# 第一章 项目导入

## 一、windows常用快捷键

|  |  |
| --- | --- |
| win+e | 打开文件资源管理器，即我的电脑 |
| win+d | 显示/隐藏桌面 |
| win+r | 打开运行窗口 |
| cmd | 打开命令行；calc打开计算器；mspaint打开画图 |
| alt+tab | 多窗口切换（向右） |
| alt+shift+tab | 多窗口切换（向左） |
| alt+F4 | 关闭当前窗口 |
| backspace | 回退到上一层文件夹 |

【练习】在d盘下创建一个目录叫做public，在public下创建一个2.html（文件类型被隐藏时，修改后缀名没有作用）

【练习】在c:/xampp/htdocs目录下，创建目录叫做day02，进入该目录，创建一个文件demo.html。全程不能用鼠标

## 二、软件工程

### 2.1 概述

IBM -> DOS(Bill Gates) -> Microsoft -> windows

软件工程包括：

①软件过程

②软件测试

③软件度量

④软件质量保证

⑤软件标准

### 2.2 软件开发流程

软件的生命周期/软件过程

#### 2.2.1 软件的定义期

①可行性研究阶段 —— 《可行性研究报告》、《项目开发计划》

分析软件各方面的可行性（IF CAN DO），例如：技术(选择一个可以实现该功能的技术)、人员、资金、时间、设备、政策法规、风俗、回报率等。

②需求分析阶段 —— 《软件需求说明书》

分析软件需要实现的功能和需要具备的性能（WHAT TO DO），例如：软件必须完成的主要/次要功能点，软件的安全性、可靠性、精度等性能需求。

#### 2.2.2 软件的开发期

③概要设计阶段 —— 架构师《概要设计说明》

从总体上对项目进行设计（HOW TO DO 1.0），例如：技术选型（软件、硬件、网络）、功能模块划分、数据结构设计、制定开发和测试计划。

④详细设计阶段

在概要设计的基础上，由各模块负责人拟定可供实施的开发指导（HOW TO DO 2.0），主要需要完成：模块的输入输出、模块内部的数据结构、模块内部的处理算法、模块内部的状态转换过程。

⑤编码实现阶段

由前后端工程师协同完成项目各部分的编码工作，例如：UI/UE设计师、前端工程师、后台工程师

⑥项目测试阶段

在软件开发完毕之后，制定详细的测试计划并严格按照测试计划对软件进行错误检查，主要分为三个阶段：单元测试、组装测试及系统测试。测试方法主要有白盒测试和黑盒测试两种。

软件测试工程师 Ada -> bug -> debug

#### 2.2.3 软件的维护期

⑦项目部署阶段

软件开发完毕并完成严格且详细的测试之后，需要从开发模式(development mode)部署到用户的生产环境即生产模式(production mode)之下。

开发模式(development mode) -> 生产模式(production mode)

⑧项目维护阶段

这一阶段是软件开发周期中持续时间最长的一个阶段，主要分为纠错性维护和改进性维护。

### 2.3 开发实例——学子商城功能需求

前台：www.codeboy.com

后台：www.codeboy.com/admin/login.html

前台子系统

商品模块、用户模块、购物车模块

后台子系统

商品模块、用户模块、订单模块

App子系统

## 三、服务器概述

### 3.1 什么是服务器？

简单的来说，服务器就是网络环境中的高性能计算机，它响应客户端的各种请求，并为其提供相应的各种服务。设计模式分为C/S模型和B/S模型。

### 3.2 硬件服务器

指提供计算服务的设备，和普通计算机类似，具有CPU、内存、系统总栈、硬盘等。由于其需响应大量请求并提供服务，在安全性可靠性上也有一定要求，所以与通用计算机相比，其各方面性能要求较高。按照大小可以分为工作站、小型机、中型机、大型机和超级计算机等。

### 3.3 软件服务器

主要指服务器应用程序，它负责监听特定端口，并接收客户端的请求。通常将其按照提供服务的不同主要分为：DNS服务器（把域名解析为计算机的IP地址）、Web服务器，数据库服务器，FTP服务器，域名服务器等等。

### 3.4 如何访问服务器？

访问网络上任一台服务器，必须提供以下三个信息：

①访问协议：规定两者之间沟通的语言，如TCP/IP、FTP等

②主机地址：服务器的域名或IP地址

③端口号：一台计算机上采用不同端口提供不同服务，所以需要告知所需服务采用的端口号。

### 3.5 搭建web服务器

#### 3.5.1 安装xampp

下载地址：www.xampp.org，默认安装路径：c:/xampp

#### 3.5.2 启动web服务器

点击Apache后的start按钮启动服务器，看到80,443端口打开

#### 3.5.3 编写网页保存在web服务器中

保存在c:/xampp/htdocs目录下，只有保存在该目录下才能被访问到

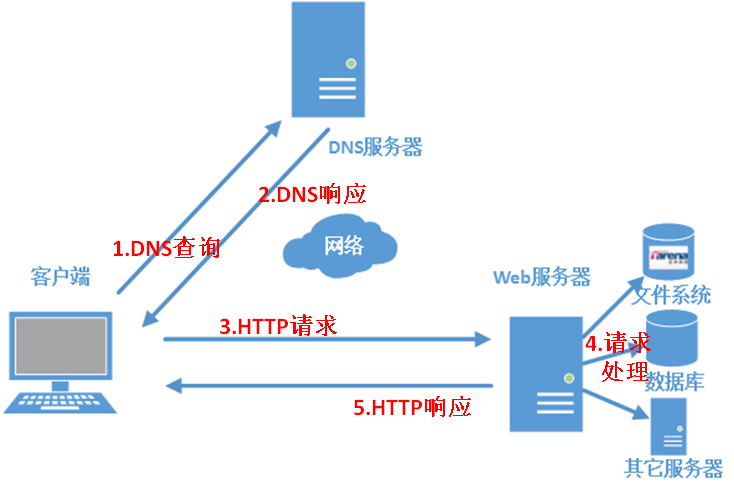
#### 3.5.4 使用浏览器访问本地/web服务器中的网页

http://127.0.0.1:80 简写 127.0.0.1/localhost/ipconfig查询本地ip进行访问

127.0.0.1永远指向本机，即使没有联网

### 【面试题】阿里面试题

Question:在浏览器地址栏中输入：www.taobao.com回车后，直到页面上出现淘宝页面，这之间的过程是怎样的？



Answer：

1.浏览器向DNS请求解析出该域名所对应的IP地址，获得后缓存该地址；（DNS查询及响应）

2.浏览器向Web服务器发起HTTP请求；

3.Web服务器接受并解析请求信息，查找指定资源并返回响应消息；

4.浏览器接收到响应消息，解析并显示。

https://blog.csdn.net/m0\_37806112/article/details/80689454

# 第二章 MySQL

## 一、项目中数据存储的方式

### 1.1 内存

速度快；但容量有限，非永久存储

### 1.2 自定义结构文件

使用简单直接；但不便于大量数据查找

### 1.3 Excel/Access等格式化文件

使用广泛；灵活性不足

### 1.4 专用文件服务器

网络存储/云存储，可靠性高；操作复杂

### 1.5 数据库服务器

存取效率高，尤其适合海量数据；操作复杂

## 二、数据库概述

### 2.1 概念

数据库：Database，以特定的结构批量存储业务数据的软件。

### 2.2 数据库发展历史

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

### 2.3 关系型数据库

RDBMS：Relation Database Manage System，大多采用C/S或B/S的形式。服务器端负责管理、维护数据，客户端程序以远程连接到服务器并发送操作指令。

①服务器端：通常以系统服务的形式启动和停止

②客户端：可能以命令行应用程序、桌面应用程序、浏览器应用

关系型数据库的逻辑结构：

Server —> Database —> Table —> Row —> Column

## 三、MySQL

### 3.1 启动并连接数据库

#### 3.1.1 下载安装XAMPP

XAMPP，Apache+MySQL+PHP+PERL，是一个功能强大的建站集成软件包，并启动其MySQL服务（服务器端和客户端）。

#### 3.1.2 启动MySQL服务器端

方法一：找到目录C:/xampp/mysql/bin/mysqld.exe双击启动MySQL服务

方法二：双击运行XAMPP，点击start启动MySQL服务

#### 3.1.3 启动MySQL客户端

方法一：找到目录C:/xampp/mysql/bin/mysql.exe双击启动MySQL客户端服务（直接连接数据库）

方法二：在XAMPP中点击shell打开命令行启动客户端（需手动连接上数据库）

【注】方法一中进入的界面等同于方法二中直接mysql命令进入的界面

#### 3.1.4 客户端连接MySQL服务器

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

简化版 mysql -uroot

其中，-h —— -host;-P —— -Post大写是为了区别于密码的p，端口号在xampp上有;-u —— -user;-p —— -password

### 3.2 常用MySQL管理命令

quit; 退出到服务器的连接

show databases; 显示服务器中当前所有的数据库

use 数据库名称; 进入/使用指定数据库

show tables; 显示当前数据库中所有的数据表

desc 表名; 描述表中有哪些列（表头）

【练习】查看MySQL服务器中提供5个数据库中各有多少个表。

information\_schema 78rows

mysql 34rows

performance\_schema 52rows

phpmyadmin 19rows

test 0rows

【练习】查看数据库phpmyadmin中，下列表有哪些列（表头）

pma\_bookmark

pma\_history

pma\_favorite

### 3.3 SQL语句

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

SQL最早是由ANSI(美国国家标准协会)提出，后提交给ISO(国际标准化组织)，最终成为数据库行业标准语言，分为多个版本：SQL87、SQL92、SQL99等，当前SQL命令被绝大多数关系型数据库所支持。

#### 3.3.1 分类

所有的SQL语句可以分为如下四类：

①DDL，Data Define Language，定义数据的结构——>CREATE/DROP/ALTER/TRUNCATE

②DML，Data Manipulate Language，操作数据，即增删改——>INSERT/DELETE/ UPDATE

③DQL，Data Query Language，查询数据——>SELECT

④DCL，Data Control Language，控制用户的权限——>GRANT/REVOKE

#### **3.3.2** 运行方式

SQL命令分为两种执行方式：

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。

②脚本模式

客户端把多行要执行的SQL命令编写在一个文本文件(eg:1.sql)中，一次性提交给服务器。适用于批量的增删改查数据。可多行顺序执行，但如遇错误只执行错误出现之前的命令。mysql -uroot < 文件绝对地址（拖拽到此即可）

【练习】使用脚本模式向数据库服务器中提交多行SQL命令，试试把其中一行写错，查看执行效果。

A：可多行顺序执行，但如遇错误只执行错误出现之前的命令。

#### 3.3.3 语法规范

①每条语句必须以英文的分号作为结束，一条语句可以跨越多行，直到遇到分号认为语句结束

②当第N条语句出现错误，则此语句本身及以后的所有语句都不会被执行

③SQL命令不区分大小写。习惯上数据库关键字均用大写，非关键字均用小写

④SQL语句中可以使用注释，分为单行注释'#'和多行注释'/\*...\*/

#### 3.3.4 常用SQL语句

①丢弃指定的数据库，如果存在的话

DROP DATABASE IF EXISTS 数据库名称;

②创建指定数据库

CREATE DATABASE 数据库名称;

③进入指定数据库

 USE 数据库名称;

④创建数据表

CREATE TABLE student(

  sid INT PRIMARY KEY,

name VARCHAR(8),

gender VARCHAR(1),

score INT

);

⑤向数据表中插入数据

INSERT INTO student VALUES(1,'Tom','M',99);

⑥修改指定行的指定数据

UPDATE student SET name='Frank',gender='M' WHERE sid=1;

⑦查询数据表中所有的数据

SELECT \* FROM student;

⑧删除数据表中的数据

DELETE FROM student WHERE sid=1;

【练习】创建03.sql文件，如果存在则先删除再新建数据库tedu\_oa，在数据库中创建保存员工信息的数据表emp，包含列编号(eid)，姓名(name)，性别(gender)，电话(phone——varchar(11))，地址(addr——varchar);插入5条记录；查询出表中的数据。

【练习】创建01\_book.sql文件，丢弃数据库dangdang，如果存在的话；创建数据库dangdang，进入该数据库；创建一个保存图书的数据表book，包含编号(bid),标题(title),作者(author),价格(price),出版时间(pubDate),是否加价购(isAddbuy);插入4条记录，删除编号为2的记录，修改编号为3的记录，查询最后的结果。

【课后练习】创建脚本文件xz.sql，删除数据库xz，如果存在的话；创建数据库xz，进入该数据库，创建数据表user，包含列uid、uname、upwd、email、phone、gender、userName(真实姓名)、regTime(注册时间2018-10-01),isOnline(是否在线，1->是，0->否);插入5行记录，删除第2行记录，查询最后的结果。

### 3.4 字符集编码

#### 3.4.1 计算机中存储英文字符

ASCII:总共128个，对所有的英文字符以及标点符号进行了编码。

Latin-1:总共256个，兼容ASCII，同时对欧洲符号进行了编码。mysql默认使用Latin-1编码。

#### 3.4.2 计算机中存储中文字符

GB2312:对常用的6千多个中文字符进行了编码，兼容ASCII。

GBK:对2W多个汉字进行了编码，兼容ASCII。

BIG5:台湾 对繁体中文进行了编码，兼容ASCII。

Unicode:对世界上主流语言中的常用字符进行了编码，兼容ASCII，不兼容GB2312、GBK、BIG5等系列。具体分为UTF-8,UTF-16，UTF-32等存储方案。

#### 3.4.3 解决中文乱码问题

产生乱码的原因：mysql中默认使用Latin-1编码，而Latin-1不存在中文编码

解决：脚本文件、客户端、服务器端三处均改为使用UTF8存储方式

①sql脚本文件另存为utf-8编码

②客户端连接服务器端使用的编码为UTF8(在.sql脚本文件最开头添加以下代码)

#设置客户端连接mysql使用的编码

SET NAMES UTF8;

③服务器端，在创建数据库时存储所使用的编码为UTF8

#使用UTF8作为数据库（服务器端）的编码

CREATE DATABASE dangdang CHARSET=UTF8;

### 3.5 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 student (name 列类型)

#### 3.5.1 数值类型——不加引号

TINYINT 微整型 占一个字节 -128~127

TB GB MB KB BYTE BIT(1 BYTE = 8 BIT)

SMALLINT 小整型 占两个字节 -32768~32767

INT 整型 占四个字节 -2^31~2^31(-2147483648~2147483647)

BIGINT 大整型 占8个字节

FLOAT(M,D) 单精度浮点型 占4个字节，最大是3.4E38，存储范围远比INT大，可能产生四舍五入。M表示总的有效位数(不包含小数点)，D表示小数点后的有效位数

DOUBLE(M,D) 双精度浮点型 占8个字节，比BIGINT存储范围大很多。也会产生四舍五入。

DECIMAL(M,D) 定点小数 不会产生计算误差

**!注意：**浮点数如果不写精度和标度，则会按照实际显示，如果有精度和标度，则会将数据四舍五入后插入，系统不报错。定点数如DECIMAL如果不设置精度和标度，刚按照默认的(10,0)进行操作，如果数据超过了精度和标度值，则会警告！

BOOL 布尔型 MySQL中不支持该类型值，会自动转换为TINIYINT(1)只能是TRUE和FALSE或者是1或0

【扩展】关于MySQL中int(M)和tinyint(M)数值类型中M值的意义

①整数型的数值类型已经限制了取值范围，有符号整型和无符号整型都有，而M值并不代表可以存储的数值字符长度，它代表的是数据在显示时显示的最小长度；

②当存储的字符长度超过M值时，没有任何的影响，只要不超过数值类型限制的范围；

③当存储的字符长度小于M值时，只有在设置了zerofill用0来填充，才能够看到效果，换句话就是说，没有zerofill，M值就是无用的。

#### 3.5.2 日期时间类型——必须添加引号

DATE 日期类型 2018-08-02

TIME 时间类型 14:51:25

DATETIME 日期时间类型 2018-08-02 14:51:25

#### 3.5.3 字符串类型——必须添加引号

VARCHAR(M) 变长字符串 不会造成空间浪费，操作速度相对比较慢。M最多存储65535

CHAR(M) 定长字符串 可能产生空间浪费，但其操作速度相对VARCHAR快一些，常用于存储固定长度的数据，例如手机号码、身份证号等。M最多存储255

TEXT 大型变长字符串 最多存储能到2G

【练习】创建02\_xuezi.sql,创建数据库xuezi,保存的编码为UTF8，进入该数据库，创建保存笔记本信息的表laptop，包含编号(lid),标题(title),价格(price),库存量(stockCount),上架时间(shelfTime),是否显示在首页(isIndex);使用适当的列类型；插入4条记录。

【练习】创建03\_tedu.sql文件，创建数据库tedu,保存的编码为UTF8,进入该数据库；创建保存公司部门信息的表dept，包含编号(did),部门名称(dname),员工数量(empCount)

10 研发部 3

20 运营部 2

30 市场部 5

创建保存员工信息的表emp，包含编号(eid),员工姓名(ename),员工性别(gender),生日(birthday),工资(salary),隶属部门的编号(deptId)

### 3.6 列约束

Constraint，约束。MySQL可以对插入的数据进行特定的验证，只有满足条件才会允许插入到数据表中，否则就会被认为是非法数据，禁止插入。例如：员工的年龄范围18~65,性别只能是男或女...

在MySQL中一共提供了6种列约束

#### 3.6.1 主键约束——PRIMARY KEY

声明了主键列上的值不能出现重复，表中查询的记录会按照主键上的值有小到大进行排序。一个表只允许有一个主键，通常设置在编号上。若主键列是整数类型，则同时可以声明刺裂上的值为[自动增长](#_2.6.7_MySQL的自增列——AUTO_INCREMENT)——如果插入数据时未指定值或指定为NULL值。则默认生成一个在当前最大值基础上+1的下一个整数值。

#### 3.6.2 唯一约束——UNIQUE

声明了唯一约束的列不能插入重复的值，但允许插入NULL而且是允许插入多个NULL值。

#### 3.6.3 非空约束——NOT NULL

所插入的值不允许为NULL(NULL和“”不等价，若输入框中不输入值是“”；而要为NULL应输入null)

#### 3.6.4 默认值约束——DEFAULT

可以使用默认值DEFAULT来为指定的列设置默认值；

使用方法：

①创建数据表时指定默认值约束

CREATE TABLE user(

uid INT PRIMARY KEY,

price DECIMAL(10,2) DEFAULT 0,

status VARCHAR(32) DEFAULT ‘等待付款’

);

②插入数据行时使用默认值

INSERT INTO VALUES(1,'Tom',DEFAULT);

③插入数据行时未明确指定值的列将使用默认值

INSERT INTO user(uid,uname) VALUES(2,'Jerry');

#### 3.6.5 检查约束——CHECK

检查约束是对输入的数据范围进行检查

CREATE TABLE student(

age TINYINT CHECK(age >= 0 AND age <= 100)

);

**!注意：**MySQL的初始InnoDB引擎默认不会执行检查约束，即MySQL不支持检查约束

#### 3.6.6 外键约束——FOREIGN KEY

声明了外键的列，取值必须在其参考的表的主键列上出现过。外键列上允许出现重复值和空值。

FOREIGN KEY(本表列名) REFERENCES 引用表名(引用列名);

【练习】创建xuezi.sql文件，新建数据库xuezi，设置UTF8编码，创建数据表xz\_laptop，包含编号(lid),标题(title),价格(price),规格(spec),详细介绍(details),上架时间(shelfTime),是否在售(isOnsale);插入5条记录。使用合适的列类型和列约束。

#### 3.6.7 MySQL的自增列——AUTO INCREMENT

自动增长，如果设置一个列为自增列，无需手动赋值，赋值为NULL，就会获取当前最大值，新纪录在当前的基础之上加1,只适用于整数型的主键上，允许手动赋值。

**!注意：**null不区分大小写；如果不设置null而是设置一个值x，那么下一条记录的值为x+1

【练习】列约束的使用

创建文件02\_tedu.sql，创建数据库tedu,使用UTF8编码，进入该数据库。创建部门表dept,包含编号(did),名称(dname);

10 研发部 20 运营部 30 市场部 40 测试部

创建员工表empt，包含编号(eid)，姓名(ename),性别(gender),生日(birthday),工资(salary)，隶属的部门编号(deptId)

插入14行记录，分别隶属部门10,20,30,其中14号员工不在任何部门

### 3.7 项目中如何存储时间

DATE #2018-08-03

03/08/2018 08/03/2018 2018年08月03日

时间存的是，距离1970年1月1日（计算机元年）距当前的毫秒数

将来存储时间列类型 BIGINT

（存储性别 BOOL 1/0 男/女 male/female man/woman...）

### 3.8 SQL查询语句

#### 3.8.1 简单查询

##### 3.8.1.1 只查询特定列

练习：查询员工的姓名、工资、生日

SELECT ename,salary,birthday FROM emp;

练习：查询员工的编号、姓名、性别、部门编号

SELECT eid,ename,sex,deptId FROM emp;

##### 3.8.1.2 查询所有的列

SELECT \* FROM emp;

##### 3.8.1.3 给列取别名

练习：查询员工的姓名、生日、收入，取别名

SELECT ename AS 姓名,birthday AS 生日,salary AS 收入 FROM emp;

SELECT ename 姓名,birthday 生日,salary 收入 FROM emp;

练习：查询员工的编号、姓名、性别，起英文别名

SELECT eid a,ename name,sex gender FROM emp;

##### 3.8.1.4 只显示不同的记录

练习：查询哪些部门编号下有员工

SELECT DISTINCT deptId FROM emp;

练习：查询公司有哪些性别的员工

SELECT DISTINCT sex FROM emp;

##### 3.8.1.5 在查询时执行计算

练习：计算2+3\*5/10

SELECT 2+3\*5/10 结果;

练习：查询出所有员工的姓名和年薪

SELECT ename 姓名,salary\*12 年薪 FROM emp;

练习：假设每个员工的工资加500，年终每个人再给5000的年终奖，查询出每个员工一年的收入是多少

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

##### 3.8.1.6 查询结果集的排序

练习：查询所有的员工，结果按照工资由低到高

SELECT \* FROM emp ORDER BY salary ASC; #ascendant

练习：查询所有员工，结果按照工资由高到低

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

练习：查询所有员工，结果按照年龄从大到小排列

SELECT \* FROM emp ORDER BY birthday ASC;

练习：查询所有的员工，结果集按照姓名排序

SELECT \* FROM emp ORDER BY ename DESC;

SELECT \* FROM emp ORDER BY ename ASC;

练习：查询所有的员工，结果集按照生日由小到大排序，如果生日相同，再按姓名由小到大排序

SELECT \* FROM emp ORDER BY birthday,ename;

练习：查询所有员工，按照工资由小到大，如果工资相同，女员工显示在男员工的前面（性别的升序，女—>0;男—>1）

SELECT \* FROM emp ORDER BY salary,sex;

**!注意：**排序默认是按照升序排列

##### 3.8.1.7 条件查询

练习：查询出编号为5的员工所有信息

SELECT \* FROM emp WHERE eid=5;

练习：查询出姓名为KING的员工的编号、工资、生日

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

练习：查询出20号部门下所有员工的信息

SELECT \* FROM emp WHERE deptId=20;

练习：查询出所有的男员工信息

SELECT \* FROM emp WHERE sex=true;

练习：查询出工资大于等于5000的员工所有信息

SELECT \* FROM emp WHERE salary>=5000;

练习：查询出1991-1-1之后出生的员工所有信息

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

练习：查询出不在10号部门的所有员工信息

SELECT \* FROM emp WHERE deptId!=10 OR deptId IS NULL;

练习：查询出暂时没有部门的员工信息

SELECT \* FROM emp WHERE deptId IS NULL;

**!注意：**NULL/null和任何值都不能进行相等/不等判定，可以使用IS NULL/IS NOT NULL进行判定

练习：查询出有明确部门的员工信息

SELECT \* FROM emp WHERE deptId IS NOT NULL;

练习：查询出工资大于6000的女员工所有信息

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

练习：查询出工资在5000~5999之间的员工信息

SELECT \* FROM emp WHERE salary>=5000 AND salary<=5999;

SELECT \* FROM emp WHERE salary BETWEEN 5000 AND 5999;(满足大于等于且小于等于才能用BETWEEN...AND)

练习：查询出工资小于4000和大于8000的员工信息

SELECT \* FROM emp WHERE salary<4000 OR salary>8000;

练习：查询出在1991年出生的员工信息

SELECT \* FROM emp WHERE birthday>='1991-1-1' AND birthday <= '1991-12-31';\

SELECT \* FROM emp WHERE birthday BETWEEN '1991-1-1' AND '1991-12-31';

练习：查询出在1990年之前和1993年之后出生的员工信息

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

练习：查询出在10号和30号部门的员工信息

SELECT \* FROM emp WHERE deptId=10 OR deptId=30;

SELECT \* FROM emp WHERE deptId IN(10,30);

练习：查询出在10、30、50、80号部门的员工信息

SELECT \* FROM emp WHERE deptId=10 OR deptId=30 OR deptId=50 OR deptId=80;

SELECT \* FROM emp WHERE deptId IN(10,30,50,80);

练习：查询出不在10和30号部门的员工信息

SELECT \* FROM emp WHERE deptId!=10 AND deptId!=30;

SELECT \* FROM emp WHERE deptId NOT IN(10,30);

##### 3.8.1.8 模糊条件查询

练习：查询出姓名中包含字母E的员工所有信息

SELECT \* FROM emp WHERE ename LIKE '%E%';(%表示大于等于0个的占位字符，\_表示一个占位字符)

练习：查询出姓名以E结尾的员工所有信息

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

练习：查询出姓名中倒数第二个字符是E的员工所有信息

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

**!注意：**SQL中提供了两个模糊查询的匹配符

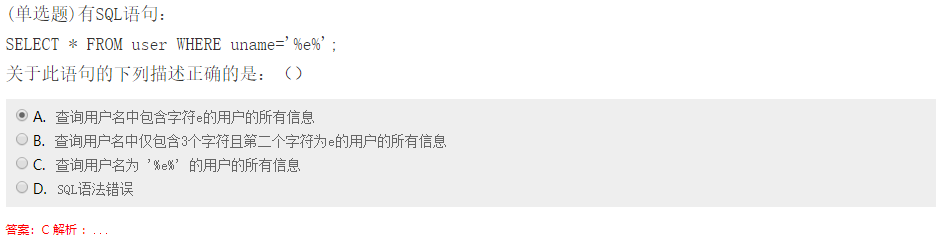
%可以匹配任意多个字符 >=0

\_可以匹配任意一个字符 =1

上述两个符号不能和=连用，只能使用LIKE连用

练习：查询出所有的男员工，对查询的结果集，按照工资升序排序。

SELECT \* FROM emp WHERE sex=1 ORDER BY salary;



##### 3.8.1.9 分页查询

分页显示：查询的结果集中记录太多，一次显示不完，可以一页一页地显示。

分页查询语句在不同的数据库中实现方法不一样。MySQL是最简单的。

SELECT \* FROM emp LIMIT start,count;

start:从结果集中开始读取的值

count:指定读取的最多行数

第1页：SELECT \* FROM emp LIMIT 0,5;

第2页：SELECT \* FROM emp LIMIT 5,5;

第3页：SELECT \* FROM emp LIMIT 10,5;

...

第n页：SELECT \* FROM emp LIMIT (n -1) \* 5,5; # start:(页码-1)\*每页数量

练习：分页查询所有员工信息，假设每页最多显示6条记录，写出查询前4页的sql语句

SELECT \* FROM emp LIMIT 0,6;

SELECT \* FROM emp LIMIT 6,6;

SELECT \* FROM emp LIMIT 12,6;

SELECT \* FROM emp LIMIT 18,6;

**!注意：**LIMIT后面的两个值不能加引号

【课后练习】

①删除所有的演示代码，根据注释重新编写SQL语句

②创建产品表三个

#### 3.8.2 复杂查询

##### 3.8.2.1 聚合查询/分组查询

练习：查询出当前所有员工的数量

SELECT COUNT(\*) FROM emp; ——推荐写法

SELECT COUNT(ename) FROM emp;

SELECT COUNT(eid) FROM emp;

SELECT COUNT(deptId) FROM emp;

【说明】函数：FUNCTION，具有特定的功能，需要接受若干个数据，最终导出特定的计算结果。

聚合查询——MySQL中提供了5个聚合函数：

COUNT()/SUM()/AVG()/MAX()/MIN()

练习：查询出10号部门员工的数量

SELECT COUNT(\*) FROM emp WHERE deptId=10;

练习：查询出10号部门所有员工工资的总和

SELECT SUM(salary) FROM emp WHERE deptId=10;

练习：查询出女员工工资的总和

SELECT SUM(salary) FROM emp WHERE sex=0;

练习：查询出所有男员工的平均工资

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

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

练习：查询出1990年出生的员工工资最大值和最小值

SELECT MAX(salary),MIN(salary) FROM emp WHERE birthday BETWEEN '1990-1-1' AND '1990-12-31';

分组查询

练习：查询出每个部门的员工数量

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

**!注意：**分组查询中SELECT后面只能跟分组条件，或者其他列的聚合函数

练习：查询出每个部门的平均工资、最大工资、最小工资

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

练习：查询出男员工和女员工的人数是多少

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

##### 3.8.2.2 子查询

示例：查询出工资比TOM高的员工的所有信息

步骤1：先查询出Tom的工资是多少——6000

SELECT salary FROM emp WHERE ename='Tom';

步骤2：在查询出工资比6000高的员工信息

SELECT salary FROM emp WHERE salary>6000;

综合：

SELECT \* FROM emp WHERE salary > (

SELECT salary FROM emp WHERE ename='Tom'

);

练习：查询出研发部所有员工的信息

步骤1：查询出“研发部”部门编号——10

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

步骤2：查询出10号部门所有的员工信息

SELECT \* FROM emp WHERE deptId=10;

综合：

SELECT \* FROM emp WHERE deptId = (

SELECT did FROM dept WHERE dname='研发部'

);

【补充重点】子查询的另一种方式：在其他表中查出数据添加到指定表中(成为新列)

SELECT \*,(

SELECT md from xz\_laptop\_pic

where laptop\_id=lid

LIMIT 1 //此处不限制1会报错

) as md

FROM xz\_laptop

where title like "%macbook%" and

title like "%i5%" and

title like "%128g%"

上述查询语句的作用：查询出xz\_laptop表中title数据符合模糊查询的行的所有列数据，并在xz\_laptop\_pic表中查出该表laptop\_id等于xz\_laptop表lid列的行数据的md列数据值(仅查出一个LIMIT 1)，并为它取别名为md。

##### 3.8.2.3 跨表查询

示例：查询出所有的员工姓名及其所在部门的名称

SELECT ename,dname

FROM emp,dept

WHERE emp.deptId = dept.did;

**!注意：**为了防止出现笛卡尔积，在多表查询的时候需要添加查询条件。

【问题】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

SQL99中的跨表查询

①内连接INNER JOIN...ON——和之前的作用一样

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

②左外连接LEFT OUTER JOIN ... ONz——显示左侧所有的记录

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

③右外连接RIGHT OUTER JOIN ... ON——显示右侧所有的记录

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

【说明】OUTER这一关键字可以省略

④全连接——FULL JOIN

MySQL不支持

##### 3.8.2.4 结果集查询

UNION关键字用于将两个查询结果集合并为一个大的结果集

UNION ALL 合并结果集，允许出现重复数据

UNION 合并结果集，重复数据仅显示一遍

示例：

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

UNION ALL

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

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

UNION

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

【课后练习】

①查询出价格在3000~5000之间的所有笔记本电脑的数量；

②查询出笔记本型号家族名称中包含“联想”的所有笔记本信息；

③查询出每个笔记本商品的主标题以及其对应的型号家族的名称

# 第三章 JS基础

学习一门编程语言路线图

①了解语言的背景，历史，特点，应用领域，现状

②搭建开发环境，编写Hello World

③声明变量和常量

④数据类型

⑤运算符

⑥逻辑结构

⑦通用小程序

⑧函数和对象

⑨第三方类库，插件，组件，框架

⑩开发项目

## 一、JS基本概念

### 1.1 JS概述

#### 1.1.1 JS概念

JavaScript是一种运行于JavaScript解释器/引擎中的解释型脚本语言

#### 1.1.2 JS执行环境——JS解释器呈现方式

客户端：使用嵌入在浏览器中的JavaScript解释器。由于浏览器过多，存在有兼容性问题

服务器端：独立安装的JS解释器。如Node.js解释器，基于谷歌公司的V8引擎。在服务器端不存在兼容性问题

【补充】Node.js下载安装

官网：https://nodejs.org 下载 8.11.3 LTS

查看NodeJS是否安装成功：

node -v

运行上述命令后，界面会显示出所安装的nodeJS版本号

运行方式

node C:/xampp/.../01.js

【练习】创建一个02.js,打印个人的姓名，使用Node.js运行，查看结果

#### 1.1.3 JS的特点

①代码可以使用任何文本编辑工具编写，语法类似于C和Java

②脚本文件无需编译，由JS引擎解释执行

③弱类型语言

④基于对象

⑤跨平台性

**!注意：**JS语言本身提供的数据类型和对象比较有限，其运行环境一般都扩展提供了更加丰富的编程接口，如浏览器为JS提供了BOM和DOM对象，Node.js为JS提供了网络访问、文件操作、数据库查询等功能对象。学习这些不同环境下的JS事注意加以区分。

#### 1.1.4 JS语法规范

①区分大小写

②JS中每一行结束可以不加分号结束，习惯上加上分号

③注释：分为单行注释(//)和多行注释(/\*\*/)

### 1.2 变量和常量

#### 1.2.1 变量

变量是用于存储信息的容器

##### 1.2.1.1 变量声明

var x=1;

var y=2,z=3; //一次能够声明多个变量

var是声明变量的关键字

x是变量名称

x=1表示把1赋值给变量x

【说明】如果声明了一个变量，没有赋值，默认是Undefined

##### 1.2.1.2 变量的命名

变量命名可以有数字、字母、下划线和美元符号，但不能以数字开头，不能使用关键字和保留字作为变量名

【补充】保留字(class、int、float)：变量名称习惯上有明确地语义，例如用户名(userName),密码userPwd,命名分为驼峰命名法和下划线命名法

【练习】使用变量来存储员工的信息，声明员工的编号，姓名，性别，生日，工资并初始化具体的信息。

var eid = 01,ename='张三',gender=1,birthday='1990-1-1',salary='12345.00';

#### 1.2.2 常量

常量就是不能改变的值，使用const关键字来声明常量，常量名习惯上使用大写

### 1.3 数据类型及其转换

MySQL:数值/日期时间/字符串

JS分为原始类型和引用类型

#### 1.3.1 原始数据类型

分为数值类型（Number）、字符串类型（String）、布尔类型（Boolean）、未定义型（Undefined）、空值类型（Null）

**!注意：**不同的JS运行环境提供的原始数据类型都是一样的，但他们提供的引用类型区别很大

##### 1.3.1.1 Number类型

数值型又分为整数和浮点数

①整数型

分为八进制（以0开头-->010），十进制，十六进制（以0x开头-->0xf）

②浮点型

使用小数点记录的数据，还可以使用指数记录的数据

【练习】查看整数20,32,54的八进制，十六进制分别是多少 24/14 40/20 66/36；使用小数点和指数来表示圆周率3.1415926 31.415926E-1 314.15926E-2

【课后练习】计算圆的周长和面积，把半径值5使用变量存储，把圆周率使用常量存储，最后打印周长和面积。

##### 1.3.1.2 String类型

常用于表示一些文本字符数据，例如：姓名、邮箱、电话、家庭住址等

特点：使用引号（单引号or双引号）把数据包含起来

①查看汉字的Unicode编码

'一'.charCodeAt(); //19968

②查看汉字的Unicode编码十六进制形式

'一'.charCodeAt().toString(16); //4e00

③查看16进制的Unicode编码对应汉字

console.log('\u4e00');

④检验基本汉字范围： 4e00 ~ 9fa5

⑤[转义字符](#_5.1.3_转义字符)

\n \r \t \\ \’ \”

##### 1.3.1.3 Boolean类型

在程序中表示真或者假的结果

取值：true/false

常用于表示是否的结果,在实际运算中：true==1,false==2

例如：是否登录，是否注册，是否是会员...

##### 1.3.1.4 Undefined类型

只有一个值是undefined。当声明了一个变量但为初始化，此时变量的值为undefined

##### 1.3.1.5 Null类型

只有一个值null,表示空，在JS中常用于释放一个对象

#### 1.3.2 引用数据类型

引用类型的值（对象）是引用类型的一个实例，在ECMAScript中，引用类型是一种数据结构，用于将数据和功能组织在一起。

引用数据类型有：Object类型、Array类型、Date类型、RegExp类型、Function类型、[基本包装类型](#_五、基本包装类型（对象）)（Boolean类型、Number类型、String类型）、单体内置对象（Global对象、Math对象）。

#### [1.3.3 数据类型检测](#_3.4.7_特殊运算符)

检测数据类型的操作符：typeof(参数) 参数表示要检测的数据

结果有：number(数值型)、string(字符串型)、boolean(布尔型)、undefined(未定义型)

#### 1.3.4 数据类型转换

##### 1.3.4.1 隐式转换

①数字+字符串：数字会被转成字符串

10 + 'kb' //10kb

②数字+布尔值：布尔值会被转成数字，true -> 1/false -> 0

1 + true = 2;

③布尔值+布尔值：布尔值会被转成数字 true -> 1/false -> 0

true + false //1

④布尔值+字符串：布尔值会被转成字符串

true + 'abc' //'trueabc'

【练习】查看输出结果

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

console.log(num1 + num2 + num3); //3sttrue

console.log(num2 + num3 + num1); //sttrue3

console.log(num3 + num2 + num1); //truest3

##### 1.3.4.2 强制转换

①强制转换成整型

parseInt('23.55t'); //23

转换规则：从第一位开始找数字，遇到非数字或小数点即结束，如果首位是非数字，则返回NaN(Not a Number)

【练习】转换15a.8 hello1 1hello

②强制转换成浮点型

parseFloat('2m3.5a5a'); //2

转换规则：从第一位开始找数字，遇到数字、小数点继续往后找，直到遇到非数字或小数点即结束。如果第一位是非数字，则返回NaN

③强制转换成数值型

Number()

Number('23.55t') //NaN

转换规则：如果被转换的值中有非数字，则结果为NaN

④转换成字符串型

toString()

20.toString(); //'20'

toString()可以将数值型和布尔型的值转换为字符串型

### 1.4 运算符和表达式

var a = 1;

由运算符连接的操作的数据，这种形式就成为表达式。

#### 1.4.1 算术运算符

+ - \* / % ++ --

% 取余，一般用于判定一个数能否被另一个数整除

++ 自增，在原来的基础之上加1

-- 自减，在原来的基础之上减1

console.log(num++); //先打印num的值，然后再执行加1

console.log(++num); //先执行加1，然后再打印num的值

【练习】以下程序的结果是？

var a = 1;

console.log(a++ + ++a); //4

console.log(a++ + ++a + a++);//3+5+5/1+3+3

console.log(a);//6

#### 1.4.2 比较(关系)运算符

> < >= <= == != ===(全等于) !==(不全等于)

产生的结果是一个布尔型的值(true/false)

**!注意：**若参加关系运算的两个数据类型不同，则会发生隐式类型转换，统一为相同类型后才能进行比较。

等于(==):比较值是否相同

全等于(===):比较值和类型是否相同，都相同才返回true

'3' > 10; //false

比较运算符的两端，只要有一个是数值型，另一个会隐式转换成数值型

'3' > '10'; //true

比较运算符的两端，如果都是字符串，比较的是Unicode码值

【练习】比较两个中文的大小

'张三丰' > '张无忌';

'三' -> 19977

'无' -> 26080

如果第一个字符相同，则继续比较第二个字符

示例：'3m' > 10

字符串'3m'会隐式转换为数值型，使用的是Number()，结果NaN

【补充】NaN与任何值（包括它本身）比较结果都是false，故为了判断一个值是否为NaN，需要借助isNaN()函数。

#### 1.4.3 逻辑运算符

&& 并且 || 或者 ! 非

&& 两个条件都满足，结果是true

|| 两个条件至少满足其一，结果是true

! 反向，逆向；非真为假，非假为真

逻辑中断(短路逻辑)

&& 如果第一个条件为false，整个结果一定为false，第二个条件就没有继续执行的必要。

|| 如果第一个条件为true，整个结果一定为true，第二个条件也没有继续执行的必要。

【练习】

var a = 4,b = 2;

a > 3 && (a = 5); //false

a < 5 || (b = 4); //true

#### 1.4.4 位运算符(了解)

01 10 11 100 101 110 111 1000 1001 1010 1011 1100 1101 1110 1111

在执行位运算的时候，十进制的值会转成二进制

按位与(&) 上下两位都是1，结果为1，否则为0

按位或(|) 上下两位只要有一位是1，结果为1，否则为0

按位异或(^) 上下两位不一致，结果为1;相同为0

按位右移(>>) 删除最后的几位，例如 3>>1 结果1，数字变小

按位左移(<<) 在末尾添加0，例如 3<<1 结果6，数字变大

【练习】10>>2 -> 2；5^8 -> 13

#### 1.4.5 赋值运算符

= +=(在原来的基础上加) -= \*= /= %=

**!注意：**在绝大多数编程语言中，a+=3由于运行时可以进行优化，执行效率都要优于a=a+3

【练习】超过500打八折，600在原来基础之上打八折

【课后练习】声明一个变量var year = 2014;判断年份是否为闰年（能被4整除但不能被100整除，或能被400整除）

A：((year%4 == 0) && (year%100 != 0)) || (year % 400 ==0)

#### **1.**4.6 三目运算符

一目运算符就是只有一个操作数或表达式：a++/a--/!false

二目运算符就是含有两个操作数或表达式：+ - \* / % > >= < <= == != === !== >> << && || & | ^ += -= \*= /= %= >>= <<=

三目运算符（条件运算符）就是含有三个操作数或表达式：

条件表达式 ? 表达式1:表达式2;

如果条件表达式结果是true，执行表达式1，否则执行表达式2

x>y ? x:y;

#### 1.4.7 特殊运算符

[typeof](#_3.3.2_数据类型检测)  instanceof void delete

【补充】instanceof 运算符是用来在运行时指出对象是否是特定类的一个实例。instanceof通过返回一个布尔值来指出，这个对象是否是这个特定类或者是它的子类的一个实例

### 1.5 逻辑结构

流程控制

程序 = 算法 + 数据

任何复杂的程序的算法都可以分为三类：顺序执行、选择执行、循环执行

选择执行：程序可以选择执行这段代码，也可以选择不执行这段代码

循环执行：在满足指定条件是反复执行某一段代码，不满足条件时自动结束循环

#### 1.5.1 分支结构

##### 1.5.1.1 if语句

if(逻辑表达式) {

语句1;

语句2;

}

语句3;

执行流程：如果逻辑表达式的结果是true，执行语句1/2;否则跳过语句1/2直接执行语句3

【练习】通过弹出提示框，输入商品的总额，如果总额满500，则打八折，最终打印商品的应付金额。

**!注意：**在if后面的大括号中的语句只有一条则可以省略大括号，否则不能省略，逻辑表达式可以直接写true/false；在逻辑表达式中，undefined、null、0、''、NaN都被认为是false

##### 1.5.1.2 if...else语句

语句1;

if(逻辑表达式) {

语句2;

} else {

语句3;

}

语句4;

执行流程：先执行语句1，判断逻辑表达式的结果，为true执行语句2；为false则执行语句3。最后再执行语句4

##### 1.5.1.3 if...else语句嵌套

语句0;

if(逻辑表达式1) {

语句1;

} else if(逻辑表达式2) {

语句2;

语句3;

} else if(逻辑表达式n) {

语句n;

} else {

语句n+1; //在以上n种情况都不满足的情况下才执行

}

执行流程：

①执行语句0；

②执行逻辑表达式1，如果结果是true，执行语句1；否则执行逻辑表达式2，如果结果是true，执行语句2；否则继续执行其他的逻辑表达式；

③如果以上所有的逻辑表达式结果都是false，则执行语句n+1

【练习】声明变量xz来保存星座信息，使用if...else语句的嵌套，来完成根据星座来打印出对应的性格

##### 1.5.1.4 多项分支语句 switch...case语句

语句0;

switch(表达式) {

case 0: //如果表达式的值是1，选择进入这个入口执行程序

语句1;

...

break; //终止，不会再向下执行程序

case 1:语句2;break;

case 2:语句3;break;

...

defalut:语句n; //以上所有值都不匹配，选择这个入口执行程序

}

**!注意：**在switch语句中，比较所执行的是全等于(===)

【练习】使用switch...case语句完成星座判断

**!注意：**if...else嵌套语句和switch...case语句的区别

if...else可以判断所有的情况，例如大于，小于，等于，不等于...

switch...case只能用于全等于的比较，局限性比较大 ；但是它的执行效率比if...else更快

#### 1.5.2 循环结构

##### 1.5.2.1 while循环

循环就是一遍又一遍执行相同或者相似的代码

例如：循环产生商品列表，打印100次'hello'

while(循环条件) {

//循环体

}

【练习】打印0~14之间所有的数字

【课后任务】

①作业：完成使用switch...case语句来判断一个人的成绩

60~69 70~79 80~89 90~99 100

【提示】让分数除以10取整(parseInt),得到的结果6,7,8,9,10

②扩展学习：学习for循环，do...while循环，以及循环嵌套

##### 1.5.2.2 do...while循环

do {

//循环体

}while(循环条件);

执行过程：

执行循环体，判断循环条件是否为true，如果为true，继续执行循环体，如果条件为false即终止循环

while和do...while的区别

①while循环是先判断再执行

②do...while是先执行一次再判断

③如果条件为false的时候，while会立即终止，而do...while会执行一次然后终止

【练习】输入密码，先输入完密码之后，才能进行判断是否正确，使用弹出提示框输入密码，如果输入的正确，结束循环，否则继续弹出提示框输入密码；无限循环弹出提示框，直到输入正确的密码结束循环。

##### 1.5.2.3 for循环

for(表达式1;表达式2;表达式3) {

//循环体

}

表达式1:循环的初始值，例如i=0

表达式2:循环的条件，例如i<10

表达式3:循环的变化(自增、自减等)，例如i++/i--

执行流程：

①初始化i的值

②判断i的值是否满足小于10

③打印i的值

④执行i++

⑤再次执行第二步

【练习】计算1~100之间所有奇数的和

【练习】计算1~100之间所有能被7整除的数，并且是奇数的和

【练习】计算20的阶乘

【练习】打印一行\*\*\*\*\*

##### 1.5.2.4 循环中的break和continue

break 结束整个循环,break之后的语句也不能执行

continue 结束这一次的循环，还执行自增或者自减

【练习】计算1~10之间所有奇数的和 //如果遇到的数字是偶数就跳过

【练习】计算1~100之间所有数字的和，当总和大于4000的时候，终止循环，输出总和，同时输出此时的i值

【练习】打印2000~2100之间所有的闰年

【补充】斐波那契数列

1 1 2 3 5 8 13 21 34 ...

