固定套路:

在tbody开头插入一个新行 var tr=tbody.insertRow(0)

在tbody结尾追加一个新行 var tr=tbody.insertRow( )

行分组可删除行:

删除tbody中i位置的行 ~~tbody.deleteRow(i)~~

坑: i 要求是行在tbody内部的相对下标位置

问题：而一个tr(行)在行分组内的相对下标位置无法自动获得

可以自动获得的tr.rowIndex是相对于整个表中的下标位置。和tbody.deleteRow()的要求错位

解决: 今后只要删除行的标准写法都是:

table.deleteRow(tr.rowIndex)

因为.前的主语变成整个table，所以deleteRow()要求就变为使用整个表中的下标。就和tr.rowIndex的意义相符了。

行分组可获取行: tbody.rows[i]

行管着格:

行可添加格: var td=tr.insertCell(i) Cell:细胞，小格子

固定套路:通常都是在行末尾追加一格: var td=tr.insertCell()

问题: insertCell只能创建td，不能创建th

vs 旧DOM:

var td=document.createElement("td");

tr.appendChild(td);

行可以删除格: tr.deleteCell(i)

行可以获取格: tr.cells[i]

#### 补充: DOM中三大对话框:

警告alert() 确认confirm() 输入prompt()

#### Form对象: 代表一个<form>元素

获取表单对象: 其实也不需要查找，就可直接获得。因为表单对于所在的网页极其重要，所以网页就将本页面内的<form>元素都保存在一个forms集合中。可用[i]获取某一个表单对象

如果页面中有多个表单元素:var form=document.forms[i]

如果网页中只有一个<form>则: var form=document.forms[0]

属性: form.elements 获得表单中所有表单元素的集合

form.elements.length 获得表单中所有表单元素的个数

form.length <=等效于=> form.elements.length

表单元素对象: 比如: <input> <select> <textarea> <button>

获取表单中的表单元素:

普通方法: var 表单元素=form.elements[id|i|name]

比如: 获得form中name=username的input元素

var txtName=form.elements["username"]

获得form中的倒数第二个表单元素——input type=button保存按钮

var btnSubmit=form.elements[form.length-2];

简写: 如果一个表单元素有name属性，可直接用.name方法获得:

var txtName=form.elements["username"]

可简写为: form.username;

方法: 表单元素.focus() 让当前表单元素自动获得焦点

#### form表单自动获得焦点:

让用户名文本框一开始就自动获得焦点:username.focus();

## 9. BOM: Browser Object Model

什么是: 操作浏览器窗口或软件的一套对象和函数

何时: 如果想操作浏览器窗口，或访问浏览器软件相关的配置信息

JS=ECMAScript + DOM + BOM

核心语法 操作网页内容 操作浏览器窗口

ES标准 W3C标准 没有标准

问题: 没有标准——极大的兼容性问题，所以用的越来越少.包括: 对象体系

#### window: 3个角色:

1. 顶替了ES标准中的global，充当全局作用域对象

比如: 浏览器中: var a=10; window.a 输出10

再比如: window.parseInt() window.parseFloat() window.Array window.Date window.RegExp

2. 包含所有ES,DOM,BOM的对象和函数(1903笔记)

比如: window.document

window.document.getElementById()

2. 保存所有除ES外的DOM和BOM的对象和函数库(上届笔记，不一样以哪个为准？)

包括:全局的BOM函数和属性:

alert() confirm() prompt() window.open() window.close() ... ...

掌控特定功能的对象:

location对象 专门掌控地址栏

history对象 掌控着前进/后退/刷新

navigator对象 掌控着浏览器的配置信息

document对象 掌控着网页的所有内容

screen 对象 掌控着和显示屏有关的信息

event对象 掌控着浏览器中发生的所有事件

3. 还代表当前正在打开的窗口

比如: window.close() 是关闭当前窗口的意思

新chrome浏览器禁止随意关闭窗口，弹出黄色警告

比如: 在F12控制台输入

window.outerWidth 可获得当前窗口的总宽度

window.outerHeight 可获得当前窗口的总高度

window中常用的全局函数或属性:

1.对话框: prompt()输入框 alert()警告框 confirm()确认框

2. 窗口大小: 2组:

1. 完整窗口大小: window.outerWidth/outerHeight

2. 文档显示区: window.innerWidth/innerHeight

文档显示区: 浏览器窗口内专门用于显示网页内容有限的白色区域。

#### 3.打开和关闭窗口: 4种:

打开窗口: ~~window.~~open() 和<a>的用法几乎完全一样

总结: 用程序打开新链接有4种效果:

1. 在当前窗口打开新链接，可后退:

html: <a href="url" target="\_self">

js: window.open("url" , "\_self")

2. 在当前窗口打开新链接，禁止后退:

js: location.replace("新url")

原理: 用新的url地址，替换history中旧的地址，保证history中只有一条url，以而禁止后退

3. 在新窗口打开新链接，可同时打开多个

html: <a href="url" target="\_blank">

js: window.open("url","\_blank")

4. 在新窗口打开新链接，只能打开一个

html: <a href="url" target="自定义新窗口名">

js: window.open("url","自定义新窗口名")

原理: 其实，每个窗口在浏览器内存中都有一个唯一的名字，用来标识这个窗口。

浏览器规定，相同name名称的窗口，只能打开一个。

其实target属性，是在给新窗口起名,新打开的同名窗口会覆盖现有的重名窗口

所以，如果target的名字是自定义的名称，则新打开的窗口，只能开一个。不能重复开。

结果: 1. 如果打开链接时，使用自定义窗口名，则反复点击链接，只能打开一个

但是，target有两个预定义好的内置名称:

\_self: 自动获取当前窗口自己的名字给新窗口

结果: 新窗口会覆盖当前窗口

\_blank: 空白，就是不给新窗口指定任何名称，浏览器会自动随机在底层分配新窗口的名称，保证不重复。

结果: 不限制打开的新窗口个数

关闭窗口: ~~window.~~close()

#### window.history:

history: 保存当前窗口打开后，成功访问过的url的历史记录的集合(数组)。

当前窗口只要成功访问过一个url，url就会被push到history中保存。

能否前进后退，取决于现在正在看的url，在history前后是否有其他已经浏览过的url

何时: 用程序执行前进，后退，刷新

如何：只给我们开放了一个函数: history.go(i)

比如: 前进一步: history.go(1)

后退一步: history.go(-1)

刷新: history.go(0)

后退两步: history.go(-2)

#### window.location:

什么是location: 专门保存地址栏中url信息的对象，并提供了打开新链接的方法。

何时: 1. 获得url地址栏中各个部分的参数时

2. 通过操作地址栏中的地址实现跳转和刷新时

属性/方法: 1.可分段获得url的各个组成部分:

<http://127.0.0.1:5500/BOM/10_location.html>?username=dingding&pwd=123456&favs=running#top

协议 主机/IP 端口 相对路径 锚点 查询字符串

location.href 完整url

location.protocol 协议部分

location.host 主机名+端口号: localhost:3000

location.hostname 主机名 localhost

location.port 端口号 3000

location.pathname 相对路径 public/products.html

location.hash 锚点地址 #top

location.search 查询字符串 ?username=dingding&pwd=123456&favs=running

想进一步获得变量值: location.search.split("=")[1]

属性/方法: 2.通过操作地址栏实现跳转和刷新

1. 在当前窗口打开新链接，可后退: location.href="新url" <=等效于=> open("新url","\_self")

2. 在当前窗口打开新链接，禁止后退: location.replace("新url")

3. 刷新当前页面: location.reload(); <=等效于=> history.go(0)

#### window.navigator: 图+原视频

什么是: 保存浏览器配置信息的对象

何时: 判断浏览器的名称和版本号

userAgent: 保存浏览器名称和版本号的字符串

如何:

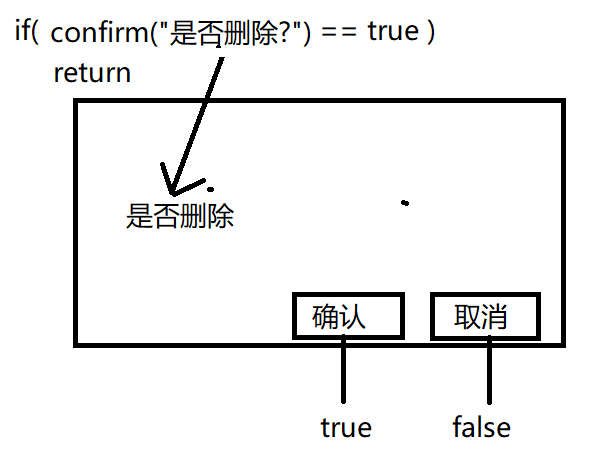
1. navigator的userAgent属性包含了浏览器的名称和版本号——字符串——没有标准