1 2 3 4 5 6 7 8 9

【课后任务】

①完成今天的复习，并整理思维导图

②完成九九乘法表（完整版）

计算斐波那契数列的第12项是多少？（选做）

使用for循环来计算第12个月的数字是多少？

③预习JS的自定义函数

##### 1.5.2.5 循环嵌套

while,do-while,for之间可以相互嵌套

## 二、函数及作用域

### 2.1 函数的声明与调用

JS中函数分为两种，一种是JS提供的，例如parseInt,parseFloat,alert...;这些是可以直接使用的。另一种是自定义函数(function)，也称为是方法(method)。自定义函数是一段已经预定义好的、可以反复执行的代码块，其中可以包含多条可执行语句。其本质上是功能完整的对象。

#### 2.1.1 普通函数的声明和调用

①声明：

function 函数名称() { //和声明变量的规则一致

//函数体——要重复执行的代码

}

②调用：

函数名称();

函数只是声明，里面的代码不执行，只有调用函数才会执行函数体中的代码

【练习】声明一个函数calc，在函数体中实现两个数字相加的结果，并打印最终的结果

#### 2.1.2 带有参数的函数声明和调用

①声明：

function 函数名称(参数列表) {

//函数体

}

②调用：

函数名称(参数列表)：

**!注意：**参数列表可以声明0个或者多个，多个之间用逗号隔开；声明时的参数叫做形参，调用时传入的参数叫做实参。形参只是负责占位，在调用的时候，实参的值会赋给形参

【练习】声明函数calc，传递两个参数，计算相加并打印最终的结果

#### 2.1.3 带有返回值的函数的声明和调用

①声明：

function 函数名称(参数列表) {

//函数体

return 返回值;

}

②调用：

函数名称(参数列表);

【说明】函数中没有添加return，返回结果是undefied；函数有return后的值为空，返回结果是undefined；return后的所有语句都不能被执行，return就是函数结束。

#### 2.1.4 [匿名函数的声明和调用](#_2.4_匿名函数与回调函数)

①声明

var 变量名称 = function (形参列表){

//函数体

}

②调用

变量名称(实参列表);

**!注意：**使用匿名函数的方式不存在函数的提升，因为函数名称使用变量表示的，只存在变量提升

【练习】声明函数getMax()，传递两个参数，计算两个参数的大小，返回最大值

【练习】声明函数getMax()，传递3个参数，返回最大值

【练习】声明函数sum()，传递1个参数，返回这个参数和1之间所有数的和

【练习】计算前n个数的乘积

【练习】声明函数isRun(),传递参数为年份，如果是闰年返回true，不是则返回false

### 2.2 变量及函数的作用域

#### 2.2.1 变量的作用域

变量的作用域指变量的可访问范围

分为：全局作用域和局部作用域（函数作用域）

全局作用域下的变量可以在任何位置访问到，而局部作用域下的变量只能在局部（函数内部）才能访问到

**!注意：**函数内部声明的变量，没有使用var关键字，声明的变量就成为了全局变量，但要访问该变量，需要事先至少调用一次该变量所在的函数。

#### 2.2.2 函数的作用域

函数和变量相同，也分为全局作用域和局部作用域

局部（函数）作用域下声明的函数，只能在函数的内部访问

全局作用域下声明的函数，可以在任何位置下访问

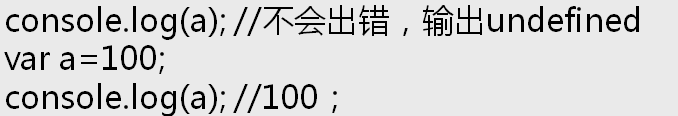
在程序执行前，声明的函数会提升到程序的最前面

#### 2.2.3 变量及函数的声明提升

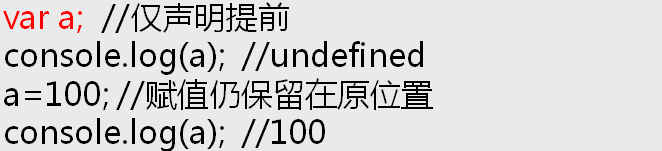
JS程序在执行前，会将所有var声明的变量和function声明的函数，预读到所在作用域的顶部（只是提升声明），但对变量的赋值还是保留在原来的位置。但编码中应尽量将变量的声明和函数的声明集中在当前作用域的顶部。

##### 2.2.3.1 变量提升的例子

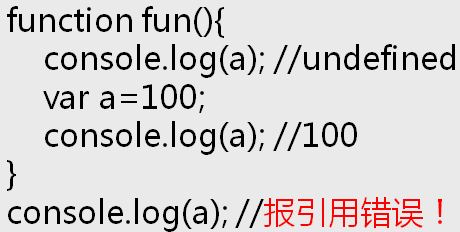
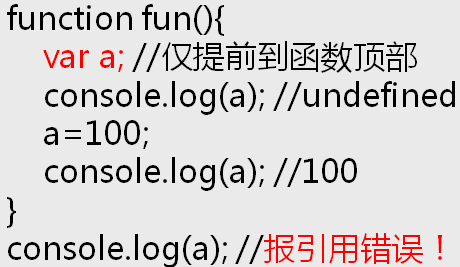
①全局变量提升



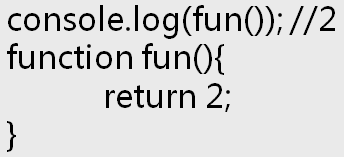
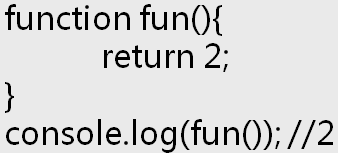
<==>



②局部变量提升

 <==> 

##### 2.2.3.2 函数提升的例子

 <==> 

**!注意：**函数提升会将函数名和函数内部逻辑一块儿提升！

#### 2.2.4 按值传递

按值传递指将实参变量的值复制一份副本给方法的形参变量。

JavaScript中为方法传递参数时，都是按值传递的。

如果传递的是原始数据类型，则在方法内修改参数变量的值，不会影响外部的实参变量。而引用类型在方法内修改参数变量的值，会影响外部的实参变量，这是因为引用类型复制的副本的值是该值的地址，在方法内部修改了该地址指向的堆内存中的值，当然会影响外部的实参变量——两者指向的是同一堆内存中的一个值。

obj1

obj2

0x4e00

0x4e00

0x4e00

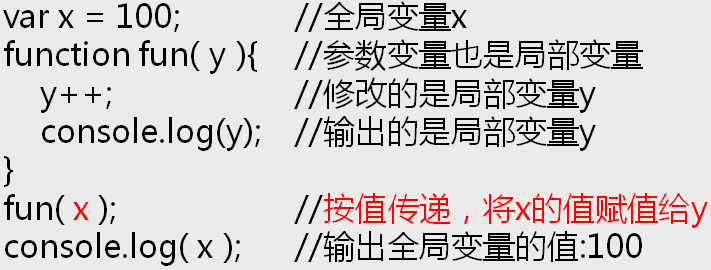
name:’cat’,

age:20

栈内存

堆内存

【例子】以下代码，改变形参值时并没有改变实参值



### 2.3 递归调用

#### 2.3.1 递归

一个函数在其定义中，直接或者间接调用自身

①一种算法，通常用于将一个复杂的问题层层转化为一个与原问题相似的较小问题来解决

②减少了代码量，但是执行速度需要考虑

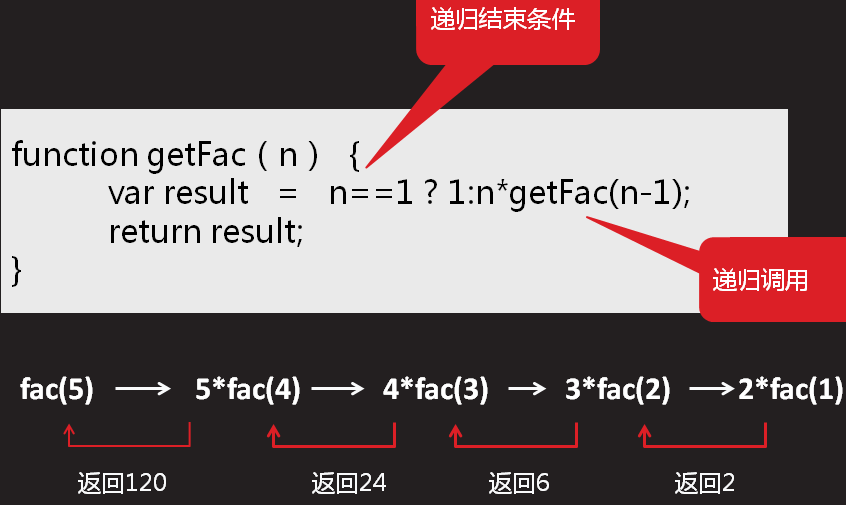
#### 2.3.2 递归的实现

①边界条件（递归结束条件）

②递归前进

③递归返回

#### 2.3.3 递归的原理



【说明】当满足边界条件时，递归返回；否则，递归前进

在函数的内部调用作甚，在使用的时候需要有结束条件

【练习】声明函数getSum(),使用递归（函数），来获取前n项之和

【练习】使用递归计算前n项的乘积

【课后任务】

①使用递归计算斐波那契数列的第n项

任何一项的结果 = 前两项的值相加

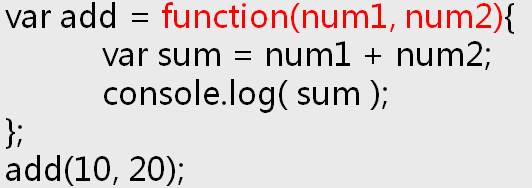
结束递归的条件：第一项和第二项返回的结果都是1

②预习匿名函数、回调函数和JS中的对象

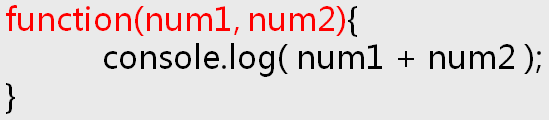
### 2.4 匿名函数

#### 2.4.1 匿名函数

匿名函数，就是定义时未直接指定名称的函数



也可以直接定义匿名函数（此方式声明后无法调用，只能用于其在内部自调用）

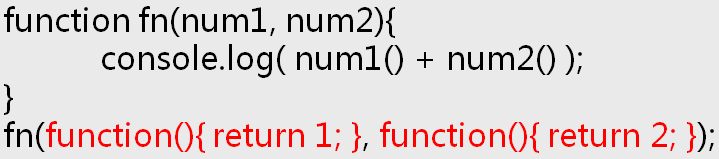


匿名函数的声明和调用见[2.1.4](#_2.1.4_匿名函数的声明和调用)

#### 2.4.2 匿名函数的应用

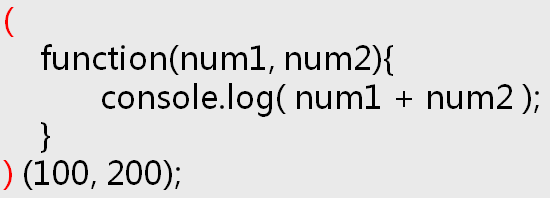
##### 2.4.2.1 回调函数

将一个函数作为另一个函数的参数叫做回调函数，且这个作为参数的函数可以是匿名函数



##### 2.4.2.2 自调函数

定义一个匿名函数即调用



作用：防止污染全局（全局污染）——全局的函数和全局的变量对整体的污染

##### 2.4.2.3 arguments对象

每个函数对象都有一个arguments属性；此属性只能在函数执行体内使用。arguments属性中保存着当前函数接收到的所有实际参数，故可以使用arguments属性处理可变数量的参数。

arguments对象的属性：

①length：返回实际传入的参数的个数

②callee：返回当前函数的引用（匿名函数可以使用该属性实现递归调用）

### 2.5 全局函数

①encodeURI() 对一个URI进行编码（主要是针对于里面的中文）

②decodeURI() 对已经编码的URI进行解码

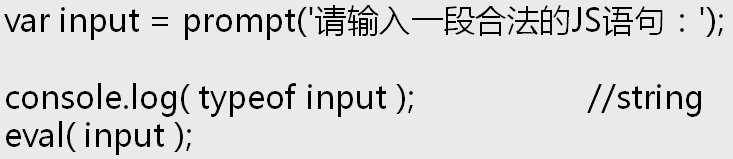
③isNaN() 检测一个值是否为NaN，是返回true，不是返回false

④isFinite() 检测一个值是否为有限值，是返回true，不是返回false(1/0返回无限Infinite)

⑤parseInt() 转换为整型

⑥parseFloat() 转换为浮点型

⑦eval() 执行字符串表达式的值；使用此函数使JS可以执行动态的JavaScript源代码



【练习】使用弹出提示框输入任意的数学计算'2+3\*5/6\*8-13',使用eval来执行这个字符串

### 2.6 运行在客户端的函数

> alert()弹出警示窗口

> prompt()弹出提示窗口

> confirm()弹出警示窗口(提供确定和取消选择按钮)

**!注意：**prompt返回的结果是字符串类型。

【练习】弹出两次提示窗口，分别输入数字，最后打印两个数字之和

## 三、对象

JS中的数据类型：原始数据类型和引用数据类型

引用数据类型：对象、函数

### 3.1 对象的定义

任何事物只要有属性，有功能都可以称为对象。

eg:电脑:大小、颜色、品牌、配置...；玩游戏、看视频、写代码、办公、作图...

JS中的对象是一组属性(property)和方法(method)的集合

#### 3.1.1 对象的通用定义

①对象是需求场景中的名词（如人、事、物）在程序中的表示

②JavaScript中，除了string、number、boolean、null、undefined等值类型数据之外，其他的数据都是对象，如数组、日期甚至函数等。

#### 3.1.2 ECMA-262标准中对象的定义

①属性的无序集合，每个属性存放一个原始值、对象或函数

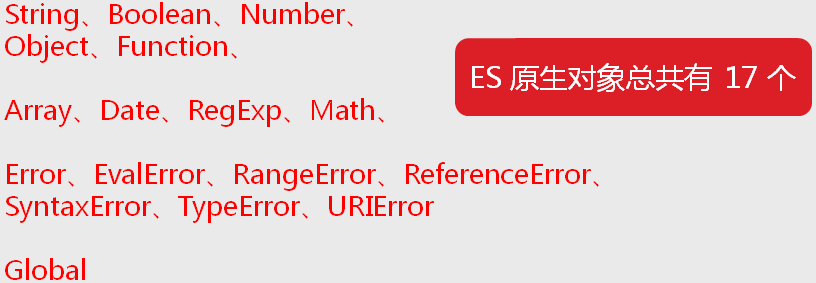
②对象是无特定顺序的值的数组

### 3.2 对象的分类

JS中的对象分为如下三大类：ES原生对象、宿主对象、用户自定义对象

#### 3.2.1 ES原生对象

ES原生对象，也称为内置对象，指JavaScript语言本身预定义的对象，在ECMAScript标准中定义，由所有的解释器厂家来提供具体实现，共17个。



**!注意：**由于标准的统一，ES原生对象的浏览器兼容性问题“不太大”

#### 3.2.2 宿主对象

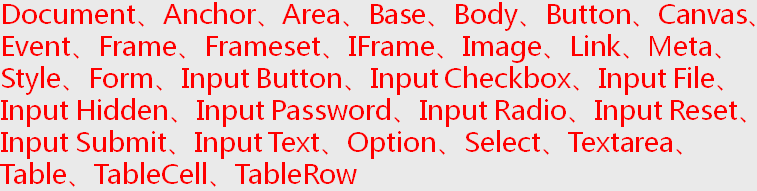
由JS解释器专门提供的扩展对象，由解释器厂家自定义并提供实现，故不同的解释器提供的扩展对象存在较大的差异。

通常浏览器中的JS解释器提供的宿主对象分为如下两类：

①BOM对象：Browser Object Model



②DOM对象：Document Object Model



除此之外，还有Node.js解释器也提供了对象，详见：https://nodejs.org/api

#### 3.2.3 用户自定义对象

由开发者自己定义的对象，兼容性问题需要由编写者注意

### 3.3 创建对象

此处创建的对象属于用户自定义对象，创建的方法主要有三种：对象直接量、内置构造函数、自定义构造函数。

#### 3.3.1 对象直接量

var person={

name:'jack',

sex:'girl',

job:function(){}

}

person.job();

①使用大括号创建对象{}

②多个属性之间用逗号隔开

③属性名和属性值之间用冒号隔开

④如果对象中出现了空格、-，属性名必须添加引号，其他的情况引号可以加可以不加

【练习】创建一本书对象，包含的属性书名bname、作者author、出版社publish、页码page、出版时间pub-time。

**!注意：**访问一个对象中不存在的属性，会返回undefined

#### 3.3.2 构造函数

①内置构造函数：

像Object和Array这样的原生构造函数，在运行时会自动出现在执行环境。

方法：var person = new Object();等

②自定义构造函数：

方法：var person = new 函数名();

function Person(){

this.name='jack';

this.sex='girl';

this.job=function(){

alert('program');

}

alert(‘123’);

}

var child=new person();

child.job ();

**!注意：**使用内置构造函数创建对象时，会自动执行该构造函数，故以上示例代码会两次弹出对话框。依次弹出123、program，原因是创建child对象时会先执行一次构造函数，执行其中的代码（不会打印job属性的匿名函数中的内容，因为没有调用该属性），打印program则是最后一行代码的缘故。其中this指的是调用函数的对象。

#### 3.3.3 工厂函数

使用构造函数创建很多对象的时候需要单独添加属性，为了便于使用这种方式，出现了工厂函数

工厂函数:解决代码重复的问题——封装

function createObj(name,age,...){

var obj = new Object();

obj.name = name;

obj.age = age;

...

return obj; }

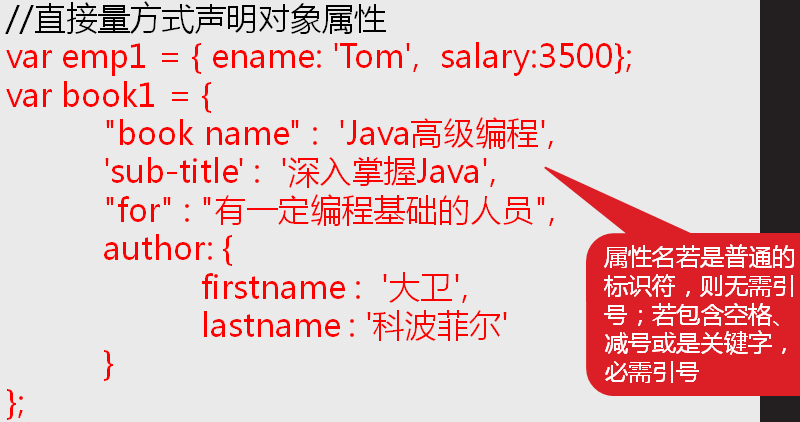
封装：把重复的代码放到函数中就成为函数封装；把重复的代码放到对象中就成为对象封装

【练习】创建一个电脑对象，品牌brand、大小size、颜色color,要求使用工厂函数

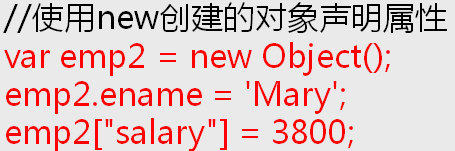
### 3.4 对象中的属性

#### 3.4.1 对象中属性的声明

①直接量中直接声明：



②使用new创建的对象声明属性



#### 3.4.2 访问属性

①对象.属性名 book.bname

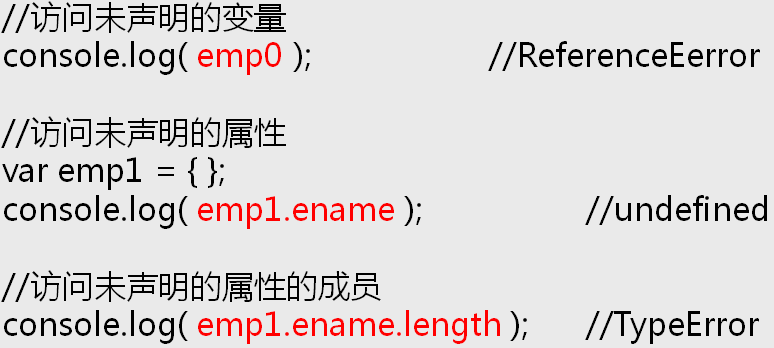
②对象['属性名'] book['bname']

属性名如果带有引号，则只能通过中括号方式访问

添加或者修改对象的属性

对象.属性名 = 属性值; 对象['属性名'] = 属性值;

**!注意：**如果要读取一个不存在的属性或属性值本身就是undefined，那得到的结果是undefined。属性访问错误的情况主要有如下三种：





#### 3.4.3 遍历属性

遍历：获取所有的属性及其值对应的属性值

for(var key in 对象){

//key表示要遍历的所有属性

//如果要获取属性值只能使用——对象[key]

}

**!注意：**只能遍历自定义的属性，不能遍历预定义的属性

【练习】创建一个汽车对象(品牌brand，类型type，颜色color，产地country),使用for-in遍历对象所有属性

#### 3.4.4 检测属性

①使用in关键字



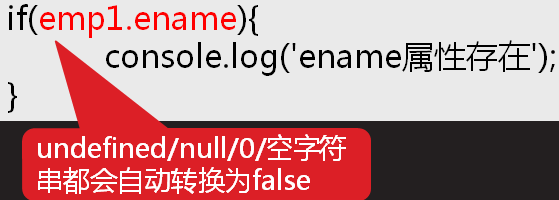
②使用对象的hasOwnProperty()方法



③使用undefined判断（注意是全等）



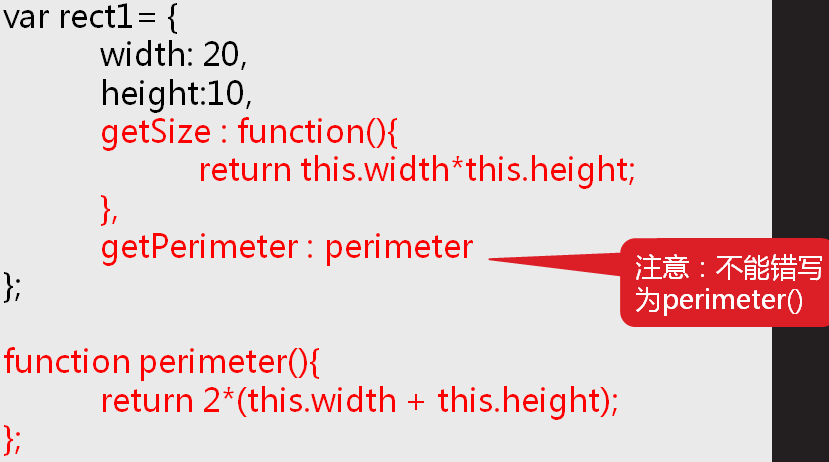
④在条件语句中直接判断



### 3.5 对象中的方法

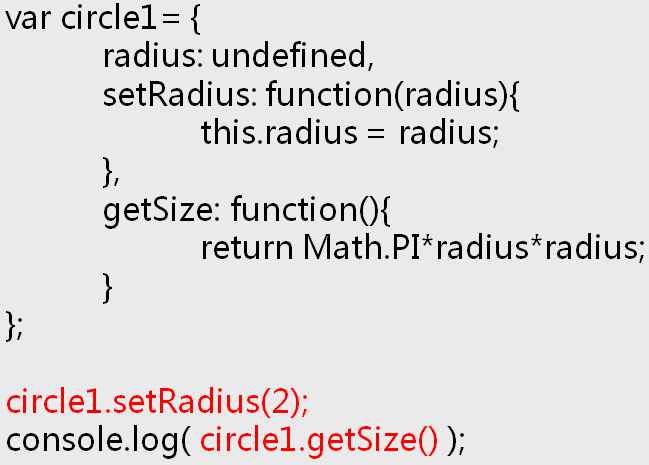
#### 3.5.1 定义方法

指对象可以实施的行为或者可以完成的功能



#### 3.5.2 调用方法

通过对象的引用调用对象的方法



成员方法的调用：person.say(); 成员方法中访问成员属性：this.name;

【课后任务】

1.复习今天的内容，完成思维导图

2.课后作业：创建一个圆对象，添加属性(半径r),添加成员方法(获取周长getLength(),获取面积getArea());

3.预习数组：数组的创建以及数组的方法

## 四、数组

数组是多个元素组成的集合——在一个变量名中存储多个值。在JS中，数组中元素的数据类型可以相同也可以不同，元素呈线性排列，下标范围[0,length-1]

### 4.1 创建和初始化数组

#### 4.1.1 创建数组

①数组直接量

var array = [元素1,元素2,...]

②内置构造函数

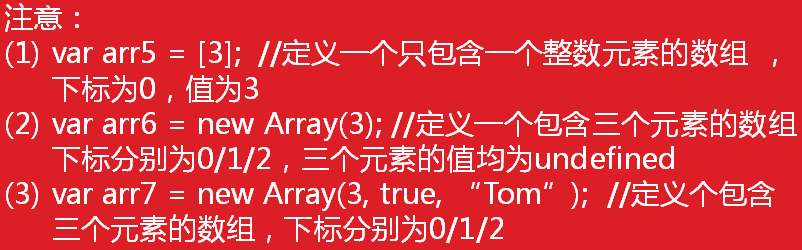
var array = new Array(元素1,元素2,元素3,...);

var array = new Array(3); //创建数组，长度是3，每一个数组元素的值是空（undefined）

【练习】

1.创建数组，保存5个国家的名称

2.创建数组，保存3个人的信息（姓名，性别，年龄）——>保存三个对象



#### 4.1.2 初始化数组

①数组定义时初始化

ex:var arr = [45,46,41];

var arr = new Array(45,46,41);

②先声明一个空数组，随后再向其中添加元素

ex:var arr = []; arr[0] = '市场部';

**!注意：**数组赋值给变量时，其实赋值的是地址；数组作为参数，也是[按值传递](#_2.2.4_按值传递)。

#### 4.1.3 null和undefined

null:特指一个变量不再指向任何对象地址

* 共同点：都是原始类型，保存在栈中变量本地
* 不同点：

- undefined表示变量声明过但未被赋过值，是所有未被赋值变量的默认值。一般不主动使用。例如：var a; //a被自动赋值为undefined

- null 表示一个变量将来可能指向一个对象，但目前暂时什么都没指向。一般用于主动释放指向对象的引用。例如：var arr = ['Scott','Alice']; arr = null;//释放指向数组的引用。数组对象成为“垃圾对象”，会被解释器回收。

### 4.2 访问数组

#### 4.2.1 设置数组的值

访问方式：数组名[下标] 下标从0开始

#### 4.2.2 获取数组元素的值

var cities = new Array('南京','杭州','青岛');

console.log(cities[0]); //南京

console.log(cities[3]); //undefined——不会抛出数组，下标越界异常

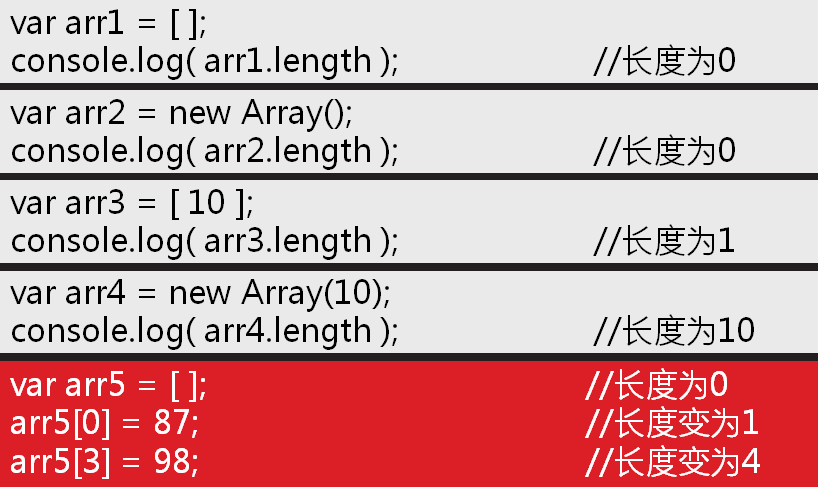
var scores = [95,88,100];

scores[2] = 98; //将值为100的元素重新赋值为98

scores[3] = 75; //在数组尾部添加一个新的元素，值为75

#### 4.2.3 获取数组元素的长度

使用length属性获取数组中元素的个数，即数组的长度



【重点】往数组中添加新的元素，即向数组末尾添加元素：arr[arr.length] = 新的元素;（arr.length不用自加喔）

【练习】信息录入，使用循环弹出提示框，每次输入一个姓名，把输入的姓名放入到数组中，并打印结果。直到输入quit，不再弹出提示框

#### 4.2.4 遍历数组 ★

* 通常使用for循环遍历数组元素（将数组的下标作为循环变量）：

也可以倒序（循环初始值i=arr.length-1）或者跳序遍历（循环的变化不加1，加上其他值）。

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

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

console.log(arr[i]);

}

* 使用for-in遍历数组元素：

【TIPS】for-in用于严格的依次遍历数组中的每一个下标

var numbers = [10,30,50,40];

for(var i in numbers){

console.log(i + "-" + numbers[i]);

}

### 4.3 数组的分类

数组分为索引数组和关联数组

* 索引数组：下标以数字的形式
* 关联数组：下标以字符串的形式

【重点】关联数组的length属性值无法获取其中元素的数量，所以关联数组只能使用for-in遍历，而索引数组可以用for循环或者for-in来遍历。

【练习】创建数组，保存学生的成绩，使用遍历数组来获取总分以及平均分是多少

【练习】求数组中的最大值 [5,10,105,32,79]

【练习】使用函数，传递一个参数，格式是数组，返回数组中的最小值

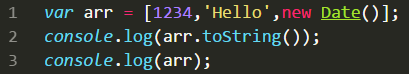
【练习】查找数组中指定元素的位置的下标

['刘备','曹操','袁绍','小乔','大乔'];查找'小乔'对应的下标

### 4.4 数组的常用方法（数组API）

js中提供的数组方法称为 数组API

* toString() 将数组转为字符串

* join([参数]) 返回字符串，参数表示元素之间的分割线，默认是逗号
* concat([参数]) 拼接多个数组
* slice(start,end) 截取数组元素,start表示开始，end表示结束（不包括end本身），[start,end)/[start,end-1];如果end为空，表示从start开始截取到最后；start和end都可取负值，start为-1指数组最后一个元素。
* splice(start,count,value...) 删除/插入/替换数组元素，start表示开始，count表示删除的数量，value表示删除后替换的元素；如果count为空，表示删除到最后,为0表示一个也不删除。返回被删除的项目，会改变原数组的值。
* reverse() 翻转数组元素，会改变原数组的值
* sort([参数]) 对数组元素进行排序，默认按照unicode码从小到大排列。

数字的大小排序，会改变原数组的值

sort(function (a,b){

//return a-b; //从小到大

//return b-a; //从大到小

});

* push(参数) 在数组的最后添加元素，原数组会发生变化
* pop() 删除数组最后一个元素，原数组会发生变化
* unshift(参数) 在数组的最前边添加元素，原数组会发生变化
* shift() 删除数组第一个元素，原数组会发生变化

**!注意：**以上七个操作后，原数组会发生变化；其余不会

【课后任务】

1.复习今天的内容，整理出思维导图

2.课后作业：

- 创建一个数组，数组包含4本书的对象（title,author,publish,page），遍历数组，以及对象，每一对象占一行

三国演义-罗贯中-1995-10-1-200

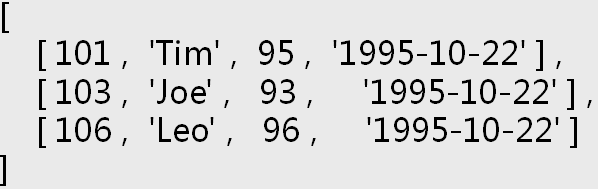
- 使用函数的形式，传递一个数组和一个元素，返回下标

3.查一查“冒泡排序”，总结原理

### 4.5 二维数组

#### 4.5.1 定义

二维数组从整体上来看，是一个数组，只是其中的每一个元素有事一个数组，即数组的数组，如：

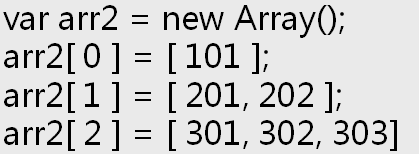


#### 4.5.2 创建二维数组

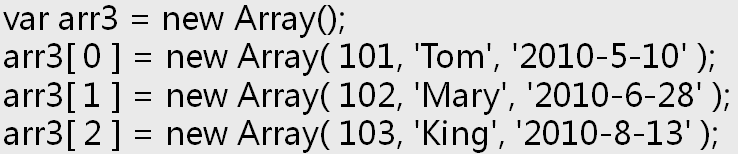
* 定义一个规整的二维数组（直接量）：



* 定义一个不规整的二维数组（直接量）：



* 定义一个形如“表格”的二维数组（内置构造函数）：



#### 4.5.3遍历二维数组

var arr = [[10,11,12],[20,21,22],[30,31,32]];

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

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

console.log(arr[i][j]);

}

}

## 五、基本包装类型（对象）

### 5.1 String对象

#### 5.1.1 创建字符串对象

* 创建引用类型字符串对象——typeof(carType)为Object



**!注意：**也可以使用直接量创建：var str = 'hello'; var str1 = String('hello');这两种方式创建的字符串对象，检测类型均为String。

#### 5.1.2 字符串API

* 将英文字符串转为大写

str.toUpperCase();

* 将英文字符串转为小写

str.toLowerCase();

* 查找某个下标对应的字符

str.charAt(index);

* 查找某个下标对应的字符的Unicode码

str.charCodeAt(index);

* 查找某个字符对应的下标，参数2开始查找的下标，找不到返回-1

str.indexOf(参数1,[参数2]);

* 查找某个字符最后一次出现位置的下标

str.lastIndexOf();

* 截取字符串

slice(start,end); start是开始的下标，end是结束的下标；范围[start,end)。如果end为空，从start到结尾；负数表示倒数第几个

* 截取字符串

substring(start,end); 和slice作用几乎一致，区别在于不支持负数写法，如果参数为负数，自动转成0

* 将字符串按照指定的字符转换为数组——切割字符串，参数指定的字符

split(separator)

* 连接字符串——连接两个或多个字符串，作用同“+”

concat(str1,str2,…,strn);

【练习】检测一个邮箱的格式是否合法？是否含有@；返回true，不是返回false

var str = '245616844@qq.com'

【练习】通过身份证号，来获取出生日期。打印\*\*\*\*年\*\*月\*\*日

123456 19960416 3465

【练习】通过邮箱地址来获取邮箱的用户名和域名

【练习】把一个英文单词的首字母转为大写，其他的字母转为小写

[jerry@sina.com](mailto:jerry@sina.com)

【练习】把一句英文每个单词的首字母全部转为大写，其它转为小写

we are family.

【练习】初始化4个英文字符，弹出提示框，如果输入不正确，继续弹框，直到输入正确，才会结束弹框。——不区分大小写

#### 5.1.3 转义字符

转义字符(\)，改变字符原来的意义

' -> \' 普通的引号

n -> \n 换行

t -> \t 制表符（键盘的tab键）



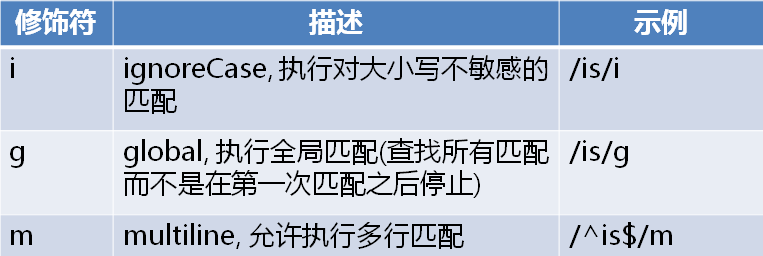
#### 5.1.4 匹配模式

* replace(参数1,参数2) 查找并替换，参数1是要查找的字符串，参数2是要替换进去的字符串

str.replace(/china/ig,'中国');

i --> ignore 忽略大小写;g --> global 全局查找

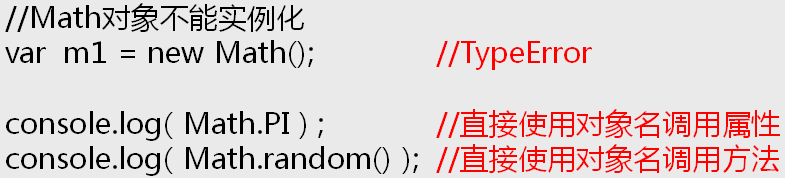
* match(参数) 查找某个字符串，返回的结果是数组
* search(参数) 查找某个字符串，返回查找到的第一个字符串的下标值；如果找不到，返回-1。只能使用i,不能使用g（找到后不再向后查找）



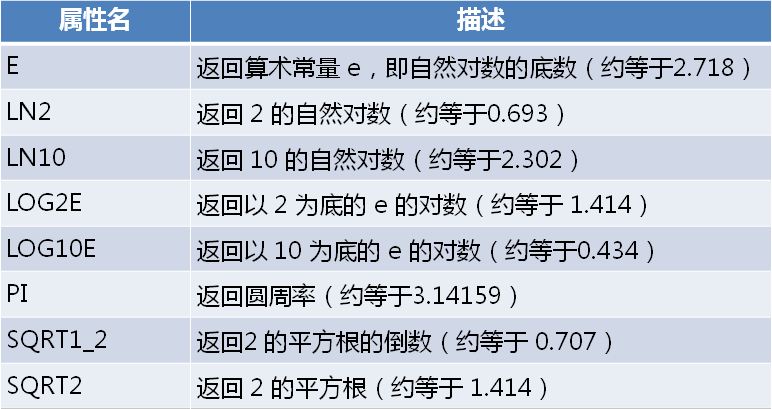
**!注意：**修饰符可以组合使用

### 5.2 Math对象

Math对象没有对应的构造函数，无法创建它的实例，故调用其属性和方法时，直接使用Math对象名即可。



#### 5.2.1 Math对象的属性



**5.2.2 Math对象的方法**

|  |  |
| --- | --- |
| **方法名** | **描述** |
| Math.abs(x) | 取绝对值 |
| Math.ceil(x) | 向上取整 |
| Math.floor(x) | 向下取整 |
| Math.exp(x) | 返回e的指数 |
| Math.log(x) | 返回数的自然对数（底为e） |
| Math.random(x) | 取随机数，范围0~1 |
| Math.round(x) | 四舍五入取整 |
| Math.max(x,y) | 返回x和y中的最大值 |
| Math.min(x,y) | 返回x和y中的最小值 |

**【课后任务】**

1.复习今天的内容，完成思维导图

2.课后作业：

使用数组保存10个人名（下标是数字），每次随机取一个（随机取下标） 0-9

3.预习Date对象以及ES6

**5.3 Date对象**

**5.3.1 获取Date对象**

* 获取系统当前的时间

var d = new Date();

* 获取某个具体时间的Date对象，月份范围0~11

var d = new Date(2018,10,20,10,20,30); //实际是'2018-11-20 10:20:30'

* 获取具体某个时间的Date对象，月份范围1~12

var d = new Date('2018-11-20 10:20:30'); //实际也是'2018-11-20 10:20:30'

* 获取具体某个时间的Date对象，月份范围1~12

var d = new Date('2018/11/20 10:20:30'); //实际也是'2018-11-20 10:20:30'

* 获取距离计算机元年的Date对象

var d = new Date(24\*60\*60\*1000);

**5.3.2 获取Date对象下的时间信息**

getFullYear()/getMonth()/getDate()/getHours()/getMinutes()/getSeconds()/getMilliseconds()/getDay()

获取年/月/日/时/分/秒/毫秒/星期

**!注意：**月范围'0~11'代表'1月~12月'；星期范围'0~6'代表'星期日~星期六'

**5.3.3 设置Date对象下的时间信息**

setFullYear()/setMonth()/setDate()/setHours()/setMinutes()/setSeconds()/setTime()

**【练习】**计算合同的到期时间等:

Q1:员工入职时间2018-5-10，合同的有效期为3年，求合同到期的时间是多少？

Q2:在合同到期前一个月续签合同，如果遇到周末，提前到上一个周五，求续签时间？

Q3:要求在续签前一周向员工发起续签提醒,求续签时间

**5.3.4 本地化Date对象**

上述获取到的Date时间对象都是0时区即格林尼治时间，以下是将该时间对象转化为本地（东八区）的时间对象的方法：

toLocaleString() //年-月-日 时：分：秒

toLocaleDateString() //年-月-日

toLocaleTimeString() //时：分：秒

**5.4 Number对象**

获取计算机存储的最大/小值

Number.MAX\_VALUE; //数值型保存的最大值

Number.MIN\_VALUE; //数值型保存的最小值

(0.1+0.2).toFixed(2); //保留小数点后2位

**5.5 Boolean对象、**

将数据转换为布尔型的值

new Boolean(值) / Boolean(值)

!!值 ---> 推荐写法

**六、错误处理**

**6.1 错误类型**

语法错误 SyntaxError:出现语法错误，整个程序不执行

引用错误 ReferenceError:使用了不存在的变量，会影响后续代码的执行

类型错误 TypeError:错误的使用了括号；会影响后续代码的执行

范围错误 RangeError:参数超出了范围；会影响后续代码的执行

**6.2 错误处理方式**

异常(错误)处理方式：因为引用错误，类型错误，范围错误会影响后续代码的执行，在程序中常用try-catch语句进行错误处理。

try {

//尝试执行的代码

} catch(err) { //捕获异常信息，err对象里存放错误信息

//出现错误后要执行的代码——错误解决方案

}

[**七、ES6新特征(了解)**](#_六、ES6新特性)

ECMAScript又称ES2016、ES2017、ES2018

**7.1 块级作用域**

在{}之间使用let声明的变量，只能在{}内部访问到，这个大括号所在的作用域就称为块级作用域。

**7.2 箭头函数**

回调函数中传递的匿名函数可以使用箭头函数:

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

return a-b;

});

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

//如果大括号之间只有一行，并且含有return，才可以简写

**7.3 模板字符串**

使用反引号``所包含的字符串就是模板字符串，在模板字符串之间可以放任意字符，例如：引号，括号，运算符...