测试: window 10的 Edge浏览器

鄙视题: 如何判断正在使用的浏览器的名称和版本号

对话框: prompt()输入框 alert()警告框 confirm()确认框

确认框的用法:



## 10. 事件:

什么是: 浏览器自动触发的或用户手动触发的页面内容、状态的改变

什么是事件处理函数: 当事件发生时自动调用的一个函数

强调: 事件处理函数中的this永远指当前触发事件的元素

绑定事件处理函数:

每个元素对象上都有一批on开头的特殊属性，这些属性会在事件发生时自动触发对应的属性。

如果希望事件发生时，按照我们的意愿执行操作，就要提前将一个自定义函数赋值给on开头的对应属性保存起来

当事件发生时，会自动调用提前保存的函数。

本质: 是元素对象上一个事件名属性。

比如: btn.onclick sel.onchange txt.onfocus

何时: 只要希望发生事件时自动执行一项任务

如何: 在事件发生前就提前绑定事件处理函数到元素的事件属性上。

如何绑定:

1. 在HTML中:

<ANY onclick="js语句/函数调用"

<script>

function 事件处理函数(){ ... }

问题: 不符合内容与行为分离的原则，不便于维护和重用

2. 在js中用=赋值方式:

元素对象.onclick=function(){ ... this 指当前触发事件的元素对象}

问题: 一个事件只能绑定一个处理函数，不灵活！

3. 在js中用添加事件监听对象的方式: --最灵活

元素对象.addEventListener("事件名",处理函数)

添加 事件 监听对象

强调: 真正的事件名是没有on前缀的，只不过在用.时，为了和其它普通属性区分，才加了on前缀。

比如: click change focus blur load

原理:

什么是事件监听对象: 一个保存元素+事件名+处理函数一套组合的对象:

比如1: btnShoot.addEventListener("click",function(){

发射普通子弹

})

btnShoot.addEventListener("click",function(){

发射跟踪导弹

})

结果: 单击btnShoot，会先后发射两种子弹

原理: 其实，浏览器中有一个巨大的事件监听队列

我们为每个元素绑定的事件监听对象都会被添加到监听队列中。

当事件发生时，浏览器采用遍历监听队列的方式，找到符合条件的事件监听对象，并执行其中的处理函数

找到几个符合条件的，就执行几个符合条件的

比如2: btnShoot.addEventListener("click",fun1);

先创建一个事件监听对象:

{ 元素: btnShoot,

事件: click,

处理函数: fun1 }

然后，addEventListener会将监听对象添加到浏览器中一个隐藏的事件队列中。

当事件发生时，浏览器会遍历事件队列中的每个监听对象，选择匹配的监听对象中的处理函数执行。

如何从元素上移除一个处理函数:

元素对象.removeEventListener("事件名", 处理函数) == 元素对象.removeEventListener("click",处理函数)

remove移除 event事件 listener监听对象 处理函数决不能用匿名函数

移除事件监听对象:

强调: 移除监听对象时，元素、事件名和处理函数都要和添加时完全相同！

坑: 如果一个处理函数可能被移除，则添加监听时，就不能用匿名函数，必须用有名称的函数。移除时，必须使用函数名，找到原函数对象，才能移除。

### \*\*事件模型:

什么是: 从点击元素触发事件开始，到所有事件处理函数执行完，所经历的过程

DOM标准规定: 从点击元素触发事件开始，到事件处理函数执行完，期间共经历三个阶段:

1. 捕获阶段:由外(document)向内(实际触发事件的元素)，记录各级父元素上的处理函数。只记录，不执行。

2. 目标触发: 优先触发目标元素上的处理函数

目标(target)元素: 最初实际触发/点击事件的那个元素

3. 冒泡阶段: 由内向外，依次触发各级父元素上记录的事件处理函数

### 事件对象:

什么是: 在事件发生时，浏览器自动创建的，收集事件相关信息的对象。并提供了改造事件的函数。(也就是:封装事件信息和操作事件的API 的对象)

何时: 1. 想获得事件信息时: 点了那个元素，点在什么位置...

2. 想改造事件默认的行为: 取消冒泡！

如何:

1. 获取: 不用自己创建,直接使用

在事件发生时，自动获取事件对象:事件对象总是作为处理函数的第一个实参默认传入！我们只要在定义处理函数时，提前定义一个形参(e)接住。

如何获取事件对象:

比如: btn.onclick=function(e){ //event事件的缩写

//当事件发生时:会自动调用处理函数，并自动传入事件对象给形参e

}

btn.addEventListener("click",function(e){ ... })

2. 事件对象包括哪些功能?

1.取消冒泡(Propagation): e.stopPropagation() stop停止 Propagation蔓延,传播

什么是: 当前元素处理函数执行完，停止继续执行父元素上的处理函数

何时: 只要不希望触发父元素上的处理函数时，都要取消冒泡。

2. 利用冒泡(事件委托/事件代理):

优化: 尽量减少事件监听对象的个数

为什么: 浏览器触发事件，是采用遍历监听数组中每个监听对象来匹配并触发处理函数的。数组中监听对象多，遍历速度慢，数组中监听对象少，遍历快！

何时:当多个平级子元素，要绑定相同的事件处理函数时

如何:只要在父元素上统一绑定一次事件处理函数即可，点击子元素时，会自动冒泡到父元素上，所有孩子就可冒泡共用。也就是说，点击子元素时会执行提前各取委托好的处理函数。

两个难题:

this已经不指子元素了，而是指父元素。因为处理函数是定义在父元素上，且是等到事件冒泡到父元素时才触发。

坑1: 如何获得最初触发事件的目标元素？

正确: e.target 记录了事件最初触发的目标元素,用e.target代替this target目标

e.target可自动获得实际点击的目标子元素,且不随冒泡而改变

总结: this->不求天长地久，只愿曾经拥有 || e.target->一旦拥有别无所求

坑2. 有可能点在正确的子元素上，也有可能点偏，点在不想要的元素上。

解决: 在正式执行逻辑操作前，必须先判断/验证e.target是不是想要的。

哪些属性可作为区分的条件: nodeName, className,... 节点名 标签名 类型名

补: 如何获取元素的标签名: 元素.nodeName

但nodeName返回全大写的标签名: BUTTON A IMG LI ... ...

### 阻止默认行为:

问题: 有些浏览器的某些元素拥有默认的行为，而默认行为不是我们想要的。

何时: 如果元素自带的默认行为不是我们想要的，就可用e.preventDefault() 来阻止默认行为的发生

比如1: bootstrap中 prevent阻止 default 默认

<a href="#content1" data-toggle="tab">

点a时，不但执行绑定的操作，还会附加修改地址栏中的url： 后加#content1锚点地址

所以，今后只要用a当做普通按钮用时，都要阻止默认行为。

比如2: <a href="#"> 默认行为会在url地址栏结尾添加#，擅自篡改地址栏。

预告: HTML5拖拽API中，几乎所有函数，一上来先是两句话:

e.stopPropagation(); 取消冒泡

e.preventDefault(); 阻止默认行为

### 获得鼠标坐标:

事件对象在事件发生时，就自动获得了鼠标在屏幕中的位置。

事件对象中包含3组坐标:

1. 相对于整个屏幕左上角的坐标: e.screenX e.screenY screen屏幕

2. 相对于浏览器文档显示区左上角的坐标: e.clientX e.clientY client客户端

3. 相对于当前所在元素左上角的坐标: e.offsetX e.offsetY offset偏移

示例:获取鼠标的坐标

pop.onclick=function(e){

console.log(e.screenX,e.screenY)

console.log(e.clientX,e.clientY)

console.log(e.offsetX,e.offsetY)

}

## 11.jQuery

### 1. 什么是jQuery day01

jQuery是第三方开发的执行DOM操作的极简化的函数库

第三方: 由除了浏览器原生的以及咱们自己编写的之外，其它组织或个人开发的框架/库——要下载

执行DOM操作: jQuery还是在执行DOM的增删改查和事件绑定操作。——学jQuery，还是在学DOM

极简化: jQuery是对DOM操作的对象和函数进行的简化

函数库: jQuery中使用函数解决一切问题。jQuery中一切都是函数！

为什么:

1. 简单:

2. 兼容性: 凡是jQuery让用的，都没有兼容性问题:

比如: IE8不支持nth-child(i),但是jQuery可随便使用nth-child(i),

因为jQuery内部其实调用的还是原生的DOM函数，且一旦发现不兼容，立刻用js程序模拟。

何时使用jQuery: 几乎所有大项目或框架的底层，都是用jQuery开发的。——仅适合于PC端项目

### 2. 如何使用:

下载: www.jquery.com

版本: 1.x 唯一兼容旧的浏览器 jquery-1.11.3.js //未压缩

优点: 可读性好，适合学习和开发之用

缺点: 体积大，不便于下载和传输

jquery-1.11.3.min.js //压缩

优点: 体积小，便于快速下载和传输

缺点: 毫无可读型。

2.x 不再兼容旧浏览器

3.x 不再兼容旧浏览器，且添加了新功能

引入: 先引入jquery-1.11.3.js，再编写自定义js代码

因为DOM操作包含大量查找操作，所以js代码必须在body结尾引入，才能保证查找到任何想要的元素。

原理: 引入jquery-1.11.3.js时，其实是在内存中创建了一个新类型叫jQuery，包括

1. 构造函数: 负责创建jQuery类型的子对象

2. 原型对象: 负责保存所有jQuery的子对象共用的简化版函数，比如: .click() .html()都保存在jQuery.原型对象中

因为DOM元素不是jQuery家的孩子，所以，DOM元素不能直接使用jQuery家简化版函数。

只要想使用jQuery操作DOM，都要先创建jQuery类型的子对象。同时，使用选择器查找到DOM树上要操作的元素。将元素保存到jQuery子对象中。

var $btn=new jQuery("#btn1")

强调: 为了将jQuery家孩子和DOM家孩子区分，jQuery家孩子的变量都要以$开头——不是必须。

jQuery家的孩子，就可以调用原型对象中的简化版函数操作自己内部封装的找到的DOM元素了。

对jQuery家孩子调用简化版函数，等效于自动对内部封装的DOM元素调用原生的函数。

比如: $btn.click() =>

会被自动转换为 => btn.addEventListener("click",...)

强调: 原DOM中事件处理函数中的this，获得是DOM家的孩子。不能直接使用jQuery家的函数。所以:

必须先var $btn=$(this)，将this放入一个jQuery家的孩子中，才能使用jQuery家简化版函数。

强调: jQuery家的函数和对象，与DOM一定不能混用！决定用jQuery，就要通篇都用jQuery。

jQuery函数的三个小脾气:

1. 几乎所有的函数都自带遍历:

因为jQuery子对象其实是一个可保存多个DOM元素的类数组对象。所以，对整个jQuery子对象调用一次简化版函数，相当于对内部的每个DOM元素都调用一次。不用自己写遍历！

2. 多数函数都是一个函数两用:

调用时如果没给新值，就获取原值。 比如： $btn.html() 获取btn的内容

调用时如果给了新值，就变为修改值。 比如：$btn.html("xxx") 修改btn的内容

3. 几乎每个函数都返回正在使用的jQuery对象，可继续使用。

强调: 因为$是即创建新对象，又查找DOM树，所以，应该尽量少用！

解决: 链式操作！

注意: 做事儿的步骤，上一步的返回值，刚好是下一步的主语

### 3. jQuery只有两种查找:

#### 1. 按选择器查找:

如何: var $元素们=$("选择器") $()里，可以写所有css选择器，都支持！

另外，jQuery也扩展了少量新的选择器——只能在jQuery的$()中使用

新选择包括:

1. 基本过滤/位置过滤:

什么是基本过滤: 先将符合条件的元素，都取出来，放在一个集合中统一编号。且编号从0开始

包括: :first :last :eq(i) :gt(i) :lt(i) :even :odd

vs 子元素过滤:

什么是子元素过滤: 按照元素在其父元素内的相对编号选取元素，下标从1开始

:first-child :last-child :nth-child(i)

2. 内容过滤:

1. 用元素内容中的文本作为筛选条件:contains(文本)

2. 用元素内部的子元素拥有的特征来鉴别父元素:has("选择器")

3. 可见性过滤: 根据元素是显示还是隐藏，来选取元素

:visible 选择未隐藏的元素

:hidden 选择隐藏的元素 对这两种有效display:none visibility:hidden; 隐藏有四种方法12\_visibility filter.html

强调: 只能找到display:none，和type="hidden"两种隐藏的元素

4. 表单元素过滤:

:input 选择所有表单元素: input textarea select button

其实jQuery为每种type都提供了专门的选择器:

:text 选择普通文本框 <input type="text"

:password 选择密码框 <input type="password"

:radio 选择单选按钮 <input type="radio"

:checkbox 选择复选框 <input type="checkbox"

:button 选择按钮 <input type="button" ......

5. 状态过滤:

:enabled 选择启用的元素

:disabled 选择被禁用的元素

:checked 选择被选中的checkbox或radio元素

:selected 选择select中被选中的option

#### 2. 按节点间关系查找:day02

父子关系

.parent() <= .parentNode

.children() 获得直接子元素

.children("选择器") 只选择符合选择器要求的直接子元素

何时: 如果只在直接子元素中查找

.find("选择器") 在所有后代元素中查找符合条件的元素

何时: 如果希望在所有后代中查找时

.children(":first-child") <= .firstElementChild

.children(":last-child") <= .lastElementChild

兄弟关系

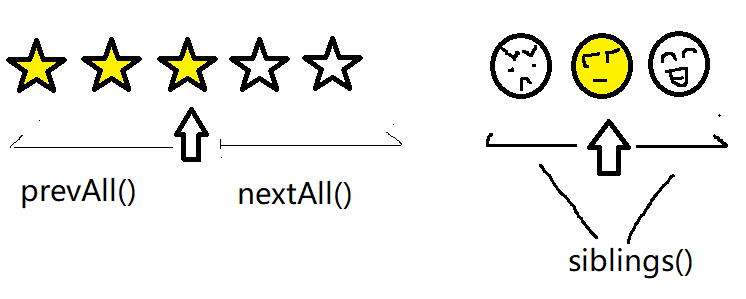
.prev() <= .previousElementSibling

.prevAll(["选择器"]) 在当前元素之前的所有兄弟中查找符合条件的

.next() <= .nextElementSibling

.nextAll(["选择器"]) 在当前元素之后的所有兄弟中查找符合条件的

.siblings(["选择器"]) 选择除自己之外的所有兄弟。不分前后。



### 4. 修改: 同DOM

内容:

1. 原始HTML片段: .html(["新内容"]) <= .innerHTML

2. 纯文本内容: .text(["新内容"]) <= .textContent

3. 表单元素的值: .val(["新值"]) <= .value

属性:

1. HTML标准属性: 2种:

核心DOM的函数：a.getAttribute()/setAttribute()

HTML DOM ： a.href

2. 状态属性: 只能用.disabled .checked修改

3. 自定义扩展属性: a.setAttribute()/getAttribute()

网页上可直接看到的属性: Attribute

HTML标准属性和自定义扩展属性

上边的都能用.getAttribute()/. setAttribute()

jquery，都改为.attr("属性名"[, "属性值"]) 一个函数两用

++++++++++++++++++++++++++++++++++++++++++++++++++++++

内存中必须用console.log()才能看到的属性: Property

HTML标准属性和状态属性

下边的都能用.直接访问——jq中没有属性，只有函数

jquery中都改为.prop("属性名"[, "属性值"])

比如:

获取a的href属性: $a.attr("href") / $a.prop("href")

修改a的data-target属性值为"content1" $a.attr("data-target","content1")

修改a的disabled为true $a.prop("disabled", true )

简写: 同时修改一个元素的多个属性:

$元素.attr或prop({

属性名: "属性值",

... : ...

})

样式:

获取或修改一个css属性的值:

$元素.css("css属性名"[, "属性值"])

如果同时修改多个css属性:

$元素.css({

css属性名: "属性值",

... : ...

})

如果批量修改css属性，都首选class来操作:

添加class: $元素.addClass("class名")

删除： $元素.removeClass("class名")

判断： $元素.hasClass("class名") => $元素.is(".class名")

在有或者没有一个class之间来回切换: $元素.toggleClass("class名")

比如: $元素.toggleClass("down");

在内部自动执行了:

if(!$btn.hasClass("down")){

$btn.addClass("down")

}else{//否则，就抬起

$btn.removeClass("down")

}

强调: .toggleClass()不能代替.addClass()或.removeClass()

如果只是想添加class，没想去掉class，首选addClass()

除非真的想在有和没有这个class之间来回切换时，才能选择toggleClass()

强调: 所有修改函数都是一个函数两用！

### 5. 添加/删除/替换/克隆

1. 添加: 2步:

1. 用html片段创建DOM元素 var $新元素=$(`html片段`)

2. 将DOM元素添加到DOM树上，将新元素添加到DOM树上

1. 末尾追加: $父元素.append ($新元素) return $父元素 等效于=> .appendChild() 简写: $父元素.append (`html片段`)

$新元素.appendTo($父元素) return $新元素 简写: $新元素.appendTo("父元素选择器")

比如:

1. 将新元素添加到父元素后，要修改父元素的宽度 $父元素.append ($新元素).css("width",500)

2. 将新元素添加到父元素后，要为新元素绑定事件 $新元素.appendTo($父元素).click(function(){ ... })

2. 开头插入: $父元素.prepend($新元素) ==作用一样，返回的值不一样== $新元素.prependTo($父元素)

3. 在现有元素前插入:

$现有元素.before($新元素) return $现有元素 => .insertBefore()

$新元素.insertBefore($现有元素) return $新元素

4. 在现有元素后插入:

$现有元素.after($新元素) return $现有元素 => .insertBefore()

$新元素.insertAfter($现有元素) return $新元素

5. 替换元素:

$现有元素.replaceWith($新元素,) return $现有元素 => .replaceChild()

$新元素.replaceAll($现有元素) return $新元素

2. 删除: $元素.remove();

3. 克隆元素: $元素.clone();

### 6. 事件绑定:

DOM中: .addEventListener("事件名", 处理函数) 添加

.removeEventListener("事件名", 处理函数) 删除

坑: 如果一个处理函数可能被移除，则绑定时就要用有名称的函数。移除时，也要用函数名来移除。

jQ中: .on("事件名", 处理函数)

对21种常见的事件，提供了更简化的写法: .事件名(处理函数)

比如: .click(function(){ ... }) ...