在模板字符串中使用变量或其他的值 —— ${变量名称}

**7.4 函数的扩展**

在ES6的新标准中，函数的参数可以给默认值以作扩展：

function fn(a,b,c=1){

return a+b+c;

}

fn(10,15,25); //50

fn(10,15); //26

**【课后任务】**

1.复习今天的内容，绘制思维导图

2.课后作业：

获取自己生日的date对象，查看是否为周末，如果是周末就提前到上一个周六，打印日期是多少？

3.预习nodeJS中模块——概念、如何使用

**第四章 Node.js**

**一、Node.js概述**

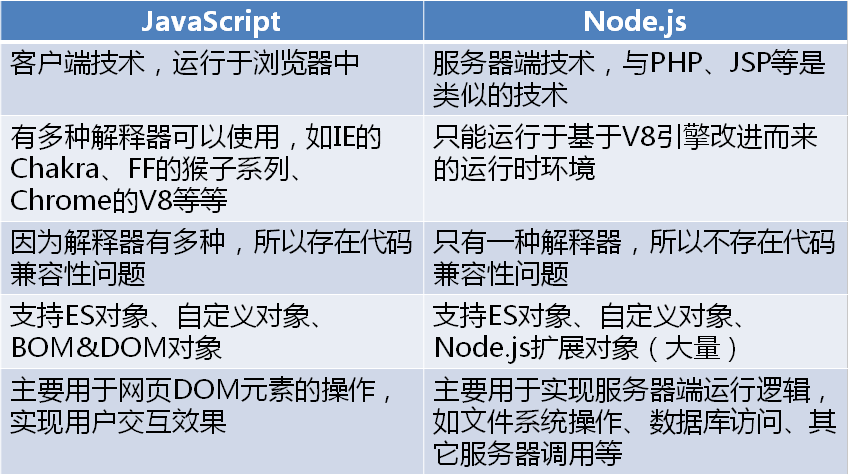
**1.1 Node.js**

**1.1.1 概念定义**

JavaScript：1995年由Netscape公司推出，后经由ECMA统一标准的脚本语言。通常狭义上理解的JS是指在浏览器内置JS解释器中运行的，主要作用是操作网页内容，完成用户交互。

Node.js：基于Google V8引擎的JS运行时环境，其运行完全脱离浏览器，可以编写独立的服务器端程序。主要用途为文件读写、网络访问、加密压缩、数据库操作等等。

**1.1.2 与JavaScript的区别**



除上表中的区别之外，Node.js和JavaScript的语法除宿主对象不同外，几乎完全一致。也就是说，两者均有原生对象和用户自定义对象，而Node.js不支持JavaScript的宿主对象（DOM&BOM）。

**!注意：**Node.js与JavaScript仅仅只是语法一致，功能完全不一致。

**1.1.3 Node.js的运行模式**

交互模式(Read-Evaluate-Print-Loop,REPL)：读取用户输入，执行运算，打印结果，继续下一次循环。在此模式下，Node.js自带的模块无需手动引入。

脚本模式：将所有语句编写在一个独立的脚本文件中，一次性执行。在此模式下，除了全局对象及其相关成员外，所有其他模块中声明的对象和方法必须使用require()引入。

**1.1.4 Node.js的特点**

Node.js的特点是：简单、避免过度设计；单线程的逻辑处理；非阻塞的异步IO处理；事件驱动编程；无锁机制，不会产生死锁；支持数万个并发连接。

基于此，Node.js适合IO密集型应用，不适用于CPU密集型应用。

**1.2 全局对象**

Node.js的全局对象为global，而JavaScript的全局对象为window。

* global

在交互模式下声明的全局函数和变量都是global对象的成员——造成了全局污染

在脚本模式下声明的全局函数和变量都是window对象的成员即都是局部作用域下——避免了全局污染

**!注意：**脚本模式下，每个.js文件都是一个独立的模块对象，其中创建的“全局变量和函数”都是该对象构造方法内的局部成员。

* console对象

console.log() 标准打印

console.info() 打印信息

console.warn() 打印警告

console.error() 打印错误

**!注意：**log()等只是console对象下定义的一个方法

**【拓展】**计算程序的运行时间

console.time('label'); 开始计时

//检测的程序

console.timeEnd('label'); 结束计时

**【练习】**检测while和do-while循环10000次，看哪个更快

* process对象(了解)

process.arch 查看CPU架构类型（'x64'）

process.platform 查看操作系统的类型（'win32'）

process.version/versions 查看nodejs的版本

process.env 查看当前计算机的环境变量

process.uptime() 查看nodejs的运行时间（单位：s）

process.memoryUsage() 查看内存的使用情况

process.pid 查看当前进程的编号

process.kill() 杀死某个编号的进程

* buffer对象

buffer对象是global对象下的一个子对象；用于将数据存储在缓冲区中，通常用于较小数据的存储。

Buffer.alloc(参数1,[参数2]) 创建buffer

参数1设置缓冲区大小，参数2是缓冲区存储的值，格式是字符串

**1.3 全局函数**

**1.3.1** [**定时器**](#_三、定时器)

**1.3.1.1 一次性定时器**

* 开启定时器：

var timer = null;

timer = setTimeout(func，delay);

func：是回调函数，间隔时间到了所执行的命令

delay：间隔的时间，单位毫秒

setTimeout():返回已经启动的定时器对象

* 清除定时器：

clearTimeout(timer);

**1.3.1.2 周期性定时器**

* 开启定时器：

var timer = null;

timer = setInterval(func，delay)

setInterval()中的两个参数及返回值和一次性定时器完全一致

* 清除定时器：

clearInterval(timer);

**【练习】**创建周期性定时器，打印10次'hello world'

**1.3.1.3 事件触发定时器(了解)**

* 本次事件循环结束时立即执行的定时器

process.nextTick(function(){});

* 下次事件循环立即执行的定时器

setImmediate() / clearImmediate();

**!注意：**客户端JS不支持事件触发定时器

**1.3.2 其他常用全局函数**

decodeURI()/decodeURIComponet()/encodeURI()/encodeURIComponent()/escape()/unescape()/parseInt()/parseFloat()/isNaN()/isFinite()/eval()

**1.4 同步/异步**

同步：程序按照顺序执行，先执行前面的，后执行后面的。后边的程序始终等待着前边的程序执行完，才执行

异步：程序在运行过程中，遇到异步处理函数（一般是系统默认比较耗时的函数），就会把这些函数先放入任务栈，等待其余所有程序执行完毕后，任务栈才向系统发送请求，开始执行任务栈中的程序。

> 异步的特点：使用了回调函数

**1.5 包和npm**

**1.5.1 概述**

Npm(Node Package Manage): 包管理工具

包：是一个目录模块，含有任意的js文件，含有一个命名为package.json的包说明文件。

npm官网：npmjs.com 使用npm命令

淘宝npm镜像：

> npm.taobao.org 使用cnpm命令，需要单独安装

> npm install cnpm –global

下载包：npm install 包名称 cnpm install 包名称

**1.5.2 npm包管理命令**

初始化package.json文件，用于记录包的安装信息。

> npm init –y

把包以运行依赖进行安装

> npm install 包名称 –save

把包以开发依赖进行安装

> npm install 包名称 --save-dev

自动安装package.json文件中所保存的包信息

> npm install

**【课后任务】**

1.复习今天的内容，整理思维导图

2.课后练习：

创建一个计算圆周长和面积的模块（功能模块），另一个模块导入功能模块，实现圆周长和面积的计算

3.预习http协议

4.自学拓展：线程和进程

**二、模块**

**2.1 模块概述**

在ES6之前JavaScript中没有模块的概念，所有要执行的JS文件运行在同一空间里，容易产生“对象覆盖”的问题。Node.js采用“Module模块”来划分不同的功能对象。每个模块内声明的变量和函数都属于该“模块作用域”。每个模块可以导出(exports)自己内部的成员供其他模块使用，也可以引入(require)并使用其他模块中导出的成员。Node.js启动时运行的第一个模块称为“主模块”。

即，nodejs中模块文件的本质，是一个构造函数方法内的执行体，其内部定义的所有变量及方法都属于它自己的局部成员。

(function(exports,require,module,\_\_filename,\_\_dirname){

//程序员写的代码

})

注释外的代码是nodejs自动添加的代码

**>** \_\_filename 文件的完整路径和文件名称

**>** \_\_dirname 导入一个模块

**2.2 导入/导出模块**

require 导入一个模块 例如：require('./07\_2.js');

module.exports 导出模块，导出类型是一个对象

**!注意：**require引入第三方模块机制是，首先查找当前目录下有无node\_modules，如果没有就往上一级目录寻找，直到找至根目录。

**2.3 模块分类**

Node.js中的模块主要分为：

> 核心模块：由Node.js提供，被编译进二进制文件，可以被解释器直接使用，加载速度最快。

> 第三方模块：基于核心模块，由组织和个人提供。

> 自定义模块：分为文件式自定义模块、目录式自定义模块。

**2.4 加载模块**

带路径的文件模块、不带路径的文件模块、带路径的目录模块、不带路径的目录模块

|  |  |  |
| --- | --- | --- |
|  | **带路径的** | **不带路径的** |
| **文件模块** | require(‘./circle.js’)  通常用于用户自定义的模块 | require(‘querystring’)  用于官方定义的核心模块 |
| **目录模块** | require(‘./2’)  在2目录下存在一个index.js或者package.json文件声明main属性 | require(‘3’)  要求把3目录放在node\_modules中。  通常用于第三方模块 |

**2.5 核心模块**

**2.5.1 querystring模块**

查询字符串模块

parse 将查询字符串解析为对象

stringify 将对象转换为查询字符串

**2.5.2 url模块**

parse 将url解析为对象

format 将对象转换为url

**2.5.3 fs模块**

fs模块，即文件系统摸块，可实现对文件（目录）操作（创建，写入，删除，读取…）

异步函数：非阻塞模式，是通过回调函数来获取结果（通常用于一些比较耗时、耗内存的代码）

同步函数：阻塞模式，使用过函数的返回值来获取结果

**2.5.3.1 查看文件状态**

在fs模块中使用stat/statSync来查看文件的状态：

> isFile() 查看状态是否为文件

> isDirectory() 查看状态是否为目录

> existsSync() 判断文件是否存在，结果返回一个布尔值

//文件系统模块

const fs = require('fs');

//1.查看文件状态

//异步函数使用回调函数来获取结果

fs.stat('./1.txt',(err,stats)=>{

    //err:查看失败所显示的信息

    //stats:把查看到的信息放入到stats中

    if(err){ //如果有错误，抛出错误，程序不会往后执行

        throw err;

    }

    console.log('查看成功');

    console.log(stats);

    //文件时目录还是文件

    console.log(stats.isFile()); //是否为文件

    console.log(stats.isDirectory()); //是否为目录

});

//使用同步函数查看文件状态

var res = fs.statSync('./1.txt');

console.log(res.isFile());

**2.5.3.2 目录操作**

* mkdir() 创建目录

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

if(err) throw err;

});

* rmdir() 删除目录

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

if(err) throw err;

});

* readdir() 读取目录

fs.readdir('./mydir',(err,files)=>{

if(err) throw err;

console.log(files); //结果是数组

});

**2.5.3.3 文件操作**

* readFile() 读取文件

fs.readFile('./mydir/1.txt',(err,data)=>{

if(err) throw err;

console.log(data.toString()); //读取的结果是buffer数组

});

* writeFile() 写入文件，不存在先创建，存在则清空再写入

fs.writeFile('./data.txt','hello',(err)=>{

if(err) throw err;

console.log('写入成功');

});

* appendFile() 追加写入文件，如果文件不存在则创建，如果已经存在，则会在文件末尾写入
* unlink() 删除文件

fs.appendFile('./data.txt','world\n',(err)=>{

if(err) throw err;

console.log('写入成功');

});

fs.unlink('./data.txt',(err)=>{

if(err) throw err;

console.log('删除成功');

});

**【练习】**检查文件’./mydir’的状态，如果不存在这个目录，创建这个目录。

**【练习】**实现计数器功能，每运行一次，数字加1

**【解题思路】**

**>** 如果文件number.txt不存在，创建文件，写入1

**>** 读取number.txt，把读取的结果+1重新存入number.txt之后打印出来

**【课后任务】**

1.复习，整理思维导图；

2.把url中查询字符串的值，并且把这些值写入str.txt（如果不存在创建）。

http://codeboy.com/list.html?lid=10&pname=apple&price=5000

3.http协议、预习express

**2.5.4 http模块**

**2.5.4.1 初识http协议**

在学习http模块之前，先了解了解http协议相关的基础知识：

HTTP协议是web服务器和浏览器之间通讯所遵循的协议，其中包含请求和响应两个步骤。

请求的方法：

> get 从服务器获取内容（默认）

> post向服务器发送内容（注册、登录）

请求的状态码：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1\*\* | 2\*\* | 3\*\* | 4\*\* | 5\*\* |
| 正在响应，还没有结果 | 成功响应 | 响应重定向到其他的url | 客户端请求错误 | 服务器运行错误 |

**2.5.4.2 http模块**

http模块可用于发送请求和构建web服务器：

* 发送请求：

> 引入http模块

> 发送请求(get方法)

http.get('http://www.codeboy.com',(res)=>{

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

console.log(buf.toString());

})

});

**!注意：**get()的参数1可以是url也可以是一个接口；res的on事件：当有数据响应的时候触发，获得的数据格式是buffer，需要toString(）进行转换

* 构建web服务器

> 引入http模块

> 创建服务器

> 开启监听

> 接受浏览器或其他客户端的连接请求

// 1.引入http模块

var http = require('http');

// 2.创建web服务器

var server = http.createServer();

// 3.开启监听(3000是端口号)

server.listen(3000,()=>{

console.log('web服务器构建成功!');

});

// 4.当有浏览器或其他客户端发送请求的时候，触发这个on事件

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

//req 浏览器请求的对象

//req.method 获取请求的方法（GET/POST...）

//req.url 获取请求的url

问题：http模块只能发送文本，不能发送文件。

解决：采用express框架构建服务器。

**三、Express框架**

//req.headers 获取请求的头信息

//res 服务器响应的对象

//writeHead(参数1，参数2) 响应的头信息 可用于重定向

//writeHead()的参数1是响应的状态码，参数2是传入的头信息对象

//write 设置响应到浏览器页面的内容

//end 结束响应

});

http模块本身有一些弊端（提供的API不够简单易用），项目中常用express框架快速构建web服务器。

Express是一个基于Node.js http模块的极简、灵活的web应用开发框架，提供一系列强大的特性，帮助开发者创建各种web和移动设备应用。

中文官网：expressjs.com.cn

下载安装：npm install express –save

**3.1 构建web服务器**

下载安装好express框架后，便可开始使用。其作用一便是可以构建web服务器，具体有两种使用方式：

**3.1.1 将express作为请求处理函数**

具体步骤如下：

// 1.引入http模块

const http = require('http');

// 2.引入express框架

const express = require('express');

// 3.创建express应用程序

var app = express();

// 4.借助http模块构建web服务器并开启监听

http.createServer(app).listen(3000,()=>{

console.log('服务器构建成功');

});

// 定义express中间件和路由

**3.1.2 将express直接作为服务器**

可以直接使用express框架构建web服务器：

// 1.引入express框架

const express = require('express');

var app = express();

// 2.使用express构建服务器并开启监听

app.listen(3000,()=>{

console.log('web服务器构建成功');

});

// 3.使用express发送请求

app.get('/index',(req,res)=>{

console.log('获取到了浏览器发送的请求');

//req 请求的对象

//res 响应的对象

});

**3.2 路由**

**3.2.1 路由定义**

http客户端发来请求消息，服务器端程序根据请求的方法和请求uri确定一个相应方法，在Express中成为定义了一个“路由”。

路由分为三部分：请求的方法（get/post…），请求的url(/login)，响应的方法（回调函数）：



**【练习】**登录的路由——请求方法get，请求的url是/login

**3.2.2 请求的对象(req)**

Express处理请求消息的异步回调函数中(即上文图示中的路由句柄)，第一个形参是一个请求消息的描述对象，从此对象中可以读取请求消息中的数据。

> req.url 请求的url

> req.method 请求的方法

> req.query 获取请求的url中的查询字符串——返回一个对象

> req.headers 获取请求的头信息

> req.body 请求主体(需要依赖特定中间件)

> req.params 路由参数对象

**3.2.3 响应对象(res)**

Express处理请求消息的异步回调函数中(即上文图示中的路由句柄)，第二个形参用于设置向客户端输出的响应数据。

> res.send() 发送一段文本到浏览器

> res.sendFile() 发送一个文件到浏览器，文件使用绝对路径（\_\_dirname）

> res.redirect() 响应重定向

**【练习】**创建一个路由，请求的方法get，请求的url为reg，发送一个注册文件给浏览器

**【课后任务】**

1.完成今天的内容，完成思维导图

2.课后练习：使用express构建web服务器，创建路由

get'/index'发送文本'this is index page'

get'/shopping'发送文件

get'/'重定向到'/index'

3.预习express中的‘中间件’和mysql模块（npmjs.com）

**3.2.4 请求的方法**

**3.2.4.1 GET请求**

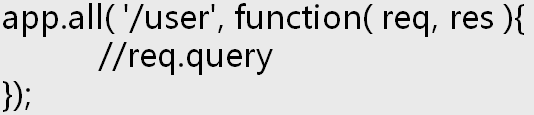
根据HTTP协议规定，GET请求表示客户端想“获得”指定资源。可以使用express路由方法中的get()方法处理客户端提交的GET请求。

**3.2.4.2 POST请求**

根据HTTP协议规定，POST请求表示客户端想“提交并保存”数据到服务器，所以都是有请求主体(body)的。可以使用express路由方法中的post()方法处理客户端提交的POST请求。

**3.2.4.3 处理所有请求**

Express路由方法中的all()方法可以用于处理客户端提交的GET、POST、PUT、DELETE等任意类型的请求消息。



**3.2.4.4 GET/POST请求的区别**

> post只能使用表单提交的时候（现阶段）；而get可以通过表单提交，也可以在浏览器地址栏直接输入url

> post不会在地址栏暴露输入的数据，相对安全；get会在地址栏显示输入的数据，相对不安全。

> post的执行速度相对于get较慢

> 一般对一些安全较高的表单使用post，例如登录、注册等；而对于一些安全较低的使用get，例如搜索关键字、查询。

**3.2.5 路由传参**

客户端在向服务器提交请求数据时，除了可以使用传统的查询字符串方式，Express还允许在请求URI中包含请求参数。

app.get('/list/:lid/:uname',(req,res)=>{

//lid就称为参数,形参

console.log(req.params);

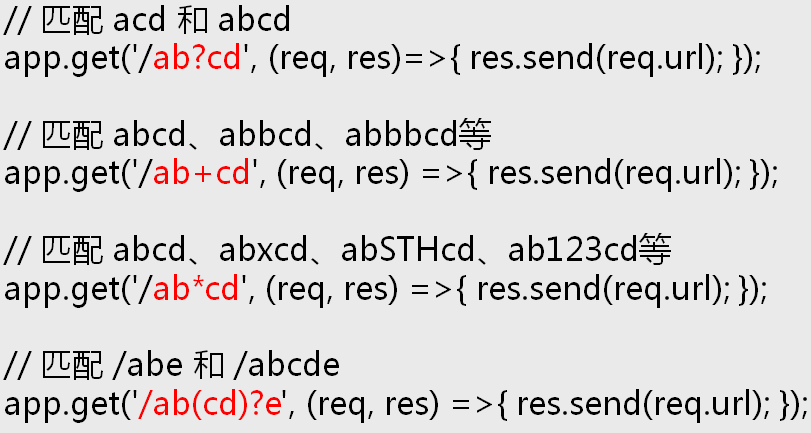
res.send('这是商品的详细信息，编号：' + req.params.lid);

});

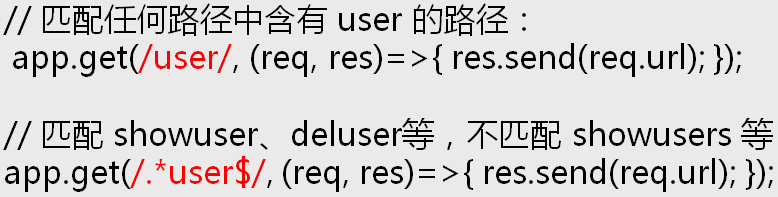
**【练习】**购物车 /shopping get，把商品的价格和商品的数量（num）以参数的形式传递给服务器，把接收的参数显示在浏览器中。

**3.2.6 路由路径匹配**

* 使用带有模式字符串的路由路径



* 使用正则表达式的路由路径



**3.3 路由器**

**3.3.1 路由器定义**

路由器，是一种可同时挂载多个路由的对象；这些路由功能相关，而且一般有着相同的路由地址前缀(如：/product/list和/product/details)。

把相关的路由归纳到一起，如：

用户模块：注册、登录、列表、修改、删除/user/delete、检索

商品模块：列表、详情、修改、删除/product/delete

这样一来就便于管理路由。

**3.3.2 挂载路由器**

// product.js

const express = require('express');

// 创建路由器

var router = express.Router();

// 将路由添加到路由器中

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

res.send('这是商品的列表界面');

});

router.get('/details',(req,res)=>{

res.send('这是商品的详情界面');

});

// 导出路由器

module.exports = router;

路由器模块编写完成后，可在主模块(如app.js)中加载并使用。

const product = require('./routes/product.js');

…

app.use('/product',product); //为路由器分配挂载地址

**3.4 中间件**

**3.4.1 概述**

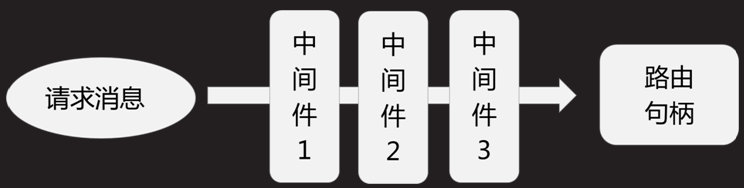
中间件(Middleware)是一个函数，它可以访问请求对象(req)，响应对象(res)和web应用中处于请求-响应循环流程中的其他中间件。

中间件主要是为主逻辑所服务，它可以①执行任何代码；②修改请求和响应对象；③终结请求-响应循环；④调用堆栈中的下一个中间件。

**!注意：**中间件不能单独使用，必须要配合其他的中间件或路由来实现一个完整的业务逻辑。

**3.4.2 中间件拦截路由**

可以在路由句柄之前或之后声明多个中间件函数，组成一种链式结构。请求消息会按照声明的顺序依次提交给每个中间件函数。中间件或路由句柄内部可以控制next()回调函数的执行以决定是否继续执行下一个中间件函数。



拦截所有的路由:app.use(fn);

拦截特定的路由（/list）:app.use('/list',fn);

**【练习】**使用中间件实现计时器，初始化一个变量用于记录访问次数——0；在前置中间件中实现访问次数+1，把+1的结果响应给浏览器。

**3.4.3 中间件分类**

中间件大致上可以分为以下五种类型：①应用级中间件 app.use(fn)；②路由级中间件——路由器的使用；③内置的中间件；④第三方的中间件；⑤错误级的中间件

**3.4.3.1 应用级中间件**

使用app.use()和app.METHOD()函数将应用层中间件绑定到应用程序对象的实例，其中METHOD是中间件函数处理的请求的小写HTTP方法（例如GET、PUT或POST）。

* 没有挂载路径的中间件，应用的每个请求都会执行该中间件，即拦截所有路由

app.use(fn); //function fn(req:any,res:any,next:any):void

app.use((req,res,next)=>{

//...

next();

});

* 挂载到/user/:id的中间件，任何指向/user/:id的请求都会执行它，即拦截特定路由

app.use('/user/:id',(req,res,next)=>{

//...

next();

});

* 路由和句柄函数（中间件系统），处理指向 /user/:id 的GET请求

app.get('/user/:id',(req,res,next)=>{

res.send('....');

next();

});

**3.4.3.2 路由级中间件**

路由器层中间件的工作方式与应用层中间件基本相同，差异之处在于它绑定到 express.Router() 的实例。

var router = express.Router();

使用 router.use() 和 router.METHOD() 函数装入路由器层中间件。

* 没有挂载路径的中间件，通过该路由的每个请求都会执行该中间件

router.use((req,res,next)=>{

//...

next();

});

* 往路由器中添加路由'/user'

router.get('/user',(req,res,next)=>{

//...

next();

});

* 导出该路由器

module.exports = router;

* 在appp.js主模块中导入路由器

const user = require('./routes/user');

* 在appp.js主模块中，将userl路由器中的路由挂载至/user目录下

app.use('/user', user);

**3.4.3.3 内置中间件**

express唯一的内置中间件——static托管静态资源。

问题：前文中的sendFile()方法只有再请求的时候，才会发送文件到浏览器 。

解决：使用static静态资源托管。此函数基于serve-static，负责提供Express应用程序的静态资源。

app.use(express.static('目录路径'));

**!注意：**可以同时托管多个静态目录，如果两个目录中有相同的文件名称，显示第一个目录下的文件。

**3.4.3.4 第三方中间件**

使用第三方中间件向 Express 应用程序添加功能。安装具有所需功能的Node.js模块，然后在应用层或路由器层的应用程序中将其加装入。

在node下安装第三方中间件bodyParser

$ npm install body-parser

* 引入第三方中间件

const bodyParser = require('body-parser');

* 调用bodyParser中间件

app.use(bodyParser.urlencoded({

extended:false

}));

* 获取post请求表单的数据

req.body //结果是一个对象

**3.4.3.5 错误级中间件**

错误处理中间件函数的定义方式与其他中间件函数基本相同，差别在于错误处理函数有四个自变量而不是三个，专门具有特征符 (err, req, res, next)：

app.use(function(err, req, res, next) {

console.error(err.stack);

res.status(500).send('Something broke!');

});

**【课后任务】**

1.复习今天的内容，完成思维导图

2.课后练习：创建一个register.html注册文件（用户、密码、邮箱、电话）；把这个文件作为静态资源托管，点击提交，使用post请求将数据发送给服务器；服务器使用body-parser中间件来获取数据，发送到浏览器‘注册成功，你好+【用户名】’。

3.复习mysql中的sql语句。

**四、MySQL模块**

此模块可以使Node.js中能够使用mysql数据库。

**4.1 安装MySQL模块**

npm install mysql

**4.2 MySQL模块的使用**

**4.2.1 普通数据库连接**

//连接mysql数据库服务器

var mysql = require('mysql'); //引入MySQL模块

var connection = mysql.createConnection({ //创建普通连接

host: '127.0.0.1',

port: 3306,

user: 'root',

password: '',

database： 'tedu' //进入mysql服务器后，进入数据库

});

connection.connect(); //建立连接

connection.query(); //数据库语句

connect.end(); //断开连接

**4.2.2 使用连接池连接**

* 创建pool.js文件，连接mysql数据库服务器并导出

//连接mysql数据库服务器

const mysql = require('mysql'); //引入MySQL模块

var pool = mysql.createPool({ //配置连接

host: '127.0.0.1', //数据库主机地址

port: 3306, //数据库服务器端口号

user: 'root', //使用数据库的用户名

password: '', //用户密码

database: 'xz', //需要连接的数据库名称

connectionLimit: 20 //数据库连接个数限制

});

//导出这个连接

module.exports = pool;

* 在路由器文件（如router/user.js）中导入上述文件模块，就可在该路由器中使用该数据库并对其中的数据进行相关操作。

//引入mysql连接池

const pool = require('../pool.js'); //../是上一级目录的意思

//数据库操作

var sql = 'INSERT INTO xz\_user VALUES(NULL,?,?,?,?,NULL,NULL,NULL)';

pool.query(sql,[$uname,$upwd,$email,$phone],(err,result)=>{

if(err) throw err;

//如何判断插入成功

//判断执行sql语句后，数据库中数据受影响的行数即可知修改成功与否

if(result.affectedRows > 0)

res.send('注册成功');

else

res.send({code:300,msg:'reg error'});

});

**【练习】**创建一个表单html，包含部门ID did,部门名称dname,和一个提交按钮。点击提交后，把数据插入到dept表中。

**【课后任务】**

1.复习mysql模块

2.完成用户删除路由，试着完成用户登录路由

**阶段项目：学子商城后台数据库搭建 ★**

**学子商城目录结构**

node\_modules…………………………………………………………………项目所依赖的包

public…………………………………………………………………项目中的静态资源文件

user\_register.html………………………………………………………用户注册页面

routes…………………………………………………………………………项目中的路由器

user.js……………………………………………………………………用户模块路由器

app.js…………………………………………………………………………项目的入口文件

xz.sql………………………………………………………………………项目的数据库文件

pool.js………………………………………………………………连接mysql数据库文件

**路由步骤**

1.构建html页面

2.构建路由

3.获取浏览器请求的数据（get-->req.query/post-->req.body）

4.验证数据是否为空（列表查询不需要，需要每页的默认大小，默认页码为1）

5.构建sql语句（增删改查）

6.执行sql语句，返回结果

7.判断结果（查询返回数组，插入和删除返回对象）

**项目开发流程**

1.安装项目中依赖的包

2.创建web服务器

3.托管静态资源

4.创建路由器文件，导出路由器

5.导入路由器，挂载到某个url下

6.导入body-parser，配置中间件（服务器下）

7.创建mysql连接，在路由器中导入模块

# 第五章 HTML

杨燕 [g-yangyan@tedu.cn](mailto:g-yangyan@tedu.cn) QQ:445190209

**第二月学习计划**

|  |  |  |
| --- | --- | --- |
| 知识点 | 天数（天） | 功能 |
| HTML5 | 2 | 搭建网页结构，实现内容的体现  http://www.cssmoban.com/  推荐书目：《高性能网站建设指南I》 |
| AJAX | 3 | 实现数据异步交互 |
| AJAX项目 | 2 | 完成用户表数据的增删改查（从客户端界面发起异步请求） |
| CSS3 | 6 | 美化HTML页面 |
| Bootstrap | 4 | CSS框架 |
| Bootstrap项目 | 3 | CSS框架 |

## 一、HTML5及WEB基础简介

### 1.1 HTML5

HTML：HTML4.01版本（1999.12~2014）

HTML5：H5

### 1.2 web与Internet

Internet：全球性的计算机互联网络，简称因特网、互联网、交互网。

提供的服务：

①www服务：world wide web万维网，通过网络的方式去访问网站

②Email：电子邮件

③BBS：电子公告板（论坛）

④FTP：文件的上传下载

⑤telnet：远程登录

### 1.3 Internet上的应用程序

#### 1.3.1 C/S程序

C:Client 客户端

S:Server 服务器端

通过指定的客户端连接服务器的程序（桌面应用程序），代表：qq、网络游戏、…、exe程序

#### 1.3.2 B/S程序

B：Browser 浏览器

S：Server 服务器

通过浏览器去访问服务器

#### 1.3.3 web应用程序

##### 1.3.3.1 web概念及作用

web：运行在Internet之上的一种B/S结构的应用程序，俗称网站

web作用：将各类信息和服务进行无缝的连接，并提供一些生动形象的用户界面

##### 1.3.3.2 web的工作原理

基于浏览器、服务器以及通信协议来实现数据的传输和展示

①通信协议：规范了数据时如何传递和打包

②服务器：

功能：存储web信息，并提供程序的运行环境；接收用户请求并给出响应；具备一定的安全功能

服务器产品：TOMCAT(JAVA)、APACHE、IIS

服务器技术：JAVA、PHP、ASP.NET、Python、Node.js（前端技术：操作数据库）

③浏览器：

功能：代理用户提交请求（UA：User Agent）；以图形化的方式解析HTML/CSS/JS

浏览器产品：Chrome、IE、Firefox、Safari、Opera

浏览器技术：HTML、CSS、JS

## 二、HTML基础认知

### 2.1 概述及其特点

HTML：HyperText Markup Language，超文本标记语言

a：普通字母 ——> <a></a>：超级文本，超链接

HTML的特点：

①用带有尖括号的标记来标识；

②以.html或.htm为后缀；

③由浏览器来解析执行；

④可以嵌套脚本语言（JavaScript）

### 2.2 基础语法

#### 2.2.1 标记

标记：又称元素或标签，在网页中表示一些功能。标记在使用时，必须使用<>括起来

##### 2.2.1.1 标记分类

封闭类型（双标记）：必须成对出现，有开始和结束标记，<标记>内容</标记>

非封闭类型（单标记）：也成为空标记，只有一个标记，既表示开始又表示结束，<标记>或<标记 />

【练习】新建一个txt文档，将文档的后缀改为.html，然后再文档中编写如下代码：

写一对<html></html>

写一对<head></head>

写一对<title></title>

写一个<meta>

##### 2.2.1.2 标记的嵌套

嵌套：在一对标记中出现另外一对(个)标记，从而形成功能的层叠

语法：

<标记1>

<标记2>

</标记2>

</标记1>

**!注意：**嵌套顺序，换行缩进；成对出现，去嵌套其他的标记

#### 2.2.2 属性和值

##### 2.2.2.1 属性

用来修饰元素

* 属性的声明必须位于开始标记里
* 一个元素的属性可能不止一个，多个属性之间用空格隔开
* 多个属性之间不区分先后顺序

##### 2.2.2.2 值

每个属性都有值

* 属性和属性的值之间用等号连接
* 属性的值包含在引号中

##### 2.2.2.3 标准属性

每个元素都有自己所特有的属性，有些属性是绝大多数元素都支持的属性，称为标准属性（或通用属性）。

id：定义元素在页面中独一无二的标识

style：在CSS中，定义行内样式

class：在CSS中，引用类选择器

title：鼠标悬停时元素上所提示的文字

##### 2.2.2.4 属性和值的语法

<标记 属性名称1="值1" 属性名称2="值2">内容</标记>

要求必须卸载开始标记中，属性和值之间用等号连接；如果有多个属性，属性之间用空格隔开

【示例】设置段落标记居中对齐<p align="center">

#### 2.2.3 注释

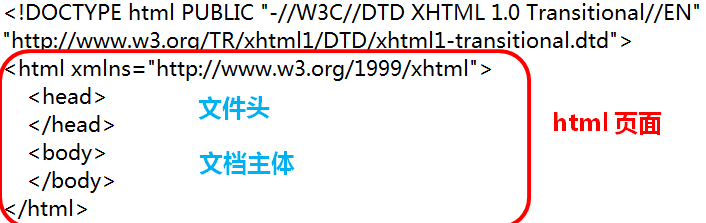
语法：<!--注释内容-->

**!注意：**使用注释时需要注意以下三点：



### 2.3 文档结构

HTML文档结构由文档类型声明和HTML页面构成，其中HTML页面又由文件头和文档主体部分组成。



**文档类型声明**

#### 2.3.1 文档类型声明

在文档的起始用DOCTYPE声明指定版本和风格，即让浏览器清楚文档的版本、类型和风格，HTML4.01规定了三种，HTML5规定了一种。

|  |  |  |
| --- | --- | --- |
| Strict DTD | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Strict//EN” “http://www.w3.org/TR/xhtml1/DTD/xhtml-strict.dtd”> | 干净的标记，免于表现层的混乱，应使用此类型并与层叠样式表（CSS）配合使用 |
| Transitional DTD | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN” “http://www.w3.org/TR/xhtml1/DTD/ xhtml-transitional.dtd”> | 包含 W3C 所期望移入样式表的呈现属性和元素。如果您的读者使用了不支持层叠样式表（CSS）的浏览器以至于您不得不使用 HTML 的呈现特性时，请使用此类型 |
| Frameset DTD | <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Frameset//EN” “http://www.w3.org/TR/xhtml1/DTD/ xhtml- frameset.dtd”> | 应当被用于带有框架的文档。除 frameset 元素取代了body元素之外，Frameset DTD等同于Transitional DTD |
| HTML5 | <!DOCTYPE HTML> | — |

#### 2.3.2 <html>元素

整个文档的包含元素，起始标签内必须包含命名空间标识符（xmlns=”http://www.w3.org/1991/xhtml”）。

<html>包含两个子元素：<head><body>

##### 2.3.2.1 <head>元素

用于为页面定义全局信息，是所有其他头元素的容器，可包含title/meta/script/style/link等。

<title></title>：定义网页的标题

<meta>：定义全局信息，如：编码格式、关键词、描述内容、元标签等

<meta charset=’utf-8’ />

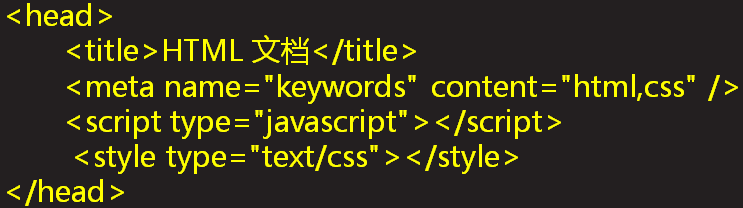
<meta name=’keywords’ content=’关键词’ />

<meta name=’Description’ content=’描述内容’ />、

<style></style>：定义内部样式

<script></script>：定义/引用js脚本

<link>：引入css文件



##### 2.3.2.2 <body>元素

显示网页的所有内容，可以包含除了html、head外所有元素

修饰body的属性：

text：定义整个网页的文字颜色

bgcolor：定义网页的背景颜色

## 三、文本标记

### 3.1 文本标记概述

#### 3.1.1 作用

文本是网页上重要的组成部分，直接书写的文本会用浏览器默认的样式显示，而包含在表集中的文本则会被显示为标记所有用的样式。

#### 3.1.2 特殊字符

用一些转义字符来表示特殊的符号：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| &nbsp; | &lt; | &gt; | &copy; | &yen; | &reg; | &times; |
| 空格 | 小于< | 大于> | © | ￥ | ® | 关闭符号× |

【练习】完成以下版权设置

Copyright © 2004 - 2018 京东<JD.com> 版权所有

ANSWER:Copyright&nbsp;&copy;&nbsp;2004&nbsp;-&nbsp;2018 &nbsp;&nbsp;京东&lt;JD.com&gt;&nbsp;版权所有

### 3.2 使用文本标记

#### 3.2.1 文本样式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 加粗文本 | 倾斜文本 | 下划线 | 删除线 | 下标 | 上标 |
| <b></b> | <i></i> | <u></u> | <s></s> | <sub></sub> | <sup></sup> |

#### 3.2.2 标题元素

让文字以醒目的方式显示，常用于文章的标题

语法：<hn></hn> n:1-6

<h1>内容</h1>

…

<h6>内容</h6>

特点：改变字体的大小；加粗显示；标题元素自带上下空白间距；独占一行

属性：align：设置内容水平对齐方式，取值left、right、center

【练习】在页面中设置一级标题中的内容靠左显示，三级标题的内容居中显示，五级标题中的内容靠右显示

#### 3.2.3 段落元素

语法：<p>内容</p>

特点：与前后的文本都换行分开；添加一段额外的垂直空白距离作为段落间距；字体无加粗和大小变化

属性：align：设置内容在水平方向的对齐方式，取值：left、right、center

#### 3.2.4 换行元素

使用该元素在任何地方创建手工换行

语法：<br>或<br />

#### 3.2.5 分区元素

分区元素用于为元素分组，常用于页面布局

* 块级分区元素

语法：<div></div>

作用：用于页面的布局

特点：单独成行

* 行分区元素

语法：<span></span>

作用：处理同一行文本的不同样式

特点：包裹的内容在一行中显示

#### 3.2.6 行内元素与块级元素

* 行内元素

多个元素在一行中显示，从左往右排列的元素，常见的行内元素：

span、i、b、u、s、sup、sub

* 块级元素

在页面中独占一行的元素，常见的块级元素：

h1~h6，p，div，结构标记

#### 3.2.7 分隔线元素

用于在页面上创建一条水平线

语法：<hr>或<hr />

属性：

width：宽度，表示水平线的宽度，取值为px或%的数值

align：水平线对齐方式，取值：left、right、center

size：表示水平线的尺寸（高度），取值为px或%的数值

color：水平线的颜色，取值为合法的颜色值

【练习】页面中添加一个水平线，设置尺寸为3px，宽度为50%，颜色为绿色，并居中对齐。

#### 3.2.8 预格式化元素

语法：<pre></pre>

作用：用于保留HTML代码中的回车和空格

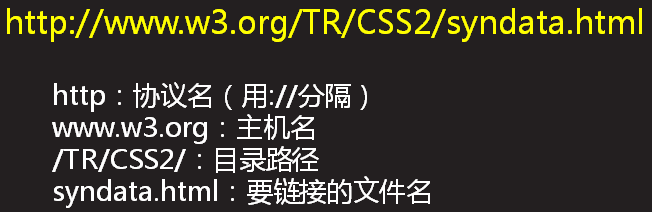
## 四、图像和链接

### 4.1 URL

#### 4.1.1 URL

URL，Uniform Resource Locator统一资源定位器，用于标识网络中的任何资源：文本、图片、音视频文件、段落、或其他超文本。

完整格式：协议+主机+文件目录结构+文件名称



#### 4.1.2 路径

路径在Web页面主要有三种形式：绝对路径、相对路径、根相对路径

##### 4.1.2.1 绝对路径

指文件从最高级目录下开始的完整的路径，无论当前路径是什么，使用绝对路径总是能找到要链接的文件，即完整的URL组成。

网络资源：使用URL的完整格式

本地：从计算机最高盘符处开始查找C:\xampp\htdocs\y.js

##### 4.1.2.2 相对路径

从当前文件所在的位置开始查找资源所经过的路径

同级目录：直接引用 ex:jd\_logo.png

子级目录：先进入，再引用 ex：img/page.jpg

父级目录：先返回，在引用 ex：../jd\_logo1.png

**!注意：**"./"当前目录；"../"父级目录；"/"根目录

##### 4.1.2.3 根相对路径

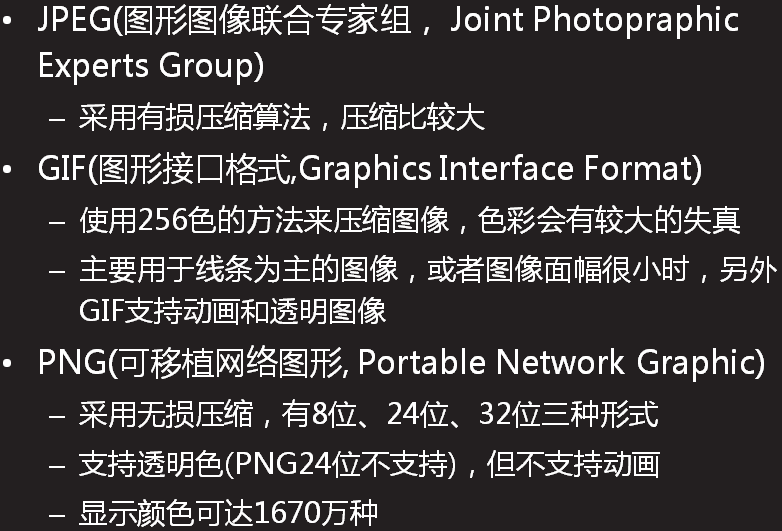
永远是从web站点的根目录处开始查找

ex：/codebody/img/a.jpg

### 4.2 图像

#### 4.2.1 图像格式

jpeg 压缩比率大 **|** png 背景透明 **|** gif 动画



#### 4.2.2 图像元素

语法：<img>

属性：

src指定要显示图像的url（必须属性）

width:指定图片的宽度，取值以px或%为单位的数字

height：指定图片的高度，取值以px或%为单位的数字

【练习】day02文件夹中的01-url.html网页需要去引用于day02平级的other文件夹中的1.jpg图片

**!注意：**img是唯一一个能够设置高宽的行内元素，如果需要更改高宽，改变其中一个值即可（约束比例），这样才不会是图片变形。

### 4.3 链接与锚点

#### 4.3.1 链接元素

语法：<a>内容</a>

属性：

href 指定链接的url

target 指定打开新网页的方式，取值：\_self默认值，在当前网页打开新的网页；\_blank在新的网页打开网页

name 锚点名称

#### 4.3.2 链接的表现形式

①资源下载：让链接的url，链接到rar/zip的文件即可

②电子邮件链接：<a href=”mailto:g-yangyan@tedu.cn”>打开邮件</a>

③返回页面的顶部：<a href=”#”>返回页面的顶部</a>

④链接到js：<a href=＂javascript:js代码＂>js代码</a>

⑤链接到其他网页：<a href=”http://www.baidu.com”>点此跳转到百度</a>

⑥链接到本地的html：<a href=”C:/xampp/docs/…/01.html”>点此跳转到01.html</a>

⑦链接到锚点：<a href=”#div”>

#### 4.3.3 锚点

①定义锚点

使用a标记的name属性定义锚点

<a name=”锚点名称”></a>

使用任意元素的id属性定义锚点

<any id=”锚点名称”></any>

②链接到锚点

链接到当前页面的锚点

<a href=”#锚点名称”></a>

链接到其他页面锚点位置处

<a href=”网页url#锚点名称”></a>

【TIPS】在html文档中，输入lorem+TAB键，可生成一段随意的文字用于填充内容。

【练习】在不同的网页中分别定义三个锚点，锚点中内容随意，然后通过a标记分别链接到三个锚点位置处。

## 五、表格和列表

### 5.1 表格

#### 5.1.1 表格的作用

按照一定的结构显示数据，由单元格按照从左往右，从上到下的方式排列到一起。数据都保存在单元格中，表格可以显示数据，也可以设置页面的布局。

#### 5.1.2 表格的语法

* 标记

表格：<table></table>

行：<tr></tr> table row

单元格/列：<td></td> table data

【练习】创建一个4\*4的表格，在网页中创建一对<table></table>标记，在<table>标记中嵌套<tr></tr>，在<tr>标记中嵌套<td>

* 属性

<table>的属性

border：设置表格的边框 取值以px为单位的数字

width：设置表格的宽度

height：设置表格的高度

align：设置表格的水平对齐方式，取值：left/right/center

bgcolor：设置背景颜色，取值为合法颜色值

cellpadding：设置单元格的内边距（内容与td之间的间距）

cellspacing：设置单元格的外边距（td边框与边框之间的间距）

<tr>的属性

align：设置当前行内容水平对齐方式

valign：设置当前行内容垂直对齐方式，取值：top/middle/bottom

bgcolor：设置当前行的背景色

colspan：跨列

rowspan：跨行

#### 5.1.3 不规则表格的使用

跨列：从指定单元格位置处开始，横向向右合并几个单元格（包括自己），被合并掉的单元格要删除

跨行：从指定单元格位置处开始，纵向向下合并几个单元格（包括自己），被合并掉的单元格要删除

#### 5.1.4 可选标记

表格标题：<caption></caption>，如果设置标题的话，则必须位于<table>下的第一个子元素。

行/列标题：<th></th>，所有td都可以用th替换

#### 5.1.5 表格的复杂应用

表格的嵌套：表格中所有被嵌套的内容只能放在td中

行分组：可以将连续的几个行划分到一个组中，并进行统一的管理

表头行：<thead></thead>，表格中最上面一行进行分组的话，可以放在表头行中

表主体：<tbody></tbody>，允许将中间的若干行放在tbody中，进行统一管理

表尾行：<tfoot></tfoot>，一般情况下将表格的最后一行放在表尾行中

### 5.2 列表

#### 5.2.1列表的作用

按照从上到下的方式来显示所有数据（或从左往右显示），并且可以在数据前增加显示的标识。

#### 5.2.2 列表的组成

由列表类型和列表项组成，其中：

列表的类型：

有序列表：<ol></ol> order list

无序列表：<ul></ul> unorder list

列表项：

用于表示列表中的数据（嵌套在列表中）

<li></li> list item

#### 5.2.3 列表的属性

##### 5.2.3.1 有序列的属性

* type 指定列表标识类型，取值：

1 默认值，以数字排序

a 小写英文字母排序

A 大写英文字母排序

I 大写罗马数字排序

i 小写罗马数字排序

* start 指定起始编号是从第 几 开始，取值为无单位的数字

##### 5.2.3.2 无序列表的属性

* type 指定无序列表的标识，取值：

disc 实心圆

circle 空心圆

square 实心方块

none 不显示任何标识

#### 5.2.4 列表的嵌套

在一个列表中又出现另外一个列表，被嵌套的列表只能在li中出现

#### 5.2.5 自定义列表

自定义列表常用于给一类事物或对名词进行解释说明。

* 语法

<dl></dl> 表示定义列表

<dt></dt> 表示列表中要解释说明的名词

<dd></dd> 表示列表中对名词解释的具体内容

ex:

<dl>

<dt>名词</dt>

<dd>解释的内容</dd>

</dl>

* 常用场合：图文混排

## 六、结构化标签

作用：用于描述网页的基本结构（取代div做布局），提升代码的语义性，方便后期阅读和维护。

常用的结构标记：

header 定义网页的头部/页眉

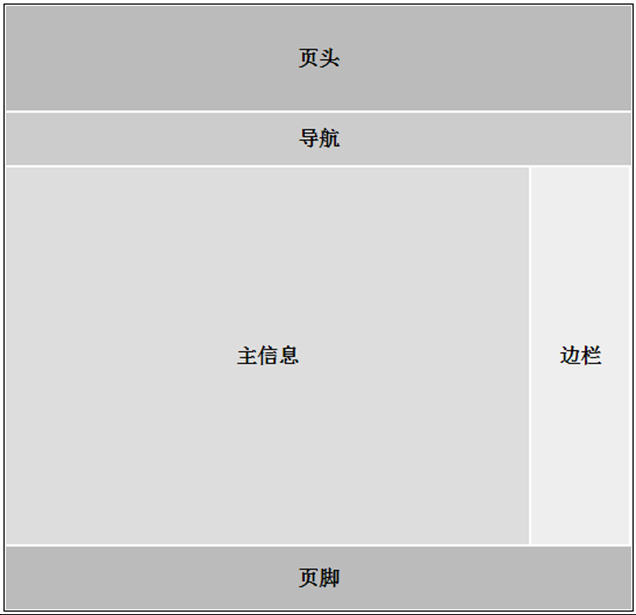
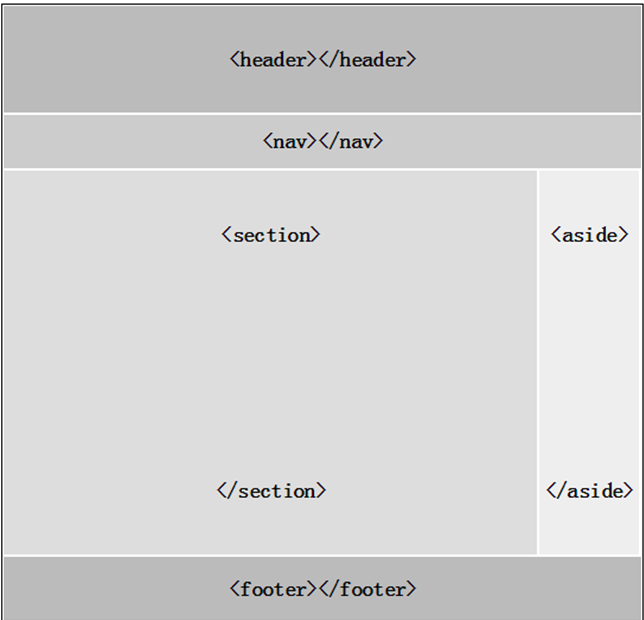
nav 定义导航连接内部

section 定义网页的主体

aside 定义网页侧边栏内容

footer 定义网页底部的内容

article 定义与文字描述相关的内容，如论坛中的帖子、微博条目、用户评论等

## 七、表单元素

作用：提供可以与用户交互的可视化控件；收集用户信息并提交给服务器

组成：前端部分：表单控件 ——> 与用户交互

服务器端部分：对提交数据的处理

### 7.1 <form>标签属性

语法：<form></form>

#### 7.1.1 action属性

定义表单被提交时发生的动作，通常定义服务器处理程序的地址（url路径），即规定当提交表单时向何处发送表单数据，默认提交给本页。

#### 7.1.2 method属性

指定表单数据的提交方式，即规定用于发送表单数据的 HTTP 方法，取值

get（默认值）：明文提交，待提交的数据会显示在地址栏上；提交数据的大小限制为2KB以内；向服务器请求数据时使用

post：提交的数据无大小限制；隐式提交，待提交的数据不会显示在地址栏上；要传递数据给服务器时使用

delete/put

#### 7.1.3 enctype属性

指定表单数据的编码方式，即允许将什么样的数据提交给服务器

application/x-www-form-urlencode：默认值，允许将任意字符提交给服务器（文件除外），即在发送到服务器之前，所有字符都会进行编码（空格转换为 "+" 加号，特殊符号转换为 ASCII HEX 值）。

multipart/form-data：允许将文件提交给服务器，不对字符编码。在使用包含文件上传控件的表单时，必须使用该值。

text/plain：空格转换为 "+" 加号，但不对特殊字符编码，只能提交普通字符

**!注意：**如果表单提交是被动的（比如搜索引擎查询），并且没有敏感信息使用GET，如果表单正在更新数据，或者包含敏感信息（例如密码）使用POST。

### 7.2 表单元素及属性

#### 7.2.1 表单元素分类

input元素

textarea元素

select元素

其他元素

#### 7.2.2 input元素

##### 7.2.2.1 HTML中的input元素类型

作用：在页面中提供各种输入的控件，如：文本框，密码框，单选按钮，复选框等

语法：<input></input>

属性：

* type 指定输入控件的类型，取值：

text 定义供文本输入的单行输入字段

password 定义密码字段

submit 定义提交表单数据至表单处理程序的按钮，表单处理程序（form-handler）通常是包含处理输入数据的脚本的服务器页面。在表单的 action 属性中规定表单处理程序（form-handler）

reset（重置，恢复到初始化状态）

button/radio/checkbox 定义按钮/单选按钮/复选框

hidden 设置该文本框内容及框隐藏不可见

file 定义可供文件上传的“按钮”，效果图见下：



**!注意：**使用该属性时，表单的method属性必须为post，enctype属性的值必须为multipart/form-data

* value 定义按钮上面的文字或各表单控件的值
* name 为控件定义名称，提交给服务器端使用（此外，还有为控件分组的作用，用于复选框中的时候需要在name值中加上中括号表示一组数据存于数组，如：hobby[]）
* disabled 禁用控件，不能操作并且也不能提交，该属性无值
* maxlength 指定限制输入的字符数
* readonly 只读，只能看，不能改
* placeholder 占位符，即默认显示在控件上的文本
* checked 用于按钮中，设置默认选中，无值属性

ex：

文本框 <input type=”text”>

密码框 <input type=”password”>

提交按钮 <input type=”submit”>

##### 7.2.2.2 HTML5新增input元素类型

* 电子邮件类型：<input type=”email”>

作用：表单提交时，会验证数据是否符合email的规范

* 搜索类型：<input type=”search”>

作用：提供了快速清除功能



* url类型：<input type=”url”>

作用：提交时，验证数据是否符合url的规范（绝对路径）

* 数字类型：<input type=”number”>

作用：只能接受数字，并且可以灵活的改变数字的值

ex:

属性：

value 默认的初始值

min 能够接收的最小数字值

max 能够接收的最大数字值

step 每次调整数字时数字的变化范围（步长），在min的基础上进行+、-step的操作

* 电话号码类型：<input type=”tel”>

作用：在移动设备中，显示为“拨号键盘”的效果

* 范围类型：<input type=“range”>

作用：提供滑块组件，允许用户选取指定范围的值

属性：

value 默认的初始值

min 能够接收的最小数字值

max 能够接收的最大数字值

step 每次调整数字时数字的变化范围（步长），在min的基础上进行+、-step的操作

* 颜色控件：<input type=”color”>

作用：拾取一个颜色的值

* 日期类型：<input type=”date”>

作用：提供一个日期控件

* 周类型：<input type=”week”>

**!注意：**各主流浏览器上述输入类型的支持情况



#### 7.2.3 textarea元素

作用：允许录入多行数据的文本框

标记：<textarea></textarea>

属性：

name 定义控件名称，提供给服务器用

readonly 只读，不能改，无值属性

cols 指定文本域的列数，即一行能显示多少个英文字符（中文减半）

rows 指定文本域的行数，即默认显示多少行数据，超出rows的行数会出现滚动条

#### 7.2.4 select元素

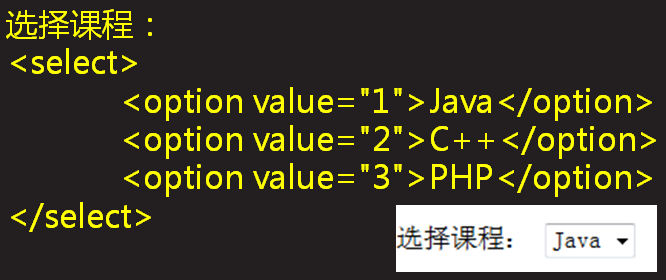
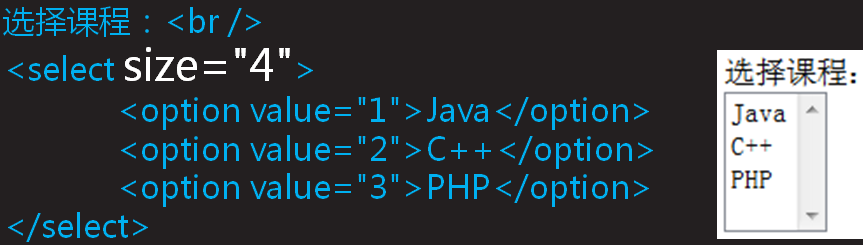
语法：<select></select> 在页面中表示一个选项框

<option></option> 选项框中具体的选项内容

select 的属性：

name 定义选项框的名称

size 定义显示选项的数量，默认值为1

multiple 设置多选 无值属性

option的属性：

value 定义选项的值

selected 设置默认选中该选项，无值属性

#### 7.2.5 其他元素

##### 7.2.5.1 label元素

作用：关联文本域表单控件

语法：<label>文本</label>

属性：for用于绑定要与label元素关联的表单控件的id值

##### 7.2.5.2 控件分组

标记：<fieldset></fieldset> 元素组合表单中的相关数据

<legend></legend> 元素为 <fieldset> 元素定义标题

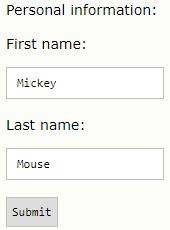
ex：

<fieldset>

<legend>标题</legend>

</fieldset>

效果图如下：

 ——>

##### 7.5.2.2 浮动框架

作用：允许在一个网页中引入另一个网页

标记：<iframe></iframe>

属性：

src 指定要引入的网页的url

width 设置框架的高度

height 设置高度

frameborder 浮动框架的边框 默认值1

# 第六章 HTTP&AJAX

## 一、HTTP协议

### 1.1 URL简介

#### 1.1.1 URL语法

作用：用于表示任意资源的位置

格式：协议+主机名称+目录结构+文件结构

完整格式：

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

* scheme：协议，方案，常见的协议：http/https…
* user：用户名，某些方案访问资源时需要指定用户名，默认为anonymouse(匿名/无名氏)
* pwd：密码，默认值为<e-mail>地址



* host：主机名，资源所在服务器的IP地址或者域名（需要DNS转换为IP地址）
* port：端口号，每项服务在服务器都对应一个监听端口号。常见协议指定了一些默认端口号（见下表），应努力避免混用。



* path：路径，资源在服务器上的存放位置，由一个斜杠将其与前面的URL组件分隔开来



* params：参数
* query：查询字符串
* frag：锚点

#### 1.1.2 URL的使用

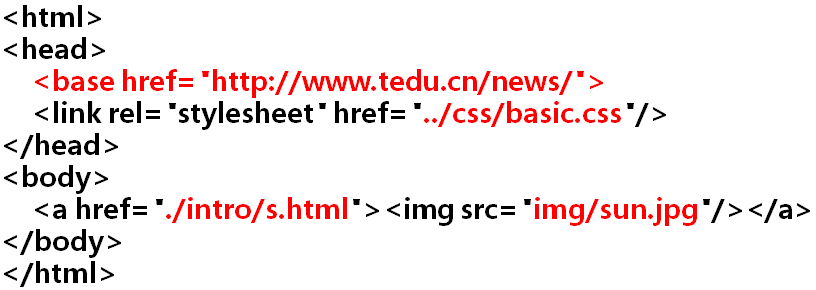
URL有两种形式：绝对URL和相对URL

* 绝对URL

以方案开头，包含要访问资源所需全部信息

* 相对URL

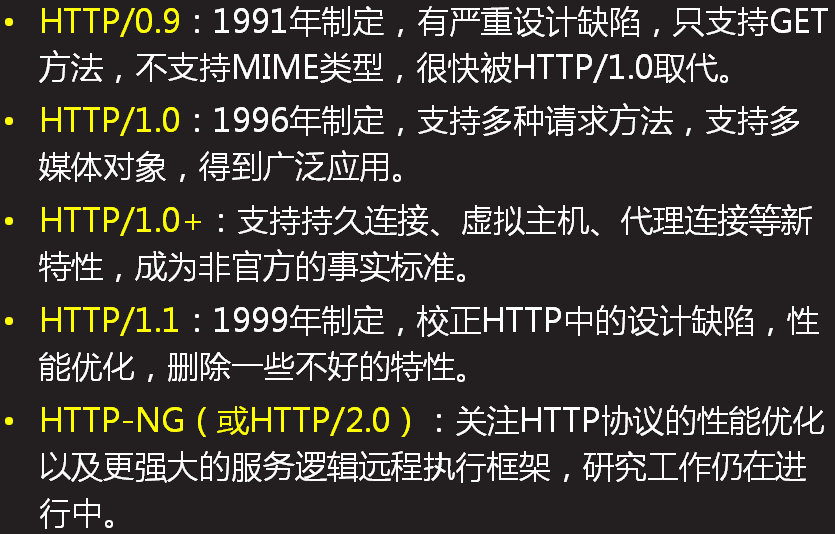
不以方案开头，URL本身包含的信息并不完整，必须参考另外一个URL（base，基础URL）才能确定其完整内容信息。



### 1.2 HTTP协议概述

HTTP，Hypertext Transfer Protocol超文本传输协议，规范数据时如何打包及传输

#### 1.2.1 历史与标准



#### [1.2.2 请求与响应流程](#_四、阿里面试题)

### 1.3 HTTP消息

#### 1.3.1 消息结构概述

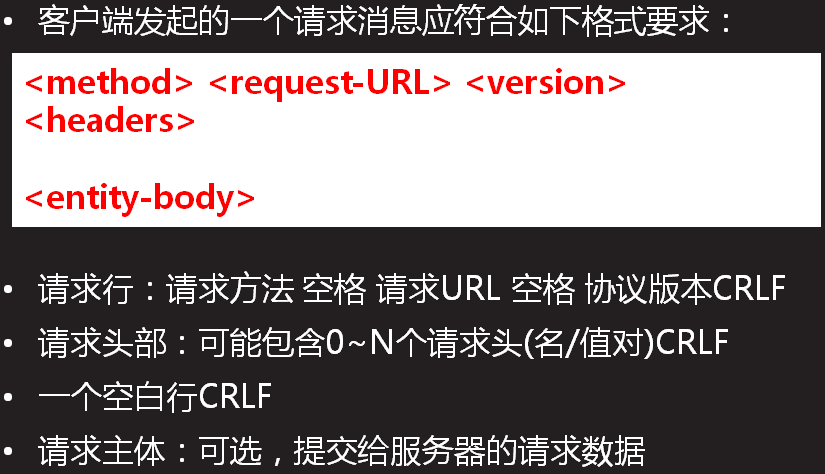
HTTP消息是简单的格式化数据块，每个消息都由三部分组成：

* Start Line:消息起始行，必需，消息基本描述信息
* Header:消息头部/报头，可能有0~N个，消息详细属性
* Body:消息主体，可选，包含数据的主体

【说明】起始行和消息头是纯ASCII字符，每行以CRLF结束；消息主体是一个可选的数据块，其中的数据可以为空，或者是字符数据（如HTML、CSS、JavaScript等字符数据），或者二进制数据（如图片、音频、视频等字节数据）。

#### 1.3.2 Request请求消息结构

Request消息，是客户端要带给服务器的数据。



请求头

请求主体

请求起始行

* 请求起始行

包括、请求URL、协议版本。

A.请求方法：

|  |  |  |
| --- | --- | --- |
| 请求方法 | 描述 | 是否有请求主体 |
| GET | 向服务器请求数据/文档 | 否 |
| POST | 向服务器发送需要处理的数据 | 否 |
| PUT | 将请求主体中的数据存储在服务器上（一般禁用） | 是 |
| DELETE | 从服务器上删除一份文档（一般禁用） | 否 |
| HEAD | 只获取文档指定的响应头部 | 否 |
| CONNECT | 测试连接 | - |
| TRACE | 对可能经过代理服务器传送到服务器上去的消息进行追踪 | 否 |
| OPTIONS | 决定可以在服务器上执行的方法（选项，保留以后使用） | 否 |

B.请求URL

C.协议版本：默认为HTTP 1.1

* 请求头

请求头可用于描述请求本身、客户端、请求主体的特征，常用的请求头有：

**-** host:localhost

作用：告诉服务器请求的是哪个主机

**-** Connection:keep-alive

作用：告诉服务器进行持久连接

**-** User-Agent：

作用：告诉服务器自己（浏览器）的类型

**-** Accept-Language：zh-cn

作用：告诉服务器自己（浏览器）能接收的自然语言

**-** Accept-Encoding：gzip

作用：告诉服务器自己（浏览器）能接收的数据压缩类型是什么

**-** Referer：http://localhost/01-login.html

作用：告诉服务器请求是来自哪个页面

* 请求主体

请求主体中保存着客户端提交给服务器，需要服务器加以处理（POST）或保存（PUT）的数据。另，如果是get方法无请求主体，如果是post方法即查看Form Data

#### 1.3.3 Response响应消息结构

Response消息是服务器返回给客户端的数据块，由响应起始行（响应行）、响应头及响应主体三部分构成。

* 响应行

响应行由协议版本、响应状态码、原因短句构成：

A.协议版本：默认为HTTP 1.1

B.响应状态码：告诉客户端（如浏览器），服务器的响应状态

**-** 1xx：100~199 提示信息

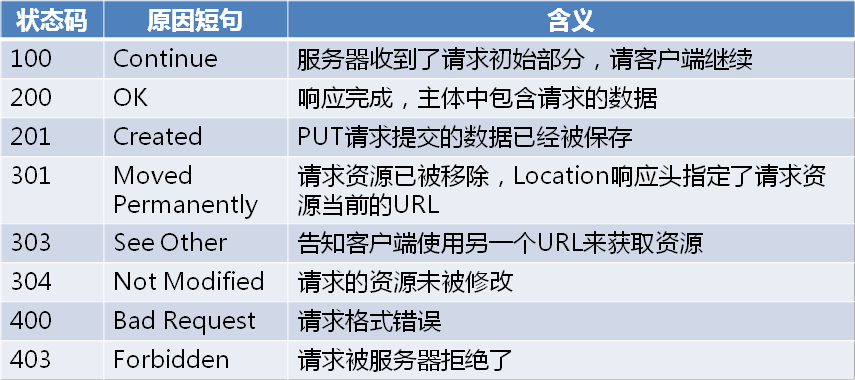
**-** 2xx：200——>响应成功 ok

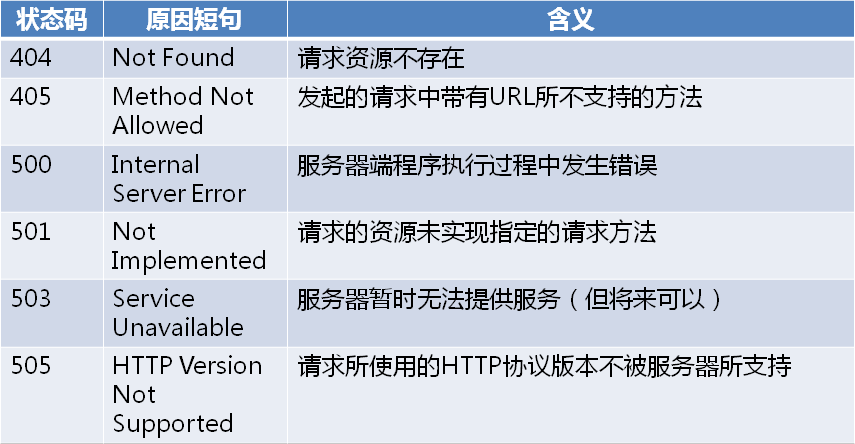
**-** 3xx：需要进行重定向，其中301是永久的重定向；302是临时的重定向；304是Not Modified请求未被修改，走缓存

**-** 4xx：客户端请求错误。404——>Not Found请求的资源不存在（文件路径出错）；403：Forbidden 权限不够；405：Method Not Allow 请求方法不被允许

**-** 5xx：服务器运行错误，500——>服务器内部错误，解决：重启xampp，mysql运行窗口重启

C.原因短句：对状态码的解释说明





* 响应头

**-** Date：Thu,30 Aug 2018 03:11:45 GMT（格林尼治时间）

**-** Connection：告诉浏览器已经启动持久连接

**-** Content-Type：响应主体的类型（MIME类型）是什么，告诉浏览器该用什么方式解析响应主体，常见的MIME类型如下：

text/html：响应回来的是html文本

text/plain：响应回来的是普通文本

text/css：响应回来的是css样式

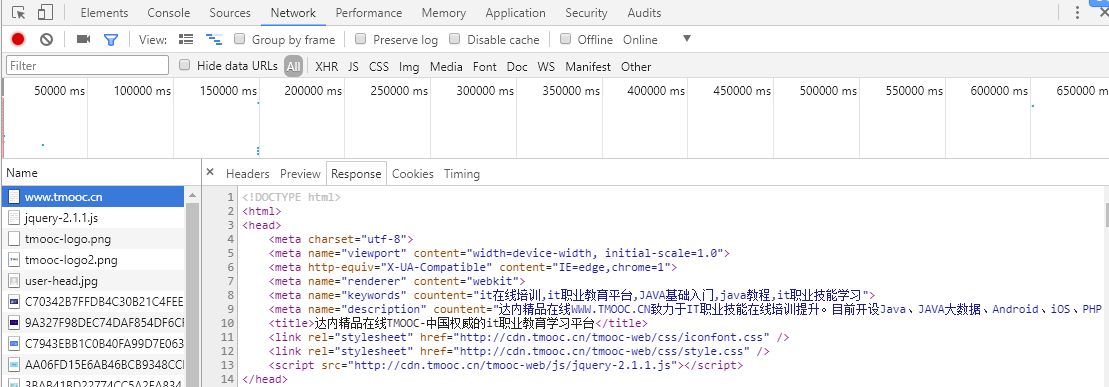
application/javascript：响应回来的是js脚本

application/xml：响应回来的数据时xml格式的字符串

application/json：响应回来的数据时json格式的字符串

image/jpg：响应回来的是图片

* 响应主体



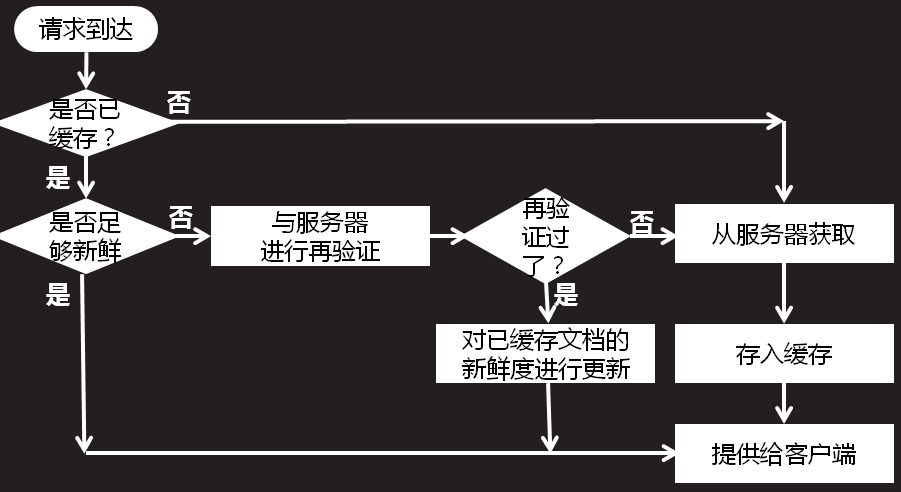
### 1.4 缓存

#### 1.4.1 缓存定义

客户端将服务器响应回来的数据进行自动保存，当再次访问时，直接使用保存的数据。

#### 1.4.2 缓存工作原理

缓存能够命中的完整流程：



#### 1.4.3 缓存优点

①减少冗余数据的传递，节省客户端流量

②降低了对服务器资源的消耗和运行要求

③降低了由于远距离造成的延时加载

#### 1.4.4 与缓存相关的消息头

* Cache-Control消息头

作用：从服务器将文档传到客户端之时起，此文档处于新鲜的秒数

语法：Cache-Control：max-age=处于新鲜的秒数

ex：Cache-Control——>max-age=3600;

* Expires消息头

作用：指定缓存过期的确切时间（格林尼治时间）

语法：Expires --- Thu,30 Aug 2018 03:11:45 GMT

#### 1.4.5 缓存消息头在网页中的设置

<meta http-equiv=”消息头” content=”值”>

ex：

<meta http-equiv=”Cache-Control” Content=”max-age=3600”>

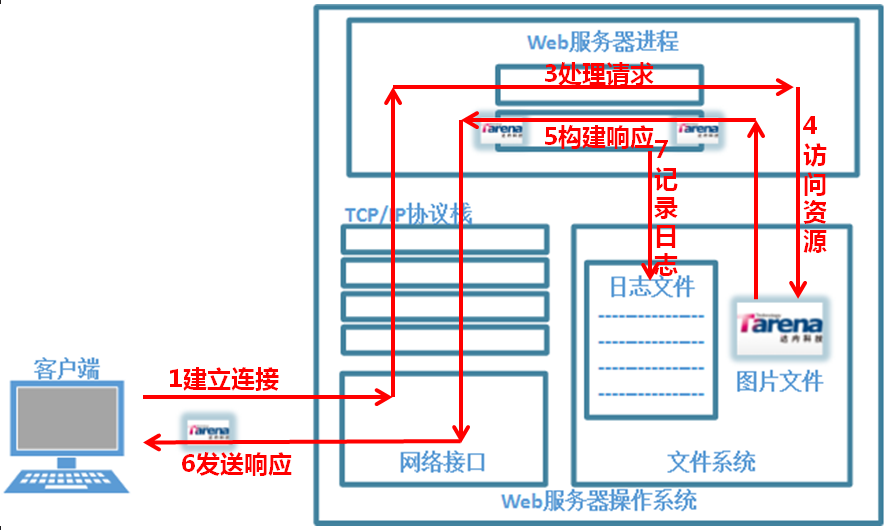
### 1.5 Cookie工作原理



### 1.6 HTTP优化

#### 1.6.1 HTTP连接过程

发起请求——>建立连接——>处理请求——>访问资源——>构建响应——>发送响应——>记录日志（服务器端完成）



#### 1.6.2 HTTP连接性能优化

* 减少域名解析次数——>减少跨站外部资源的引用
* 减少连接创建次数——>使用keep-alive避免重复连接
* 减少请求次数——>合理设置缓存（设置Expires时间、资源合并）
* 提高服务器端运行速度——>提高数据运算及查询速度
* 减少响应数据的长度——>启用压缩

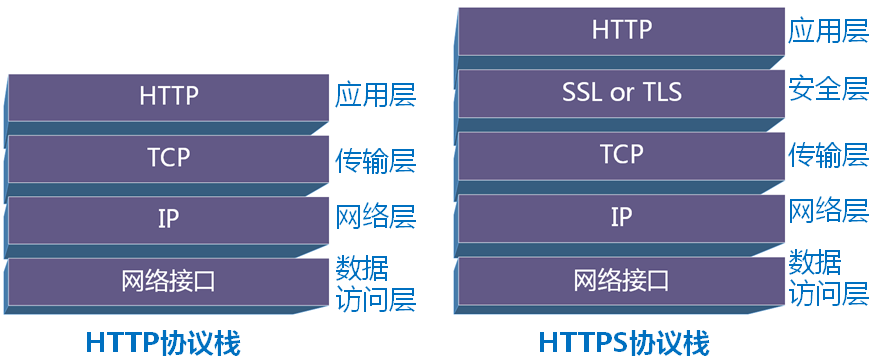
#### 1.6.3 安全的HTTP协议

HTTPS协议：安全版本的HTTP协议

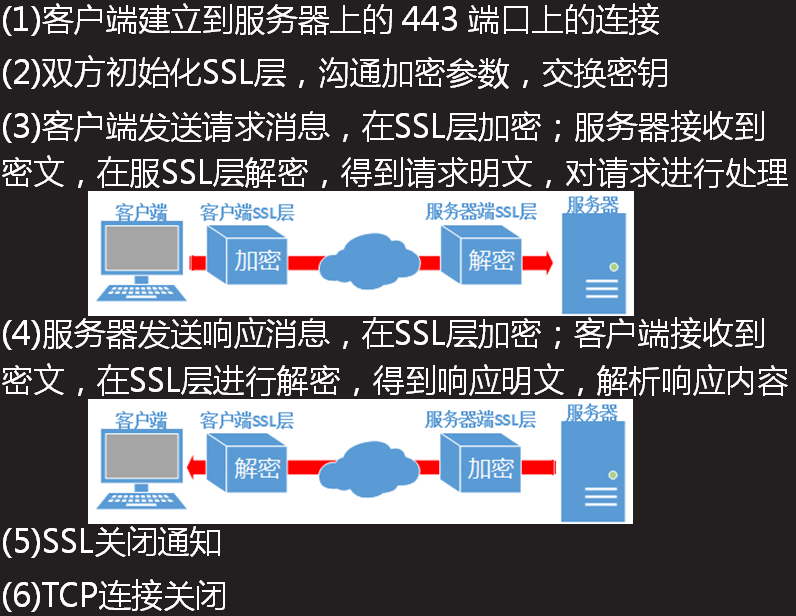
客户端发送请求消息，在SSL层加密，服务器接收到密文，在SSL层解密，得到请求明文，对请求进行处理（SSL：为数据通讯提供安全支出）。

服务器发送响应消息时，在SSL层加密，客户端接收到密文，在SSL进行解密，得到响应明文，解析响应内容。

总的来说，HTTPS协议将消息发送给TCP层之前，先交给了一个安全层（由SSL或TLS协议实现），安全层负责对消息进行加密和解密操作。



#### 1.6.4 安全连接建立过程



## 二、DOM操作（为AJAX做铺垫）

### 2.1 完整的JS的组成

* JS核心：ECMAScript
* DOM：Document Object Model

文档对象模型：让JS动态的操作页面的元素

* BOM：Browser Object Model

浏览器对象模型，让JS动态的操作浏览器

### 2.2 使用JS获取页面上的某个元素

①为元素增加id属性

<div id=”d1”>文本内容</div>

②在JS中，允许通过元素的id来获取元素

document.getElementById(“元素的id”);

### 2.3 获取/设置某元素中的内容

DOM对象的一个属性：innerHTML，表示的就是标签中的所有内容

【练习】在页面添加一个div元素，内容随意；在页面中添加一个按钮，点击时，获取div中的文本，并打印在控制台；再添加一个按钮，点击时将div中的文本内容更改为“这是我的第一个dom练习！”

### 2.4 获取/设置表单控件的内容

在DOM，是通过DOM对象的value属性来获取/设置表单控件的内容的。

步骤：

通过id获取表单控件元素

通过DOM对象的value属性获取/设置值。

【练习】在页面中创建一个文本输入框（id=”uname”），div（id=”d1”），一个按钮。单机按钮时，将文本输入框中的数据，显示在div中。

### 2.5 简化document.getElementById()

根据指定的“id值”，获取对应的“HTML元素”

参数：要获取元素的id

返回值：获取的对应的HTML元素

function $(id){

return document.getElementById(id);

}

### 2.6 HTML元素的事件

什么是事件：允许通过用户的行为来激发的操作就是事件

常用事件：onclick 鼠标单击事件

其他事件：

* 文本框/密码框获取焦点事件

onfocus：获取焦点事件

* 文本框、密码框失去焦点事件

onblur：失去焦点事件

## 三、AJAX ★

### 3.1 同步和异步

* 同步（Synchronous）：在一个任务进行中时，不能开启其他的任务。

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

出现场合：

①地址栏输入网址访问网页

②a标记的默认跳转

③submit按钮的表单提交

* 异步（Asynchronous）：在一个任务进行中时，可以开启其它的任务

异步访问：浏览器在向服务器发送请求时，不耽误用户在网页上做其他操作。

出现场合：

①用户名重复验证

②搜索建议

③聊天室

④股票走势图

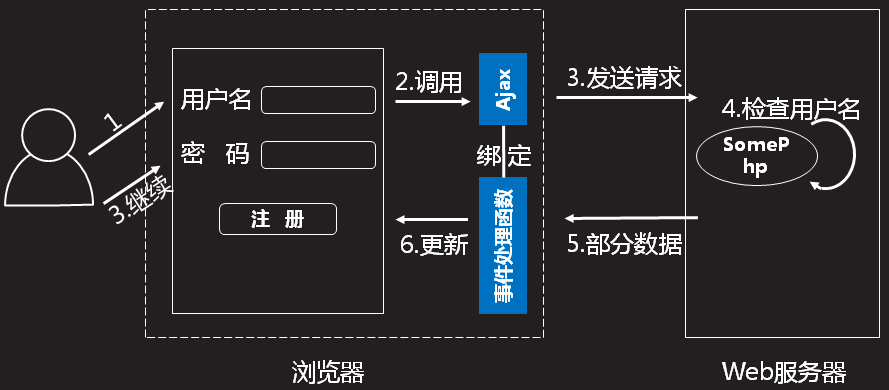
### 3.2 AJAX定义及工作原理

#### 3.2.1 AJAX定义

AJAX：Asynchronous Javascript And Xml.（异步的JavaScript和XML）

本质：使用js提供的异步对象（XMLHTTPRequest），异步地向服务器发送请求，并接收响应的数据。在AJAX中，可以无刷新的更改页面中布局的内容。

#### 3.2.2 AJAX工作原理



### 3.3 AJAX对象属性和方法

#### 3.3.1 AJAX异步对象的创建

* 标准创建：var xhr = new XMLHttpRequest();
* IE8以下：var xhr = new ActiveXObject(“Microsoft.XMLHttp”);

通过window.XMLHttpRequest来判断浏览器是否支持标准创建。如果不支持标准创建，那么window.XMLHttpRequest的值为null

if(window.XMLHttpRequest){

var xhr = new XMLHttpRequest();

}else{

var xhr = new ActiveXObject(“Microsoft.XMLHttp”);

}

【练习】将创建异步对象的步骤封装到一个函数中，函数名：createXhr()，在函数体内将创建好的异步对象返回到函数外。

function createXhr(){

var xhr = null;

if(window.XMLHttpRequest){  
 xhr = new XMLHttpRequest();

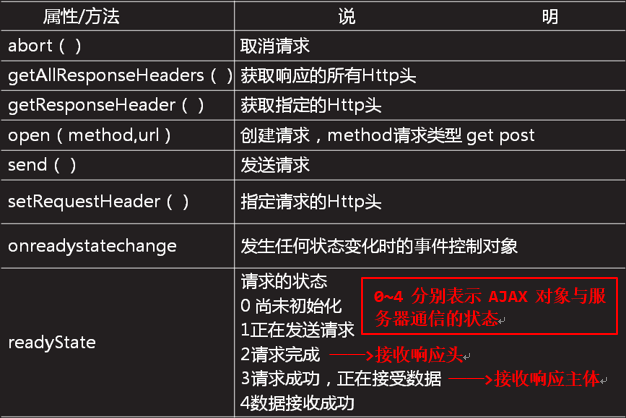
else

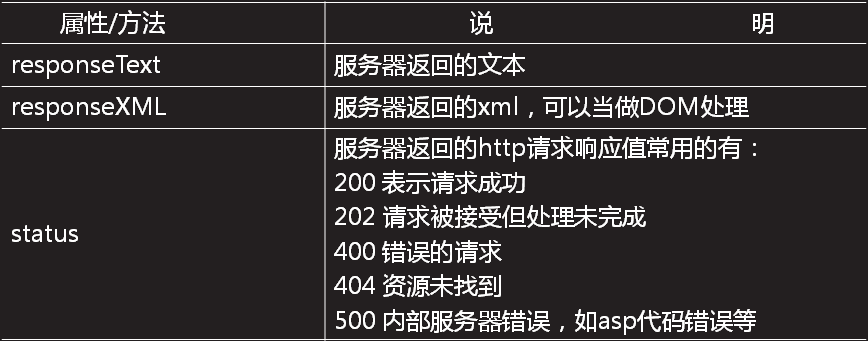
xhr = new ActiveXOject(‘Microsoft.XMLHttp’);

return xhr;

}

#### 3.3.2 AJAX对象的属性和方法





#### 3.3.3 onreadystatechange事件

作用：绑定一个事件处理函数，该函数来处理readystatechange事件。当AJAX对象的readyState值发生了改变，比如由0->1，就会触发readystatechange事件。

语法：

xhr.onreadystatechange=function(){

if(xhr.readyState == 4 && xhr.status == 200){

var result = xhr.responseText;

console.log(result);

}

}

### 3.4 AJAX的使用 ★

#### 3.4.1 异步请求的步骤

* 创建AJAX对象（xhr）

使用封装好的创建xhr的函数：[createXhr()](#_3.3.1_AJAX异步对象的创建)

* 绑定监听事件

使用onreadystatechange事件监听xhr对象的请求状态，即readyState的值变化。只有当readyState等于4并且status等于200时才表示①服务器接收请求数据（req）成功并且②正确处理之后③给出了响应。

【TIPS】当readyState的值为4的时候，才表示所有的响应都已经接收完成；200，表示服务器已经正确处理了请求并且给出了响应。代码见[3.3.3](#_3.3.3_onreadystatechange事件)，其中匿名函数是为异步对象xhr绑定的事件处理函数，它会执行五次（readyState会从0开始变化到4）。

* 打开连接

使用异步对象xhr的open方法，以指定的请求方法打开指定URL，并告诉服务器是否为异步方法。

作用：打开连接

语法：xhr.open(method,url,isAsyn);

**-** method:请求方法：string类型---get/post

**-** url请求地址：string类型。[特殊情况：](#_3.4.2_使用AJAX发送GET请求)发送GET请求时，需要传入数据并打开该数据指定的URL时,代码应如下：



**-** isAsyn：boolean类型

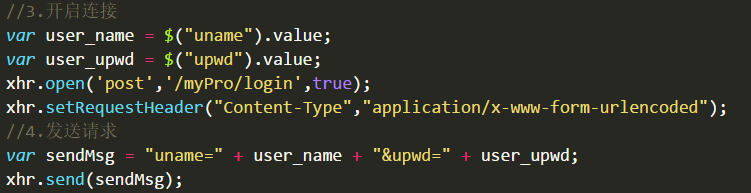
- true表示发送异步请求，当AJAX对象发请求时，用户仍然可以对当前页面做其他操作。

- false表示发送同步请求，当AJAX对象发请求时，浏览器会锁定当前页面，用户不能对当前页面做其他操作。

* 发送请求

使用异步对象xhr的send方法

语法：xhr.send(null/body) ——>body 请求主体，即需要提交的数据

**!注意：**如果没有提交的数据（[GET方法](#_3.4.2_使用AJAX发送GET请求)），则send方法内传递null。若要传递数据（[POST](#_3.4.3_使用AJAX发送POST请求)/PUT…），最佳实践是事先将需提交的数据用一个字符串变量（如sendMsg）存储在一起之后，用xhr.send(sendMsg)从客户端传递至服务器端进行获取（服务器端：req.body），从而使数据能够在服务器端进行相关操作。

【总结】以上异步请求四个步骤的具体代码如下：

//1.创建异步对象（采用[封装好的函数](#_3.3.1_AJAX异步对象的创建)进行创建）

var xhr = createXhr();

//2.绑定监听事件（回调事件处理函数）

xhr.onreadystatechange = function(){

if(xhr.readyState == 4 && status == 200){

var res = xhr.reponseText;

//DOM操作

}

}

//3.打开连接

xhr.open(method,URL,isAsyn);

//4.发送请求

xhr.send(body/null);

【练习】在demo.js文件中创建接口（ajaxExer），在接口中响应“这是我的第一个ajax练习！”；新建一个网页，网页中创建一个按钮和一个div，点击按钮时异步的向ajaxExer接口发送请求，并将响应回来的结果显示在div元素中。

#### 3.4.2 使用AJAX发送GET请求

使用AJAX发送GET请求和POST请求的差别主要在于异步请求步骤3和步骤4，即打开连接和发送请求这两步。其中，GET请求的一般写法为：

//3.打开连接

xhr.open(‘get’,’/myPro/deleteUser’,true); /\*不需要传参\*/

xhr.open(‘get’,’/myPro/deleteUser?uid=’+uid,true); /\*需要传参\*/

//4.发送请求

xhr.send(null);

【练习】在demo.js中创建login接口，在接口将用户名称获取，并响应输出获取到的用户名；创建一个login.html，在网页中创建一个用户名输入框和一个按钮，点击按钮时，异步的将用户名称（将用户名以参数的方式拼接在url后）提交给服务器端（login接口）。

#### 3.4.3 使用AJAX发送POST请求

使用AJAX发送GET请求和POST请求的差别主要在于异步请求步骤3和步骤4，即打开连接和发送请求这两步。其中，POST请求的一般写法为：

//3.打开连接

xhr.open(‘post’,’/myPro/login’,true);

/\*设置以下代码才能使服务器端接收到数据，具体原因见补充知识点\*/

xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

//4.发送请求

/\*方法一\*/

var user\_name = $("uname").value;

var user\_upwd = $("upwd").value;

var sendMsg = "uname="+user\_name+"&upwd="+user\_upwd;

xhr.send(sendMsg);

/\*方法二\*/

xhr.send(“uname = ” + value1 + “,upwd = ” + value2);

【补充】关于AJAX请求头的设置详解：

<https://blog.csdn.net/qlcql/article/details/51206972>

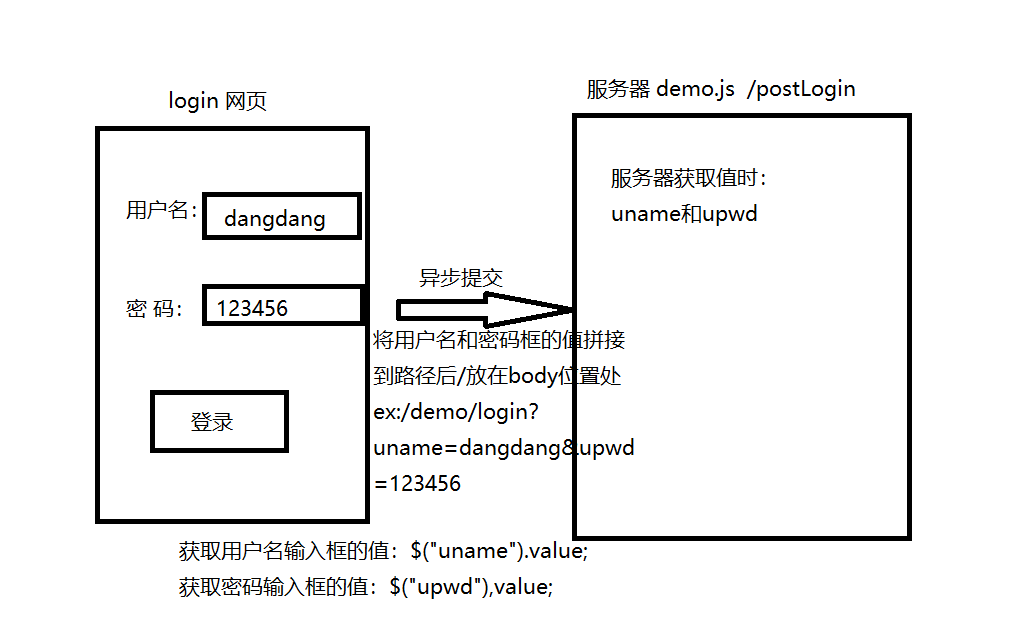
<https://blog.csdn.net/xmLove93/article/details/78839901>

【练习】

1.在demo.js中创建一个list接口，在接口红去完成xz用户表的查询，并将结果相应回前端网页。

2.新建一个05-list.html网页，在网页中添加一个按钮，点击按钮时异步的向服务器端的list接口发起请求，并取回结果数据打印在控制台。

### 3.5 AJAX前后端异步传值图解



## 四、JSON

### 4.1 JSON定义

JSON：JavaScript Object Notation，js对象表示法，即以js对象的数据格式表现出来的字符串。

### 4.2 JSON语法

#### 4.2.1 JSON数据格式

JSON中用一对{}来表示一个对象，对象中的属性名称必须用“”（双引号）引起来，值如果是字符串的话，也必须用“”引起来。整个式子表现出来的是字符串，所以在最外层加单引号。

【例子】

普通字符串：var tom = “汤姆”;

JSON字符串：var tom = '{“name”:”汤姆”}';

普通数组：var arr = [“tom”,”lilei”,”lucy”];

JSON数组：

‘[

{

“name”:”tom”,

“age”:20

},

{

“name”:”lilei”,

“age”:21

},

{

“name”:”lucy”,

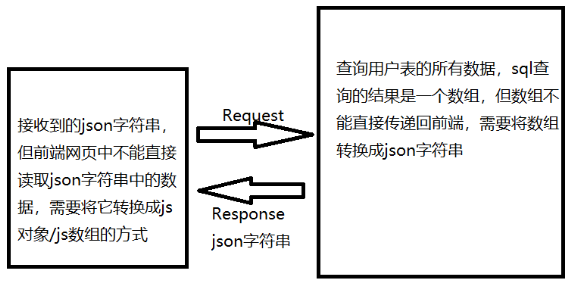
“age”:18

}

]’

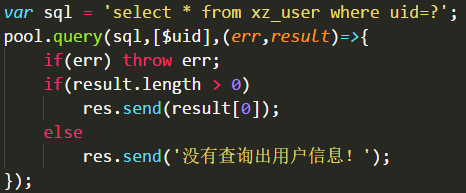
#### 4.2.2 JSON应用 ★

应用场景：服务器端响应回来的响应消息（res）是一个JSON字符串，如要使用res中的具体数据，应先使用JSON.parse()来将JSON字符串转换为js对象之后，才能取出相应数据给客户端显示。



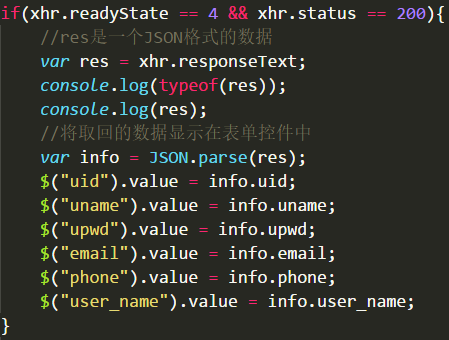
var obj = JSON.parse(“从后端响应回来的json字符串”);

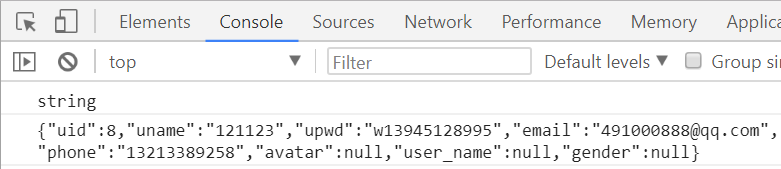
【说明】为了进一步说明JSON.parse()的用法，举出如下例子以便理解：



左图是客户端getInfo()函数中的代码（绑定到指定元素的onclick事件中以触发），xhr.responseText中的数据就是上面服务器端响应回来的result[0]，要使用它就必须先将其转为js对象，才能调用具体数据。

左图是服务器端的js代码，按照客户端传过来的uid查询该用户的所有信息之后用result[0]作为响应消息传回客户端，它是一个JSON格式的字符串





左图是上述客户端代码console.log()语句打印出来的数据

## 五、XML

### 5.1 XML定义

eXtensible Markup Language,可扩展标记语言

XML的标记没有被预定义过，需要自行定义

XML的宗旨：做数据传递，而非数组显示

### 5.2 XML语法

XML可以保存独立的\*.xml文件，也可以以字符串的方式出现

* XML的声明需要放在文件的顶端

- 以?开头并且以?结束

- 声明必须从文档的0行0列位置开始

- 声明只有三个属性：

version 指定XML文档版本。必须属性，一般为1.0

encoding 指定当前文档的编码。可选属性，默认值为utf-8

standalone 指定文档独立性。可选属性，默认值为yes，表示文档是独立文档；如果为no则表示当前文档不是独立文档，会依赖外部文件。

<?xml version=”1.0” encoding=”utf-8” standalone=”yes”?>

* 标记必须成对出现 <name></name>
* 标记严格区分大小写，开始和结束标记必须一致
* 标记允许被嵌套，注意嵌套顺序
* xml标记都允许自定义属性，格式与html一致，但属性值，必须要“”括起来
* 每个xml文件，必须有一个根元素

【练习】完善student.xml文件的数据，并新建一个网页点击按钮时异步的向student.xml文件发起请求，将xml数据取回前端网页进行展示

### 5.3 使用AJAX请求XML数据

请求的步骤同[3.4.1](#_3.4.1_异步请求的步骤)

**-** 创建异步对象

var xhr = createXhr();

**-** 绑定监听事件

**-** 打开连接

**-** 发送请求

### 5.4 解析XML文档对象的内容

核心方法：getElementsByTagName(“标签名称”);

返回值：返回一个包含指定元素们的“类数组”

【课后作业】

1.准备工作：在xz文件夹中创建了myPro文件夹，然后在routes文件夹创建了myPro.js路由，并将路由挂载到指定目录下，并静态托管了myPro文件夹

2.完成登录功能（异步发起请求）：

在myPro.js中创建login接口，获取客户端传递过来的用户名称和密码，并将用户名和密码拿到数据库中完成查询并给出提示（登录成功/用户名或密码错误）。

在myPro文件夹中新建一个网页01\_login.html，在网页中添加用户名和密码框，点击按钮时将用户名和密码异步地提交给服务器，完成登录功能。

3.将取回的用户列表数据以表格的方式展示在网页中（参考05\_list.html）。

4.在myPro.js中创建deleteUser接口，在接口中接收从客户端传递过来的用户uid，并根据uid去数据中完成用户的删除功能，给出提示（删除成功/删除失败）。

5.新建04-register.html页面，用异步的方式将数据提交给myPro.js中的register接口，完成注册功能，在注册之前需要判断用户名是否已存在（在myPro.js中创建selectUname接口，根据前端提交过来的用户名去数据库中查询用户名是否已存在）。

### 【Project】

1.完成登录功能

2.将用户数据查询出来，以表格方式展示在网页中

3.在myPro.js中创建deleteUser接口，在接口中接收从前端传递过来的用户uid,并根据uid去数据中完成用户的删除功能，给出提示（删除成功/删除失败）。

4.点击"02-list.html"页面中的修改连接，将用户uid传递给03-sel-update.html,在03-sel-update.html网页中加载出当前用户的信息，并展示在表单控件中。

5.在03-sel-update.html网页中修改输入框中的信息后点击"修改"按钮,依赖于form表单用post方法将修改后的数据提交给myPro.js中的update接口，完成数据库用户信息的修改。

# 第七章 CSS3基础

## 一、CSS概述

### 1.1 什么是CSS

CSS:Cascading Style Sheet层叠样式表/级联样式表，简称样式表。

### 1.2 CSS的作用

设置HTML网页元素的样式（修饰网页）

### 1.3 HTML与CSS的关系

HTML：构建网页内容

CSS：修饰网页

### 1.4 HTML属性与CSS属性的使用原则

W3C建议尽量使用CSS属性来取代HTML的属性

## 二、CSS语法规范

### 2.1 使用CSS的方式

#### 2.1.1 内联方式（行内样式）

特点：将CSS样式定义在HTML的开始标记中

语法：<any style=”CSS样式的声明”></any>

样式声明：由样式属性和值组成，CSS属性名与其值之间用冒号连接，多个样式声明之间用分号分隔。

常用的属性和值：

* 设置文本颜色的属性和值

属性：color

值：颜色的英文单词（red,blue,yellow…）

* 设置背景颜色的属性和值

属性：background-color

值：颜色的英文单词

* 设置字体大小的属性和值

属性：font-size

值：以px/pt为单位的数字（24px）

#### 2.1.2 内部样式

在网页的头元素中增加一对<style></style>，在<style></style>标记中声明该网页的所有样式。

语法：

样式的规则：由选择器和样式声明组成

<head>

<style>

样式规则

</style>

</head>

选择器：规范页面中哪些元素能够使用定义好的样式

【例子】元素选择器：由元素名称作为选择器

语法：元素名称{样式声明}

ex:div{

color:red;

font-size:40px;

background:pink;

}

#### 2.1.3 外部样式

独立于任何网页的位置处，声明一个样式文件（\*.css），在css文件中保存样式规则。

使用步骤：

* 创建样式表文件（.css作为后缀）
* 在文件中编写样式规则
* 在网页中引入外部样式表

<link rel=”stylesheet” href=”CSS文件url”>

【练习】新建一个网页，页面中创建p,div好h3分别放上一段文本，然后用外部样式表修饰：p元素的样式为红色字体，大小为30px；div元素中文本的样式为蓝色字体，40px；h3元素中的文本样式为绿色，背景颜色为灰色。

### 2.2 CSS的样式特点

#### 2.2.1 继承性

子元素继承父元素的样式（有层级关系），大部分的样式可以直接继承。

#### 2.2.2 层叠性

可以为一个元素定义多个样式规则，规则中的属性在不冲突的情况下，可以同时都应用到当前元素上。

#### 2.2.3 优先级

在样式声明有冲突时（重复），则按照样式的优先级来应用样式

优先级由低到高：

浏览器默认设置——>低

样式表（内部样式和外部样式）——>中：就近原则

内联样式——>高

#### 2.2.4 调整优先级

!important 规则

将!important加在属性值的后面，与值之间用空格隔开

ex:color:red !important;

【练习】网页中新建一个div元素，用内部样式设置其字体大小为40px，字体颜色为红色，然后用外部样式设置其字体大小为70px，字体颜色为蓝色，将外部样式引入到网页中查看页面的效果，然后交互内外样式的位置，再查看页面的效果，最后用！important去改变优先级。

## 三、选择器 ★

### 3.1 通用选择器

语法：\*{样式声明}

作用：可以匹配页面上的任意元素

**!注意：**通用选择器的效率较低，尽量少用

### 3.2 元素选择器

语法：元素名称{样式声明}

作用：设置页面上某种元素的样式

ex:div{…}

p{…}

header{…}

### 3.3 类选择器

作用：定义页面上某个元素或某些的样式（谁想用谁就可以使用）

特点：通过元素的class属性来引用类选择器

语法：

**-** 声明：.类名{样式声明}

**-** 引用：<any class=”类名”></any> .类名{样式声明}

**!注意：**定义类名时，点不能省略，类名不能以数字开头，不能包含特殊字符，但可以包含-和\_

类选择器的特殊用法：

**-** 多类选择器：让元素可以引用多个类选择器

<any class=”类名1 类名2 …”></any>

**-** 分类选择器：将元素选择器和类选择器结合使用

元素名称.类名{样式声明}

ex:div.test{color:red;}

【练习】页面中新建的div和p元素，内容随意用类选择器为所有元素设置字体为红色；用分类选择器为p元素设置背景颜色为黑色。

### 3.4 id选择器

作用：设置指定id元素的样式（专属定制）

语法：#id{样式声明}

### 3.5 群组选择器

作用：将多个选择器放在一起进行样式的声明

语法：选择器1，选择器2，选择器3…{样式声明}

### 3.6 后代选择器

作用：通过元素的后代关系匹配元素

后代：一组或多级嵌套

语法：选择器1 选择器2 选择器3 …{样式声明}

【课后作业】按以下布局完成样式设置：用后代选择器完成第一个span元素样式的设置，字体大小为32px，文本颜色为红色，背景颜色为黄色。

<div>

<p class="one">

<span>这是第一个span元素</span>

</p>

<p>

<span>这是第二个span元素</span>

</p>

</div>

### 3.7 子代选择器

作用：通过元素的子代（一层嵌套）关系匹配元素

语法：选择器1>选择器2{样式声明}

### 3.8 伪类选择器

作用：匹配元素的不同状态

语法：所有的伪类以:开始——> 选择器:伪类选择器{样式声明}

伪类选择器分类：

* 链接伪类

:link 匹配元素尚未访问时的状态

:visited 匹配元素访问过的状态

* 动态伪类

:hover 匹配鼠标悬停在元素上时的状态

: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.在页面中增加a标记，设置一下效果：链接未访问时，颜色为红色，没有下划线（text-decoration:none）；鼠标悬停时文本颜色为黄色；被激活时为橙色；访问后为绿色。

2.在页面中再增加一个输入框，默认文本颜色为灰色，字体为斜体（font-style:italic）：1）被激活时，字体为非斜体（fon-style:normal）；2）获取焦点时，文本颜色为红色。

### 3.9 选择器优先级

权值：标识当前选择器的重要程度，权值越大的优先级越高

**-** 元素选择器 1

**-** 类选择器/伪类选择器 10

**-** ID选择器 100

**-** 内联样式 1000

**!注意：**多个选择器的权值加到一起，大的优先级越高；若权值相加后相等，以定义的先后顺序为准，后定义的覆盖前面定义的样式。

## 四、尺寸与边框

### 4.1 单位

#### 4.1.1 尺寸单位

px:像素（1024\*768）

in:英寸（1in=2.54cm）

pt:磅（1pt=1/72in） 一般用于表示文字大小

cm:厘米

mm:毫米

em:相对单位（相对于元素本身字体的大小乘以倍数，如：2em）

rem:与em用法大致一样，只不过是相对于根元素字体的大小乘以倍数，如:3rem

%:相对单位（50%）

#### 4.1.2 颜色单位（取值）

* 英文单词

red,blue,pink,purple,yellow,orange…

* rgb(r,g,b)

rgb(0,0,255)

* #rrggbb

由6位16进制的数字和字母表示一个颜色：由0~9和a~f组成

#eeeeee 灰色



* #rgb

#eeeeee ——> #eee

* #rgba(r,g,b,alpha)

alpha:透明度，取值0~1的小数，值越小越透明

### 4.2 尺寸属性

作用：改变元素的宽度和高度

语法：

* 宽度width

min-width:最小宽度

max-width:最大宽度

* 高度height

min-height:最小高度

max-height:最大高度

* 页面中允许设置尺寸的元素：

①所有块级元素都允许设置尺寸，如:div、p、h1-h6、结构标记、ul、ol、dl

②行内块元素允许设置尺寸，如:表单控件元素（单选/复选框除外）

③本身具备width和height属性的元素，如:img、table

**!注意：**大部分的行内元素是无法修改尺寸的

### 4.3 溢出处理

作用：当内容多，元素区域小的时候，就会产生溢出效果，默认都是纵向溢出。

属性：

overflow/overflow-x/overflow-y

取值：

visible 可见的，溢出可见，默认值

hidden 隐藏的，溢出的内容全部隐藏，不可见

scroll 显示滚动条，仅产生溢出时可用

auto 自动，溢出时自动显示滚动条并可用

【练习】参考视频中效果，完成溢出处理（边框的设置：border:1px solid black;）

### 4.4 边框

#### 4.4.1 边框的简写方式

border:width style color;

width:边框的宽度，以px为单位的数字

style:边框的样式，取值：solid实线、dotted点线、dashed虚线

color:边框颜色，合法的颜色值，也可以为transparent透明色

**!注意：**取消边框border:0px/none;

【练习】在网页中创建一个id为main的div元素，设置尺寸为200px\*200px，并设置四个方向的边框为1px的宽度，实线，橙色。

#### 4.4.2 定义单边样式

只设置某一条边的三个属性

语法：border-top/right/bottom/left:width style color

【练习】在上面练习的基础上，设置上边框为2px，颜色为蓝色，边框样式为虚线（dottoed）

#### 4.4.3 设置单个属性

设置四条边框的某一个属性

语法：border-color: red; / border-style: dashed; / border-width: 5px;

#### 4.4.4 设置单边的单个属性

只设置某一条边的某一个属性

语法：border-top-style: solid;

#### 4.4.5 边框倒角

将元素的四个角，由直角倒成圆角

语法：border-radius:值; 取值：以px为单位的数字或百分比%

单角：border-top-left-radius/border-bottom-left/border-top-right/border-bottom-radius

#### 4.4.6 边框阴影

语法：box-shadow

取值：

h-shadow:阴影在水平方向的偏移，必须值；取值为正往右偏移

v-shadow:阴影在垂直方向的偏移，必须值；取值为正往下偏移

blur:阴影模糊距离，取值越大，模糊效果越明显

spread:阴影的尺寸，指定要在基础阴影上扩充出来的大小距离

color:阴影的颜色，合法的颜色值

inset:将默认的外阴影改为内阴影

【练习】

1.在上午溢出练习基础上，当鼠标悬停在图片上时，为图片添加向右下方偏移的阴影效果，偏移尺寸为5px，阴影颜色为灰色。

2.设置一个圆形的外发光效果（外阴影）。

#### 4.4.7 轮廓

轮廓指的是边框的边框，绘制与边框外的一条线。

属性：outline:width style color

outline:0/none 取消轮廓

## 五、框模型 ★

### 5.1 框模型概述

框：页面上的元素皆为框

框模型：box model盒模型，定义了元素框处理元素的内容，内边距，外边距以及边框的一种计算方式。

外边距：元素与元素之间的间距

内边距：元素边框与元素内容之间的间距

* 框模型的计算方式

元素实际占地宽度=左右外边距+左右内边距+左右边框+width

元素实际占地高度=左右外边距+左右内边距+左右边框+height

### 5.2 外边距

#### 5.2.1 基本用法

定义：围绕在元素边框外的空白间距，就是外边距

属性：

margin:定义某元素四个方向的外边距

margin-top/right/bottom/left:定义单边

取值：

以px为单位的数字

取值为负数时元素向着相反的方向移动

取值为%（占父元素宽或高的百分比）

取值为auto（自动计算外边距的值，实现块级元素的水平居中对齐）

【练习】页面中添加一个div元素，这是尺寸和边框，查看默认效果，然后设置div的左外边距60px，查看页面效果，再讲做外边距设为-60px查看页面效果；上外边距做同样设置。

#### 5.2.2 简写方式

margin: valule; 设置四个方向的外边距

margin: v1 v2;上下 左右

margin：v1 v2 v3; 上 左右 下

margin: v1 v2 v3 v4;上 右 下 左（顺时针）

#### 5.2.3 自带外边距的元素

p、h1~h6、body、ul、ol、dl、pre

通过CSS Reset的方式来重置具备外边距的元素

p,h1,h2,h3,h4,h5,h6,body,ul,ol,dl,pre{margin: 0;}

#### 5.2.4 外边距的特殊效果

* 外边距合并

当两个垂直（上下）外边距相遇时，他们将合并成一个，最终的距离取决于两个外边距中较大的那个，如果两个值相同，取其中一个。

* 行内元素和行内块级元素

**-** 行内元素垂直外边距无效，但左右外边距有效并会相加

**-** 行内块元素，设置垂直外边距时，整行元素都会跟着发生变化。

【练习】在页面中创建两个div，尺寸都为200\*200，分别设置不同的背景，在第二个div中添加一个子元素div,设置其尺寸为100\*100，同时设置背景颜色。设置子元素的上外边距为50px，观察页面效果。

* 外边距溢出

在特殊的条件下，为子元素设置上边距时会作用到父元素上：

**-** 父元素没有上边框

**-** 为第一个子元素设置上边距时

解决方案：

**-** 为父元素增加上边框，弊端：对父元素的高度有影响

**-** 使用父元素的内边距来取代子元素的上外边距，弊端：对父元素的高度有影响

**-** 在父元素的第一个子元素的位置处，增加一个空<table></table>

【练习】在页面中创建两个div，尺寸200px\*200px，分别设置不同的背景：再在第二个div中添加一个子元素div，设置其尺寸为100\*100，同时设置背景颜色；最后设置子元素的上边距为50px，观察页面效果

【课后作业】

1.完成学子商城登录页面效果（参考http://www.codeboy.com/login.html，背景图先用灰色背景替代）

2.整理今天所学内容思维导图

3.预习明天内容

### 5.3 内边距

#### 5.3.1 基本用法

元素边框与内容之间的间距，内边距会扩大元素边框占地区域。

属性：

padding:value 四个方向的内边距

padding-top/bottom/right/left:value 定义单边

取值：

**-** 以px为单位的数字；

**-** 以%为单位的数字，表示相对于父元素的占比。

#### 5.3.2 简写方式

- padding:值 表示四个方向的内边距

- padding:v1 v2 v1表示上下，v2表示左右

- padding:v1 v2 v3 v1上，v2左右，v3下

- padding:v1 v2 v3 v4 v1上，v2右,v4下，v3左

#### 5.3.3 box-sizing属性

作用：指定框模型的计算方式

取值：

* content-box

默认值，采用默认的方式计算元素的占地尺寸。

实际宽度=左右外边距+左右内边距+上下边框+width

实际高度=左右外边距+左右内边距+上下边框+height

* boder-boxtransform

元素的尺寸，包含border以及padding的值。

实际宽度=width+margin

实际高度=height+margin

## 六、背景

以单一颜色或图片来填充元素作为背景。

### 6.1 背景颜色

background-color:值 值为合法的颜色值

**!注意：**背景颜色从边框处开始填充。

### 6.2 背景图片

background-image:值 值为图片的url

ex：background-image:url(img/bj02.jpg);

#### 6.2.1 背景图平铺

background-repeat:值

取值：

- repeat 平铺

- no-repeat 不平铺。只显示一次图片

- repeat-x 只在水平方向平铺

- repeat-y 只在竖直方向平铺

#### 6.2.2 背景图片尺寸

background-size:值

取值：

- width height

- width% height% 该属性是以父元素的百分比来设置图片的宽度和高度的，第一个值是宽度，第二个值是高度。如果只设置一个值，那么第二个值会被设置为 “auto”（约束比例）;

- cover 将背景图等比放大，直到背景图完全覆盖元素所有区域为止。

- contain 将背景图等比放大，直到背景图与元素任意一边重合为止。

#### 6.2.3 背景图片固定

背景图片固定在网页某个位置，一致在可视化区域中，不会随着滚动条而发生位置的变化。

background-attachment:值

取值：

- scroll 默认值，滚动

- fixed 固定

#### 6.2.4 背景图片定位

改变背景图在元素中的位置。

background-position:值

取值：

- x y

x为背景图在水平方向的偏移距离，取值为正向右移动，反之向左移动。

y为背景图在垂直方向的偏移距离，取值为正向下移动，反之向上移动。

- x% y%

以页面左上角为原点，相对于父元素的百分比

### 6.3 背景属性简写方式

background:color url() repeat attachment position

### 6.4 渐变

渐变是指多种颜色平缓变化的一种显示效果

渐变的主要因素：色标——一种颜色及其出现的位置

一个渐变是由两个或两个以上的色变组成。

#### 6.4.1 线性渐变

以直线的方式来填充渐变色。

background-image:linear-gradient(angle,color-point1,color-point2,...)

* angle 表示渐变填充个方向或角度，取值

- 0deg / to top 从下向上填充

- 180deg / to bottom 从上向下填充

- 270deg / to left 从右向左填充

- 90deg / to right 从左向右填充

* color-point 表示色标颜色及其位置，取值：

ex：

- red 0% 在填充方向开始的位置处颜色为红色

- green 50% 在填充方向一半位置处，颜色为绿色

- blue 200px 在填充方向上200px的位置处，颜色为蓝色

#### 6.4.2 径向渐变

以圆形的方式来填充渐变色。

background-image:radial-gradient(size at position,color-point1,color-point2...)

* size 半径，以px为单位的数字
* position 圆心的位置，取值：

- x y 具体数字

- x% y% 元素款、高的占比

- 关键词 如left/center/right top/center/bottom

#### 6.4.3 重复渐变

将线性渐变或者径向渐变重复几次出现。

* 重复线性渐变：

background-image:repeating-linear-gradient(angle,color-point)

* 重复径向渐变

background-image:repeating-radial-gradient(size at position,color-point)

**!注意：**在重复渐变色标位置用绝对值，不能用%。

#### 6.4.4 浏览器兼容性

部分浏览器低版本不支持渐变属性，需要通过设置浏览器前缀的方式使其支持渐变。

* firefox -moz-
* IE -ms-
* Oprea -o-
* chrome&safari -webkit-

ex：

background:-ms-repeating-linear-gradient(0deg,red,yellow,

blue)

【TIPS】查询是否兼容的一个第三方网站：<https://caniuse.com/>

## 七、文本格式化属性

### 7.1 字体属性

#### 7.1.1 字体类型

font-family:值

取值：字体名称列表，中间用逗号隔开

ex：font-family:"微软雅黑","黑体","宋体",Arial;

#### 7.1.2 字体大小

font-size:值

取值：以px/pt为单位的数字

ex：font-size:32pt;

#### 7.1.3 字体加粗

font-weight:值

取值：

- bold 加粗

- normal 正常

- value 无单位的整百倍数字（400=normal，900=bold）

#### 7.1.4 字体样式

font-style:值

取值：normal/ italic 正常/斜体

#### 7.1.5 小型大写字母

将小写字母变为大写，但大小与小写字母一致。

font-variant:值

取值：normal/small-caps 正常/小型大写字母

### 7.2 字体属性的简写方式

font:style variant weight size family;

**!注意：**使用简写方式时，必须设置family的值，否则无效。

### 7.3 文本属性

#### 7.3.1 文本颜色

color:值

取值：合法的颜色值

#### 7.3.2 文本排列

指定文本、行内、行内块元素的水平对齐方式。

text-align:值

取值：

left/center/right/justify 左/中/右/两端对齐

#### 7.3.3 文字修饰（线条）

text-decoration:值

取值：

- none 无任何线条修饰

- underline 下划线

- overline 上划线

- line-through 删除线（s标记）

#### 7.3.4 行高

line-height:值 设置一行数据所占的高度。

取值：

- 以px为单位的数字 表示行高是多少；

- 无单位的数字 表示行高是当前字体大小的倍数

**!注意：**如果行高设置得比文本高度大，那么该行文本将在指定行高内呈现垂直居中的效果。

#### 7.3.5 首行缩进

text-indent:值

取值：以px为单位的数字

#### 7.3.6 文本阴影

text-shadow: h-shadow v-shasow blur color

【课后作业】

1.创建一个400\*400px的表格，4行4列：

①设置每个单元格的尺寸为100\*100;

②设置表格和单元格的边框为1px solid #000;

③尝试着为每个td增加上外边距10px。

2.完成"课后作业.png"效果图中的布局（注意细节修饰）

3.可以开始做个人项目（参考http://www.cssmoban.com）

## 八、表格

### 8.1 表格常用属性

* 边距：padding
* 边框：border
* 尺寸：width/height
* 文本格式化：font-\*、text-\*、line-height
* 背景属性：颜色、图片、渐变
* vertical-align：指定单元格数据的垂直对齐

取值：top/middle/bottom 上/中/下

### 8.2 表格特有属性

#### 8.2.1 边框合并

属性：border-collapse

取值：

- collapse 默认值，即分离边框模式

- separate 边框合并

#### 8.2.2 边框间距

作用：设置单元格之间或单元格与表格之间的距离

属性：border-spacing

取值：

- 指定一个值，水平和垂直的间距相同

- 指定两个值，值1表示水平的间距，值2表示垂直的间距

**!注意：**只有再分离边框的模式下才能设置边框的间距，即border-collapse为collapse

#### 8.2.3 标题位置

属性：caption-side

取值：

- top 默认值，标题在表格正上方

- bottom 标题在表格的正下方

#### 8.2.4 显示规则

作用：用来帮助浏览器指定如何布局一张表，也就是指定td尺寸的计算方式

属性：table-layout

取值：

- auto 默认值，即自动表格布局，列的尺寸实际上是由内容来决定的

- fixed 固定表格布局，列的尺寸由设定的值为准

【区别】自动表格布局和固定表格布局：

* 自动布局

- 单元格大小适应内容（特点）

- 自动布局会比较灵活（优点）

- 表格比较复杂时，加载速度较慢（缺点）

- 适用于不确定每列大小并且表格不太复杂时（应用场合）

* 固定布局- 单元格的尺寸取决于设定的值

- 任何情况下都会加速显示表格（优点，计算机不用去计算，而是按照给定值直接显示）

- 固定表格不够灵活（缺点）

- 适用于每一列的大小确定时使用

## 九、定位/浮动定位 ★

### 9.1 定位概述

定义：指的是改变元素在页面中的位置

分类：按照定位的效果，可以分成以下几类：

- 普通流定位（默认定位方式）

- 浮动定位

- 相对定位

- 绝对定位

- 固定定位

### 9.2 定位相关属性

#### 9.2.1 定位属性

属性：position

取值：

- static 静态，默认值

- relative 相对定位

- absolute 绝对定位

- fixed 固定定位

**!注意：**将元素的position设置为relative/absolute/fixed任何一个值，那该元素就称为"已定位"元素

#### 9.2.2 偏移属性

top/bottom/left/right

以上四个数学取值均为数字（px单位）

ex:

top: 150px; 元素向下偏移150px

bottom: -200px; 元素向下偏移200px

left: 100px; 元素向右偏移100px

**!注意：**只有"已定位"元素才能使用偏移属性

### 9.3 普通流定位

普通流定位又称“文档流定位”，页面中的元素默认定位方式。

* 每个元素在页面中都是在父元素左上角开始显示
* 每个元素在页面中一定都有自己的空间
* 块级元素都是按照从上往下排列的，每个元素独占一行
* 行内元素以及行内块元素都是按照从左往右的方式来排列

### 9.4 浮动定位 ★

#### 9.4.1 浮动定位的特点

如果将元素的定位方式设置为浮动定位，元素将具备以下特点：

* 元素将被排出文档流之外（脱离了文档流），不再占页面空间，其他未浮动的元素将上前补位
* 浮动元素会停靠在其父元素的左边或右边，或其它已经浮动元素的边缘上
* 浮动定位解决的问题：让多个块级元素在一行中显示

#### 9.4.2 浮动定位的语法

属性：float

取值：

- none 默认值，即无任何浮动效果

- left 左浮动，让元素停靠在父元素的左边，或挨着左侧已经浮动的元素边缘

- right 右浮动，让元素停靠在父元素的右边，或挨着右侧已经浮动的元素边缘

【练习】页面中创建三个元素，id分别为d1，d2，d3设置其尺寸各500px,分别设置背景颜色为blue,green,yellow,依次让d1,d2,d3实现右浮动，观察效果，然后再依次将d1,d2,d3分别实现左浮动观察效果。

#### 9.4.3 浮动引发的特殊效果

* 当父元素显示不下所有已浮动元素时，最后一个将换行，但是，有可能被“卡住”
* 元素一旦浮动起来后，宽度将以内容为主（未指定宽度情况下）

- 如果设置元素宽度为auto,元素内部又无内容撑开，该元素不可见

* 元素一旦浮动起来，将变为块级元素；允许设置尺寸，正常处理垂直外边距
* 文本、行内元素，行内块元素时采用环绕的方式排列的，是

#### 9.4.4 清除浮动

元素一旦浮动起来之后，就会对后续元素带来影响（后续元素要上千补位），那么就可以通过清除浮动的方式来解决。

属性：clear

取值：

- left 清除当前元素前面的元素左浮动带来的影响

- right 清除当前元素前面的元素右浮动带来的影响

- both 清除当前元素前面的元素任何一种浮动带来的影响

- none 默认值，不做任何清除操作

#### 9.4.5 浮动元素对父元素高度带来的影响

元素的高度，都是以未浮动元素的高度为准，浮动元素是不占页面空间的，所以一旦子元素都浮动起来，那么父元素高度为0

解决方案：

- 设置父元素高度

弊端：不是每次都知道父元素的高度

- 设置父元素也浮动

弊端：不是任何时候父元素都需要浮动，而且浮动元素会影响后续元素（补位）

- 设置父元素的overflow

取值：hidden/auto

弊端：如果有内容要溢出显示的话，也会一同隐藏起来

- 在父元素的最后一个子元素后面，追加一个空的块级元素，并设置其clear:both;

### 9.5 相对定位

定义：元素会相对于它原来的位置偏移某个距离

语法：

position：relative；

配合着偏移属性使用

使用场合：元素位置的微调时使用

**!注意：**相对定位的元素不会脱离文档流

### 9.6 绝对定位

定义：元素会相对于特定祖先元素或body的位置偏移某个距离

特点：

- 绝对定位的元素会脱离文档流，不占页面空间

- 绝对定位的元素会相对于离它最近的已定位的祖先元素，如果没有已定位的祖先元素，那么元素就会相对于body去实现位置的初始化

- 绝对定位的元素若为行内元素则定位后会变为块级元素

【TIPS】一般子元素要求实现绝对定位，父元素通常都设置为相对定位

语法：

position：absolute；

配合偏移属性来使用

使用场合：弹出效果or有堆叠效果的元素

### 9.7 固定定位

定义：将元素固定在页面的某个位置，位置不会随着滚动条而发生位置的变化，始终固定在可视化的区域中

语法：postion:fixed 配合偏移属性来使用

**!注意：**

- 固定定位脱离文档流，不占页面空间

- 固定定位的元素会变成块级元素，允许修改尺寸

- 固定定位的元素永远都是相对于body去实现位置的初始化

### 9.8 堆叠顺序

一旦将元素变为已定位元素的话，元素们有可能会出现堆叠的效果。如果想要改变默认的堆叠顺序，可以使用：z-index

属性：z-index

取值：无单位的数字，数字越大越靠上

**!注意：**

- 父子元素间，z-index无效，永远都是子元素压在父元素上。

- 只有已定位的元素才能使用z-index

## 十、显示

### 10.1 显示方式

决定元素在网页中的表现形式（块级、行内、行内块等）

属性：display

取值：

- none 不显示元素，即隐藏元素

- block 让元素表现的与块级元素一致

特点：从上往下排列，元素独占一行，并可以设置尺寸和垂直外边距

- inline 让元素表现得与行内元素一致

特点：排列方式从左往右在一行中显示，不能修改尺寸（img除外）

- inline-block 让元素表现的与行内块一致

特点：排列方式如同行内元素，但是具备块级元素的特征，即允许修改尺寸，也可以设置垂直外边距，但会影响整行。

- table 让元素表现的如同table

特点：尺寸以内容为主；每个元素独占一行；允许修改尺寸

**!注意：**一般将行内元素显示为块级或行内块，很少将块级转换为行内

### 10.2 显示效果

#### 10.2.1 显示/隐藏

属性：visibility

取值：

- visible 可见的，默认值

- hidden 元素不可见，即隐藏

【练习】在网页中新建两个div，id分别为d1和d2，设置各自的尺寸和背景颜色，先设置一个div的display:none，查看页面效果，注释掉第一步的代码，然后再设置第一个div的visibility:hidden；在查看页面效果

**!注意：**display:none和visibility:hidden的区别：

display:none会脱离文档流，不占页面空间；visibility:hidden不脱离文档流，占据页面空间。

#### 10.2.2 透明度

属性：opacity

取值：0-1 值越小越透明

**!注意：**rgba()与opacity的区别：

opacity作用于元素，跟元素相关的颜色都会发生透明度的变化，rgba()只作用于当前某一个颜色值

### 10.3 垂直对齐方式

属性：vertical-align

使用场合：

- 表格中使用

取值：top/middle/bottom

- 图片中使用

取值：top/middle/bottom/baseline基线对齐（默认值）

**!注意：**编写网页时，通常都会将图片的垂直对齐方式更改为除baseline外的值

### 10.4 光标

作用：改变鼠标悬停在元素上时鼠标的状态

属性：cursor

取值：

- default 箭头

- pointer 小手状态

- crosshair + （十字准线）

- text I

- wait 等待

- help 帮助

## 十一、列表

### 11.1 列表项标识

属性：list-style-type

取值：

- none 无标识/disc/circle/square

### 11.2 列表项图像

属性：list-style-img

取值：url(图片路径)

### 11.3 列表项位置

属性：list-style-position

取值：

- outside 默认值，将标识放在li的外面

- inside 将标识放在li的里面

### 11.4 列表属性的简写

属性：list-style

取值：type url() position

常用方式：list-style: none;

【课后作业】

1.复习今天所学内容，重点-浮动定位

2.完成"课后作业.png"中的布局效果（样式参考www.codeboy.com网站首页头部）

3.做个人大项目（参考 <http://www.cssmoban.com>）

# 第八章 CSS3高级

## 一、 复杂选择器

### 1.1 兄弟选择器

兄弟元素：具备相同父元素的平级元素之间成为兄弟元素

ex:

<div>

<p></p>

<p></p>

<span></span>

<h6></h6>

</div>

#### 1.1.1 相邻兄弟选择器

作用：获取紧紧挨在某元素后的兄弟元素

语法：选择器1+选择器2{}

**!注意：**兄弟选择器，只能向后找，不能向前找

#### 1.1.2 通用兄弟选择器

作用：获取某元素后所有满足条件的兄弟元素

语法：选择器1~选择器2{}

【练习】通过兄弟选择器设置样式：

1.h3与div之间有30px的外边距

2.从第二个div开始，每个div都有10px的上外边距

<h3>静夜思</h3>

<div>床前明月光</div>

<div>疑是地上霜</div>

<div>举头望明月</div>

<div>低头思故乡</div>

### 1.2 属性选择器

id,class,style,title,name,type,value等

作用：允许通过元素所附带的属性及值来匹配页面元素

#### 1.2.1 基本写法

语法：[attr] attr:属性

作用：匹配页面中所有附带attr属性的元素

ex:[id] 匹配页面中所有附带id属性的元素

[class] 匹配页面中所有附带class属性的元素

#### 1.2.2 elem[attr]

elem 表示任意元素

attr 表示任意属性

作用：匹配页面中所有附带attr属性的elem元素

ex：div[id] 匹配带有id属性的div元素

#### 1.2.3 [attr1][attr2]…

作用：匹配同时附带多个属性的元素

ex:[id][class] 匹配页面中同时附带id属性和class属性的元素

#### 1.2.4 [attr=value]

作用：匹配页面中attr属性值为value的元素

[attr~=value] 获取属性值中包含指定词汇（完整单词）的元素

[attr^=value] 匹配属性值以指定值开头的每个元素

[attr$=value] 匹配属性值以指定值结尾的每个元素

[attr\*=value] 匹配属性值中包含指定值的每个元素

ex:[id="one"]{ color:red;} 等同于 #one{ color：red;}

### 1.3 伪类选择器

#### 1.3.1 目标伪类

作用：突出显示活动的HTML锚点元素，匹配被激活的锚点

语法：

选择器:target{…}

#### 1.3.2 结构伪类

作用：通过元素的结构关系来匹配元素（上下级嵌套）

##### 1.3.2.1 :first-child

匹配的元素时属于其父元素中的第一个子元素

ex: p:first-child ——>p1

<div>

<p>p1</p>

<p>p2</p>

<p>p3</p>

</div>

##### 1.3.2.2 :last-child

匹配的元素时属于其父元素中的第一个子元素

ex: p:last-child ——>p3

【练习】

- 创建一个4\*4的表格，并填充内容

- 设置第一行的数据背景颜色为黄色

- 设置最后一行的数据背景颜色为红色

##### 1.3.2.3 :nth-child(n)

匹配的元素时属于其父元素中的第n个元素

ex: tr:nth-child(3){ color:red;}

**!注意：**n值从1开始，即nth-child(1)为其父元素下的第一个子元素

【练习】在上面表格的基础上，设置表格第三行第三列中的文字为粗体并倾斜显示，字体大小为30px

##### 1.3.2.4 :empty

匹配页面中没有子元素的元素（不能有任何内容，包括空格）

##### 1.3.2.5 :only-child

匹配的元素是属于其父元素中的唯一子元素

#### 1.3.3 否定伪类

将满足指定选择器的元素给排出在外

语法：:not(selector){…}

ex:

### 1.4 伪元素选择器

/\* 除第一行以外的所有字体都变为斜体，茶色 \*/

tr:not(:first-child){

font-style: italic;

color: teal;

}

/\* 除第一列以外的所有字体都变为粗体 \*/

td:not(:first-child){

font-weight: bold;

}

#### 1.4.1 基础伪元素

* ::first-letter/:first-letter

作用：用于匹配元素的首字符

* ::first-line/:first-line

作用：用于匹配元素的首行

**!注意：**如果首字符与首行样式冲突时，优先使用首字符样式

* ::selection

作用：匹配被用户选取的内容（拖动鼠标选中一段文本、图像…内容）

**!注意：**只能修改文本颜色和背景颜色

【练习】创建一个div元素，内容随意：

- 设置div中首字符样式为加粗，红色，大小为32px，右外边距为20px

- 将首行的文本设置为橙色，斜体

- 用户选取内容时文本颜色为蓝色，背景为灰色。

#### 1.4.2 内容生成的伪元素

使用CSS动态的向元素中插入一段内容

##### 1.4.2.1 伪元素及其特有属性

* ::before/:before

作用：匹配到某元素的内容（文本or标签）区域之前

ex:<p>(:before)床前明月光</p>

* ::after/:after

作用：匹配到某元素的内容（文本or标签）区域之后

ex: <p>床前明月光(:after)</p>

属性：content

作用：向匹配到的位置处增加的内容

取值：

- 字符串，用""引起来

- url(图片路径)

【练习】

<p>学而时习之，不亦说乎</p>

<p>有朋自远方来，不亦乐乎</p>

输出显示：

子曰：学而时习之，不亦说乎

——摘自《论语》

子曰：有朋自远方来，不亦乐乎

——摘自《论语》

##### 1.4.2.2 实际应用

* 上外边距溢出

- 在父元素的第一个子元素位置处（父元素选择器::before）

- 添加一个table标记，并且标记的内容为空

ex:

选择器::before{  
 content:"";

display: table;

}

* 解决浮动元素对父元素高度带来的影响

- 在父元素的最后一个子元素位置处

- 添加一个块级元素（display:block;）

- 要设置其clear: both;

## 二、弹性布局 ★

### 2.1 弹性布局概念

弹性布局（Flexible layout）是一种布局方式，主要解决某元素中“子元素”的布局方式，为布局提供最大的灵活性。

### 2.2 弹性布局的相关概念

* 弹性布局容器：子元素的父元素称为容器
* 项目：要实现布局的子元素
* 主轴：项目们排列方向的一根轴
* 交叉轴：与主轴垂直相交的一根轴

### 2.3 弹性布局用法

#### 2.3.1 指定为弹性（flex）布局

将元素变为flex容器后，那么所有子元素都将变为flex项目，都允许按照弹性布局的方式排列。

属性：display

取值：

- flex 将块级元素变为容器

- inline-flex 将行内元素变为容器

**!注意：**元素设置为flex容器之后，子元素的float,clear,vertical-align将失效；元素变为flex容器之后，子元素尺寸允许被修改（把子元素变为块级）。

#### 2.3.2 容器属性

* flex-direction

作用：指定容器的主轴及其排列方向

取值：

- row 默认值，即主轴为x轴，起点在左端

- row-reverse 主轴为x轴，起点在右端

- column 主轴为y轴，起点在顶端

- column-reverse 主轴为y轴，起点在底端

* flex-wrap

作用：当一个主轴排列不下所有的项目时，项目是否换行显示

取值：

- nowrap 默认值，即空间不足时也不会换行，项目自动缩小

- wrap 换行，第一行在上方

- wrap-reverse 换行并反转，第一行在下方

* flex-flow

作用：是flex-direction和flex-wrap的缩写形式

取值：

- row nowrap 默认值

- direction wrap

* justify-content

作用：定义项目在主轴上的对齐方式

取值：

- flex-start 在主轴的起点对齐

- flex-end 在主轴的终点对齐

- center 在主轴上居中对齐

- space-between 两端对齐

- space-around 每个项目两端的间距相同（两个项目之间间距要大一些，因为它是两个项目间距之和）

* align-items

作用：定义项目在交叉轴上的对齐方式

取值：

- flex-start 在交叉轴的起点对齐

- flex-end 在交叉轴的终点对齐

- center 交叉轴上居中显示

- baseline 交叉轴上基线对齐

- stretch 如果项目未设置尺寸，在交叉轴上沾满所有空间

【练习】网页中添加一个div，id为parent，设置parent的边框,在parent中再添加两个子元素div，id分别为d1和d2设置d1和d2的尺寸和背景颜色,将父元素变为容器，并设置d1和d2在容器中水平、垂直都居中显示，并且当容器缩小时，不改变项目的尺寸。

#### 2.3.3 项目属性

该组属性只能设置在某一个项目元素上，只控制一个项目，不影响其他项目

* order

作用：定义项目的排列顺序，值越小，越靠近起点，默认值为0

取值：无单位的整数数字

* flex-grow

作用：定义项目的放大比例，如果容器有足够的剩余空间，项目按照比例放大（将剩余空间大小按指定比例分）

取值：

- 整数数字，无单位

- 默认为0，即不放大；

- 取值越大，占据剩余空间的比例越多

* flex-shrink

作用：定义项目缩小比例，即容器空间不足时，项目按照比例缩小

取值：

- 整数数字，无单位

- 默认值为1

- 空间不足时，则等比缩小

- 取值为0，则不缩小

* align-self

作用：定义当前项目在交叉轴的对齐方式

取值：

- flex-start 起点对齐

- flex-end 终点对齐

- center 居中

- baseline 基线对齐

- stretch 未设定项目高度时沾满父元素空间

- auto 自动继承父元素align-items的属性值

## 三、转换

### 3.1 转换概述

改变元素在页面中的位置，大小，角度以及形状的一种方式。

转换分为2D转换和3D转换：

2D转换：只在x轴和y轴上发生转换效果

3D转换：增加z轴的转换效果

### 3.2 转换属性

#### 3.2.1 设置转换

属性：transform

取值：

- none 默认值，元素无任何的转换效果

- transform-function 表示一个或多个转换函数

如果是多个转换函数，中间用空格隔开

#### 3.2.2 设置转换原点

属性：transform-origin

取值：center center(中心点) / 0 0(左上角)

- 数值/百分比/关键词

- 2个值表示原点在x轴和y轴的位置

- 3个值表示原点在x轴、y轴和z轴的位置

**!注意：**默认的原点是在元素的中心位置 center center/50% 50%

### 3.3 2D转换

#### 3.3.1 位移

作用：改变元素在页面的位置

属性：transform

取值：

- translate(x) 指定元素在x轴上位移的距离，取值为正，元素右移

- translate(x,y) 指定元素在x轴和y轴上位移的距离，取值为正，元素右下移

- translateX(x) 指定元素在x轴上位移的距离

- translateY(y) 指定元素在y轴上位移的距离

#### 3.3.2 缩放

作用：改变元素在页面的尺寸

属性：transform

取值：

- scale(value) 表示横向和纵向缩放比例一致。value取值为数字，默认值为1，即不缩放。

value>1 放大

0<value<1 缩小

value<0 放大（水平和垂直方向都翻转180度）

- scale(x,y) x:横向的缩放比例；y:纵向的缩放比例

- scaleX(x) 只在横向进行缩放

- scaleY(y) 只有纵向进行缩放

#### 3.3.3 旋转

作用：改变元素在页面上的角度

属性：transform

取值：

- rotate(ndeg) n值为正，顺时针旋转；反之

**!注意：**元素旋转时，坐标轴也会跟随旋转，会影响到旋转后的位移效果；转换原点会影响最后的转换效果

【课后作业】在页面中创建两个div，id分别为d1和d2,设置其尺寸为200\*200px,添加不同背景;

- 将两个div都定位到父元素的左上角显示;

- 将d1先旋转45deg,再向右位移50px;

- 将d2向右位移50px，再旋转45deg，对比两个div的效果。

#### 3.3.4 倾斜

作用：改变页面中元素的形状（正方形->平行四边形）

属性：transform

取值：

- skewX(xdeg) 让元素向着x轴的方向产生倾斜的效果，实际改变的是y轴的角度

x值为正，y轴逆时针倾斜

x值为负，y轴顺时针倾斜

- skewY(ydeg) 让元素向着y轴的方向产生倾斜的效果，实际改变的是x轴的角度

y值为正，x轴顺时针倾斜

y值为负，x轴逆时针倾斜

- skew(x) 等同于skewX(xdeg)

- skew(x,y) 同时在x,y轴产生倾斜

【练习】页面创建一个200\*200px的元素，设置背景颜色，当鼠标悬停在元素上时，钙元素向右移动200px，向下移动200px，旋转135deg，再向着x轴倾斜45deg，同时放大1.5倍。

### 3.4 3D转换

#### 3.4.1 透视距离

模拟人的眼睛到3D转换元素之间的距离

属性：perspective

**!注意：**该属性要加在3D元素的父元素上

#### 3.4.2 3D旋转

属性：transform

取值：

- rotateX(xdeg) 以x轴为中心轴，旋转元素的角度（爆米花机器，烤羊腿）

- rotateY(ydeg) 以y轴为中心轴，旋转元素的角度

- rotateZ(zdeg) 以z轴为中心轴，旋转元素的角度

- rotate3D(x,y,z,ndeg) x,y,z取值为大于0的数字时，表示该轴参与旋转，取值为0则表示不参与旋转。

## 四、过渡

### 4.1 过渡概述及用法

使用CSS样式在一段时间内平缓变化的效果

属性：transition-property

作用：指定哪个属性值在变化的时候需要用过渡的方式来体现

取值：

- all 能使用过渡的属性，一律用过渡来体现

- 具体属性名

ex:

- transition-property:background;当背景的属性在发生变化的时候用过渡来体现。

- transition-property:all; 所有属性发生变化时都用过渡体现

### 4.2 允许设置过渡效果的属性

* 颜色属性
* 取值为数字的属性
* 转换属性
* 渐变属性
* visibility属性

### 4.3 指定过渡时长

作用：指定在多长时间内完成过渡操作

属性：transition-duration

取值：以s或ms为单位的数值

### 4.4 指定过渡的速度时间曲线函数

属性：transition-timing-function

取值：

- ease 默认值，慢速开始，快速变快，慢速结束

- linear 匀速

- ease-in 慢速开始，加速结束

- ease-out 快速开始，慢速结束

- ease-in-out 慢速开始和结束，中间先加速后减速

### 4.5 指定过渡的延迟

属性：transition-delay

取值：以s或ms为单位的数值

### 4.6 过渡属性的编写位置

将过渡放在元素声明的样式中，既管去，又管回

将过渡放在触发操作中，只管去，不管回

### 4.7 过渡的简写方式

属性：transition:property duration timing-function delay;

ex:

transition:background 2s linear; //正确

**!注意：**property和duration是必须值

【练习】1.在页面中创建一个100\*100px的元素

2.鼠标悬停时，5秒钟之内，向右移动500px的同时旋转3圈

3.鼠标移除时，还能滚回来。

## 五、动画

### 5.1 动画概述

使用元素从一种样式逐渐的变化为另一种样式，即将多个过渡效果效果整合到一起

动画是通过“关键帧”，来控制元素变化的每一步

关键帧：

- 动画执行的时间点

- 在该时间点上的样式是什么

### 5.2 使用动画

#### 5.2.1 声明动画

为动画起名，定义关键帧

语法：

@keyframes 动画名称{

/\*定义 关键帧\*/

0%{

//动画开始时元素的样式

}

25%{

//动画执行到1/4时的样式

}

50%{

//动画执行到一半时的样式

}

100%{

//动画执行结束时的样式

}

}

#### 5.2.2 调用动画

将声明好的动画，通过使用动画名称的方式来添加给元素

##### 5.2.2.1 调用动画的属性

* animation-name

作用：指定调用动画的名称

* animation-duration

作用：指定动画执行一个周期的时长

取值：以s或ms为单位的数字

* animation-timing-function

作用：指定动画播放速度时间曲线函数

取值：ease/linear/ease-in/ease-out/ease-in-out

* animation-delay

作用：指定动画的延迟时间

取值：以s或ms为单位的数字

* animation-iteration-count

作用：指定动画的播放次数

取值：

- 具体的数字

- infinite 无限次播放

* animation-direction

作用：指定动画的播放方向

取值：

- normal 默认值，正常播放0%~100%

- reverse 逆向播放100%~0%

- alternate 轮流播放（奇数次正向播放，偶数次逆向播放）

* animation-fill-mode

作用：规定动画在播放前或播放后的状态

取值：

- none 默认值

- forwards 动画播放完成后，将保持在最后一个帧的状态上

- backwards 动画播放前，延迟时间内，动画保持在第一帧的状态上

- both 是forwards和backwards的结合体

* animation-play-state

作用：指定动画是处于播放还是暂停的状态上

取值：paused/running 暂停/播放

##### 5.2.2.2 调用动画的简写方式

animation:name duration timing-function delay iteration-count direction;

#### 5.2.3 浏览器兼容

如果要兼容低版本浏览器的动画效果，可以使用加前缀：

ex:

@-webkit-keyframes 动画名{…}

@-moz-keyframes 动画名{…}

@-ms-keyframes 动画名{…}

@-o-keyframes 动画名{…}

## 六、CSS优化

### 6.1 CSS优化原则

* 尽量减少HTTP请求的个数
* 页面顶部引入css文件
* 将css和js放到外部独立的文件中

### 6.2 CSS代码优化

* 合并样式
* 缩小样式文件
* 代码压缩
* 选择更优的样式属性值
* 页面中尽量避免空的src和href

# 第九章 Bootstrap

官网：<https://getbootstrap.com/>

中文官网：<https://bootcss.com/>

## 一、响应式布局

### 1.1 响应式网页

Responsive web page:响应式网页，可以根据浏览设备的不同（PC/PAD/PHONE），自动地更改布局，图片文字效果，不会影响用户的浏览体验。

#### 1.1.1 响应式网页必须做到

- 布局：不能固定宽度，必须是流式布局

- 文字和图片大小随着容器大小而改变

- 使用媒体查询

#### 1.1.2 响应式网页存在的问题：

代码的复杂大大提升，适用于内容不太多的页面

#### 1.1.3 如何测试响应式网页

- 方式一：使用真实设备测试

好处：测试效果真实可靠

不足：测试任务量较大

- 方式二：使用第三方模拟测试软件

好处：无需添置太多真实设备，方便测试

不足：测试效果有限

- 方式三：使用Chrome等浏览器自带模拟设备

好处：简单方便

不足：测试效果有限，必须进一步验证

#### 1.1.4 如何编写响应式网页

##### 1.1.4.1 声明元标签（viewport-手机适配）

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">

- viewport元标签：

早期3G手机为例浏览大尺寸的网页，只能强行把网页缩小，导致文字，图片等变小

IOS提出了“视口”的概念，视口用于盛放网页内容，超过部分，滑动显示

Android也借用了此概念

width=device-width 表示视口的宽度就是设备的宽度

initial-scale=1.0 表示视口是否缩放，1.0不缩放

maximum-scale=1.0 表示允许用户缩放的最大倍率

minimum-scale=1.0 表示允许用户缩放的最小倍率

user-scalable=0 表示是否允许用户手动缩放，取值yes/no/1/0

##### 1.1.4.2 所有的容器/文本/图片 使用相对尺寸不使用绝对尺寸

ex:

div{  
 /\*width: 1000px; 绝对值\*/

width: 100%; //相对值

}

##### 1.1.4.3 流式布局+弹性布局

实现方法：float+flex

##### 1.1.4.4 使用CSS3媒体查询技术——响应式布局必须条件

Media：媒体，指的是浏览网页的设备，如：screen(pc/pad/phone),tv(电视),print(打印机)

Media Query：媒体查询，可以自动根据当前设备类型的不同，或者特性（方向，尺寸，解析度）不同，有选择性的执行部分的css样式，忽略其他部分。

* 媒体查询的两种使用方法

① 据媒体查询结果使用不同的外部css文件

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

<link media="screen and (min-width:992px)" rel="stylesheet" href="pc.css">

<link media="screen and (max-width:991px) and (min-width:768px)" rel="stylesheet" href="pad.css">