.off("事件名", 处理函数)

事件委托/事件代理:

$(父元素).on("事件名","选择器",function(~~e~~){

//on执行了: this=e.target;

var $tar=$(this); ~~//$(e.target);~~

~~if($tar.is("选择器")){~~//on把选择器拿走自动判断去了，结果，凡是进入function中的，一定都是符合要求的

直接写正常的事件处理逻辑

~~}~~

})

两个福利: 1. on拿走了选择器，自动判断，咱们不用再写if

2. this->e.target，不用写e和e.target

总结: $有4种用途:

1. $("选择器") 创建jQuery对象，并查找DOM元素，封装进jQuery对象中

2. $(DOM元素) 直接将获得的DOM元素，包装为一个jQuery对象，可调用jQuery家的简化版函数

比如： $(this) $(e.target) $(txt)

3. $(`html片段`) 创建jQuery对象，同时用html片段创建新的元素

4.$(function(){....} ) 绑定事件，在DOM内容加载后就自动触发。

day03

鄙视题:你觉得我们公司的官网有哪些可能优化的地方:

F12->network->刷新页面

1.请求次发:>50次，算多的

可以说，减少请求次数:

1.用CSS精灵图，减少图片的张数

2.尽量减少ajax请求的次数，尽量一次获得全部元素，看个别进度条比较长的图片

可以说，降低图片质量，减小图片体积

压缩/合并js和css代码，减少代码？？？？

console.log(e.keyCode); 查询按钮的编号

1. 事件:

页面加载后，自动执行:

问题1: 如何保证无论js代码写在网页的开头还是结尾，都能正常执行呢？

解决1: 将所有的事件绑定代码都写在window.onload=function(){ 中 }

比如://当窗口的内容都加载完之后

window.onload=function(){

//才查找按钮绑定事件

$("#btn1").click(function(){

alert("疼！");

})

}

问题2: window.onload=function(){}是用赋值方式绑定事件。整个窗口只能绑定一个处理函数。如果多个js文件中都包含window.onload=function(){}，结果只有最后一个会覆盖之前所有。如何让多个js文件中的window.onload=function(){}并存？

解决: $(window).on("load",function(){

//.addEventListener

//可进一步简写为:

$(window).load(function()

})

问题3: window.onload必须等待所有网页内容（HTML, JS, CSS, 图片）都加载完才能触发，才能绑定事件。用户有可能等不及要使用元素的功能！

如何让用户不必等待css和图片，也能提前用上网页功能？

解决: jQuery:

//当 DOM树 准备就绪 就提前触发

$(document).ready(function(){

$("#btn2").click(function(){

alert("疼！");

})

})

结论: 今后绝大多数页面初始化或事件绑定操作，都应该放在DOM树加载完就提前触发。

简写:

$(function(){

$("#btn2").click(function(){

alert("疼！");

})

})

总结: 今后所有jq代码，都应该放在$(function(){ ... })

今后见到$(function(){ ... })就是"DOM树加载后就提前执行"的意思

从此$(function(){ ... })代替了匿名函数自调成为所有程序的外层容器代码。

鼠标事件:

mouseover 鼠标进入 mouseout 鼠标移出

问题: mouseover 和mouseout 默认是冒泡的。

也就是说反复进出子元素同样会反复触发父元素上的事件

且不能用e.stopPropagation()解决，因为事件绑在最外层父元素上。不可能阻止所有子元素上来的冒泡。也不可能给每个子元素都绑定事件阻止冒泡。

解决: 就希望进出子元素，不再反复触发父元素上的事件

jQuery中都是用.mouseenter()和.mouseleave() 代替mouseover和mouseout

简写: 如果同时绑定鼠标进入和鼠标移出事件时:

jQuery中可只绑定一个: .hover()

绑定一个.hover()，等效于同时绑定了mouseenter()和mouseleave()

但是hover中需要2个函数。一个给mouseenter，另一个给mouseleave。

$("#target")

.hover(//=mouseenter+mouseleave

function(){//给mouseenter

$(this).addClass("hover")

},

function(){//给mouseleave

$(this).removeClass("hover")

} )

极其特殊的简写: 如果两个处理函数刚好可以改为一致的，则只需要写一个处理函数即可。一个函数即给mouseenter，又给mouselieave

比如: $("#target")

.hover(//=mouseenter+mouseleave

function(){//即给mouseenter又给mouseleave

$(this).toggleClass("hover")

} )

模拟触发: 即使没有点击按钮，也可执行按钮上的事件处理函数

如何: $(处理函数所在的元素).trigger("事件名") trigger 触发

//可简写为: .事件名()

### 7. 动画: 2大类:

1. 简单动画: 固定的三种动画效果:

1. 显示隐藏: .show() .hide() .toggle()

默认: 代替display:block和display:none实现瞬间显示隐藏

所以，如果只是想显示隐藏，不带动画效果，show(), hide(), toggle()还是很好用的！

如果给函数添加ms数参数，就变成带动画的版本

2. 上滑下滑: .slideUp() .slideDown() .slideToggle()

3. 淡入淡出: .fadeIn() .fadeOut() .fadeToggle()

简单动画函数致命的两个问题:

1. 在jQuery源代码中写死的动画效果，无法维护。 vs css动画: 极其便于维护

2. 都是用js定时器模拟的: 效率低！ vs css动画: 由专门的绘图引擎解析，效率高！

2. 万能动画: 可对任意css属性应用过渡的动画函数

$元素.animate({

css属性: 目标值,

... : ...

},动画持续时间)

执行效果: animate自动获得当前元素状态和css属性值，再与目标值比对，执行过渡动画效果。

问题: 只支持单个数值的css属性。不支持: 颜色过渡， CSS3变换

排队和并发:

排队: 多个css属性先后依次变化:

如何: 对同一个元素，先后调用多次animate，就实现排队执行。

原理：其实，每个元素都有一个动画队列: [ ]

animate其实不是播放动画的意思，而是将现有动画定义假如到元素的动画队列中

动画何时执行取决于队列中前边有没有其他未执行完的动画。只有之前的动画都执行完，当前动画才自动开始

并发: 多个css属性同时变化

如何: 放在一个animate中的多个css属性默认并发变化

动画结束后自动执行:

其实每个动画函数都有最后一个形参，是一个回调函数。这个回调函数会在动画播放完成后自动被调用！

停止动画: .stop()

坑: 只能停止队列中当前正在播放的一个动画。队列中后续动画依然继续执行。

解决: .stop(true) 清空队列

选择器: :animated 专门用于判断或选取正在播放动画的元素

### 8. 类数组对象操作:

问题: jQuery返回的都是类数组对象，经常需要查找/遍历。但是在原生js中，类数组对象很受歧视。因为不是纯正的数组类型，所以数组家的函数，类数组对象都不能用。

解决: jQuery模仿原生js中的indexOf和forEach，重新定义了两个index()和each()函数，专门给类数组对象用。

包括:

1. 遍历jQuery对象中封装的每个DOM元素:

$(...).each() 仿原生js中的arr.forEach()

何时: 只要遍历jQuery结果中的每个DOM元素时

如何:$(...).each(function(i, elem){

//i 正在遍历的当前位置

//elem 正在遍历的DOM元素

//想用elem调简化版函数，必须$(elem)一下

})

2. 查找: 在jQuery对象中查找一个DOM元素出现的位置

var i=$(全部元素).index(要找的元素)

比如: 在所有li中找一个li的位置 var i=$("ul>li").index($li)