<link media="screen and (max-width:767px)" rel="stylesheet" href="phone.css">

**!注意：**此方法较少用，所有css都会被浏览器加载，每个css文件中可能会存在重复代码

【课后作业】

1.完成媒体查询练习，效果图参考"课后练习"文件夹中的效果。

2.开发个人项目

② 根据媒体查询的结果执行同一个css文件中的不同代码块

@media screen and (min-width:992px){

//选择器{样式}

}

## 二、Bootstrap css基础

简洁、直观、强悍的前端开发框架，让web开发更迅速、简单。

Bootstrap内容：

* 如何使用-入门
* 全局css样式表
* 组件
* JS插件
* 定制（scss）

### 2.1 Bootstrap入门

设置基本模板：

### 2.2 全局CSS类

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

<link rel="stylesheet" href="bootstrap.css">

<script src="jQuery.js"></script>

<script src="popper.js"></script>

<script src="bootstrap.js"></script>

#### 2.2.1 特点

* 所有标记的默认样式都统一化——重写

body,p,h1~h6,ul,dl等重写样式

* 设置box-sizing:border-box

#### 2.2.2 按钮开关

* 不同样式按钮：

.btn 基本按钮

.btn-success 成功

.btn-danger 危险

.btn-warning 警告

.btn-info 信息

.btn-primary 主要按钮

.btn-secondary 次要按钮

.btn-dark 黑色

.btn-light 浅色

.btn-link 链接

* 不同边框按钮：

.btn-outline-primary 主要按钮

.btn-outline-success 成功

.btn-outline-info 信息

…

* 不同大小：

.btn-lg 大按钮

.btn-sm 小按钮

.btn-block 块级按钮

#### 2.2.3 图片相关

.rounded 圆角图片

.rounded-circle 椭圆图片

.img-thumbnail 缩略图

.img-fluid 响应式图片

#### 2.2.4 文本相关

.h1~.h6 标题样式

.text-danger/success/info/dark/light/primary/muted 文字颜色

.text-uppercase/lowercase/capitalize 大写/小写/首字母大写

.text-left/center/right/justify/nowrap 对齐方式

.font-weight-bold/light/normal 加粗/更细/普通

.font-italic 斜体

#### 2.2.5 列表相关

.list-unstyled 去除标识的效果

.list-inline 行内列表

.list-group/.list-group-item 创建列表组

.active 激活状态

.disabled 禁用状态

.list-group-item-danger/success/warning/info/primary/dark/light… 列表项颜色

#### 2.2.6 表格相关

.table

.table-bordered 带边框的表格

.table-striped 隔行变色的表格

.table-hover 带鼠标悬停的表格

.table-responsive 响应式表格（用于table的父元素上）

.table-danger/success/warning/info/primary/dark/light… 表格颜色

#### 2.2.7 辅助类

* 浮动类：

.float-\*-left/.float-\*-right 左/右浮动

.clearfix 清除浮动

* 显示：

.visible/invisible 可见/隐藏

* 背景：

.bg-success/info/danger/warning/dark…

* 边框：

.border/border-0/border-top/border-bottom/border-left/border-right 边框设置

.border-success/info/danger/warning/dark... 边框颜色

.round/rounded-0 边框圆角

.rounded-top/bottom/left/right

* 居中对齐：

.m-auto 设置居中对齐

* 边距：

.m-0/1/2/3/4/5/ml/mr/mt/mb 来设置外边距

.p-0/1/2/3/4/5/pl/pr/pt/pb 来设置内边距

* 宽度：

.w-25/50/75/100 来设置元素的宽度

* 高度：

.h-25/50/75/100 来设置元素的宽度

### 2.3 栅格布局 ★

#### 2.3.1 web页面布局方式比较

web页面的布局方法有三种：

* table布局

优点：简单、容易控制

缺点：语义错误，渲染效率低

* div+css布局

优点：语义正确，渲染效率高

缺点：控制起来比较麻烦

* bootstrap的栅格布局

优点：简单、容易控制，语义正确，渲染效率高（div+css）

缺点：不适合过于复杂的页面

#### 2.3.1 栅格布局的使用

* 最外层需要使用容器

.container 定宽的容器

.container-fluid 变宽的容器

* 容器中添加行

.row(一行均分为12等份)

.col(每个列都需要指定宽度的占比n/12)

#### 2.3.3 屏幕尺寸的划分

* 针对不同的屏幕需要使用不同的列，如：

大屏幕：4/12 三列 左中右

中等屏幕：6/12 两列 左右

小屏幕：12/12 一列

.col-xl-1/2/3/…/12

.col-lg-1/2/3/…/12

.col-md-1/2/3/…/12

.col-sm-1/2/3/…/12

##### 2.3.3.1 尺寸划分标准

|  |  |  |  |
| --- | --- | --- | --- |
| 尺寸简写 | 尺寸全称 | 适配设备 | 具体适配的屏幕宽度值 |
| xl | Extra large | 超大屏幕 | width>=1200px |
| lg | large | 大pc屏幕 | 992px<=width<1200px |
| md | medium | 中等pc或pad | 768px<=width<992px |
| sm | small | 小屏幕 | 576px<=width<768px |
| xs | Extra small | 超小屏幕 | width<576px |

**!注意：**col不添加数字时，自动布局同一行的每个列的宽度占比相等，如：两个col，各占50%，三个各占33.3%，可以用col设置布局时的等宽效果。

col-xl-\* 只能在xl屏幕下有效

col-lg-\* 在lg/xl屏幕下都有效

col-md-\* 在md/lg/xl屏幕下都有效

col-sm-\* 在sm/md/lg/xl屏幕下都有效

【总结】列可以在当前屏幕以及更大的屏幕下有效（xs除外）

##### 2.3.3.2 响应式布局

一个div可以指定在不同屏幕下的不同占比

<div class="col-lg-3 col-md-6 col-sm-12"></div>

#### 2.3.4 列偏移

作用：可以使用“列偏移”实现指定列以及其后的列向右偏移效果

语法：列偏移通过offset-\*-\*类来设置

- 第一个\*号可以是sm/md/lg/xl，表示不同屏幕尺寸之下的显示规则

- 第二个\*号可以1到11的数字，这些类会把一个列的做外边距增加\*列

ex:.offset-md-4 在中等屏幕下向右偏移4列

#### 2.3.5 栅格布局可以嵌套

<div class="row">

<div class="col">

<div class="row">

<div class="col">显示内容</div>

</div>

</div>

</div>

### 2.4 弹性布局

使用.d-flex/.d-inline-flex类创建一个弹性容器

* .flex-row 设置弹性子元素水平显示，默认效果（左对齐）
* .flex-row-reverse 设置右对齐显示
* .flex-column 设置子元素垂直方向显示（顶部对齐）
* .flex-column-reverse 设置子元素垂直方向显示（底部对齐）
* .justify-content-\* 用于修改弹性子元素的排列方式，\*号的值有：

start,end,center,between,around

响应式flex类：

* d-sm/md/lg/xl-flex 根据不同屏幕来设置flex类，从而实现响应式布局
* flex-\*-row \*号表示sm/md/lg/xl
* flex-\*-column

### 2.5 表单

.form-group 堆叠表单（全屏宽度）-垂直方向

.form-inline 内联表单-水平方向

样式类：

.form-control/form-check/form-check-inline/form-check-input/form-check-label

### 2.6 组件

#### 2.6.1 下拉菜单

下拉菜单 必须 三级结构

<div class="dropdown">

<button data-toggle="dropdown">触发元素</button>

<ul class="dropdown-menu">菜单项元素（隐藏）</ul>

</div>

【TIPS】data-toggle="dropdown"会被bootstrap.js选中，添加事件监听：标题.dropdown.header; 分割线：.dropdown-divider;禁用：.disabled

#### 2.6.2 按钮组

* 在元素添加.btn-group类创建按钮组

<div class="btn-group">

<a href="#" class="btn btn-success">按钮1</a>

<a href="#" class="btn btn-info">按钮2</a>

…

</div>

* 使用.btn-group-lg/sm类来设置按钮的大小
* 使用.btn-group-vertical类来创建垂直的按钮组

#### 2.6.3 信息提示框

提示框可以使用.alert类，后面加.alert-success/info/danger/primary

<div class="alert alert-danger w-25">

<span data-dismiss="alert" class="close">X</span>

提示信息框

</div>

data-dismiss：驳回数据，关闭信息提示框

bootstrap.js监听事件

#### 2.6.4 导航

Bootstrap提供了三种形式的导航：水平导航、选项卡导航、胶囊导航

* 水平导航

创建一个简单的水平导航，可以在ul元素上添加.nav类，在每个li选项上添加一个.nav-item类，在每个链接上添加.nav-link类

【TIPS】通过设置ul的justify-content-\*来控制导航的对齐方式；

<ul class="nav">

<li class="nav-item">

<a href="#" class="nav-link">内容</a>

</li>

</ul>

设置ul的flex-colum类用于创建垂直导航的效果；

.nav-justified类可以设置导航项等宽显示。

* 选项卡导航

使用.nav-tabs类可以将导航转换为选项卡，ex:

<ul class="nav nav-tabs">

<li class="nav-item">

<a href="#" class="nav-link">内容</a>

</li>

</ul>

如果要设置选项卡是动态可切换的，可以在每个链接上添加data-toggle="tab"属性，然后在每个选项对应的内容上添加.tab-pane类

<ul class="nav nav-tabs">

<li class="nav-item">

<a href="#one" class="nav-link" data-toggle="tab">内容</a>

</li>

</ul>

/\*以下内容是点击选项卡时要切换的内容\*/

<div class="tab-content">

<div class="tab-pane" id="one">具体内容</div>

</div>

* 胶囊导航

.nav-pills类可以将当行设置为导航效果

要实现胶囊导航的动态切换需要设置data-toggle属性值为pill

#### 2.6.5 导航栏

使用.navbar类创建导航栏，后面紧跟.navbar-expand-xl/lg/md/sm类来创建响应式导航栏（大屏幕下水平铺开，小屏幕下垂直堆叠）。导航栏上的选项可以使用ul元素并添加navbar-nav类，然后在li元素上添加.nav-item类，a元素上添加.nav-link类。

#### 2.6.6 折叠

控制内容隐藏和显示，在a标记或button元素上添加data-toggle="collapse"属性，data-target="#id"属性是对应折叠的内容，ex:

<button class="btn" data-toggle="collapse" data-target="#demo"> </button>

<div id="demo" class="collapse">具体内容</div>

#### 2.6.7 折叠导航栏

要创建折叠导航栏，需要在按钮上添加.navbar-toggler类，并设置data-toggle="collapse"与data-target="#id"；然后再设置了class="collapse navbar-collapse"类的div上包裹导航内容（链接），内容所在的div元素的id匹配在按钮的data-target属性上。

<nav class="navbar navbar-expand-sm bg-warning">

<!--不折叠的内容-->

<a href="#" class="navbar-brand">Bootstrap中文网</a>

<button class="navbar-toggler navbar-light" data-toggle="collapse" data-target="#main" type="button">

<!--在小屏幕下按钮上的三条短横线效果-->

<span class="navbar-toggler-icon"></span>

</button>

<div id="main" class="collapse navbar-collapse">

<ul class="navbar-nav">

<li class="nav-item"><a href="#" class="nav-link">Bootstrap2中文文档</a></li>

<li class="nav-item"><a href="#" class="nav-link">Bootstrap3中文文档</a></li>

<li class="nav-item"><a href="#" class="nav-link">Bootstrap2中文文档</a></li>

<li class="nav-item"><a href="#" class="nav-link">Less教程</a></li>

</ul>

</div>

</nav>

#### 2.6.8 卡片

通过.card与.card-body类创建一个卡片：  
 .card-header类用于查看卡片的头部

.card-footer类用于创建卡片的底部

ex:

<div class="card">

<div class="card-header">头部</div>

<div class="card-body">内容</div>

<div class="card-footer">底部</div>

</div>

**!注意：**使用data-parent属性来确保所有的折叠元素在指定的父元素下，这样才能实现在一个折叠选项显示时隐藏其他项的内容。

【练习】卡片+折叠=手风琴效果

#### 2.6.9 媒体对象

处理图片或视频等内容的布局，应用场景有博客、评论、商品列表等。

创建媒体对象在容器上添加.media类，然后将多媒体内容放在子容器上，子容器需要添加.media-body类，可以使用align-self-\*相关类设置多媒体对象的图片显示位置（上中下）

<div class="media">

<img src="a.jpg">

<div class="media-body">内容</div>

</div>

#### 2.6.10 轮播

创建轮播：.carousel

为轮播添加指示符：.carousel-indicators

添加要切换的内容：.carousel-inner

指定每个图片的样式：.carousel-item

<div class="carousel" data-ride="carousel" id="demo">

<ul class="carousel-indicators">

<!--指示符-->

<li class="active" data-target="#demo" data-slide-to="0"></li>

<li data-target="#demo" data-slide-to="1"></li>

<li data-target="#demo" data-slide-to="2"></li>

<li data-target="#demo" data-slide-to="3"></li>

</ul>

<!--轮播图片-->

<div class="carousel-inner">

<div class="carousel-item active">

<img src="../img/5.jpg" alt="">

#### 2.6.11 模态框

</div>

<div class="carousel-item">

<img src="../img/6.jpg" alt="">

</div>

<div class="carousel-item">

<img src="../img/7.jpg" alt="">

</div>

<div class="carousel-item">

<img src="../img/8.jpg" alt="">

</div>

</div>

<!--左右箭头-->

<a href="#demo" class="carousel-control-prev" data-slide="prev">

<span class="carousel-control-prev-icon"></span>

</a>

<a href="#demo" class="carousel-control-next" data-slide="next">

<span class="carousel-control-next-icon"></span>

</a>

</div>

模态框是覆盖在父窗体上的子窗体，模态框可以在不离开父窗体的情况下有一些互动，提供信息交换等。

<button data-toggle="modal" data-target="#id"></button>

<div class="modal" id="main">

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header">头部</div>

<div class="modal-body">主体</div>

<div class="modal-footer">底部</div>

</div>

</div>

</div>

#### 2.6.12 其他小组件

##### 2.6.12.1 徽章

badge-\*

##### 2.6.12.2 巨幕

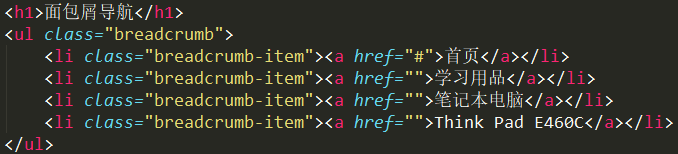
.jumbotron

##### 2.6.12.3 分页

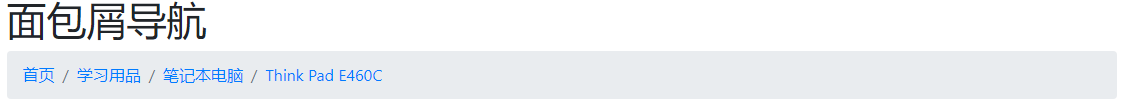
可以在ul元素上添加.pagination类，然后在li上添加.page-item类，a标记上添加.page-link

##### 2.6.12.4 面包屑导航

.breadcrumb

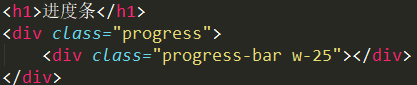


效果图：



##### 2.6.12.5 进度条

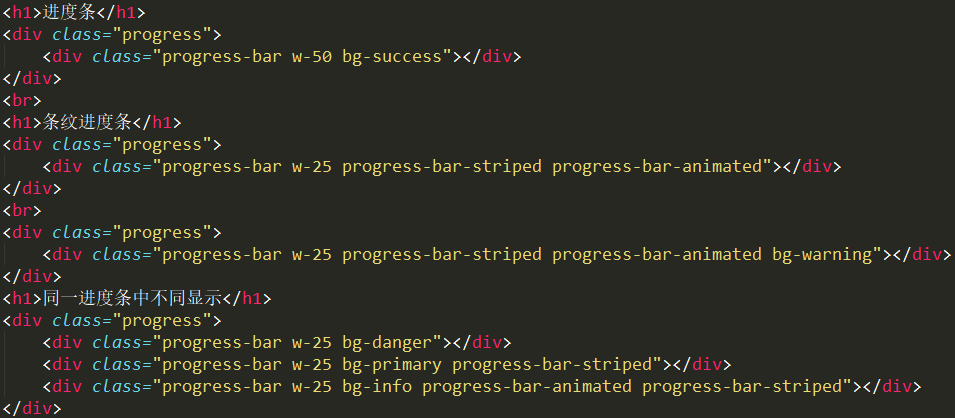
使用.progress类创建进度条



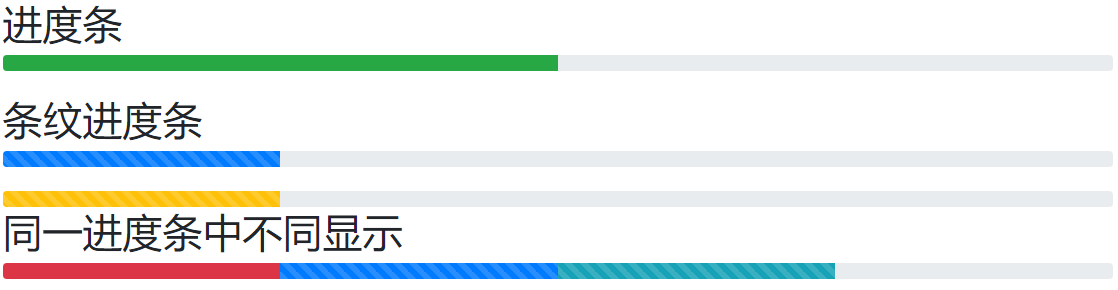
添加宽度（用百分比设置，表示进度条所在的位置）

使用.progress-bar-striped类设置条纹进度条

使用.progress-bar-animated类设置进度条的动画



效果图如下：



【重点】媒体查询技术，栅格布局，scss定义

## 三、 SCSS定制

ex:css文件

### 3.1 CSS缺点

语法不够强大，没有变量和合理的复用机制，使用逻辑上相关的属性值必须重复编写，导致难以维护。

动态样式语言为css赋予了动态语言的特性，极大地提高了样式语言的可维护性。

常用的动态样式语言：

* scss/sass 都属于css的预处理器（用一种专门的编程语言，进行web页面样式的设计，再通过编译器转化为正常的css文件，以供项目使用）
* less
* stylus

### 3.2 SCSS概述

scss是一款强化css的辅助工具，它在css语法的基础上增加了变量、嵌套、混合、导入、函数等高级功能，这些拓展相对于css更强大与优雅。

scss作用：有助于更好地管理样式文件，以及更高效的开发项目。

### 3.3 SCSS在服务器端的使用

* 安装NodeJS解释器
* 安装scss编译程序

- 在命令行界面输入：npm install -g node-sass

- 检查scss程序能否正常使用：node-sass -v（能看到版本号，说明可用）

* 编译scss文件

- 创建一个scss文件，编写动态样式代码

- 通过以下命令行来完成编译

|  |  |
| --- | --- |
| 在webstrom中用Alt+F12打开命令行界面 | |
| 刷新编译后的css文件 | ctrl+alt+y |
| 单文件的转换命令 | node-sass scss/input.scss css/output.css |
| 多文件的转换命令 | node-sass scss文件夹名称 -o css文件夹名称 |
| 单文件监听命令 | node-sass -w scss/input.scss css/output.css |
| 多文件监听命令 | node-sass -w scss文件夹名称 -o css文件夹名称 |

### 3.4 SCSS的基础语法

#### 3.4.1 变量

使用$符号标识表里，变量命名规范遵循css中选择器的命名规范，不能以数字开头，不能包含特殊字符（可以包含\_和-），尽量见名知意。

ex：

$jd\_red: #f10215; //颜色变量

$normal\_width: 100px; //数值变量

$before\_content: "子曰："; //字符串变量

$border\_style: solid; //样式变量

$my\_border: 1px $border $jd\_red; //声明变量时，变量的值也可以是其他变量

**!注意：**

- 声明变量时，变量值也可以引用其他变量；

- 变量定义在{}规则外，整个样式表中都可以引用；定义在{}规则内，则只能在规则块中使用

- 声明重复变量，只有最后一处有效，且会覆盖前面的样式，尽量不要重复

- !default规则，如果变量已经声明赋值了，那就用它声明的值，否则就用当前这个值的默认值

ex:

#### 3.4.2 嵌套规则

div{

dorder:$my\_border;

$width:200px; //局部变量，只能在当前选择器中使用

width:$width; //在此可以调用

}

nav{

width:$width; //在此不能调用$width

}

##### 3.4.2.1 在选择器中嵌套选择器

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| #main div.top h1{color:red;}  #main div.top p{margin: 10px;}  #main div.middle{  background:gray;} | #main{  div.top{  h1{color: red;}  p{margin:10px;}  }  div.middle{  background:gray;  }  } |

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| div a {color: red; }  div a:hover {color: blue; } | div {  a {color: red;  &:hover {color: blue;}  }  } |

在嵌套规则中，有时候需要使用嵌套外层的父选择器，可用&代表嵌套规则外层的父选择器，改写上述代码如下：

##### 3.4.2.2 群组选择器嵌套

ex:nav a,header a,section a{color: blue;}

用scss方式改写：

nav,header,section{a{color:blue;}}

##### 3.4.2.3 属性嵌套

在scss中除了选择器，属性也可以嵌套：

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| div{  border-style: solid;  border-width: 1px;  border-color: #000;  } | div{  border:{style:solid;width:1px;color:#000;}  } |

【练习】创建一个200\*200的div，正方形，红色背景，鼠标悬停时过渡到黄色背景，并且变为圆形，用scss来编写代码。

#### 3.4.3 导入文件

在scss中，局部文件名以\_开头，在scss编译时就不会编译这些以\_开头的文件，而只是把这些文件用作导入，引入局部文件时可以不写文件的全名，即省略下划线，并且局部文件的后缀也可以不写。同一个局部文件可以在不同的主文件中引用。

引入局部文件需要用@import

ex:定义了一个局部文件\_myColors.scss,引入该文件:@import "myColors"

**!注意：**@import里的css样式会在HTML页面内容先加载完后，才加载到页面。而link标签是一边加载HTML一边加载CSS。

#### 3.4.4 混合器

##### 3.4.4.1 普通混合器

使用@mixin定义混合器。

语法：@mixin 混合器名称{样式声明}

使用混合器：

选择器{@include 混合器名称}

##### 3.4.4.2 带参混合器

@mixin 混合器名称(参数1，参数2，…){

属性1:参数1;

属性2:参数2;

…

}

使用带参混合器：

选择器{ @include 混合器名称(实参1，实参2，…)}

【练习】定义一个带参的混合器，抽离出字体大小，颜色样式，粗细，在div和p元素中使用混合器，传入不同的参数，查看页面的效果。

#### 3.4.5 继承

通过@extend实现继承

继承是说一个选择器可以继承另一个选择器定义的所有样式

ex:

.round{

border:1px solid red;

border-radius:8px;

}

.round-shadow{

@extend .round;

box-shadow:0 0 5px 1px;

}

【课后作业】

1.练习scss中的转换命令。

2.在选择器.my\_border中定义1px的实线边框为透明色，元素的宽度为400px，距离左边有30px的外边距，在选择器.my\_bottom\_border中继承.my\_border样式，并且底边框为红色，鼠标悬停时底边框为蓝色。

3.定义一个带参混合器，将元素的宽度，高度，内边距和背景颜色作为参数传入到混合器中，然后在div和h3元素中使用混合器并传入不同实参，查看页面效果。

#### 3.4.6 运算

##### 3.4.6.1 数字

加减乘除、取整等运算（+,-,\*,/,%），如果必要会在不同单位间转换值（前提是scss能转）。

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| $my\_width: 5%;  $my\_max\_width: $my\_width\*4;  $other\_size: $my\_width+$my\_max\_width;  div{  width: $my\_width;  height: 100px; }  p{  width: $my\_max\_width; }  h1{  width: $other\_size; } | div{  width: 100px;  height: 100px; }  p{  width: 400px; }  h1{  width: 500px; } |

* 加法运算

“+”号也可以用于拼接字符串

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| p:before{  content:"Microsoft"+yahei;}  h2{  font-family: A+"rial"; } | p:before {  content: "Microsoftyahei"; }  h2 {  font-family: Arial; } |

如果用有引号的字符串去连接无引号的字符串，运算结果是有引号的；相反，无引号的字符串去连接有引号的字符串，运算结果则没有引号。

* 除法运算

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| div{  font-size: 10px/80px;  $width:100px;  width: $width/2;  height: (500px/2);  margin: 5px+10px/2px;} | div {  font-size: 10px/80px;  width: 50px;  height: 250px;  margin: 10px; } |

在css中通常起到分隔数字的用途，同时也赋予除法的功能，以下情况会被视为除法运算：

- 如果值，或值的一部分，是变量或者是函数的返回值

- 如果值被圆括号包裹时

- 如果只是算术运算的一部分

* 运算表达式与其他连用时，用空格做连接符

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| p{  margin: 5px+10px auto;  } | p{  margin: 15px auto;  } |

* 在有引号的文本字符串中使用#{}插值语句可以添加动态的值

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| p:before{  content:"I am #{5+15} years old!" } | p:before {  content: "I am 20 years old!"; } |

##### 3.4.6.2 颜色

颜色值的运算时分段计算的，也就是分别计算红色，绿色以及蓝色的值。

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| p{color: #010203+#040506;} | p{color: #050709;} |

计算：01+04——>05；02+05——>07；03+06——>09

**!注意：**值包含alpha，必须有相等的alpha值才能进行运算，因为算术运算不会作用于alpha。 ex:rgba(125,25,10,.5)+rgba(345,65,90,.5)

【练习】定义两个变量分别为10px和20px，两个变量相加后乘以2，将结果赋值给p选择器的width属性。

#### 3.4.7 函数

scss定义了多种函数，有些甚至可以通过普通的css语句调用。

ex:rgba(10,30,40,.7);在普通的css中也可以使用

##### 3.4.7.1 颜色函数

- hsl(hue,saturation,lightness);

hue:色调，取值0-360,0-120红色，120-240绿色，240-360蓝色

saturation:饱和度，取值为0%~100%

lightness:亮度，取值为0%~100%

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
|  |  |

##### 3.4.7.2 数字函数

- round(value); 将数值进行四舍五入，转换成最接近的整数

- ceil(value); 向上取整

- floor(value); 向下取整

- min(v1,v2,…,vn); 找出数值之间的最小值

- max(v1,v2,…,vn); 找出数值之间的最大值

- random(); 获取随机数

##### 3.4.7.3 字符串函数

- unquote(string) 删除字符串的引号

- quote(string) 添加字符串的引号

- To-upper-case(string) 将字符串小写字母转换为大写字母

- To-lower-case(string) 将字符串大写字母转换为小写字母

##### 3.4.7.4 自定义函数

scss支持自定义函数，使用关键词@function定义，并能在任何属性或scss中使用；需要调用@return输出结果。

语法：@function 函数名($param){

@return 运算表达式;

}

【练习】定义一个带参函数getSize($param),函数体内定义两个变量，值分别为60px和10px，函数最终返回$param乘以最大值，加上$param-1乘以最小值的结果，并将结果赋值给header元素的height属性。

##### 3.4.7.5 控制指令

* 当@if的表达式返回值不是false或null时，表示条件成立，执行{}内的代码

ex:

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| p{  @if (1>2) {boder-radius:5px;}  @if (5>3) {boder-radius:10px;}  @if (null) {boder-radius:20px;}  } | p {  boder-radius: 10px; } |

* @if声明后面可以跟多个@else if声明，或者一个@else声明

|  |  |
| --- | --- |
| **SCSS文件** | **CSS文件** |
| $type:moon;  nav{  @if $type==sun{  color:yellow;  }@else if $type==mars{  color: red;  }@else if $type==moon{  color: blue;  }@else { color: black;}  } | nav {  color: blue; } |

【练习】声明一个函数，在函数中判断：如果参数大于1返回值为200px，如果参数等于1返回值为100px，否则返回0，并在div元素上调用函数，查看运行结果。

# 阶段项目：学子商城首页商品页界面 ★

<http://www.codeboy.com/>

搭建项目结构：css文件+js文件+图片资源文件+网页

1.用Bootstrap的模板页创建一个header.html，同时创建一个header.css文件，存放到css文件夹中。

2.创建一个基础样式文件base.css，编写所有网页都会用到的样式代码

3.新建footer.html和footer.css布局网页底部

4.在index.html中引入网页的头部和底部

【课后作业】继续完成首页index.html的布局：轮播图、楼层布局

张东 [zhangdong@tedu.cn](mailto:zhangdong@tedu.cn)

课前准备：

1.犀牛书——《JavaScript权威指南》

2.关注微信公众号——前端大全

3.上届笔记——打印

# 第十章 JS高级

## 一、正则表达式

### 1.1 正则表达式概述

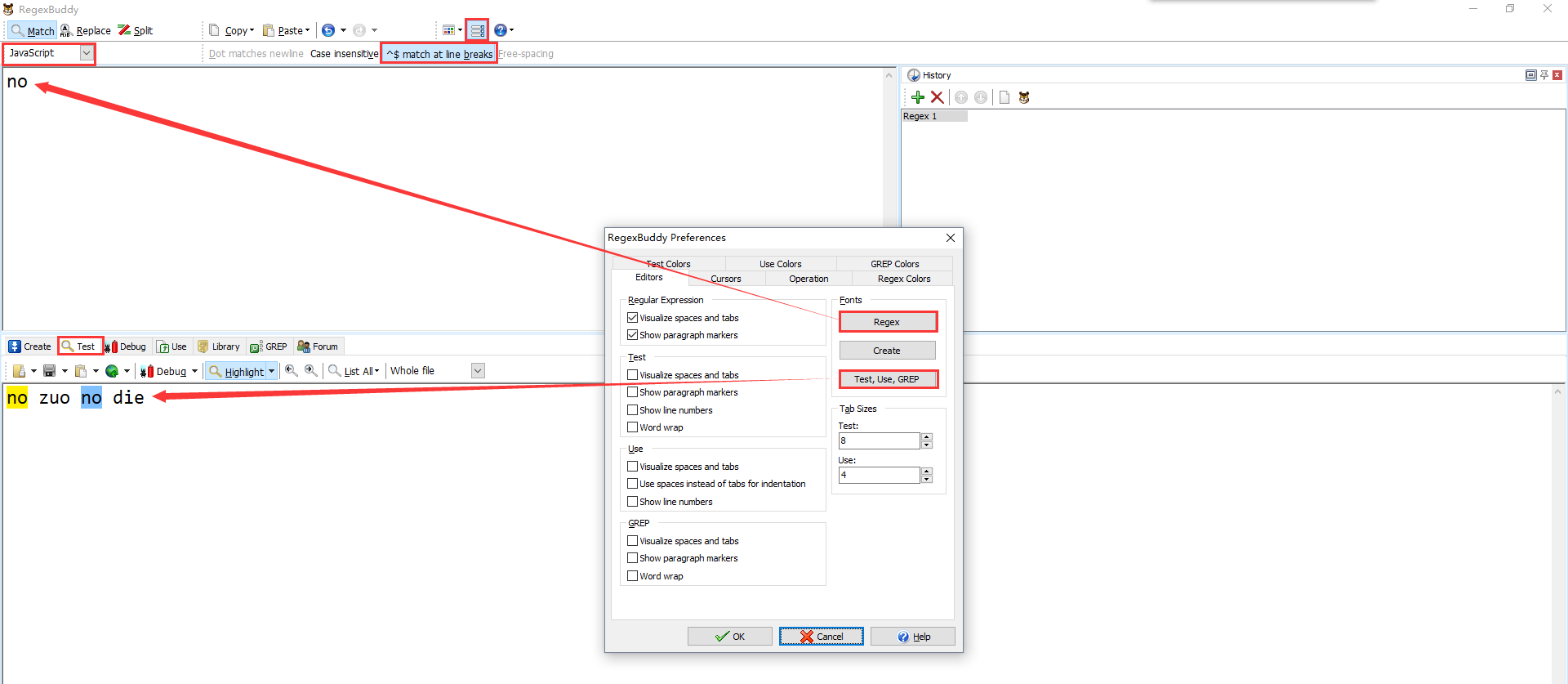
正则表达式(Regular Expression)是规定字符串中字符出现规律的规则。

用法：

1.模糊查找多种关键词

2.表单验证：格式验证

（传统方式模糊查找关键词）



【TIPS】最简单的正则表达式其实是关键词本身

### 1.2 使用正则表达式

#### 1.2.1 字符集（备选字列表）

定义：规定一位字符上备选字符列表的集合

何时：如果一位字符，有多种备选字时，就需要用到备选字符集：

使用：[备选字列表]

只要这一位置上的字符，匹配中括号中其一，就算符合规则

**!注意：**一个[]只能匹配一位字符！

简写: 如果备选字符列表是连续的，可简写为[x-x]

|  |  |
| --- | --- |
| **字符内容** | **备选字列表** |
| 一位数字 | [0123456789]——>[0-9] |
| 一位小写字母 | [abcdefg…xyz]——>[a-z] |
| 一位大写字母 | [ABCDEFG…XYZ]——>[A-Z] |
| 一位字母 | [A-Za-z] |
| 一位字母或数字 | [0-9A-Za-z] |
| 一位汉字 | [\u4e00-\u9fa5] |

【练习】车牌号: 1位汉字 1位大写字母 · 五位大写字母或数字

[\u4e00-\u9fa5][A-Z]·[A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9][A-Z0-9]

#### 1.2.2 预定义字符集

对常用字符集的一种简写，包括：

\d 一位数字 ——>[0-9]

\w 一位字母或数字或\_ ——>[a-zA-Z0-9\_]

\s 一位空字符——>空格、tab

. 通配符——>匹配除换行字符外的任何单个字符

**!注意：**如果规则完全匹配时，才能使用预定义字符集；如果规则不完全匹配，则退一步用自定义[字符集];预定义字符集不用加括号

问题：一个字符集只能规定一位字符的变化，即无法灵活定义出现的次数

#### 1.2.3 量词

定义：专门规定一位字符集出现次数的规则

使用：必须放在一个字符集之后，修饰相邻的前一个字符集，只能规定一位字符集出现的次数，有以下两种使用方法：

* 有明确数量边界

{n,m} 至少n次，最多m次

{n,} n次以上，多了不限

{n} 必须n次

【练习】车牌号：1位汉字 1位大写字母 · 五位大写字母或数字

[\u4e00-\u9fa5][A-Z]·[A-Z0-9]{5}

* 没有明确数量边界

\* 可有可无，多了不限

? 可有可无，最多一次

+ 至少一次，多了不限

#### 1.2.4 选择和分组

选择：本质上就是“或”

使用：在多个规则之间选其一匹配即可，具体用法如下：

规则1|规则2

分组：将多个字符集分为一组

量词默认仅修饰相邻的前一个字符集，故使用上有缺点，于是出现了分组（将规则应用到多个字符上）。

使用：(多个规则)

**!注意：**\是转义字符

【练习】身份证号：15位数字 2位数字 1位数字或X或x；后三位整体可有可无，最多一次

/\d{15}(\d{2}[0-9Xx])?/

【练习】手机号：+86或0086（可有可无最多一次） 空格（可有可无多了不限） 1 [3-8] 9位数字

/(\+86|0086)?\s\*1[3-8]\d{9}/

【作业】微信：weixin wei xin w x

/(微|w(ei)?)\s\*(信|x(in)?)/i

*var* msg = "咱们微信联系!";

*var* arr = msg.match(/(微|w(ei)?)\s\*(信|x(in)?)/i);

if(arr!=null){

*console*.log(arr);

*console*.error(`在位置：${arr["index"]}上发现敏感词：${arr["0"]}，关小黑屋！`);

}else{

*console*.log("成亮说：" + msg);

}

【作业】百度上找一个简单的电子邮件格式，用正则表达式写出来

#### 1.2.5 匹配特殊位置

字符串开头 ^ 比如，开头的空字符：^\s+

字符串结尾 $ 比如，结尾的空字符：\s+$

单词边界 \b 比如，选择一个单词no：\bno\b

只要仅匹配开头或结尾的关键词时，使用前两个；同时匹配开头或结尾的空字符：^\s+|\s+$

### 1.3 正则表达式API

#### 1.3.1 查找

##### 1.3.1.1 查找一个固定的关键词出现的位置

var i=str.indexOf("关键词"[,fromi])

var i=str.lastIndexOf("关键词")

【缺点】不支持正则，只能查找一个固定的关键词

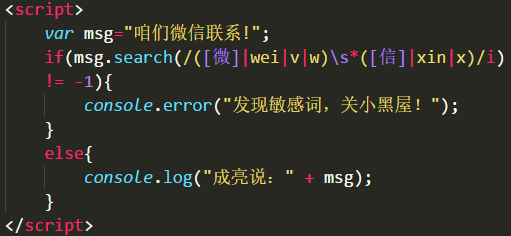
##### 1.3.1.2 用正则查找一个关键词出现的位置

var i=str.search(/正则/) /xxx/是正则表达式的标识

返回关键词的位置i,如果找不到，返回-1

【缺点】默认正则表达式是区分大小的

在第二个/后加后缀i ignore忽略以解决其区分大小写的问题



【缺点】只能返回位置，不能返回关键词内容

##### 1.3.1.3 用正则查找关键词的内容

* 只查找一个关键词的内容和位置

var arr=str.match(/正则/i)

返回值: 返回一个数组: [ 0: "关键词", index: i ]，找不到返回null

问题：只能找第一个关键词

* 查找所有关键词的内容:

var arr=str.match(/正则/ig) g global 全部

返回值: [ "关键词1","关键词2",...]，找不到返回null

arr.length 代表关键词个数！

*var* msg1 = "老师：请用小红、我的、朋友造句。小亮：小红是我的朋友。小明：朋友，小红是我的";

*var* arr1 = msg1.match(/小[\u4e00-\u9fa5]/ig);

*console*.log(arr1);

**!注意：**先判断不是null，再使用。

【缺点】此方式只能查找出关键词内容，不包含其对应下标值。

##### 1.3.1.4 查找每个关键词的内容和位置:

RegExp.exec(str)

#### 1.3.2 替换

##### 1.3.2.1 简单替换

将所有符合正则表达式的关键词都替换为一个统一的新值

str=str.replace(/正则/ig,"替换值");

*var* str="明明喜欢我，却不告诉我。别理我，我想静静。静静是谁？你先告诉我明明是谁！";

*var* arr = str.match(/明[\u4e00-\u9fa5]/g);

str = str.replace(/明[\u4e00-\u9fa5]/g,"\*\*");

*console*.log(str);

*console*.log(`共替换${arr!=null?arr.length:0}处`);

##### 1.3.2.2 高级替换

根据不同的关键词，选择替换不同的新值

str=str.replace(/正则/ig,function(kw){

//function(keyword)每找到一个关键词，就自动调用一次

//每次调用时，自动传入本次找到的关键词给keyword

//根据kword不同，返回不同的新值

return 新值;

});

##### 【**面试题**】将每个单词的首字母改为大写

// 将每个单词的首字母改为大写

*var* str1 = "you can you up!";

// 找出首字母

// var arr1 = str1.match(/\b[a-z]/ig);

// console.log(arr1);

*var* arr1 = str1.replace(/\b[a-z]/g,*function*(*kword*){

return kword.toUpperCase();

})

*console*.log(arr1);

#### 1.3.3 删除

其实就是替换为空字符串trim()

*var* str = " hello world ";

//删除str开头的空字符

*function* ltrim(*str*){

return str.replace(/^\s\*/,"");

}

//删除str结尾的空字符

*function* rtrim(*str*){

return str.replace(/\s\*$/,"");

}

// 删除str开头和结尾的空字符

*function* trim(*str*){

return str.replace(/^\s\*|\s\*$/g,"");

}

*console*.log(

`|${ltrim(str)}|`, // |hello world |

`|${rtrim(str)}|`, // | hello world|

`|${trim(str)}|` // |hello world|

)

#### 1.3.4 切割

##### 1.3.4.1 简单切割

分隔符是固定的

var arr=str.split("分隔符")

<script>

*var* email = "zhangdong@tedu.cn";

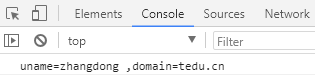
*var* arr = email.split("@");

*var* uname = arr[0];

*var* domain = arr[1];

*console*.log("uname="+uname+" ,domain="+domain);

</script>

结果如右图：

固定用法: 将字符串打散为字符数组

var chars=str.split("")

##### 1.3.4.2 高级切割

分隔符是变化的

var arr=str.split(/正则/i)

<script>

*var* html = `

<li>82</li>

<li>91</li>

<li>73</li>

<li>66</li>`

*var* html = html.replace(/^\s\*<li>|<\/li>\s\*$/g,"");

*console*.log(html);

*var* scores = html.split(/<\/li>\s\*<li>/);

*console*.log(scores);

scores.sort(*function*(*a*,*b*){return a-b;});

*console*.log(scores);

</script>

## 二、RegExp对象

### 2.1 RegExp对象概述

专门封装一条正则表达式，并提供用正则表达式执行验证和查找的API

今后只要在程序中保存和使用正则表达式，都用正则表达式对象。

### 2.2 创建RegExp对象

#### 2.2.1 用 / 创建

正则表达式一成不变的情况下，采用这种直接量的方式创建：

var reg=/正则表达式/ig

#### 2.2.2 用new RegExp创建

如果正则表达式不是固定的，需要动态生成：

var reg=new RegExp("正则表达式","ig")

实例：

<script>

//服务器端

//["我去","我靠","卧槽"]

//客户端

*var* msg="我靠，你家真大！";

//用ajax获取服务器端数据

*var* kwords=["我去","我靠","卧槽"];

//console.log(kwords.join("|"));

*var* reg = new RegExp(kwords.join("|"),"ig");

if(msg.search(reg)!=-1){

*console*.error("发现敏感词，关小黑屋！");

}else{

*console*.log("成亮说："+msg);

}

</script>

### 2.3 RegExp对象的API

#### 2.3.1 字符串格式验证

验证一个字符串的格式是否符合正则表达式要求

var bool=reg.test(str);

问题：默认在str中只要能找到符合reg要求的子内容，就算通过！

解决：凡是验证，必须前加^，后加$。

#### 2.3.2 查找关键词

即查找每个关键词的内容，又查找每个关键词的位置。

var arr=reg.exec(str);

在str中查找下一个（exec()每次可自动从下一个开始找！）符合reg要求的关键词的内容和位置；只返回本次找到的一个关键词的内容和位置，即arr:[ 0: "关键词", index: i ]；如果找不到，返回null

【原理】reg对象中有一个lastIndex属性，表示下次开始查找的位置。exec每执行一次，就自动修改reg.lastIndex为本次的index+关键词的.length。

【应用】可用循环反复调用exec，不断找下一个！

【强调】reg必须加g，才能先后找出所有符合正则表达式的值！

*var* str = "明明喜欢我，却不告诉我。别理我，我想静静。静静是谁？你先告诉我明明是谁？";

*var* reg = /明明|静静/g;

//.lastIndex

do{

*var* arr = reg.exec(str);

if(!arr){

break;

}else{

// console.log(arr);

*console*.log(`在位置${arr["index"]}，找到关键词${arr[0]},下一次从${reg.lastIndex}开始查找`)

}

}while(true);

## 三、Function

### 3.1 Function概述

什么是：保存一段可重用的代码段的程序结构

何时：只要一段代码可能被反复使用时

### 3.2 创建Function

#### 3.2.1 声明创建

function 函数名(形参,...){

函数体;

return 返回值

}

【TIPS】函数本身缺少必须的值需定义参数；调用者需要获得执行结果时需定义返回值

**!注意：**由于函数是对象，因此函数名实际上也是一个指向函数对象的指针

问题：[声明提前——广受诟病的缺陷！](#_2.2.3_变量及函数的声明提升)

声明提前: 在程序开始执行前，引擎会将var声明的变量和function声明的函数（函数体也会提升并运行），提前到当前作用域的顶部，集中创建，赋值留在原地。

//EX4:

*function* fun(){*console*.log(1);}

fun(); //2

*function* fun(){*console*.log(2);}

fun(); //2

解决: 改为赋值方式创建函数

#### 3.2.2 赋值创建

var 函数名=function (...){ ... }

#### 3.2.3 new创建

var 函数名=new Function("形参1","形参2",...,"函数体")

### 3.3 Function重载

#### 3.3.1 重载概述

重载（overload），指相同函数名不同参数列表的多个函数，在调用时，可根据传入参数的不同，自动选择匹配的函数执行。

即当一件事，可根据给的参数不同，执行不同的方法时，使用重载。

作用：减少API的数量，减轻使用者的负担

问题：JS不支持重载语法，因为JS不允许多个同名函数同时存在，最后一个同名函数会覆盖之前所有的同名函数。

解决：借助[arguments对象](#_2.4.2.3_arguments对象)

#### 3.3.2 arguments对象

arguments对象不需要创建，可直接使用。它是每个函数内自带的，专门接收所有传入函数的参数值的类数组对象（形如数组的对象）。

【TIPS】类数组对象VS数组

相同点：都有 ①下标；②.length属性； ③可用for循环遍历

不同点：类型不同——API不通用

*function* pay(){

*console*.log(arguments);

*console*.log(typeof(arguments));

if(arguments.length == 0){

*console*.log("手机支付");

} else if(arguments.length == 1){

*console*.log("现金支付");

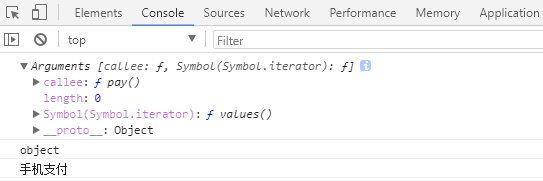
} else if(arguments.length == 2){

*console*.log("银行卡支付");

}

}

pay();

运行结果：

### 3.4 匿名函数

#### 3.4.1 匿名函数概述

创建函数时不指定函数名的函数

作用：①节约内存；②临时使用时划分临时作用域

#### 3.4.2 匿名函数的应用

##### 3.4.2.1 回调函数

回调函数是指，自己定义的但是交给别的函数去调用，比如：

* arr.sort(function(a,b){return a-b;});
* xhr.onreadystatechange=function(res,req){…}
* str.replace(/正则表达式/,function(){…});

##### 3.4.2.2 匿名函数自调用

匿名函数自调用即指，定义函数后，立即调用自己——为了避免全局污染（一个变量不再使用，但空间未被释放）。

匿名函数自调用会划分临时作用域，避免使用全局变量。

<script>

// 匿名函数自调用

(*function*(){

*var* start = new Date();

alert(`开始加载页面内容，at:${start}`);

})();

</script>

<h1>Welcome</h1>

<h3>Welcome to my page.</h3>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Rem aspernatur natus enim cupiditate magnam laudantium praesentium quae soluta, odit illum ea minus doloribus, quod possimus harum corporis, error vel cumque?</p>

<script>

(*function*(){

*var* end = new Date();

alert(`页面加载完成，at:${end}`);

})();

//测试：F12-->console:输入start回车，看start是否遗留在内存，再输入end回车，看end是否遗留在内存

</script>

【总结】今后禁止使用全局变量；所有js代码都应该用匿名函数自调包裹起来。

### 3.5 作用域和作用域链

#### 3.5.1 概述

为了防止内外变量之间互相干扰，出现了作用域的概念。

作用域（scope），一个变量的可用范围。从本质上来说，每个作用域都是一个保存变量的对象。作用域主要分为两种：

* 全局作用域：保存全局变量，从程序任何位置都可以访问的作用域。

> 全局变量特点：随处可用，可反复使用

* 函数作用域：保存局部变量，仅函数内部可访问的作用域。

> 局部变量特点：仅函数内部访问，不可反复使用（释放后不存在，故不能反复使用）。

**!注意：**js中没有块级作用域

*var* a = 10;

*function* fun(){ //函数作用域

/\*var \*/a = 100;

a++;

*console*.log(a); //101

}

fun();

*console*.log(a); //101

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

sum += i;