简写: 如果在一个父元素内，找某个子元素的位置

可简写为: $子元素.index() ——默认就只在当前父元素内找

### 9. 添加自定义函数:

何时: jQuery提供的函数不够用时

如何: 其实就是在jQuery的原型对象中添加一个新函数。所有jQuery类型的子对象都可共用！

### a.总结: $的原理: 共有四种用法:

1. $("选择器") 创建jQuery对象，并查找DOM元素

2. $(DOM元素) 将已经获得的DOM元素包装为jQuery对象

3. $("html片段") 创建jQuery对象，同时创建新的DOM元素

4. $(function(){ ... }) 绑定事件，在DOM内容加载后就自动触发

### b.鄙视题: 你觉得我们公司的官网有哪些可以优化的地方:

F12->network->刷新页面

看请求次数: >50次，算多的

可以说，减少请求次数:

1. 用css精灵图，减少图片的张数

2. 尽量减少ajax请求的次数，尽量一次获得全部元素

看个别进度条比较长的文件

可以说，降低图片质量，减小图片体积

压缩/合并js和css代码，减少代码的文件个数和体积

### c.常用事件列表:

click 单击

dblclick 双击

change 选中项改变 常用于select元素

blur 失去焦点

focus 获得焦点

keydown 键盘按下未抬起

keyup 键盘按键按下后抬起

mousedown 鼠标按键按下

mouseenter 鼠标进入元素 代替 DOM中 mouseover

mouseleave 鼠标移出元素 代替 DOM 中mouseout

mousemove 鼠标移动事件

mouseout DOM中的鼠标移出事件

mouseover DOM中的鼠标移入事件

mouseup 鼠标按键抬起

scroll 滚动条发生滚动时触发

input 文本框中输入文字改变时触发

day04 2019/6/21

### 1. 封装自定义插件:

组件/插件: 拥有独立的HTML，css，js和数据的可重用的页面区域

为什么: 重用!

何时: 如果页面中一个功能区域可能被反复使用，都要将其封装为一个插件。

如何: 2种情况:

1. jQuery官方插件: jQueryUI

官网下载: jquery-ui.css images/ jquery-ui.js

在页面中引入:

<link rel="stylesheet" href="css/jquery-ui.css"

<!--先引入jquery，再引入jquery-ui-->

顺序如下:

<script src="js/jquery-1.11.3.js"

<script src="js/jquery-ui.js"

<script>自定义代码</script>

html中: 按照插件的要求，定义HTML内容结构，不需要加任何class

js: 查找插件的HTML父元素，对父元素调用一次插件函数即可自动添加样式和行为。

$("my-accordion").accordion();

原理: 侵入性: 插件函数根据自己的需要，自动为元素添加class和行为

好处: 简单 不好: 不可维护

2. 封装自定义插件:

前提: 已经用html，css和js实现了插件的样式和功能。

如何:

1. 将插件相关的css，剪切到一个独立的css文件中保存。

强调: 如何避免插件/组件间class冲突:

一个组件内的选择器，都要以插件的父元素的class作为查找的开头/起点。

2. 定义独立的js文件：

在jquery的原型对象中添加插件函数

1. 添加样式：侵入: 悄悄自动添加

2. 添加行为：使用自定义插件: 同jQueryUI的用法。

### 2. ajax:

jQuery中已经封装好了现成的ajax函数，我们可直接调用

$.ajax({

url:"url路径",

type: "get/post",

data:"uname=dingding&upwd=123456",

//如果本次请求没有参数，可省略data

dataType: "json",

//按照国际标准所有服务端接口，都应该返回json格式的字符串。但是也不排除例外

//如果返回的不是json，就不要写dataType:"json"，会出错。

//因为dataType:"json"是自动执行JSON.parse()的意思。如果服务端返回的字符串不是json格式，则JSON.parse()会报错！

//success函数是一个回调函数，会在请求成功结束后，自动调用。参数result，会自动获得服务端返回的值。而且已经被JSON.parse()，范围为js对象了。

success:function(result){

... ...

}

})

//ES 6 promise中

.then(function(result){

//result 获得的就是服务端返回的数据

})

强调: success: 和 .then() 二选一即可！

### 3. \*\*\*\*\*跨域:

什么是: 一个域名下的网站，向另一个域名下的服务端发送请求。

比如:

网页 服务端

http://www.a.com -------> http://www.b.com

http://a.tedu.com --------> http://b.tedu.com

http://localhost:8080 --------> http://localhost:3000

http://www.12306.cn(:80) --------> https://www.12306.cn(:443)

http://127.0.0.1:3000 (IP) --------> http://localhost:3000(域名)

http://tedu.cn/index.html(相对路径)--------> http://tedu.cn/index/login(相对路径) 不算跨域！

问题: 浏览器禁止ajax请求跨域获取数据

原理:

1. 浏览器其实是发出了请求的。

2. 请求也确实成功了，返回的结果，也拿回来了

3. 因为响应结果中带着响应头，记录着数据来源的IP地址。所以，当浏览器查验服务端返回的数据时，发现响应头中的IP不是当前网页的IP，不让用！

报错: Access to XMLHttpRequest at 'http://localhost:3000/' from origin 'http://127.0.0.1:5500' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

CORS: Cross Origin Resources Sharing cross跨 origin源头resources资源 sharing共享 策略 也可叫同源、跨域。

浏览器禁止与当前网页不同来源的数据被使用

解决: 2种:

1. 简单: 单靠 服务端配置响应头

在服务端程序中，在发送消息之前，手动修改响应头:

node.js ：

res.writeHead(200,{

"Access-Control-Allow-Origin":"http://客户端网页地址栏中的地址"

})

2. 公司: JSONP —— 同时修改客户端和服务端程序才能可支持。

前后端分离: 见视频。

## 12.VUE

正课: 1. 什么是Vue 2. 如何使用Vue 3. MVVM 4. 绑定语法 5. 指令

### 1. 什么是Vue:

渐进式的基于MVVM的 执行数据操作的 纯前端js框架

渐进式: 可以逐步在项目中使用Vue框架

可以和现有传统技术很好的结合

全家桶: 如果要使用一种框架，就只能使用框架的所有技术。不能和其它技术混搭。

基于MVVM？

执行数据操作的: 终极简化对网页执行的数据操作: 增删改

纯前端js框架: 不要nodejs，仅靠浏览器就可运行项目

### 2. 如何使用:

官网: cn.vuejs.org

下载: 版本: 2.6 2种:

1. 下载vue.js文件，引入网页中——前三天

开发版(未压缩): 体积大，可读性好，带错误提示

生产版(压缩): 体积小，可读性差，去掉了所有错误提示

2. 脚手架代码——最后一天做项目

如何引入: 1. <script src=" js/vue.js"> 2. <script>自定义代码</script>

如何使用: 3大步:

1. 定义页面的HTML内容:

要求: 必须包含在一个<div id="app"></div>内

必须用{{变量}}标记处，要使用数据的位置

如有事件处理函数，就用@click绑定事件处理函数

比如: <div id="app">

<button @click="add">click me({{n}})</button>

2. 在自定义程序中先定义页面所需的所有数据。

要求: 必须包含在一个data:{}对象中

比如: 页面上有一个{{n}}，表示一处n需要发生变化，所以var data={ n:0; }

3. 创建new Vue()对象示例，将数据和页面元素绑定起来

var vm=new Vue({

el:"#app",

data: data,

//结果: data中的变量是什么值，页面中就显示什么值

//data中的变量被改成什么值，页面中就自动变成什么值

//如果需要事件处理函数，都要定义在methods:{}中

methods:{

add:function(){//当单击按钮时，自动触发add函数

this.n++; //修改data中的n+1，页面上的n自动跟着变化

}

}

})

事件: 2步:

1. 在页面的html元素上用@click="add" 绑定事件处理函数

2. 事件处理函数，不要定义在new Vue的外边

应该定义在new Vue中专门的methods属性内:

new Vue({

el:"#app",

data: data,

methods:{

add:function(){

this.变量++

}

}

})

总结: 页面需要什么，Vue就定义什么

比如: <button @click="add">click me(0)</button>

new Vue({ ↑ ↑

add:function(){} {{n}}

<button @click="sub">- <span>1 <button @click="add">+

new Vue({ ↑ ↑ ↑

sub:function(){} {{n}} add:function(){}

### 3. MVVM模式:

传统的DOM:

HTML: 只负责静态内容，不会自动变化

CSS: 只负责静态样式，不会自动变化

JS: 即要负责内容的变化，又要负责样式的变化

问题: 1. 步骤繁琐且重复: 查找，绑定，遍历，替换，拼接字符串

2. 不便于维护：jQuery中，如果页面结构或内容发生变化，则js中选择器和HTML片段都要跟着修改

现代框架: MVVM模式

1. 界面/视图View: 包括静态的HTML+CSS

2. 数据模型Model: 页面中所需的所有数据的整体

3. 控制器/视图模型ViewModel: 自动将数据模型Model中的变量，填充到界面中所需的位置

总结: 什么是M V VM模式: 页面需要什么，模型就定义什么。

视图模型会自动将模型中的数据填充到页面中。且模型数据发生变化时，视图模型会自动更新页面。

优点: 1. 没有任何重复的代码

2. 界面和模型松耦合，界面变化，不需要修改模型，甚至不需要修改ViewModel控制器。——及其便于维护

ViewMode的原理:

当new Vue将模型对象和页面元素绑定在一起时，内部自动构建两大子系统:

1. 响应系统:

自动将data中的每个变量变成访问器属性get()和set()

今后，所有对变量的读写，自动都被访问器接管，由访问器属性代为执行。

比如：修改变量的值时，自动调用变量的set方法，修改实际的变量值。

而每个变量的set方法，当变量发生改变时，都会自动发送通知：xxx变量的值变成xxx了

通知是交给另一子系统，虚拟DOM树的

2. 虚拟DOM树：

当new Vue()创建对象时，会扫描el:"#app"指向的父元素div，及其子元素。然后，仅找到可能发生变化的元素和属性，保存在一个简化版的虚拟DOM树结构中：

比如：

var virtualDOM={

element: div, //<div id="#app"

id:"#app",

children:[

{

element: button, //<button>click me({{n}})

innerHTML: "click me {{n}}",

@click:"add"

},

//其它可能变化的元素

]

}

什么是虚拟DOM树: 仅保存可能变化的元素的简化版树结构对象，同时预先封装了要对元素执行的DOM操作。

当响应系统发来某变量被修改的通知后，虚拟DOM树先遍历自己内部的元素，找到受这个被修改变量影响的元素。然后自动调用预先定义好的DOM操作，更新实际DOM树上的元素。其余未受变量影响的元素，保持不变。

虚拟DOM树的优点:

1. 仅包含可能变化的元素，内容精简，便于快速遍历，查找变化的内容。

2. 封装了DOM操作，自动执行DOM操作，无需开发人员重复编码。

this: 凡是进入new Vue中的，无论是data中的变量，还是methods中的方法，都被打散，直接隶属于new Vue，变为相邻的平级成员。所以，在方法中想访问data中的数据，必须加this.

### 4. 绑定语法:

学名: 插值语法 Interpolation == {{}}

什么是: 专门用于将一个变量的值自动插入到页面的指定位置==让HTML可以自动找到程序中的变量的特殊语法

为什么: 因为传统的HTML是静态的，缺少动态变化的能力。导致js当中要想操作HTML，需要大量重复的代码。

何时: 凡是页面上可能发生变化的地方，都用{{}}绑定语法。

如何: 2步:

1. 先找页面中所有可能发生变化的地方有几处

2. 再在模型数据中定义相同数量的变量:

new Vue({

el:"#app",

data:{

变量名:值,

... : ...,

}

})

强调: HTML中有几处变化，data对象中就要有几个变量与之对应。

3. 在HTML中，可能发生变化的位置用绑定语法定义变量: {{变量或表达式}}

强调: 其实{{}}的用法和模板字符串中${}的用法完全一样！

能写：变量, 算术计算，关系/逻辑运算，函数调用, 访问数组元素, 三目——凡是有返回值的js表达式都能

不能: if else while for——都是程序结构，没有返回值！

结果: 运行时，HTML中的所有{{}}会自动去data中找同名的变量使用。且内存中的data中的变量值发生变化，HTML中的{{}}的值自动变化！用户最终看到的是{{}}中的变量或者表达式计算后的值！而看不见双花括号——节省了大量重复的查找和修改操作。——多亏了MVVM中的ViewModel中的两大子系统: 响应系统和虚拟DOM树。

### 5. 指令：directive

什么是: 为HTML添加更多新功能的Vue预置的自定义属性

为什么: 因为原来HTML缺少程序必须的功能: 判断/分支结构，循环......功能。所以只能依靠js中反复查找，反复修改来控制HTML元素的内容和状态。

一个程序基本的功能:

变量:

判断:

循环:

何时: 只要想在html中使用类似变量，判断，循环等程序的功能时，就用指令:

Vue中共定义了13种指令:

1. 绑定发展：v-bind:

什么是: 专门用于绑定属性值的指令

何时: 只要属性值需要根据变量动态变化时，就要用v-bind或:简写

为什么: {{}}只能用于绑定元素内容(innerHTML)中的值,不能用于动态绑定属性值

如何: <img v-bind:src="pm25<100?'img/1.png':

pm25<200?'img/2.png':

pm25<300?'img/3.png':

'img/4.png'">

去{{}}换v-bind:

其实可简写: <img ~~v-bind~~:src="...

去{{}}换:

执行时: Vue会拿着data中的pm25变量，到有:的属性中执行三目运算，选择其中一个符合条件的字符串，作为替换src属性值的路径。

2. v-if v-else-if v-else

2. 控制元素的显示隐藏:

1. 控制一个元素的显示隐藏: <ANY v-show="判断条件"

只要条件满足，就显示元素。一旦条件不再满足，就隐藏元素！

2. 控制多个元素，多选一显示:

<ANY v-if="判断条件" 单用时

v-if从现象上看，和v-show是完全一样的

但是原理不一样

鄙视: v-show vs v-if的差别:

v-show 用display:none 控制显示隐藏

v-if 用删除节点的方式，控制显示隐藏

如果一个元素频繁需要显示隐藏,v-show的效率高！

和v-else-if v-else 配合使用:

<元素1 v-if="条件1"  
 <元素2 v-else-if="条件2"

... ...

<元素n v-else

强调: v-if和v-else-if和v-else之间禁止插入一切其他元素。必须连续写！

原理: Vue会自动判断每个条件：

哪个条件符合，就只显示哪个条件的元素。其余元素都不显示。

如果前一个条件已经满足，则后续判断都不再执行。所以，不可能同时显示多个元素。

如果所有条件都不满足，会显示v-else的元素

==

原理: 其实所有带v-if v-else-if v-else的元素都没有在页面上。而是根据条件，动态决定临时添加哪个元素到页面——靠添加删除DOM元素的方式控制元素的显示

强调: 之间不能插入其它无关的元素

3. v-show

根据条件决定当前一个元素是显示还是隐藏

如何: <ANY v-show="条件">

当条件满足时，当前元素就显示，当条件不满足时，当前元素就不显示。

比如: <button v-show="i<count" @click="next">下一页</button>

问题: 如果用在多个元素控制显示隐藏，则必须把条件反复写在每个元素上

vs v-if:

v-show采用display:none/block方式控制显示隐藏，不修改DOM树上的节点——效率高

v-if 采用动态添加删除元素的方式控制显示隐藏——效率低

3. 根据数组反复生成多个相同结构的HTML元素:

其实就是为HTML添加循环功能:

如何:

1. data中必须先定义一个可遍历的数组

2. 在HTML中使用v-for遍历数组，反复生成多个相同结构的元素，并动态绑定元素的内容。

语法: <要反复生成的元素 v-for="(elem,i) of 数组" :key="i"

强调: 特例: v-for中的循环变量可被v-for自己或子元素用于绑定！

4. v-for:

遍历数组中每个元素，每遍历一个数组元素就自动创建一个HTML元素，并且在HTML元素中还可绑定遍历出的数组元素内容

何时: 根据数组反复生成多个相同结构的元素时

如何:

1. 只需要写一个要生成的元素作为模板

2. 在要重复生成的元素上加v-for

比如:

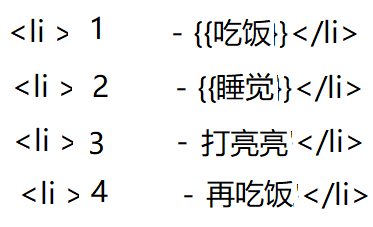
data中: tasks:["吃饭","睡觉","打亮亮","再吃饭"]

<ul>

<!--遍历tasks数组中的每个元素，将每个元素的值临时保存在变量task中，每遍历一个tasks数组中的元素，就创建一个相同结构的li，并动态绑定循环变量的值到元素内容中-->

<li v-for="(task,i) of tasks">{{i+1}} - {{task}}</li>

<!--结果-->



</ul>

5. 事件绑定: v-on:click="处理函数"

简写：@click="处理函数"

强调: 1. 处理函数必须写在new Vue中的methods中

2. 处理函数中的this不再指向当前按钮对象，而是指整个vue对象

3. 处理函数中要操作data中的变量必须加this.