}

*console*.log(sum); //5050

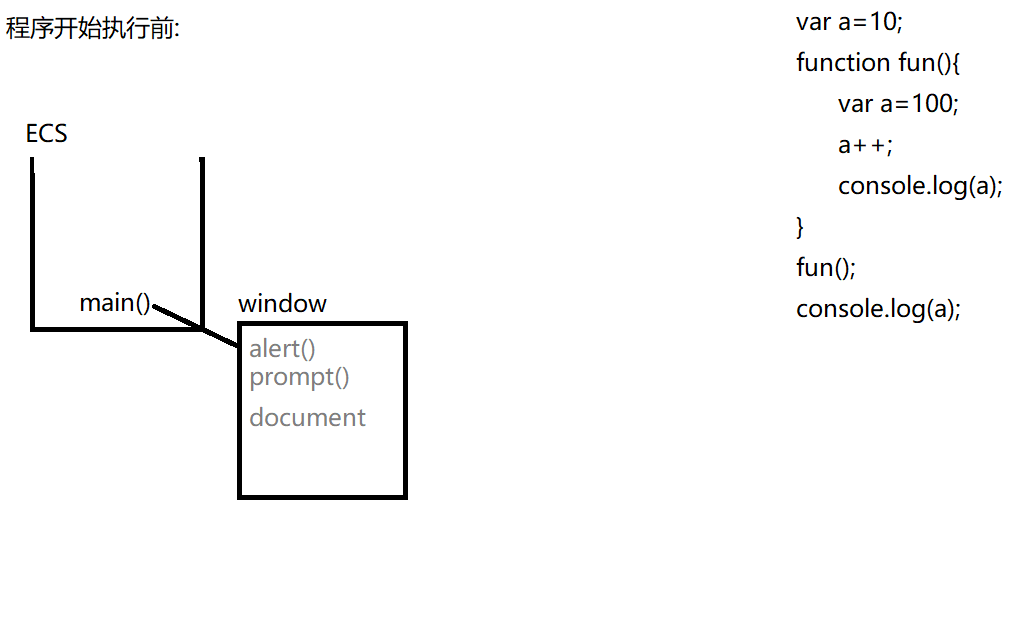
#### 3.5.2 函数的生命周期

* 程序开始执行前

- 创建执行环境栈数组：用于记录将来调用的函数

- 调用浏览器软件的主程序main()函数

- main()函数会创建并引用window对象，此时window对象已经包含了很多内置的函数和对象，如：alert() prompt() document



* 定义函数时

- 在window中添加函数名变量

- 在window外创建函数对象，保存函数定义

- 函数名变量引用函数对象

- 函数对象的scope属性指回自己诞生自的对象window



* 调用函数时

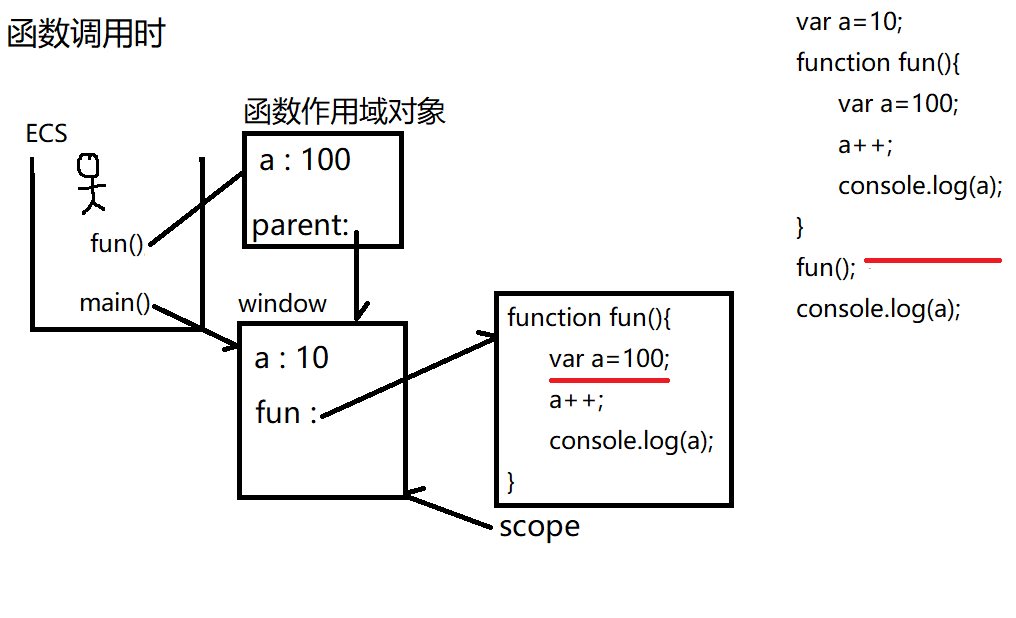
- 在执行环境栈数组中添加本次函数调用的记录

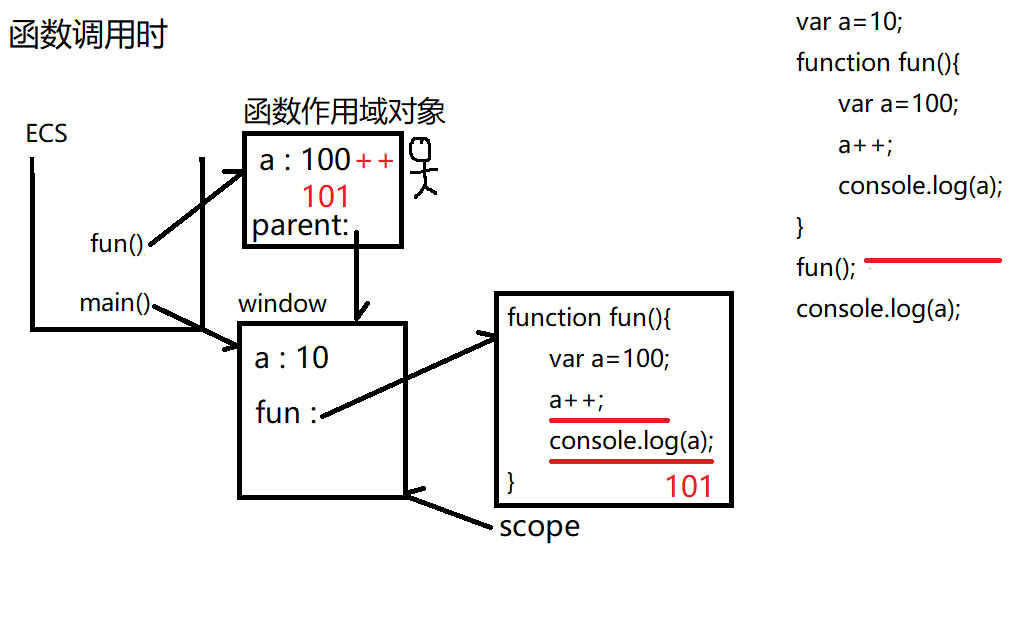
- 为本次函数调用创建临时作用域对象——函数作用域对象

- 在函数对象中添加函数中声明的所有局部变量

- 调用时变量使用顺序:

只要自己有，就用自己的；局部没有，才去全局找



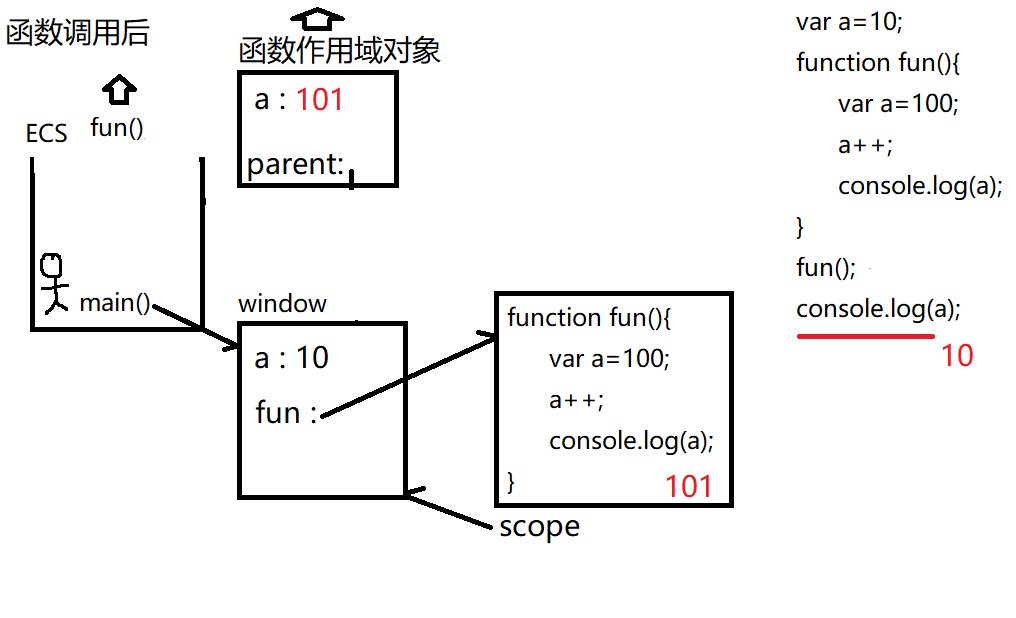


* 函数调用后

- 执行环境栈中本次函数调用的记录出栈

- 导致函数作用域被释放

- 导致局部变量一同释放！——局部变量不可重用



#### 3.5.3 作用域链

作用域链是指，由多级作用域逐级引用，形成的链式结构。

* 作用:

- 保存所有全局和局部变量

- 控制着变量的使用顺序: 先局部，后全局

### 3.6 闭包 ★

#### 3.6.1 闭包概述

闭包，指既重用变量，又保护变量不被污染的一种机制。

定义：一个函数调用了不属于自己的局部变量。

全局变量和局部变量都有不可兼得的优缺点:

> 全局变量: 优: 可重复使用

缺: 随处可用——极容易被污染

> 局部变量: 优: 仅函数内可用，不会被污染

缺: 不可重用

**!注意：**今后尽量少用或不主动使用全局变量

鉴于此，当想为一个函数永久绑定一个专属的可重用变量时，可采用闭包。

#### 3.6.2 创建闭包

创建闭包的步骤主要为以下三步：

> 用外层函数包裹要保护的变量和使用变量的内层函数

> 外层函数将内层函数返回到外部

> 使用者调用外层函数，获得内层函数对象，保存在变量中反复使用

// 1.用外层函数包裹要保护的变量和使用变量的内层函数

*function* outer(){ //外层函数

*var* i = 1; //受保护的变量

// 2.将使用受保护变量的内层函数返回给外层函数，以供调用

return *function*(){ //内层函数

*console*.log(i++);

}

}

// 3.调用外层函数，获得内层函数对象，保存在变量中反复使用

*var* getNum = outer();

getNum();

getNum();

i = 1; //window.i

getNum();

【TIPS】程序中的function意思是“创建函数”，这是一个“动作”。同理，“[]”也是“创建数组”之意。

【重点】闭包形成的原因：外层函数的作用域对象在调用后无法释放；但也因此闭包比普通函数多占用内存

##### 【笔试题】if([] == [])——> false

“[]”是创建数组的意思，此题意为创建两个数组，而“==” 比较的是两个数组的地址，当然不相等。

#### 3.6.3 闭包笔试题解决思路

画简图

* 找受保护的变量
* 找外层函数共向外抛出了几个内层函数对象——>3中情况

> return function(){…}

> 直接给全局变量赋值

> 将函数包裹在对象或数组中返回

*function* fun(){

*var* n = 999;

nAdd = *function*(){n++;}

return *function*(){

*console*.log(n);

}

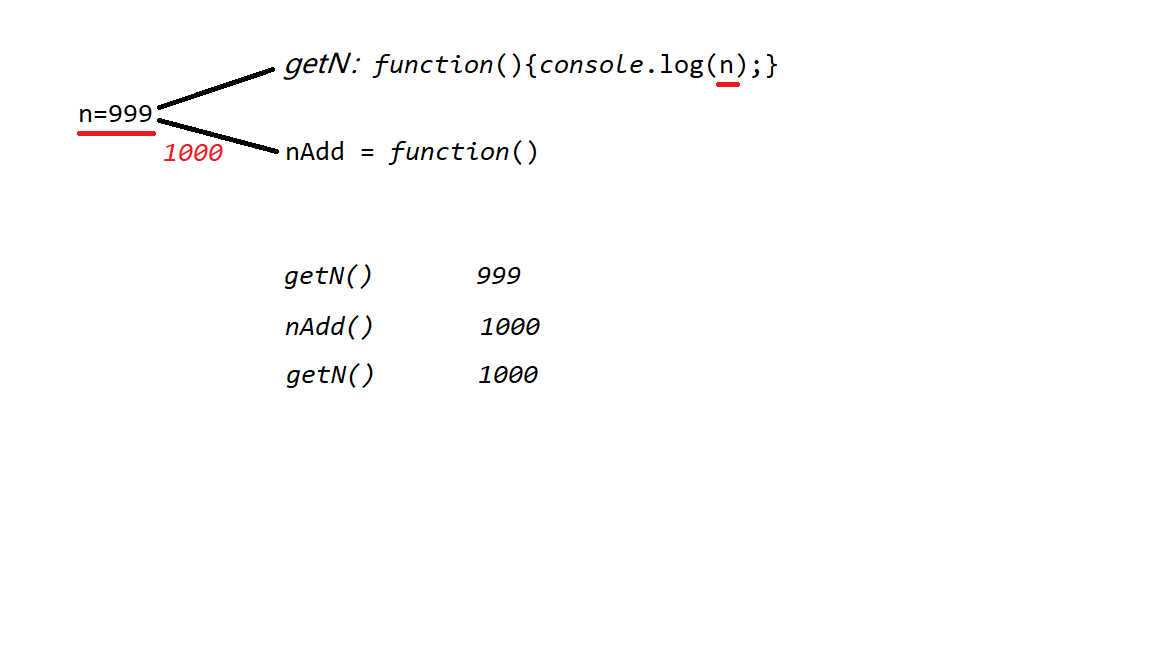
}

*var* getN = fun();

getN(); //999

nAdd(); //n++

getN(); //1000



## 四、面向对象OOP

### 4.1 概述

面向对象编程（Object Oriented Programming，OOP，面向对象程序设计）是一种计算机编程架构。

程序都是先用对象结构描述现实中一个具体事务，再按需使用事务的属性和功能。

作用：便于大量数据的管理和维护

**!注意：**今后所有程序都使用面向对象的方式实现。

### 4.2 面向对象三大特征

封装、继承、多态

#### 4.2.1 封装

* 定义：创建一个对象结构来集中保存一个事物的属性和功能
* 作用：便于大量数据的使用和维护
* 何时：只要使用面向对象方式编程，都要先创建对象。
* 如何：创建方式（3种）

##### 4.2.1.1 直接量创建

即用{}创建：

var obj={

属性名:值,

… :…,

方法名~~:function~~(){ //ES6:方法名(){…}

… …

}

}

* 对象是描述现实中一个事物的属性和功能的程序结构

事物的属性会成为对象中的属性

事物的方法会成为对象中的方法

* 访问对象成员：成员={属性+方法}

先用对象名找到对象，再用.找到对象中的成员。属性其实就是保存在对象中的变量，用法和变量完全一样

* 访问对象的属性：对象.属性名
* 访问对象的方法：对象.方法()

方法其实就是保存在对象中的功能函数，用法和函数完全一样

问题：对象自己的方法内要使用对象自己的属性

错误解法: 直接使用属性名

不加任何前缀的变量，默认只能在作用域链中查找，不能擅自进入对象中获得对象的成员。

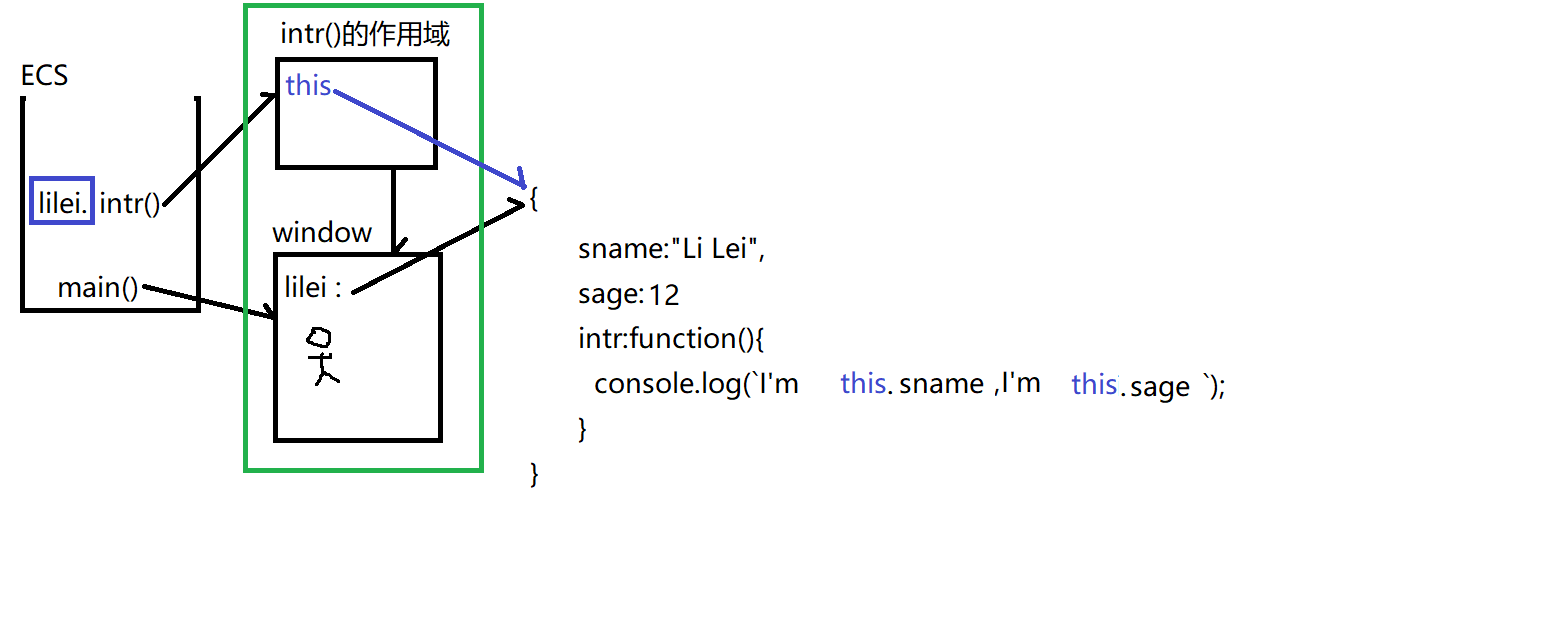
正确1:对象.属性名（紧耦合，不推荐）

正确2:this.属性名

this本质是保存在函数作用域中的一个关键词，是自动指向正在调用该函数的.前的对象。即.前是谁，this就自动指谁。

**!注意：**今后，只要对象自己的方法，要访问自己的成员，都必须加this.

下图是this关键词的指向图解：



【EX】用直接量创建对象，并使用clone()创建对象副本

*var* lilei = {

sname: "Li Lei",

sage: 11,

intr(){

*console*.log(`I'm ${this.sname},I'm ${this.sage}`);

}

}

*function* clone(*obj*){

*var* newObj = {};

for(*var* key in obj){

// key:"sname","sage","intr"

// console.log(key);

// 问题1：以下三种写法，哪种正确？

newObj[key] = obj[key]; //正确

// newObj["key"] = obj["key"]; //key——>undefined

// newObj.key = obj.key; //key——>undefined

}

return newObj;

}

// 问题2：clone()函数和直接赋值lilei给Obj2有什么区别？

*var* obj2 = clone(lilei);

*console*.log(obj2);

// var obj2 = lilei;

// console.log(obj2);

问题：只适合创建一个单独的对象，如果需要反复创建同一类型的多个对象时，代码会很繁琐。

解决：使用构造函数

##### 【笔试题】this关键词指向的对象

// 笔试题

*var* a=10;

*var* obj={

a:20,

fun:*function*(){

*var* a=10;

*console*.log(this.a);

}

}

*var* fun = obj.fun; //fun:function(){ var a=10; console.log(this.a);}

obj.fun(); //20 this->obj

fun(); //10 this->window

// 如果一个函数不用.调用，则this->window

上述代码改为下面的代码可以更直观的理解：

// 笔试题

*var* a=10;

*var* obj={

a:20,

fun:*function*(){

*var* a=10;

*console*.log(this.a);

}

}

*var* funs = obj.fun; //fun:function(){ var a=10; console.log(this.a);}

obj.fun(); //20 this->obj

funs(); //10 this->window

// 如果一个函数不用.调用，则this->window

##### 4.2.1.2 用new创建

创建步骤分为以下两步:

/\*1.先创建空对象\*/

var obj=new Object();

/\*2.再向空对象中强行添加新属性\*/

obj.属性名=值;

obj.方法名=function(){…}

第1步可简写：new可省略, ()也可省略，但必须保留其一

【原理】其实，js对象底层也是关联数组

* 即可用["属性名"]访问元素，有可简写为.属性名(.属性名<==>["属性名"])

何时用. : 如果属性名是写死的obj.sname

何时用[""] : 如果属性名是动态获得的obj[key](key——>"sname")

* 访问不存在的属性，不报错，而是返回undefined
* 强行给不存在的属性赋值，不报错，而是强行添加该属性
* 遍历对象每个属性也可以用for-in循环

for-in循环中的key值是字符串型

问题：以上两种方式，只适合创建一个单独的对象，如果需要反复创建同一类型的多个对象时，代码会很繁琐。

解决：使用构造函数

##### 4.2.1.3 用构造函数

构造函数是描述一类对象同一结构的函数。当需要反复创建多个相同结构的对象时，采用构造函数以便于代码重用。

/\*1.定义构造函数\*/

function 类型名(){

this.属性名 = 值;

this.… = …;

this.方法名 = function(){

…this.…

}

}

/\*2.调用构造函数\*/

var student = new Student();

第2步中的new共做了4件事：

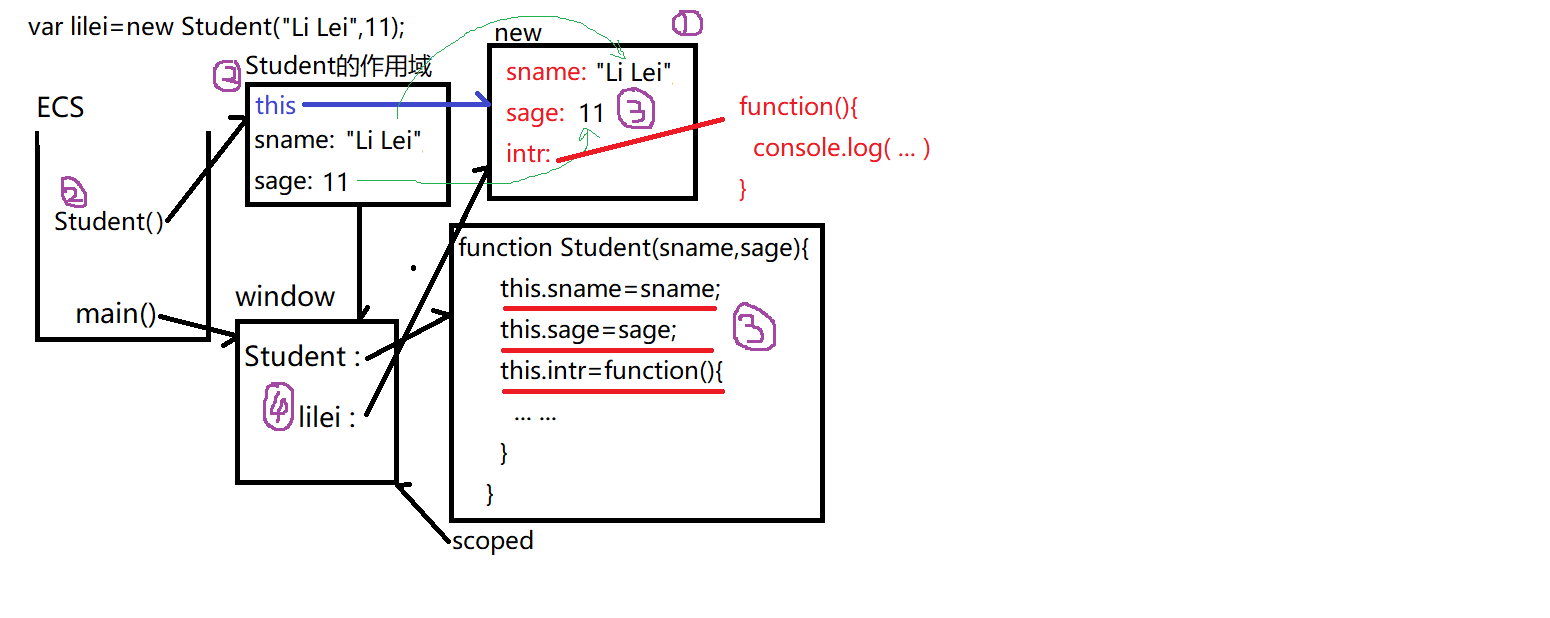
* 创建一个新的空对象
* [？](#_4.2.2.2_使用继承)
* 用新对象调用构造函数

new可将构造函数中的this自动指向新对象

构造函数通过强行赋值新属性，构造函数通过强行赋值新属性的方式，为新对象添加新属性；

* 返回新对象的地址给变量。

过程图如下：



问题：构造函数中的方法定义，每new一次，就会反复创建副本——浪费内存！也不便于维护

解决: 1. 构造函数中不应再包含方法定义！

2. 方法定义放哪儿？

#### 4.2.2 继承

##### 4.2.2.1 继承概述

定义：父对象的成员，子对象无需重复创建，就可直接使用。如果多个子对象要使用相同的方法时，使用继承可以使代码重用，进而节约内存。

js中继承都是通过原型对象实现的：

原型对象定义：集中存储所有子对象共有成员的父对象

如何获得原型对象：买一赠一，即每创建一个构造函数，都自动附赠一个原型对象。

##### 4.2.2.2 使用继承

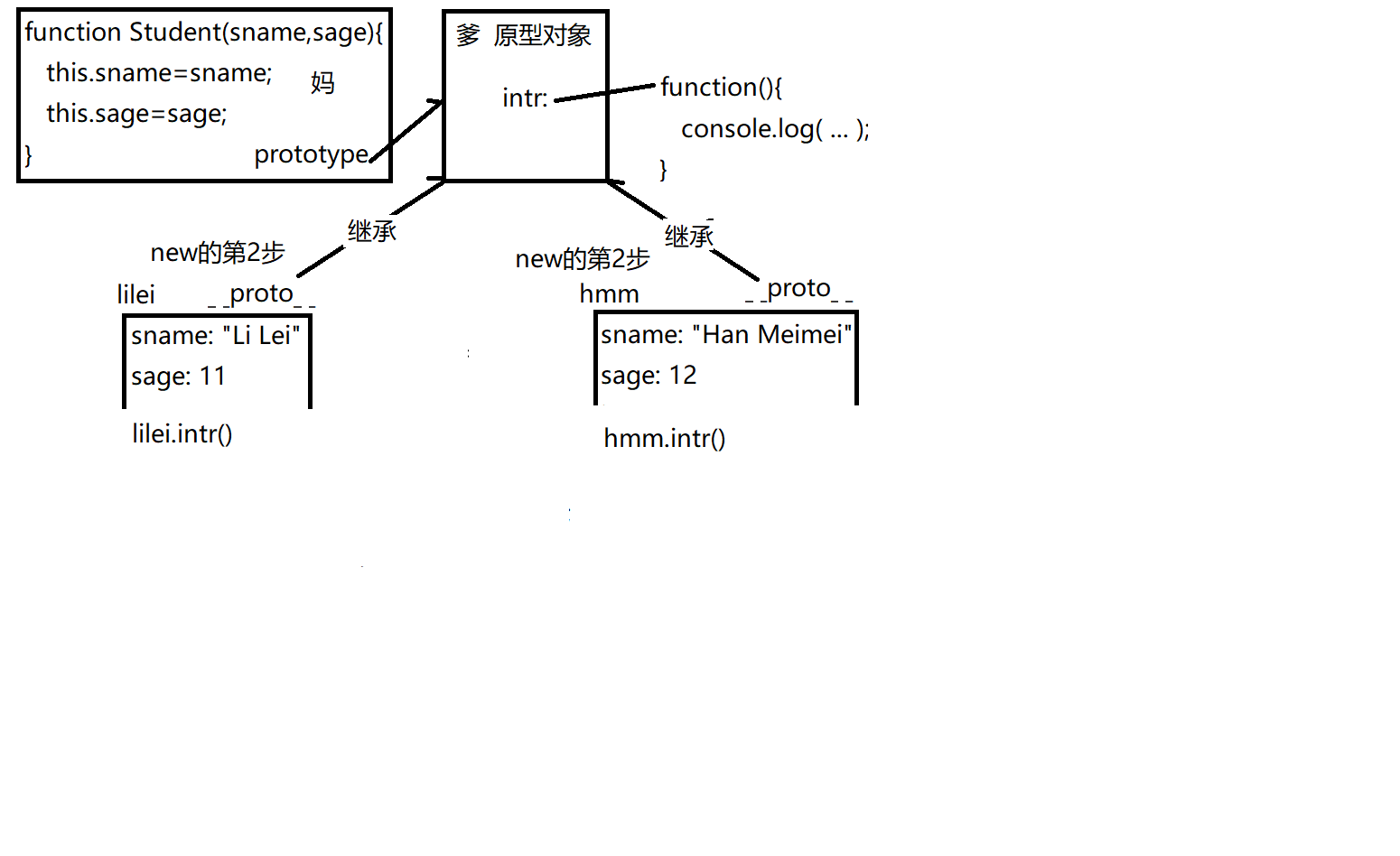
即new的第2步：

* 让新子对象继承构造函数的原型对象，即：

子对象.\_\_proto\_\_ = 构造函数.prototype

* 向原型对象中添加共有成员：

构造函数.prototype.方法名=function(){…}



【TIPS】自有属性和共有属性

自有属性：保存在子对象本地，仅当前对象自己所有的属性

共有属性：保存在原型对象中，归所有子对象共有的属性

区别：

获取属性值：二者没有差别；子对象.属性名

修改属性值：可以修改自有属性，但无法修改共有属性（如果强行用子对象修改共有属性，会给子对象添加一个同名的自有属性，从此子对象与共有属性再无关系）

*Student*.prototype.className="初一2班";

// 通过子对象输出其共有属性：一致*console*.log(lilei.className,hmm.className);//"初一2班" "初一2班"

// 用原型对象修改共有属性

*Student*.prototype.className="初二2班";

// 再用子对象输出其共有属性：一致

*console*.log(lilei.className,hmm.className);//"初二2班" "初二2班"

// 某个子对象想修改共有属性

lilei.className = "初三2班";

// 再用子对象输出其共有属性

// 结果：不一致，仅为lilei添加自有属性className

*console*.log(lilei.className,hmm.className);//"初三2班" "初二2班"

// 再修改共有属性

*Student*.prototype.className = "高一2班";

// lilei因为有自有属性className，所以与共有className再无关系*console*.log(lilei.className,hmm.className);//"初三2班" "高一2班"

##### 4.2.2.3 内置类型/对象

内置类型/对象定义: ES标准规定的，浏览器/平台已经实现的内置类型/对象，包括以下11种: Number、String、Boolean、Array、Date、RegExp、Error、Function、Object、Math{}、global{} (在浏览器中被window代替) (Math{}、global{}不能使用new)



凡是能new的都是一种类型

每种内置类型，同样也包含2部分:

* 构造函数：专门用于创建该类型的子对象
* 原型对象：专门保存该类型所有子对象的共有成员

将共有方法添加到内置类型的原型对象中

this->将来调用该方法的.前的子对象

为类型添加一种新函数：

*Array*.prototype.sum = *function*(){

*console*.log("调用自定义的sum函数");

// var arr = this;

*var* sum = 0;

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

sum += this[i];

}

return sum;

}

*var* arr1 = [1,2,3];

*var* arr2 = [1,2,3,4,5];

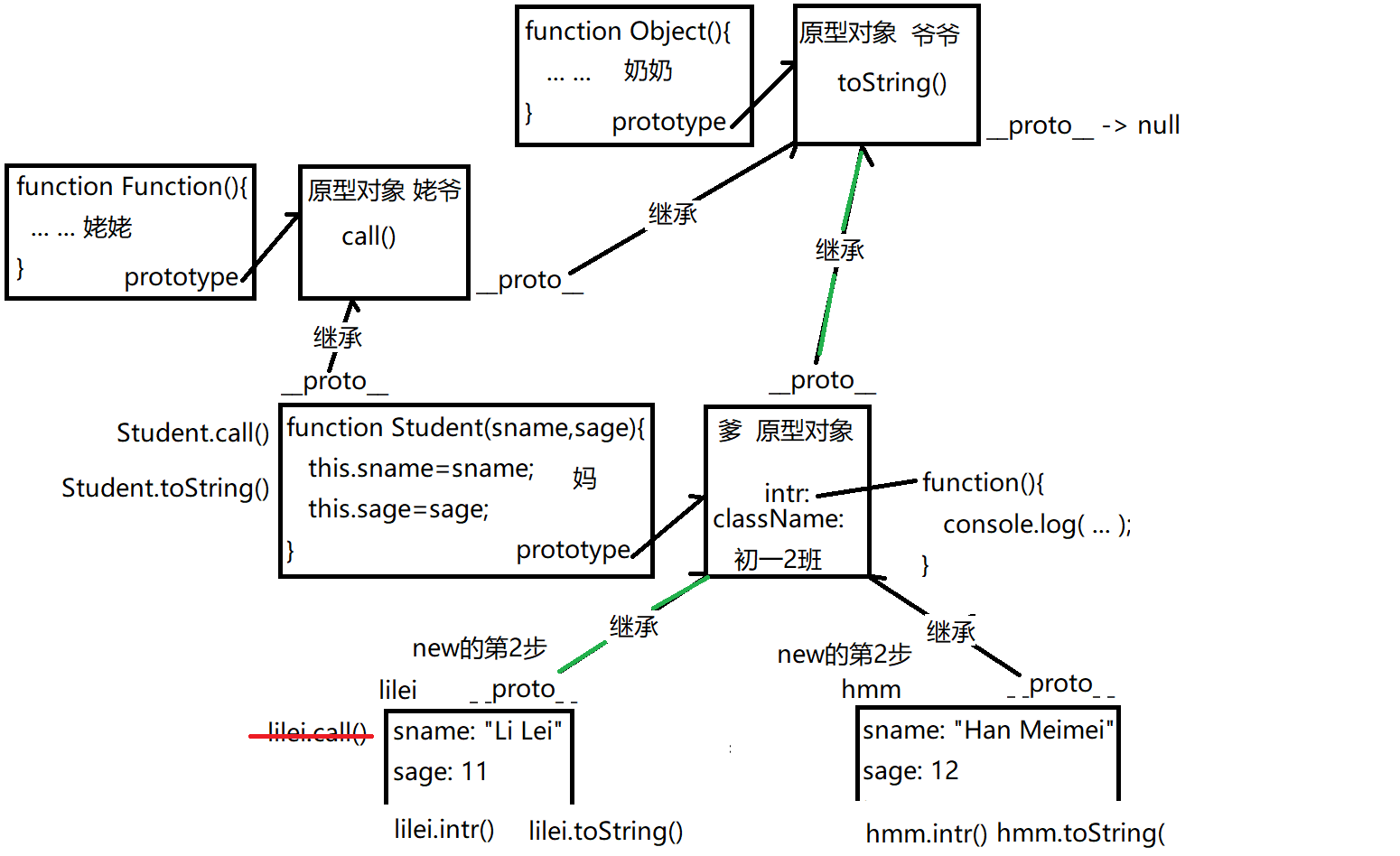
*console*.log(arr1.sum(),arr2.sum());

##### 4.2.2.4 原型链

定义：由多级父对象逐级继承，形成的链式结构

作用：保存着所有对象的成员；控制着成员的使用顺序：先自有，再共有

下图是一个完整的原型链结构：



##### 【笔试题】原型链：如何判断一个对象是不是数组！有几种方式。 ★

导入：如下是各数据类型使用typeof操作符的结果

*var* n=5,s="hi",b=true,nu=null,un;

*var* obj1={"x":1,"y":2},obj2=[1,2,3], obj3=new Date(),

obj4=*function*(){return 1};

*console*.log(

typeof n, //number

typeof s, //string

typeof b, //boolean

typeof nu, //object

typeof un, //undefined

typeof obj1, //object

typeof obj2, //object

typeof obj3, //object

typeof obj4 //function

);

问题：typeof仅能区分原始类型和function，无法进一步区分每种对象的类型。

解决：解决方法有如下四种

1.判断原型对象：

obj.\_\_proto\_\_ == Array.prototype;

Array.prototype.isPrototypeOf(obj);

2.判断构造函数：

使用instanceof运算符，instanceof 运算符与 typeof 运算符相似，用于识别正在处理的对象的类型。与 typeof 方法不同的是，instanceof 方法要求开发者明确地确认对象为某特定类型。

通常来讲，使用 instanceof 就是判断一个实例是否属于某种类型；另外，更重的一点是 instanceof 可以在继承关系中用来判断一个实例是否属于它的父类型。

【TIPS】对象创建后更改继承的原型对象，再用上述两种方法验证其是否为数组，输出true。

obj.constructor == Array;

obj instanceof Array;

3.判断隐藏的class属性：

Object.prototype.toString.call(obj1) == "[Object Array]";

4.Array.isArray(obj1):

直接返回boolean值，原理同第三种。

<script>

// 方法一

*var* obj1={"x":1,"y":2},

obj2=[1,2,3],

obj3=new Date(),

obj4={"a":3,"b":4};

obj4.\_\_proto\_\_ = *Array*.prototype;

*console*.log(

*Array*.prototype.isPrototypeOf(obj1),

*Array*.prototype.isPrototypeOf(obj2),

*Array*.prototype.isPrototypeOf(obj3),

*Array*.prototype.isPrototypeOf(obj4)

);

// 方法二

*console*.log(

obj1 instanceof *Array*,

obj2 instanceof *Array*,

obj3 instanceof *Array*,

obj4 instanceof *Array*

);

// 方法三

*console*.log(

*Object*.prototype.toString.call(obj1) == "[object Array]",

*Object*.prototype.toString.call(obj2) == "[object Array]",

*Object*.prototype.toString.call(obj3) == "[object Array]",

*Object*.prototype.toString.call(obj4) == "[object Array]"

);

// 方法四

*console*.log(

*Array*.isArray(obj1),

*Array*.isArray(obj2),

*Array*.isArray(obj3),

*Array*.isArray(obj4)

);

</script>

##### 4.2.2.5 自定义继承

自定义继承主要有3种方式：

* 仅修改一个对象的父对象：

子对象.\_\_proto\_\_=father;

Object.setPrototypeOf(子对象，father)

* 修改多个对象的父对象：

构造函数：protype=father;

时机：必须在创建子对象之前就改变

* 两种类型间的继承：

问题：两种类型间包含部分相同的树形结构和方法定义。

解决：定义抽象父类型，主要分为2步

1. 定义抽象父类型:

父类型的构造函数中保存相同的属性结构

父类型的原型对象中保存相同的方法定义

2. 让子类型继承父类型:

让子类型原型对象继承父类型原型对象

在子类型构造函数中调用父类型构造函数

##### 【笔试题】坦克大战飞机

构建如下图所示的程序代码：



代码如下：

*function* Enemy(*fname*,*speed*){

this.fname = fname;

this.speed = speed;

}

*Enemy*.prototype.fly=*function*(){

*console*.log(`${this.fname}以时速${this.speed}飞行`);

}

*function* Plane(*fname*,*speed*,*score*){

// 用call()更换调用Enemy构造函数的对象

// window.Enemy --> this.Enemy

Enemy(fname,speed);

this.score = score;

}

*Plane*.prototype.getScore=*function*(){

*console*.log(`击落${this.fname}得${this.score}分`);

}

*Object*.setPrototypeOf(*Plane*.prototype,*Enemy*.prototype);

*var* f16 = new Plane("f16",1000,5);

f16.fly();

f16.getScore();

*console*.log(f16);

问题：在子类型构造函数中直接调用父类型构造函数，无法将父类型的属性加入到新对象中。

原因：子类型构造函数中直接调用父类型构造函数，导致父类型构造函数中的this默认指向window，导致父类型的属性都泄露到全局。

解决：当一个函数中的this不是想要的时候，可用call()，将函数中不想要的this换成想要的对象。

函数名.call(替换this对象，参数值列表…)

* 可立刻调用函数
* 将函数中的this，临时替换为想要的"对象"
* 将参数值传给函数

故可将上述笔试题红框中的代码改为：

Enemy.call(this,fname,speed);

#### 4.2.3 多态

* 定义：同一个函数，不同情况下表现出不同的状态
* 包括两种形式：

- [重载(overload)](#_3.3_Function重载)

- 重写(override):如果子对象觉得从父对象继承来的成员不好用，可在子对象本地定义同名成员，来覆盖父对象的成员。

### 4.3 静态方法/实例方法

有时在调用函数时，无法确定调用函数的实例是什么。当希望不确定实例对象也能调用方法时，可将其定义为静态方法。

静态方法：不需要创建子对象实例，用构造函数就可直接调用的方法

静态方法都要定义在构造函数对象上，而不是添加在原型对象中

实例方法：必须创建该类型的子对象，用子对象才能调用的方法

##### 【笔试题】何时使用静态方法，何时使用实例方法

* + 如果规定必须某个类型的子对象才能调用的方法，就用实例方法

实例方法保存在该类型的原型对象中

* + 如果不确定调用该方法的数据类型是，就用静态方法

静态方法保存在构造函数中

## 五、ES5新特性

### 5.1 严格模式

#### 5.1.1 概述

定义：比普通js运行机制要求更严格的模式

#### 5.1.2 严格模式解决的问题

作用：解决js中部分广受诟病的缺陷。

|  |  |
| --- | --- |
| **js中的缺陷** | **严格模式下** |
| 1.未声明的变量也可赋值，并自动创建在全局——全局污染  2.静默失败：执行不成功，也不报错  3.普通函数调用中的this默认指向window  4.允许使用递归 | 1.禁止给未声明的变量赋值  2.将静默失败升级为错误  3.普通函数调用中的this不再指向window，而是undefined  4.强烈建议不使用递归 |

#### 5.1.3 使用严格模式

使用：两种使用方式

* 整个js文件/<script>标签内启用严格模式:

在顶部添加: "use strict";

* 仅在函数内启用严格模式:

在函数内顶部加: "use strict";

**!注意：**今后所有js程序都应该在严格模式下运行

### 5.2 保护对象

#### 5.2.2 保护对象概述

定义：让使用者按规则有限度的使用对象

原因：js中的对象，默认没有任何限制

何时：如果想控制使用者对对象的使用时

#### 5.2.2 保护对象的属性

##### 5.2.2.1 对象属性的分类

* 命名属性：凡是用.可直接访问到的属性

分为数据属性和访问器属性

> 数据属性：实际存储属性值的属性

> 访问器属性：不实际存储属性值，仅提供对其他数据属性的保护

* 内部属性：不能用.直接访问到的属性，比如：class

##### 5.2.2.1 保护数据属性值

* 数值属性的特性：ES5将每个简单的属性变成了一个缩微的对象，每个属性对象有包含4个特性：

{

value:属性值, //实际存储属性值

writable:true/false, //控制是否可修改

enumerable:true/false, //控制是否可被隐藏/for in遍历

//强调: 仅控制for in遍历，不能控制.直接访问

configurable:true/false //控制是否可删除该属性

//控制是否可以修改前两个特性；

//一旦改为false，不可逆！

//今后修改前两个属性，都要带上configurable

}

* 查看一个属性的四个特性：

var id\_attrs=Object.getOwnPropertyDescriptor(eric,"id");

console.log(id\_attrs);

* 保护命名属性中的数据属性值主要有两种方式：

**>1** 设置属性的四种特性保护数据属性的值；

修改属性的四个特性：

Object.defineProperty(对象,"属性名",{

writable:false

});

问题：一次只能修改一个属性的四个特性

解决：使用Object.defineProperties()

Object.defineProperties(对象, {

属性1:{

writable:false

},

属性2:{

writable:false,

configurable:false

}

…

});

问题：无法使用自定义规则保护属性

解决：用访问器属性保护数据属性

**>2** 设置访问属性作为“保镖”保护数据属性的值。

只要用自定义规则保护属性时，都需要用到访问属性，具体步骤如下：

①先定义一个隐藏的要保护的数据属性(enumerable:false;)

②定义访问器属性，保护数据属性：

【强调】访问器属性只能用defineProperty()/defineProperties()添加，不能用直接量添加，即：

Object.defineProperties(对象, {

其他属性…,

访问器属性:{

get(){

//…

},

set(){

//…

},

enumerable:false,

configurable:false

}

});

#### 5.2.3 保护对象的结构

定义：禁止擅自删除或添加对象的属性

共分为三个层次：禁止添加新属性、密封和冻结

##### 5.2.3.1 防扩展

禁止添加新属性

语法：Object.preventExtensions(obj)

原理：每个对象都有一个内部属性：extensible:true;

preventExtensions(obj)设置obj的extensible为false

##### 5.2.3.2 密封

在禁止添加新属性的基础上，进一步禁止删除现有属性

语法：Object.seal(obj);

原理：设置obj的extensible为false且自动设置obj的每个属性的configurable为false。

##### 5.2.3.3 冻结

在密封的基础上禁止修改属性值

语法：Object.freeze(obj)

原理：设置obj的extensible为false且自动设置obj的每个属性的configurable和writable均为false。

### 5.3 Object.create()

定义：基于一个现有父对象，直接创建子对象，并为子对象扩展自有属性

何时: 如果没有构造函数，也想创建子对象时

使用步骤:

> 创建一个新对象

> 让新对象继承父对象

> 为新对象扩展自有属性

var child = Object.create(父对象,{

属性1:{

writable:false

},

属性2:{

writable:false,

configurable:false

})

创建一个新对象child，并让child继承father，还可给child添加新自有属性。

### 5.4 替换this的三种方法

如果一个函数执行时，其中的this不是想要的，可用以下三种方法进行替换：

#### 5.4.1 .call()

作用：立即调用一次函数，临时替换this

语法：函数.call(替换this的对象, 实参1, ... ... )

特点: 临时绑定，即时调用，只调用一次

问题：只能传入单个参数

#### 5.4.2 .apply()

作用：替换this的对象为单个元素1,元素2,...，顺便打散数组参数为单个值，

何时: 如果函数需要传入多个参数值，但参数是放在数组中给定的。

语法: 函数.apply(替换this的对象,数组)

#### 5.4.3 .bind()

作用：基于原函数，为某个对象，创建一个专属的新函数，并永久绑定this为指定对象

> 创建一个和原函数一模一样的新函数

> 永久替换新函数中的this为指定对象

语法：var 新函数=函数.bind(替换this的对象)

固定场景: 专门用于绑定回调函数中的this

##### 【笔试题】回调函数中this指向的问题

*var* a = 10;

*var* obj = {

a:20,

fun:*function*(){

*var* a=30;

*console*.log(this.a);

}

}

// obj.fun=function(){var a=30;console.log(this.a);}

setTimeout(obj.fun,100); //10

setTimeout(obj.fun.bind(obj),100); //20

【总结】替换this实例

*function* calc(*base*,*bonus1*,*bonus2*){

*console*.log(`${this.ename}的总工资是：${base+bonus1+bonus2}`);

}

*var* lilei = {ename:"Li Lei"};

*var* hmm = {ename:"Han Meimei"};

// call()

calc.call(lilei,10000,1000,2000);

// apply()

*var* bonus = [3000,4000,5000];

calc.apply(hmm,bonus);

// bind()

// calc\_l = function calc(base,bonus1,bonus2){console.log(`${this.ename}的总工资是：${base+bonus1+bonus2}`);}

*var* calc\_l = calc.bind(lilei,10000);

//this-->lilei,base-->10000

calc\_l(1000,3000);

calc\_l.call(hmm,10000,3000,5000); //Li Lei的总工资是：23000

*var* nums=[3,6,2,1];

*console*.log(Math.max.apply(null,bonus));

### 5.5 数组API

#### 5.5.1 查找元素在数组中的位置

var i=arr.indexOf(elem[, fromi])

//返回值: elem的位置i, 如果找不到，返回-1

兼容：为了兼容有些不支持ES6的浏览器，可在Array类型的原型对象中添加自定义的indexOf()方法，代码如下：

// 兼容不支持es6的浏览器——添加自定义的indexOf()方法

*Array*.prototype.indexOf = *function*(*elem*,*starti*){

// console.log("调用自定义的indexof");

if(starti === undefined) starti = 0;

// this-->将来调用indexOf()方法的.前的对象

for(*var* i = starti;i < this.length;i++){

if(this[i] === elem){

return i;

}

}

return -1;

}

#### 5.5.2 判断数组中的元素是否符合要求

判断数组中是否符合要求，分两种：

* every() 判断数组中是否所有元素都符合要求

arr.every(function(elem,i,arr){

return 判断条件;

});

其中：

> function会在每个元素上自动执行一次

> elem：自动获得当前元素值

> i：自动获得当前位置

> arr：自动获得当前正在遍历的数组对象

**!注意：**回调函数function(elem,i,arr)前没有用.调用，所以this默认指向window,故不能用this代替arr->当前数组。

* some() 判断数组中否包含符合要求的元素

arr.some(function(elem,i,arr){

return 判断条件;

});

#### 5.5.3 遍历数组

对每个元素执行相同的操作：

forEach() 对原数组中每个元素执行相同的操作，执行后，再存入原数组中

arr.forEach(function(elem,i,arr){

//对数组元素进行操作

arr[i] = 新值;

});

map() 取出原数组中每个元素，执行相同操作后，放入新数组返回，原数组不变。

var newA = arr.map(function(elem,i,arr){

//对数组元素进行操作

});

兼容：与上述同理，在今后的编程中也应考虑浏览器兼容问题，应在其原型对象中添加自定义的对应方法。具体代码如下：

*Array*.prototype.forEach = *function*(*callback*){

*console*.log("调用自定义的forEach");

*var* arr = this;

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

if(arr[i] !== undefined)

callback(arr[i],i,arr);

}

}

*Array*.prototype.map = *function*(*callback*){

*console*.log("调用自定义的map");

*var* newArr = [];

*var* arr = this;

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

if(arr[i] !== undefined)

newArr[i] = callback(arr[i],i,arr);

}

return newArr;

}

**!注意：**上述代码中，在循环里先对遍历到的数组元素进行判断，以避免稀疏数组造成的NaN问题。



// var nums = [1,2,3,4,5];

*var* nums = [1,,3,,5]; //稀疏数组

nums.forEach1(*function*(*elem*,*i*,*arr*){

arr[i]=elem\*2;

});

*console*.log(nums);

#### 5.5.4 过滤和汇总

filter()过滤 复制出原数组中符合条件的元素组成新数组返回（即原数组保持不变）

var subArr = arr.filter(function(elem,i,arr){

return 判断条件;

});

reduce()汇总 对数组中所有元素进行统计，并返回统计结果

其中，base是赋给prev的初始值。

var result = arr.reduce(function(prev,elem,i,arr){

//prev: 截止当前元素之前的临时汇总值

return prev+elem;

},base);

## 六、ES6新特性

不改变原理的基础上，尽量简化代码

### 6.1 let

let作用：代替var用于声明变量

var的问题：

> 声明提前

> 没有块级作用域

let原理：匿名函数自调!