==

4. 事件绑定: v-on:事件名="处理函数"

可简写为: @事件名="处理函数"

强调:

1. 处理函数必须定义在:

new Vue({

el:"#app",

data:{ ... },

methods:{

处理函数(){

this.data中变量

}

}

})中的methods:{}结构中

2. 其实可以传参: @事件名="处理函数(实参值)"

3. 也可以用事件对象e：用法和DOM中完全一样

获得e: methods:{

事件处理函数(e){ ... }

}

后续:

获得目标元素，实现事件委托： e.target

取消冒泡: e.stopPropagation()

阻止默认行为: e.preventDefault();

键盘事件中获得键盘号: e.keyCode

获得鼠标坐标位置: e.screenX, e.screenY e.clientX, e.clientY e.offsetX, e.offsetY

6. v-html v-text: 代替{{}}绑定元素的内容

{{}}的问题: 2个

1. 始终保持变量中的字符串原始样子输出

即使绑定HTML片段，也只能原样显示

解决: 用v-html代替{{}}

2. 如果new Vue加载慢时，客户会短暂看到{{}}

解决: 用v-text代替{{}}

因为v-text是属性，即使慢，暂时绑定不出来，页面上也不会出现绑定语法。

缺点: 对字符串拼接和过滤器支持不好

==

避免用户短暂看到{{}}

问题: 如果new Vue加载慢，则可能短暂看到{{}}

解决: v-cloak 可让元素在new Vue加载之前暂时隐藏。当new Vue加载完成后，会自动查找页面中的所有v-cloak斗篷删除该属性，让元素显示出来。

强调: v-cloak没有属性值！

问题: v-cloak空有属性名，没有配套的样式

解决: 只能自己手写！用属性选择器: [v-cloak]{display:none}

强调: v-cloak不能改名！改名了，vue就找不到了！

其实，除了v-cloak外，还可用v-text代替{{}}绑定元素内容，避免短暂看到{{}}

如何:

<ANY v-text="`js模板字符串语法，用${变量}动态生成内容`"

比如:

<h1 v-text="`姓名: ${uname}`"></h1>

<h2 v-text="`性别: ${sex==1?'男':'女'}`"></h2>

原理: v-text在未绑定时，是一个浏览器不认识的属性，所以，显示不出来。直到new Vue加载完，认出v-text，才用V-text的内容代替元素的innerHTML内容。

v-cloak vs v-text:

v-cloak: 需要额外添加css样式，且需要选择放在哪个元素上，隐藏什么范围内的元素。整个网页都隐藏，用户体验还是不好的。

v-text: 仅对当前元素有效，不会影响其它元素的显示不显示。用户体验比v-cloak要好。但是使用反引号的语法，很晦涩。

6. 绑定HTML片段:

问题: 使用{{}}绑定HTML片段，VUE不会解析HTML片段为网页内容。而是原样显示HTML代码

解决: 今后，只要绑定一段HTML片段，必须用v-html。

如何: <ANY v-html="变量"></ANY>

7. v-cloak: cloak隐身斗篷

专门用于在new Vue加载慢时，暂时隐藏元素

如何: 2步:

1. 在要隐藏的元素上添加v-cloak属性，不用给值

2. 手动在网页顶部提前定义[v-cloak]{display:none}

原理: 当new Vue加载完之后，自动查找所有v-cloak属性，并移除。

8. v-once:

仅在页面加载时绑定一次变量值。之后，即使变量发生变化，也不更新页面

如何: <ANY v-once>

原理: 所有要更新的元素都保存在虚拟DOM树中

标有v-once的元素，在首次绑定后，就从虚拟DOM树中移除。下次即使变量再变化，扫描虚拟DOM树，找不到标有v-once的元素了。

==

只在页面加载之初，绑定一次。之后及时数据变化，也不反复更改页面：v-once

强调: v-once没有任何属性值，写在元素中，就起作用

原理:

在构建虚拟DOM树时，会扫描到v-once的元素，所以首次绑定，能绑定v-once的元素的内容。

首次绑定后，v-once的元素会从虚拟DOM树中移除。从此，再出发变量改变时，不会再修改v-once的元素。

9. v-pre:

万一内容的正文中，有{{}}，但是不是用于绑定的，会出错。

解决: v-pre 可阻止内容中的{{}}被编译，而是让{{}}原样显示。

如何: <ANY v-pre>

总结:

1. 元素内容随变量自动变化: {{}}

用 {{变量或表达式}} 绑定

也可用v-text="`变量或表达式`"

2. 如果元素的属性值要变化: 用 :属性名="变量或表达式" 绑定

3. 只有一个元素控制显示隐藏: v-show="条件"

4. 多个元素选其一显示: v-if="条件" v-else-if="条件" v-else

5. 反复生成多个相同结构的元素时:

v-for="(elem, i ) of 数组" :key="i"

6. 只要绑定事件处理函数都用: @事件名="处理函数"

7. 只要不希望用户短暂看到{{}}语法: v-cloak 或 v-text

代替{{}}绑定普通元素内容时: v-text ——避免短暂看到{{}}

希望所有{{}}的元素，在new Vue加载完之前暂时隐藏: v-cloak

8. 只要绑定的内容是HTML片段时: v-html

9. 只要希望只在页面加载之初绑定一次，之后不再变化，就用v-once.

10. 如果内容中包含不想被vue编译的{{}}正文，可用v-pre阻止VUE编译元素的内容。

正课:

1. 双向绑定:

2. 计算属性:

3. 绑定样式:

4. 自定义指令:

5. 过滤器:

6. axios:

7. new Vue的生命周期

### 1. 双向绑定:

什么是单向绑定: 只能将程序中的修改更新到页面中。但是无法将页面中的更改，更新回程序中。

只能从Model -> View 不能从View -> Model

比如: {{}} v-text : 等等 都是单向绑定

比如: <input :value="变量"

结果: 变量中的值可正常显示在文本框中，但是文本框内容修改，变量的值不会改变。

什么是双向绑定: 即能将程序中的修改更新到页面中，又能将界面中的修改，反向更新会程序中

何时: 只要希望通过表单元素的修改，来更改程序的内容时。——只要绑定表单元素，都用双向绑定。

如何：不要用: ， 要用v-model:属性名="变量"

原理: v-model其实就是自动为表单元素绑定了oninput或onchange事件。在这些内容更改事件中，自动修改当前Vue对象中的模型变量。

普通绑定为什么不能更新程序中的数据：因为没有自动添加事件，导致界面上更改，Vue框架不知道，自然就无法更近模型变量。

其它表单元素的双向绑定:

1. 单选按钮radio: 要绑定checked

不绑定value，因为radio中的value是写死的，固定不变的。等着被选择。

绑定checked: 因为是否选中radio，取决于checked属性的值。用户更改选中项，改的也是checked属性值。——radio只能绑定v-model:checked="变量"

绑定时:

<input type="radio" name="sex" value="1" v-model:checked="变量"

<input type="radio" name="sex" value="0" v-model:checked="变量" ~~"变量==值"~~

小脾气: v-model在radio中：

从程序绑定到页面时: 会自动用checked后绑定的变量值和当前radio写死的value值做比较，返回bool值决定当前radio是否选中。时不用自己在checked后写完整条件！只写一个变量名即可！

改变选中状态时，v-model会将选中的radio的写死的value值更新回程序中绑定的变量上

2. 单个checkbox绑定: 也绑定checked属性

<input type="checkbox" v-model:checked="isAgree"

3. select元素绑定: 绑定的是select元素整体的value属性

因为，无论选择了哪一个option，当前选中项的值都 变成select元素的value属性值。

如何: <select v-model:value="变量">  
 <option value="值1">

<option value="值2">

... ...

原理:当从Model->View绑定时: 用变量的值和每个option的value做比较，按个相等，就选中哪一个  
 当修改select下的选中项后，将当前选中的option的value值更新回变量去

其实，以上绑定都有一个共同的简写: 只写v-model即可，可省略:value :checked等。v-model会个识别元素中起决定作用的可变的属性，自动绑定。

watch监控机制:

什么是: Vue中的watch监控机制可监控vue所有模型变量的变化。只要模型变量变化，就自动触发一个同名的函数。在函数中可调试变量的值或执行实时操作

何时: 希望只要变量的值变化，就立刻做一件事儿时

如何:

new Vue({

el:"#app", //找要监控的整个页面空间

data:{

变量1:值1

}, //所有页面需要的模型变量

methods:{ ... }, //所有事件处理函数和一些自定义函数

created(){ ... }, //当new Vue()对象加载完自动执行

watch:{ //定义监控变量的函数

变量1(){//必须同名，只有同名的函数，才能监视同名的变量

执行一切js或vue的操作

}

}

})

### 2. 计算属性:

什么是: 不实际属性值，绑定时，动态通过其他现有属性，计算出自己的属性值。

何时: 页面上需要一个值，但是，这个值数据库中没有直接保存，而是需要动态计算出来。

如何: new Vue({

el:"#app",

data:{ },

methods:{ },

created(){ },

watch:{ },

computed:{

属性名(){

//通过其他变量计算获得最后结果

return 结果

}

}

})

使用计算属性：计算属性虽然定义成函数，但是，用法和普通模型变量完全一样！不要加()

特点:

1. 只要依赖的其它变量变化，计算属性都会自动重新计算

2. 一个属性如果多个位置使用，其实只计算一次。

原理: 计算属性的值，每计算一次，结果都会被Vue框架缓存起来，反复使用。除非依赖的变量发生改变时，才重新计算，然后再重新缓存，反复使用。

vs 函数: 也可以实现计算的效果

问题: 不会被缓存！每用一次就重复计算一次！

总结: 如果只关心计算结果时，首选计算属性。如果不关心结果，甚至没有返回值，只关心执行的过程时，首选函数。

### 3. 绑定样式: 2种:

1. 动态绑定单个css属性: 2种:

1. 用字符串方式，绑定整个style属性：

<ANY :style="变量"  
 data:{

变量:"top:50px; left:50px"

}

缺点: 在字符串中不便于修改某一个css属性

2. 都用对象方式，绑定style:

<ANY :style="变量"

data:{

变量:{

top:"50px", //必须px

left:"50px" //必须px

}

//绑定时，被自动翻译为字符串:

//"top:50px; left:50px"

}

强调: 固定的style和动态绑定的style可同时存在

比如：

<div style="position:fixed" :style="divStyle">

data:{

divStyle:{

top:"50px", //必须px

left:"50px" //必须px

}

}

运行时，结果: 动态绑定的style会和写死的style合并为一个style:

<div style="position:fixed; top:50px; left:50px"

2. 动态绑定class 2种:

1. 将整个class属性当做一个字符串绑定

问题: 不便于控制一个class的启用或禁用

2. 今后也要使用对象格式绑定:

<b class="固定不变的class" :class="变量名"

data:{

变量名:{

class1: true/false,

class2:true/false,

... ...

}

}

运行时:

1. 将变量中值为true的class转为字符串:

比如: { down:true, fade:true }

-> "down fade"

{down:false, fade:true}

-> "fade"

2. 用翻译好的class字符串和元素上规定的class=""，合并为一个class

如果想用程序切换使用/不使用某个样式类，只要修改类名对应的true/false就行

值为true，就是应用该样式类

值为false，就是不应用该样式类

### 4. 自定义指令:

何时: 当希望元素自动执行一件事，但是vue中的指令不够用，就可以自定义指令。

如何:

1. 定义一个新的自定义指令:

Vue.directive("指令名",{//指令名不应该带v-前缀

//会在当前元素加载后，自动执行

inserted(elem){

//elem: 接住的是当前指令所在的DOM元素

//可对elem执行任何原生DOM的函数

}

})

强调: 应该定义在不属于任何new Vue()的外部全局

将来所有new Vue()都可以共用这个自定义指令。

2. 使用指令: <ANY v-指令名

但是，使用指令时，必须加v-前缀

原理: 当<ANY>这个元素在HTML中加载完之后，自动去调用指令中携带的inserted()函数

### 5. 过滤器:

什么是: 将不能直接使用的变量值，过滤成可以给人看的新值的函数

何时: 只要变量的原始值不是想要的，都可用过滤器过滤成想要的值给人看。

如何:

1. 定义过滤器:

Vue.filter("过滤器名称",function(oldVal){

return 新值

})

2. 使用过滤器:

在绑定语法中都可用"|"将过滤器连接在变量之后

运行时: 原始变量值->过滤器->oldVal参数->return过滤后的新值->给人看

过滤器参数:

何时: 当根据不同值，选择过滤成不同的新结果时

如何: 2步:

1. 定义过滤器时添加形参: function(oldVal,lang="cn"){

2. 使用过滤器时传入实参: {{sex2|sexFilter("en")}}

强调: 定义时添加过滤器形参，从第二个形参开始添加。但是使用过滤器时，自定义实参值，却从第一个实参传入。但是还不会出错！因为过滤器函数在调用时，自动会将原始值作为第一个实参传入。我们添加的自定义实参，只能排在第二。

其实程序中的过滤器也可以用|串联多个:

{{变量|过滤器1|过滤器2|... }}

坑：定义后一个过滤器时，拿到的值不是原始值，而是上一个过滤过滤后的中间产物。

### 6. axios

什么是: 专门发送ajax请求的基于Promise的函数库

为什么: 总共有5种方法

1. 然哥四步/五步:

2. 可以自己封装

3. jQuery库中的$.ajax()

将来在Vue项目中，几乎不会用jQuery。

如果仅仅为了发ajax请求而引入jQuery函数库，特别小题大做。

4. Vue官方有一个vue.resource组件，但已停止维护。

5. axios: Vue官方推荐的最好的ajax函数库。

何时: 今后只要在任何位置想发ajax请求，都可用axios

如何:

1. 引入axios.js

2. 调用axios.get()函数发送请求

axios.get("url",{

params:{//相当于$.ajax中的data:{ ... }

参数1:值1,

... : ...

}

//运行时params中的参数会被翻译为字符串，拼接到url的结尾:

"url?参数1=值1&参数2=值2&..."

}).then( result=> {

为this保持和外部VUe一致，必须用箭头函数

result返回的不只是服务端响应结果

真正的响应结果包含在result.data中

})

### 7. new Vue的生命周期: 面试会考

什么是: 一个new Vue()的加载过程:

何时: 如果希望在加载过程某个阶段，自动执行一项任务时。

如何: new Vue()的声明周期包括4个阶段，8个函数

钩子函数——其实就是在不同阶段自动执行的回调函数

前: beforeCreate(){ ... }

1. 创建(create)阶段: 创建new Vue()对象和data()对象

已经有data对象了，但是没有虚拟DOM树

可以发送ajax请求

因为没有虚拟DOM树，所以暂时不能用DOM操作

后: created(){ ... }

前: beforeMount() { ... }

2. 挂载(mount)阶段: 创建虚拟DOM树，将data中的变量值开始向DOM树上绑定

即有data对象，又有虚拟DOM树

即可发送ajax请求，又可执行DOM操作

后: mounted(){ ... }

//后两个阶段不是必须:

前: beforeUpdate(){ ... }

3. 更新(update)阶段: 当data中的变量值被改变时才触发

后: updated(){ ...}

前: beforeDestroy(){ ... }

4. 销毁(destroy)阶段: 当调用专门的$destroy函数销毁一个组件时，才触发

后: destroyed(){ ... }

# === END ===

重要的网站:

Iconfont 阿里马马矢量图标，选择图标。 现在它已经替代了雪碧图。

jquery-ui.css jquery官方插件下载网址

images/ jquer-ui.js

http:v4bootcss.com

https://cn.vuejs.org vue框架的官网  
案例目录

day01 2019/6/17

1.用户名和密码框，输入内容的设置 day01 14\_html\_val.html

2.点击“我同意本站的使用条款” 解锁input输入功能 day01 13\_form state selector.html

3.四种隐藏方式 day01 12\_visibility filter.html

day02 2019/6/18

1.点击小图片，下方my-big中显示大图片 day02 3\_attr2.html

2.双态按钮 day02 4\_class.html

3.使用属性选择器实现标签页头的切换 day02 7\_tabs.html 不完善，用不了

4.对话框的淡入淡出,下沉，卷轴 day02 05\_class2.html 有视频

5. 添加随机颜色的方块，并且可以删除 day02 9\_append.html

6.战机替换 day02 10\_replace\_clone.html

7.子射子弹/跟踪导弹的简写 day02 12\_bind\_unbind.html 有视频

8.单击图片，切换下一张 day02 2\_attr.thml

9实现点击目标元素，改变其它元素的样式 day02 6\_traverse.html

day03 2019/6/19

1.加载完成后，绑定事件,和页面优化 day02 14\_ready.html

2.鼠标进入移出，绑定事件 day 02 15\_mouse.html

3.鼠标进入移出，绑定事件的简写 day02 16\_hover.html

4.百度查找功能，三种方法的查询，回车，点击，输入 day02 17\_trigger.html 文件报错，id忘记写#了

5.品牌的精简显示，与全部显示 day03 2\_show\_hide2.html

6.手风琴效果 day03 3\_accordion.html 有视频 accordion-7

7.星星动画移动 day03 5\_stars.html

8.星星闪三次后消失 day03 6\_star.html

9. 请给每个不足60分的成绩+10分，并将超过90分的成绩用绿色背景标识出来 day03 9\_each.html

10.打分程序 day03 10\_index.html 选中的元素及选中元素之前的元素都是红色，之后的颜色都是绿色

11.自定义jQuery函数对象 day03 11\_sum.html 求和

day04 2019/6/20

1.用插件函数做手风琴效果 day03 12\_widget\_accordion.html

2.自己封闭插件做手风琴效果 day03 14\_accordion.html 有视频 css + js + html

英文翻译

constructor 构造函数 prototype 原型 milliseconds 毫秒 random 随机 round 圆，循环

sqrt 平方根 trigger触发 cross跨 origin源头 resources资源 haring共享 question问题