### 6.2 参数增强

#### 6.2.1 默认参数

语法：function fun(形参,…,形参=默认值)

强调：带默认值的形参，必须放在形参列表的末尾

#### 6.2.2 剩余参数

作用：代替arguments

arguments的问题：

> 不是纯正的数组类型，无法使用数组API

> 只能获得全部参数，不能有选择的获得部分

剩余参数的优点：纯正的数组且可有选择的获得部分参数

如何使用剩余参数：

> 定义函数时：function fun(形参,…,…数组名)

> 函数中使用剩余函数时，直接用“数组名”

兼容：将arguments转换为数组：强行调用数组的slice

var arr = Array.prototype.slice.call(arguments); //复制全部参数

Array.prototype.slice.call(arguments,starti); //复制starti（包括starti）之后的所有参数

//上一行代码可简写为：

[].slice.call(arguments);

#### 6.2.3 散播

作用：散播(spread),代替apply，专门打算数组参数为单个值

apply的问题：本意是替换this，不是专门打算数组用的

使用spread：

调用时: fun(实参1,..., ...数组)

Math.max (...arr)

兼容:fun.apply(obj,数组)

### 6.3 箭头函数

专门简化回调函数或匿名函数自调

> 去掉function，换=>

> 如果只有一个参数，可省略()

> 如果函数体只有一句话，可省略{}，如果仅有的一句话还是return，就要继续省略return

如果希望回调函数中的this和外部的this保持一致时，就可以改为箭头函数。

*var* lilei = {

sname: "李雷",

friends: ["tom","jack","jerry"],

intr(){

// this->lilei

this.friends.forEach(/\*function(\*/f/\*){\*/*=>*

// this->window 改为箭头函数后 this随外层this的指向

*console*.log(`${this.sname}认识${f}`)

/\*}\*/)

}

}

lilei.intr();

**!注意：**如果不希望内外this通用时，就不能改为箭头函数。

### 6.4 模板字符串

作用：代替+号做字符串拼接

【TIPS】ESLint中规定不允许使用+拼接字符串，必须用模板字符串

使用:

> ``包裹字符串模板

> 内部支持换行

> 只是动态生成内容: ${js表达式}

### 6.5 解构

定义：将一个对象中的成员，提取出来，单独使用

作用：

> 简化使用对象的成员

> 简化批量赋值

何时：将一个大的对象，拆成多个小成员单独使用时

使用解构：

#### 6.5.1 数组解构

下标对下标，将数组中的元素，提取出来，分别保存到多个变量中

var[变量1,变量2, … ]=[元素1,元素2, … ]

0 1 0 1

变量1=元素1

变量2=元素2

…

#### 6.5.2 对象解构

属性对属性

var {属性1:变量1,属性2:变量2, … }={属性1:值1,属性2:值2, … }

变量1=值1, 变量2=值2

简写：如果等号左边的{}中属性名和变量名相同，可只写一个。

var {属性1,属性2, …}=对象

#### 6.5.3 参数解构

本质是特殊的对象解构

何时: 定义函数时，函数的参数个数和顺序都不确定的情况下使用。

使用:

> 定义函数时，将所有参数定义在一个对象结构中

function fun({属性名1:形参1, 属性名2:形参2, ... ...})

> 调用时:

fun({ 变量1, 变量2,...})

其中, 变量名（属性名）要和参数名保持一致，但是，顺序和个数不必一致！

优点: 调用函数时，参数的个数和顺序与函数定义时无关！

### 6.6 for-of

for of：简化版的for循环，专门用于遍历数组

遍历数组的三种方式：

**>** for循环（最灵活）

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

var elem = arr[i];

}

问题：无法遍历关联数组和对象。

**>** forEach

arr.forEach((elem,i,arr)=>{})

问题：无法控制遍历的顺序和步伐

**>** for-of

for(var elem of arr){}

问题：无法获得当前位置i，也就不能获取当前元素值

for-in专门遍历自定义下标名称的关联数组和对象

for-of专门遍历数字下标的数组和类数组对象

### 6.7 class封装对象

传统构造函数和原型对象方法是分别独立定义的，不符合封装的要求。而class可以封装一个类型的构造函数及其原型对象方法的程序解构。今后，只要创建一种类型都要将构造函数和原型对象方法包装在class中

语法：创建一个对象

var obj = {

属性名：变量名 //如果属性名和变量名同名可省略不写变量名

方法名：function(){…}

}

#### 6.7.1 使用class封装对象

* 创建class：

> 用class{}包裹住构造函数和对应的原型对象方法

> 构造函数名提升为class名，构造函数去掉function，并更名为固定的constructor

> 原型对象中的方法可省略.propotype前缀和function，直接定义在class下的方法，默认保存在原型对象中继承

* 使用class

用法同构造函数完全一致，也可用new

#### 6.7.2 class中的访问器属性

使用前提：依然需要一个隐藏的数据属性被保护，实际存储数据。

* 定义访问器属性:

相同属性名的一对儿get/set方法自动组成一个访问器属性，并保存在原型对象中。

class xxx{

构造函数(){ ... }

get 属性名(){ return ...}

set 属性名(value){ ... }

}

#### 6.7.3 class中的静态方法

* 定义静态方法:

class 类型名{

构造函数(){ ... }

static 方法名(){ ... }

}

* 调用静态方法:

类型名.方法名()

#### 6.7.4 继承

* 让子类型继承父类型:

class 子 extends 父{

...

}

//extends代替了Object.setPrototypeOf

* 子类型构造函数中借用父类型构造函数

constructor(...){

super(...) //代替了.call(this)

//super指代父类型，super()相当于调用父类型的构造函数

//super()不需要传this

}

### 6.8 Promise

问题：回调地狱(callback hell)，即为了保证多个异步任务必须顺序执行，而形成的很深的嵌套调用结构

解决：使用Promise

#### 6.8.1 Promise使用

* 定义Promise对象封装要执行的异步任务

**!注意：**

new Promise(function(open){

//异步任务

//在异步任务最后一步之后，打开开关open()

})

> new Promise在创建对象同时，就自动执行内部的异步任务

> 如果不open(),则后续的then不执行！

* 将多个Promise对象，用.then连起来

Promise1

.then(function(){

return Promise2

})

...

.then(function(){

最后一项任务

})

#### 6.8.2 错误处理

* 在new Promise中，如果出错，则调用err开关:

new Promise(function(open,err){

//异步任务

//如果正常执行完，调用open()

//如果出错，调用err("错误消息")

})

**!注意：**调用err()后，后续then不执行

* 在顺序调用的结尾，只添加一个.catch()

.catch(function(err){ ...err变量接住"错误消息" })

* 前后两个异步调用间传参

前一个异步调用，通过open(参数)方式传参

后一个异步调用，通过.then(function(参数){ ... })方式接受

#### 6.8.3 用ES7简化Promise

ES6的Promise的问题: .then()依然使用嵌套方式

.then(function(){

return 异步调用的函数

})

解决: await:

前提: await修饰的函数必须也是return new Promise()的

使用:

##### 6.8.3.1 await控制程序按要求顺序执行

(async *function*(){ //async表示大的任务是异步的

try{ //代替.catch(function(err){}) 做错误处理

*var* result =

await ajax("get","http://localhost:8000/liang",*res=>*{

*console*.log(res); //liang的最后一步

}) //await表示等待程序执行完，阻塞大任务中的每一步

//如果异步调用中通过open(参数)返回了数据，可用var result=来接住异步任务返回的数据，在后续步骤中继续使用。

*console*.log(result);

*var* result =

await ajax("get","http://localhost:8000/yaner",*res=>*{

*console*.log(res);

})

*console*.log(result);

*var* result =

await ajax("get","http://localhost:8000/dong",*res=>*{

*console*.log(res);

})

*console*.log(result);

}catch(err){

*console*.error("出错啦！"+err);

}

})()

##### 6.8.3.2 等待多个异步任务执行后才执行

*Promise*.all([

ajax("get","http://localhost:8000/liang",*res=>*{

*console*.log(res); //liang的最后一步

}), //return new Promise

ajax("get","http://localhost:8000/yaner",*res=>*{

*console*.log(res);

}), //return new Promise

ajax("get","http://localhost:8000/dong",*res=>*{

*console*.log(res);

}) //return new Promise

]).then(*function*(){

*console*.log("比赛结束！");

}).catch(*function*(*err*){

*console*.error("出错啦！"+err);

})

# 第十一章 DOM

## 一、DOM概述

DOM定义：文档对象模型(Document Object Model)，专门操作网页内容的API标准

JS = ECMAScript(3.1/5/6/7) 核心语法

+DOM 专门负责操作网页内容

+BOM 专门负责操作浏览器窗口

DOM标准：为了统一所有浏览器操作网页内容的API

DOM包括五件事：增、删、改、查、事件绑定

## 二、DOM树

### 2.1 DOM树概述

定义：内存中存储所有网页内容的树形结构，而网页内容也是由层级关系的，所以产生了DOM树结构。

DOM树的产生：

> 当浏览器得到一个.html页面加载时，先在内存中

> 浏览器变读取网页内容，便在document下创建各级子节点对象。

### 2.2 节点对象的三个共同属性

#### 2.2.1 节点类型nodeType

值为整数。辨别获得的节点对象的类型时使用

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点名称 | document | element | attribute | text |
| 值 | 9 | 1 | 2 | 3 |

问题：无法进一步识别元素的具体名称

#### 2.2.2 节点名称nodeName

想进一步辨别元素的具体名称时用。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点名称 | document | element | attribute | text |
| 值 | #document | 全大写标签名 | 属性名 | #text |

#### 2.2.3 节点值nodeValue

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点名称 | document | element | attribute | text |
| 值 | null | null | 属性值 | 文本 |

## 三、查找

### 3.1 不需要查找就可以直接获得的节点元素

根节点 document

<html> document.documentElement

<head> document.head

<body> document.body

<form> document.forms[id/i]

### 3.2 按节点间关系查找

何时：如果已经获得一个元素，要找周围元素时

节点树：包含所有网页内容(元素、文本等)的树结构

#### 3.2.1 父子关系

node.parentNode 获得node的父节点

——> node.parentElement

node.childNodes 获得node的所有直接子节点

——> node.children

node.firstChild 获得node的第一个直接子节点

——> node.firstElementChild

node.lastChild 获得node的最后一个直接子节点

——> node.lastElementChild

**!注意：**node.parentNode和node.parentElement的区别：

<https://blog.csdn.net/qq_36010298/article/details/78636770>

#### 3.2.2 兄弟关系

node.previousSibling 获得node的上一个兄弟节点

——> node.previousElementSibling

node.nextSibling 获得node的下一个兄弟节点

——> node.nextElementSibling

问题：会受到看不见的空字符的干扰！

解决：元素树：仅包含元素节点的树结构

> 优点：不包含空字符节点，不受空字符干扰

**!注意：**元素树不是一棵新树，只是节点树的子集

##### 【笔试题】遍历指定父元素下所有子元素节点的步骤：

**>** 先仅遍历直接子元素节点

问题：childNodes和children返回的是动态集合(不实际存储属性值，每次访问集合，都重新查找DOM树)

优点：首次查找速度快，因为不必返回所有数据

缺点：反复访问集合，会导致反复查找DOM树

所以：for(var i = 0;i < children.length;i++)不好

解决：for(var i = 0,len=children.length;i < len;i++)

**>** 对每个直接子元素节点，调用与父元素节点完全相同的函数

### 3.3 按HTML查找

#### 3.3.1 按id查找

var elem = document.getElementById(id);

**!注意：**

**>** 只能用document上调用

**>** 每次只能找到一个元素对象

【TIPS】只有id属性能够通过id值直接获取该元素，因为id属性值会被自动声明为全局变量，但这种方法存在问题，一般不推荐在程序中使用。仅是在调试中图简便才使用。

#### 3.3.2 按标签名查找

var elems = parent.getElementsByTagName(标签名);

**!注意：**

**>** 可在任意父元素上调用

**>** 返回动态集合，

**>** 不仅查找直接子元素，且查找所有后代元素

#### 3.3.3 按name属性查找

var elems = document.getElementByName("name");

**!注意：**

**>** 只能在document上调用

**>** 专门用于查找表单元素的值

**>** 返回动态集合

#### 3.3.4 按class属性查找

var elems = parent.getElementsByClassName("class");

**!注意：**

**>** 可在任意父元素上调用

**>** 返回动态集合

**>** 不仅查找直接子元素，且查找所有后代元素

**>** 只要用修饰当前元素的多个class中任意一个class，都可找到当前元素

### 3.4 按选择器查找

#### 3.4.1 仅查找一个元素

var elem = parent.querySelector("选择器");

#### 3.4.2 查找多个元素

var elems = parent.querySelectorAll("选择器");

**!注意：上述两种查找均**

**>** 可在任意父元素上调用

**>** 不仅查找直接子元素，且查找所有后代元素

**>** 受制于浏览器的兼容性限制

**>** 返回非动态集合(返回完整数据的集合，即使反复访问集合，也不会反复查找DOM树)

优点：即使反复访问集合，也不会反复查找DOM树

缺点：首次查找效率低

##### 【笔试题】按HTML vs 按选择器，如何选择？

**>** 返回值：按HTML查找，返回动态集合；按选择器查找，返回非动态集合

**>** 首次查找效率：按HTML查找，效率高；按选择器查找，效率低

**>** 易用性：按HTML查找，简单好用；按选择器查找，繁琐

**总结：**如果只靠一个条件就可以找到元素时，首选按HTML查找；如果查找条件复杂，首选按选择器查找。

## 四、修改

### 4.1 修改内容

修改元素的HTML代码片段内容：.innerHTML

修改元素的纯文本内容：.textContent (存在浏览器兼容问题)

vs.innerHTML:innerHTML会删除内嵌标签，并将转义字符翻译为正文

textContent 给什么就原封不动输出什么

修改表单元素的值：value

### 4.2 修改属性

#### 4.2.1 修改HTML标准属性

* 核心DOM API：

.getAttribute("属性名");

.setAttribute("属性名","值");

.hasAttribute("属性名");

.removeAttribute("属性名");

* HTML DOM API：

对核心DOM API中常用的API进行的简化，即HTML DOM已经提前将所有标准属性做成了内存中元素对象的属性，默认值都是""：elem.属性名

特例：class

**!注意：**ES标准中已经有了class属性，所以DOM中将网页的class属性，更名为className

#### 4.2.2 修改状态属性

enabled disabled selected checked

值是bool类型！不能用核心DOM操作

#### 4.2.3 修改自定义扩展属性

自定义扩展属性：程序员自发添加的自定义属性

何时使用自定义扩展属性：

> 在客户端临时缓存业务数据

> 代替元素/class等选择器，用于查找元素绑定事件

问题：元素和class做选择器添加行为的弊端，即元素和class极有可能频繁修改，导致js程序失效

解决：今后，所有事件绑定都要查找自定义属性

如何自定义扩展属性：

> 添加自定义扩展属性：在开始标签中：data-属性名 = "值".

> 访问自定义扩展属性：肯定不能用HTML DOM，因为内存中的对象上没有自定义扩展属性。

- 用核心DOM API

- HTML5标准：elem.dataset.自定义属性名

dataset可自动收集开始标签中的data-开头的自定义属性

### 4.3 修改样式

只有一种样式可修改：内联样式

elem.style.css属性="值";

相当于<elem style="css属性:值">

**!注意：**css属性名都要去横线变驼峰，例如：z-index ——> zIndex

问题：上述获取样式的方式，不能用.style，仅能获得内联样式

解决：获得计算后的完整样式

var style = getComputedStyle(elem对象);

var value = style.css属性;

**!注意：**计算后的属性是只读的！禁止修改！因为来源不确定，可能牵一发而动全身。

【总结】如果修改样式: .style.css属性；如果获取样式: getComputedStyle(elem)

## 五、添加/删除

### 5.1 添加

#### 5.1.1 添加元素方法

* 先创建空元素对象：

var a = document.createElement("a");

效果：<a></a>

* 设置关键属性：

a.href = "http://tmooc.cn"

a.innerHTML = "go to tmooc"

效果：<a href="http://tmooc.cn">go to tmooc</a>

* 将新元素添加到DOM树上(3种)：

// 将a追加到parent下子元素的末尾

parent.appendChild(a);

// 将a插入到child之前

parent.insertBefore(a,child);

// 用a替换child

parent.replaceChild(a,child);

#### 5.1.2 优化添加方法 ★

问题：在添加元素中，因尽量减少操作DOM树的次数。因为每操作一次DOM树，就会导致重排重绘（效率极低）

解决：使用以下两种方式进行优化

1)如果同时添加父元素和子元素，应现在内存中将子元素加到父元素，最后再一次性将父元素添加到DOM树

2)如果父元素已经在页面上，只是添加多个平级子元素，要用文档片段（内存中临时保存多个子元素的虚拟父元素）

> 创建文档片段对象

var frag = document.createDocumentFragment();

> 将子元素加入到文档片段

frag.appendChild(child);

> 将文档片段添加到DOM树

parent.appendChild(frag);

**!注意：**frag将子元素添加到DOM树后，自动释放，不会成为页面上真实的元素。

### 5.2 删除

parent.removeChild(child)

## 六、HTML DOM常用对象

HTML DOM对部分复杂的元素的API进行了简化

### 6.1 Image

代表页面上一个<img>元素，可用以下简写创建<img>元素：

var img = new Image();

### 6.2 Select/Option

代表页面上一个<select>元素及其内部的<option>元素，它们各自有以下的属性、方法和事件。

#### 6.2.1 Select属性

* .selectedIndex 获取当前选中项的下标位置
* .value 获得当前选中项的值：

> 如果选中的option有value，则返回value

> 如果选中的option没有value，则返回内容

* .options 获得当前select下所有option的集合

.options.length 获得当前select下option的个数

* .length <==> .options.length

#### 6.2.2 Select方法

* .add(opt)

问题：不支持文档片段

* .remove(i) 删除i位置的option

#### 6.2.3 Select事件

onchange = function(){…} 当选中项发生改变时

#### 6.2.4 Option创建

代表select的下一个option元素

* 创建：

var opt = new Option(text,value);

### 6.3 Table/…

Table代表页面上一个<table>元素，Table管着行分组：

#### 6.3.1 行分组

* 创建：

var 行分组 = table.createThead()

.createTbody()

.createTfoot()

* 删除：

table.deleteThead();

table.deleteTfoot();

* 获取：

table.tHead();

table.tFoot();

table.tBodies[i]

#### 6.3.2 行分组下的行

* 创建：

// 在行分组内i位置插入一个新行

var tr = 行分组.insertRow(i)

【TIPS】固定套路：一般来讲都是在开头结尾添加行

> 在开头插入新行：var tr = 行分组.insertRow(0)

> 在结尾追加新行：var tr = 行分组.insertRow()

* 删除：

// 删除行分组中i位置的行

行分组.deleteRow(i)

**!注意：**i要求是行分组内的下标位置

问题：无法自动获得要删除行相对于行分组内的下标位置

解决：tr.rowIndex 记录着tr在整个表中的下标位置，所以应用table去删除指定行,

不能用行分组.deleteRow(tr.rowIndex)。

// 删除表格中i位置的行

table.deleteRow(tr.rowIndex);

* 获取：

行分组.rows[i];

table.rows[i]

#### 6.3.3 行下的单元格

* 创建：

var td = tr.insertCell(i)

【TIPS】固定套路：一般来讲新格都是追加在末尾：tr.insertCell()

* 删除：tr.deleteCell(i)
* 获取：tr.cells[i]

### 6.4 Form/…

#### 6.4.1 Form

Form代表页面上一个<form>元素

* 获取：var form = document.forms[i/id]
* 属性：

> .elements 获得form中的表单元素(input select textarea button)

.elements.length 获得form中表单元素的个数

> .length <==> .elements.length

* 方法：form.submit() 手动调用提交表单

#### 6.4.2 表单元素

* 获取：var elem = form.elements[i/id/name]

**!注意：**如果表单元素上有name属性，则可用form.name直接获取到

* 方法：

> .focus() 让当前元素获得焦点

> .blur() 让当前元素失去焦点

**!注意：**页面中只能有一个元素获取焦点，即当下一个元素获取焦点时，之前的元素自动失去焦点。

# 第十二章 BOM

## 一、BOM概述

BOM,Browser Object Model 浏览器对象模型，专门操作浏览器窗口的API。

问题：没有标准！

BOM包括：

> window:包含了所有ES+DOM+BOM的API

> location

> history

> document

> navigator

> screen

> event

## 二、BOM常用对象

### 2.1 window对象

#### 2.1.1 window对象的角色

> 代表ES中的global，充当全局作用域对象

> 封装ES+DOM+BOM的所有API

#### 2.1.2 window对象的属性

* 窗口大小

> 整个浏览器的窗口大小：outerWidth/outerHeight

> 文档显示区域大小：innerWidth/innerHeight

#### 2.1.3 window对象的方法

##### 2.1.3.1 open()

打开一个新窗口

* 在当前窗口打开新链接，可后退

html中的写法：<a href="url" target="\_self">

js中的写法： open("url","\_self");

* 在当前窗口打开新链接，禁止后退

js中的写法： location.replace("新url");

【原理】用新url代替history中当前旧的url来禁止后退

* 在新窗口打开新链接，可打开多个

html中的写法：<a href="url" target="\_blank">

js中的写法： open("url","\_blank")

* 在新窗口打开新链接，只能打开一个

html中的写法：<a href="url" target="自定义窗口名">

js中的写法： open("url","自定义窗口名");

【原理】浏览器中每个窗口在内存中都有一个唯一的name，可认为指定窗口名，如果不指定，浏览器就随机生成。

**!注意：**浏览器规定，相同窗口名的窗口，只能打开一个，后打开的同名窗口会把先打开的窗口覆盖掉

【TIPS】内置窗口名：

> \_self 用当前窗口自己的名字打开新窗口

> \_blank 不指定窗口名，让浏览器随机分配

##### 2.1.3.2 close()

关闭当前窗口

### 2.2 history对象

history：保存当前窗口打开后，成功访问过的url的历史记录栈（数组）

可用程序控制前进和后退：

history.go(1) 前进一步

history.go(-1) 后退一步

history.go(0) 刷新

### 2.3 location对象

保存当前窗口正在访问的url地址的对象

#### 2.3.1 属性

.href 完整url

.protocol 协议

.host 主机名+端口号

.port 端口号

.pathname 相对路径

.hash #锚点地址

.search ?查询字符串

#### 2.3.2 方法

* 在当前窗口打开新链接，可后退：

location.assign("新url") <==> location.href="新url"

* 在当前窗口打开新链接，禁止后退：

location.replace("新url");

用新url代替history中当前旧的url来禁止后退

* 刷新

> 普通刷新：优先从浏览器本地缓存中获取资源，缓存中没有或过期，才从服务器获取；

history.go(0)/F5/location.reload(/\*false\*/) false：不强制

> 强制刷新：跳过浏览器本地缓存，总是从服务器端下载新资源

location.reload(true) true：强制

### 2.4 navigator对象

保存浏览器配置信息的对象，常用于获取客户端浏览器和操作系统信息，故当想查看浏览器的配置信息时，会用到此对象。

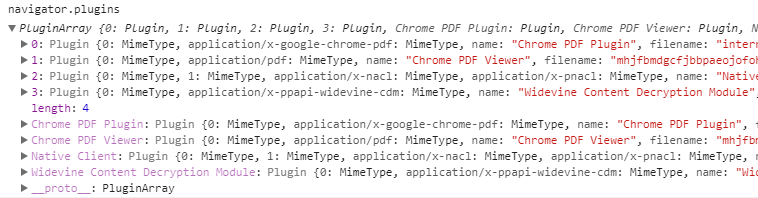


#### 2.4.1 属性

##### 2.4.1.1 .plugins

返回一个plugins[]对象集合，保存浏览器已安装的插件信息

**!注意：**虽然 plugins[] 数组是由 IE 4 定义的，但是在 IE 4 中它却总是空的，因为 IE 4 不支持插件和 Plugin 对象。



##### 2.4.1.2 .cookieEnabled

判断当前浏览器是否启用cookie

> cookie：客户端本地持久保存用户个人信息的小文件

程序内存中的数据（变量，数组，对象）都是临时存储，程序关闭，内存释放！故只要希望在客户端持久保存一个数据，就应该使用cookie

原理：

##### 2.4.1.3 userAgent

保存当前浏览器名称和版本号的字符串

判断浏览器种类和版本时使用

##### 【笔试题】判断浏览器种类及版本

[..\DOM\day03\11\_navigator.html](../DOM/day03/11_navigator.html)

## 三、定时器

### 3.1 周期性定时器

让程序每隔一段时间间隔，反复执行一项任务

使用：

> 任务函数：让定时器每次执行什么样的任务

> 启动定时器：var timer = setInterval(任务函数,间隔ms)

> 停止定时器：clearInterval(timer);

- 用户手动停止

- 自动停止：在任务函数中判断临界值，一旦到达临界值，则自动调用clearInterval()。

### 3.2 一次性定时器

让程序等待一段时间后，延迟执行一项任务，执行任务后自动关闭定时器

使用：

> 任务函数：让定时器每次执行什么样的任务

> 启动定时器：var timer = setTimeout(任务函数,延迟ms)

> 停止定时器：clearTimeout(timer);

## 四、事件

### 4.1 事件概述

事件定义：浏览器自己触发的或用户手动触发的页面状态的改变

事件处理函数：事件发生时自动调用的函数

何时：只要希望发生事件时，自动执行一项任务

使用事件处理函数：先将处理函数绑定到元素的事件属性上，当事件发生时自动调用

### 4.2 事件绑定方法

#### 4.2.1 在HTML中绑定

<elem on事件名="js语句">

问题：不符合内容与行为分离的原则，不便于维护

#### 4.2.2 在js中绑定

* 赋值方式

elem.on事件名 = function(){

… …

}

问题：一个事件属性只能绑定一个处理函数

* 添加事件监听

elem.addEventListener("事件名",处理函数)

elem.removeEventListener("事件名",处理函数)

【原理】

> 创建一个事件监听对象，封装元素、事件名和处理函数

> 将事件监听对象假如浏览器的事件队列中

> 当事件发生时，浏览器通过遍历找到符合条件的监听对象，调用其处理函数执行。

**!注意：**移除时必须获得原处理函数对象

### 4.3 事件模型

定义：事件发生时的执行过程

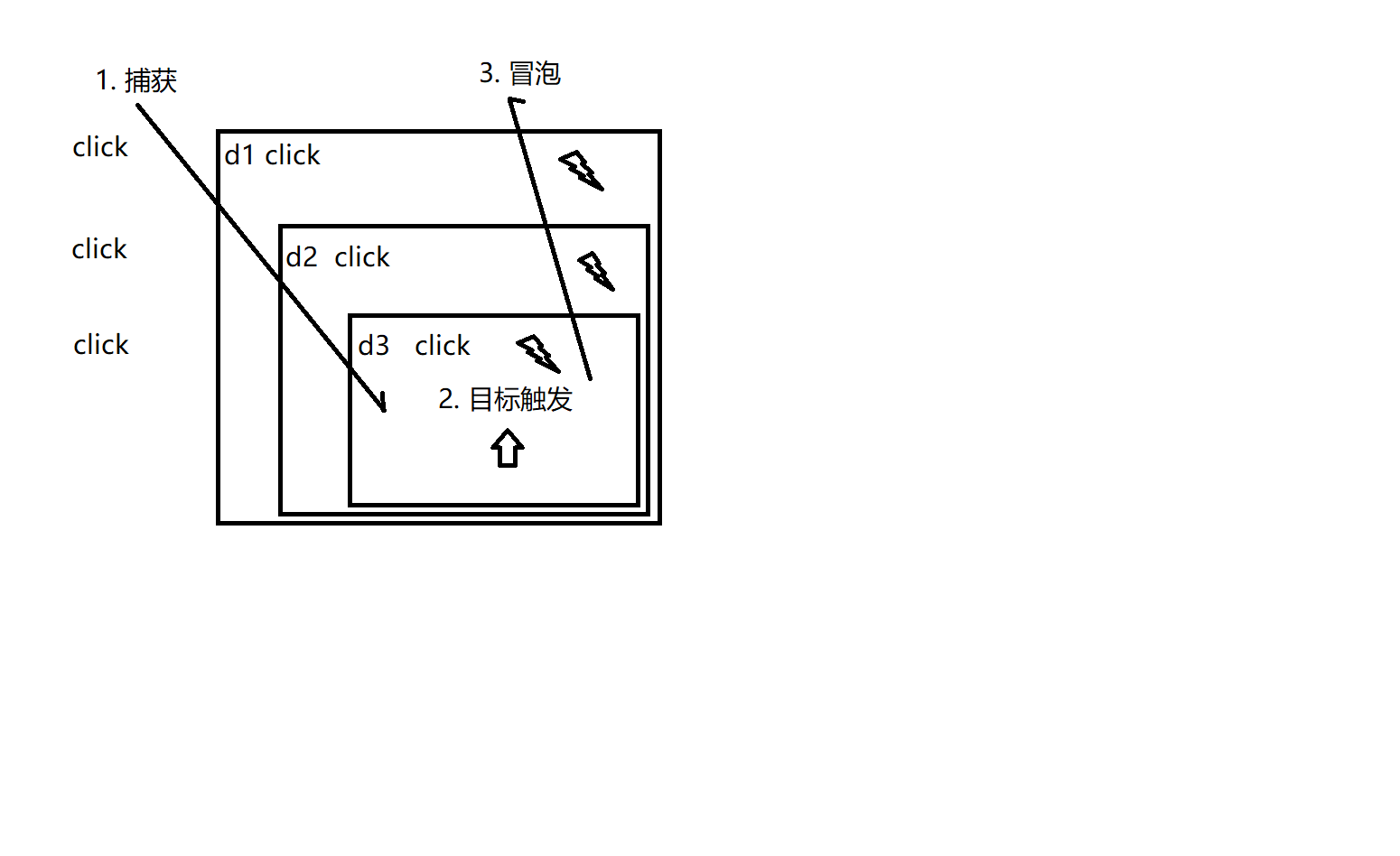
在DOM标准中的事件模型分3个阶段：

* 捕获：由外向内，记录各级父元素绑定的处理函数
* 目标触发：优先触发目标元素上的处理函数

目标元素：实际触发事件的元素

* 冒泡：由内向外依次触发各级父元素上的处理函数

冒泡产生原因：所有浏览器的开发者都认为点在内层元素上，等效于点在外层元素上



### 4.4 事件对象

定义：事件发生时，自动创建的，封装事件信息，并提供操作事件的API

何时：只要希望获得事件的信息或操作事件的行为时

#### 4.4.1 获得事件对象

事件对象总是默认作为处理函数的第一个参数自动传入：

function(e){

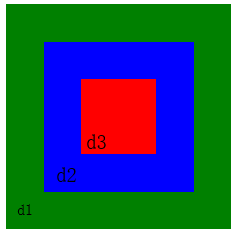
//…

}

#### 4.4.2 事件对象API

##### 4.4.2.1 取消冒泡

e.stopPropagation();



d1.onclick = function(e){

alert("我是：" + this.id);

e.stopPropagation();

}

d2.onclick = function(e){

alert("我是：" + this.id);

e.stopPropagation();

}

d3.onclick = function(e){

alert("我是：" + this.id);

e.stopPropagation();

}

##### 4.4.2.2 利用冒泡

> 优化代码：浏览器通过遍历触发事件，监听个数多会导致遍历变慢，故应尽量减少事件监听的个数

> 使用：如果多个子元素要绑定相同的事件，则只需要在父元素上绑定一次即可，所有元素共用。

问题1：事件绑定在父元素上，故事件处理函数中的this不能在反馈到具体子元素上，即this->父元素。

解决1：e.target 自动获得目标元素，且不随冒泡而改变。

问题2：不是所有目标元素都是想要的

解决2：都要先判断目标元素是否符合要求，再执行操作。

var table = document.getElementById("data");

table.onclick = function(e){

var btn = e.target;

if(btn.nodeName == "BUTTON"){

//…

}

}

##### 【笔试题】利用冒泡 vs 普通绑定

* + 绑定位置不同：

> 普通绑定：绑在子元素上

> 利用冒泡：绑在父元素上

* + 事件监听的个数不同：

> 普通绑定：多

> 利用冒泡：只有一个

* + 对动态生成的新元素：

> 普通绑定：无法自动获得事件处理函数

> 冒泡绑定：可自动获得事件处理函数

##### 4.4.2.3 阻止默认行为

如果希望不执行HTML元素自带的行为时，可通过e.preventDefault()阻止其默认行为。HTML中主要有以下三种默认行为的情况：

* a标签

默认行为：<a href="#xxx">自动在地址栏结尾添加#xxx锚点地址

<body>

<a href="#top">回到顶部</a>

<script>

document.body.children[0].onclick = function(e){

e.preventDefault();

alert("hhh");

}

</script>

…

</body>

* 类型为submit的input标签

默认行为：<input type="submit">会自动提交表单

form.onsubmit = function(e){

var rName = vali(txtName,/^\w{1,10}$/);

var rPwd = vali(txtPwd,/^\d{6}$/);

// 如果验证不通过

if(!(rName&&rPwd))

e.preventDefault();

}

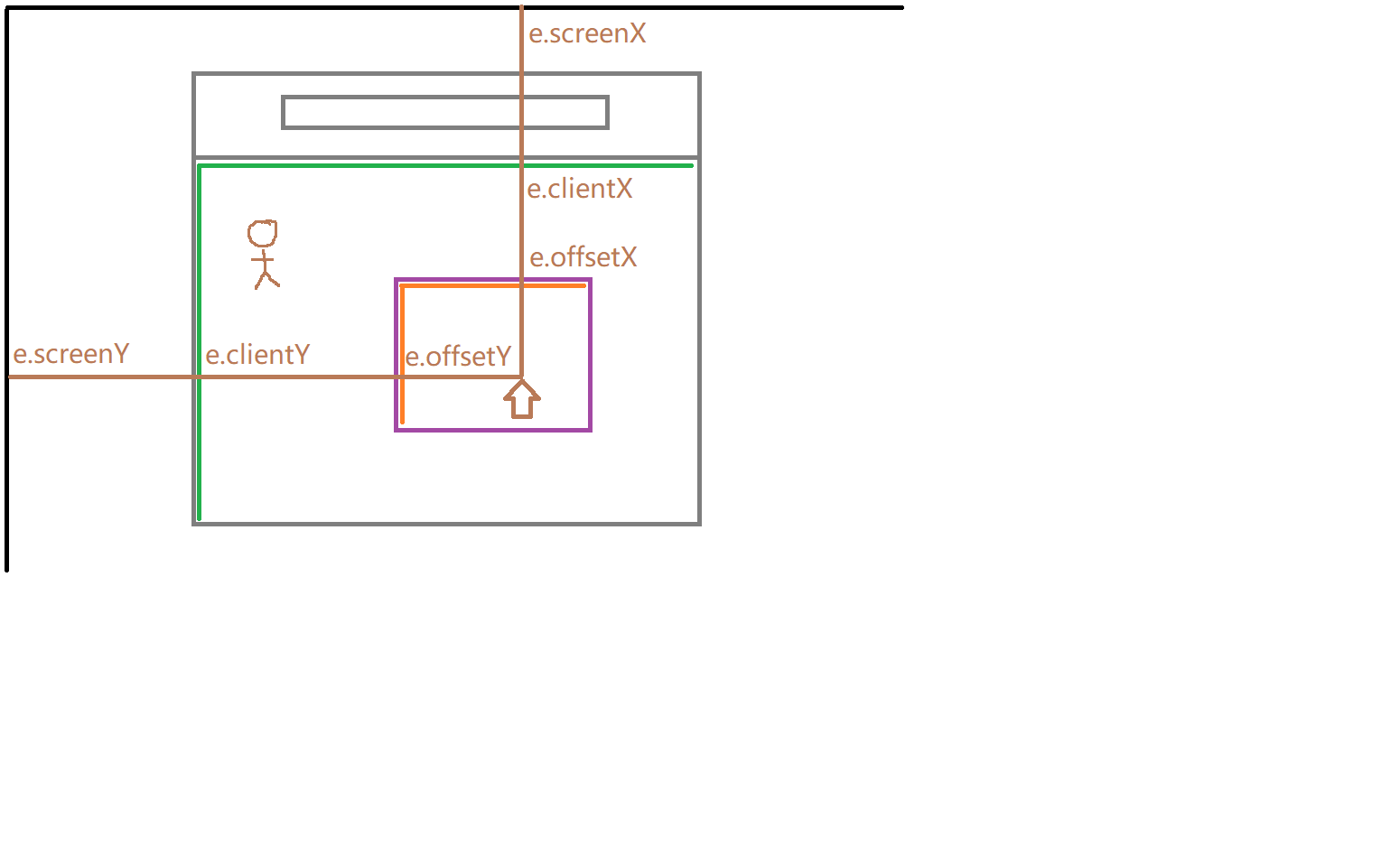
具体实例代码：[..\DOM\day04\4\_valiWithCSS\_onsubmit.html](../DOM/day04/4_valiWithCSS_onsubmit.html)

* 取消默认拖拽行为

下一阶段讲

##### 4.4.2.4 获得鼠标位置

* 相对于屏幕左上角：e.screenX e.screenY
* 相对于文档显示区左上角：e.clientX e.clientY
* 相对于当前元素左上角：e.offsetX e.offsetY



【实例】图片跟随鼠标移动[..\DOM\day04\5\_pop1.html](../DOM/day04/5_pop1.html)

##### 4.4.2.5 页面滚动事件

响应页面滚动事件

window.onscroll=function(){

//获得页面滚动的距离：scrollTop

var scrollTop=document.documentElement.scrollTop;

if(scrollTop>=500){

toTop.style.display="block";

}else{

toTop.style.display="none";

}

}

【课后作业】

> 图片自动轮播效果：[..\DOM\day03\6\_images.html](../DOM/day03/6_images.html)

> 自动关闭页面：[..\DOM\day03\7\_close.html](../DOM/day03/7_close.html)

> 完成计算器功能：[..\DOM\day04\3\_calc.html](../DOM/day04/3_calc.html)

# 阶段项目：学子商城后台数据搭建 ★

将public文件夹和xz文件夹分开以更好的理解客户端和服务器端代码，public-->客户端；xz-->服务器端。

## 一、首页数据搭建

### 第一步：服务器端新建首页的路由index.js

* 文件路径：xz/routes/index.js

1.引入需要用到的组件：Express框架、封装好的数据库pool线程池

var express = require("express");

var pool = require("pool.js"); //填写pool.js的相对路径

2.利用引入的Express框架创建路由router

var router = express.Router();

3.编写路由代码：

router.get("url",(req,res)=>{

//数据库操作

var sql = "…";

pool.query(sql,[],(err,result)=>{ //第二个参数为sql传入参数

if(err) throw err;

//res.write(result)是以下代码的简写形式，但无法修改指定值

res.writeHead(200,{

// 定义内容类型：json数据格式，中文采用utf-8

"Content-Type": "application/json;charset=utf-8",

// 允许跨域请求

"Access-Control-Allow-Origin": "\*" // \* --> all

});

res.write(JSON.stringify(result)); //将响应信息转为json格式

res.end(); //结束响应并将响应信息发送到客户端

})

})

4.导出写好的路由

module.exports = router;

第二步：

# 第十三章 jQuery

## **一、**jQuery**概述**

### 1.1 jQuery定义

jQuery是第三方提供的执行DOM操作的极简化的函数库

> 第三方：先下载，才能使用

> 执行DOM操作：学jQuery还是在学DOM的增删改查，事件绑定

> 极简化：jQuery是对DOM API的终极简化

> 函数库：jQuery使用函数解决一切问题！

### 1.2 使用jQuery的原因

在项目中使用jQuery的主要原因有以下两方面：

> API极简化(4个方面)：增删改查，事件绑定，动画，ajax

> 解决了大部分兼容性问题，凡是jQuery让用的都没有兼容性问题(jQuery内部自带浏览器检查机制，如遇不支持的语法，jQuery会使用相应的替换语法完成需求的功能)

### 1.3 下载jQuery

jQuery有以下三个版本：

**!注意：**各版本之间无好坏之分，根据需求进行选择

* 1.x 兼容旧浏览器

> 压缩版：去掉空字符、注释、极简化变量名

体积小，便于下载；可读性差

生产环境中才使用

> 未压缩版：包含完备的空字符、注释及见名知意的变量名

可读性好；体积大，下载慢

学习和开发时使用

* 2.x 不兼容旧浏览器
* 3.x 不兼容旧浏览器

##### 【笔试题】3.x版本新特性

> 所有代码运行在严格模式下

> 支持for-of循环遍历（以前用.each()函数）

> 新的动画API：requestAnimationFrame()

显示器刷新频率：50次/s

setInterval(function(){…},100)

若指定100次/s，会浪费定时器功能，即相当于有一半动画不能显示出来。而requestAnimationFrame()可以不指定定时器时间，而自动将其设置为显示器当前的刷新频率。

> 支持Promise(以前的版本用回调函数)

### 1.4 使用jQuery

#### 1.4.1 引入jQuery

引入项目本地文件夹中的jQuery.js文件

引入CDN网络中共享的jQuery.js文件

#### 1.4.2 jQuery原理

引入jQuery.js其实是在window中添加一个新类型jQuery，包括：

> 构造函数：创建该类型的子对象

> 原型对象：保存该类型的子对象共用的API

要使用jQuery是执行DOM操作的API，所以jQuery对象都要封装一个或多个DOM元素

#### 1.4.3 jQuery对象及其创建

定义：jQuery对象是封装一组DOM元素对象，并提供操作DOM对象的简化版API的类数组对象。

创建：有两种方法创建jQuery对象：

* 用选择器查找DOM元素，并封装进新创建的jQuery对象中。

var $jq = new jQuery("选择器");

简化1：将new关键词封装进jQuery对象内部

function jQuery(){

return new xxx();

}

所以：创建jQuery对象时，不用写new

var $jq = jQuery("选择器");

简化2：将jQuery对象赋值给$，简化代码的字符个数window.$ = jQuery

所以：可用$直接代替jQuery对象

var $jq = $("选择器");

* 直接将获得的DOM元素封装为jQuery对象

var $jq = $(DOM 元素);

## 二、jQuery操作DOM

### 2.1 jQuery API三大特性

* 一个函数两用：

> 没给新值，默认获取

> 给了新值，默认修改

* 自带遍历效果：

对jQuery对象调用一次API，等效于自动对内部每个DOM元素都调用一次API。

* 几乎每个API都返回正在使用的jQuery对象：

可使用链式操作精简代码

### 2.2 查找元素

#### 2.2.1 按选择器查找

如何：var $jq = $("选择器");

包括：jQuery支持所有CSS选择器，并扩展了部分新选择器

##### 2.2.1.1 基本选择器

同CSS

5个：#id .class element \* select1,select2

##### 2.2.1.2 层级选择器

同CSS

4个：父 后代，父>子代，兄+弟，兄~弟

##### 2.2.1.3 子元素过滤选择器

同CSS

4个：:first-child :last-child :nth-child :only-child

##### 2.2.1.4 基本过滤选择器(jQuery新增)

:first 找到页面中第一个元素

:last 找到页面中第一个元素

:gt(i) 从i位置开始往后查找出所有该元素(不包括i位置)

:lt(i) 从i位置开始向前查找出所有该元素(不包括i位置)

:eq(i) 找到页面中i位置的元素

:even 找出页面中所有偶数位置上的元素(包括0)

:odd 找出页面中所有奇数位置上的元素(奇数位置偶数行)

特点：

> 先取出所有符合条件的元素放入一个集合，在集合中统一编号

> 编号从0开始

**!注意：**如果一个样式既能用css又能用jQuery，优先用css。

##### 2.2.1.5 属性过滤选择器

同CSS

[属性名] [属性名=值] [属性名^=属性值] [属性名$=属性值] [属性名\*=属性值] [属性名!=属性值]

【TIPS】同时满足多个选择器条件可写为：多个选择器不加分割地连续使用

##### 2.2.1.6 内容过滤选择器(jQuery新增)

用元素的内容作为条件进行查找或判断

:contains(文本) 内容中包含指定“文本”的元素

:has(selector) 包含符合selector要求的子元素的父元素

:parent 选择不为空的元素

:empty 选择空元素

##### 2.2.1.7 可见性过滤器(jQuery新增)

:visible 选择可见的元素

:hidden 选择隐藏的元素，只识别display:none和input type="hidden"

##### 2.2.1.8 表单元素过滤和状态过滤

* 表单元素过滤：

:input 选择所有表单元素(input button select textarea)

> 每类input都有一个对应的选择器：

:text :password :submit :radio :checkbox

* 状态过滤：

:enabled :disabled :selected :checked

#### 2.2.2 按节点关系查找

##### 2.2.2.1 父子关系

$().parent() //.parentNode

$().children([selector]) //.children

$().find(selector) 在所有后代中查找符合selector要求的子元素，必须提供selector参数

$().children().first() //.firstElementChild

$().children().last() //.lastElementChild

##### 2.2.2.2 兄弟关系

$().prev() //.previousElementSibling

$().next() //.nextElementSibling

$().prevAll([selector]) //之前所有兄弟

$().nextAll([selector]) //之后所有兄弟

$().siblings([selector]) //除自己之外所有兄弟

### 2.3 修改

#### 2.3.1 修改内容

* 原始HTML片段：$().html([新HTML片段]) .innerHTML
* 纯文本内容：$().text([新文本]) .textContent
* 表单元素的值：$().val([新值]) .value

#### 2.3.2 修改属性

* HTML标准属性：$().attr("属性名"[,值]) .get/.setAttribute()
* 状态属性：$().prop("状态属性"[,值]) .属性名
* 自定义扩展属性：$().attr("属性名"[,值])
* 同时修改多个属性值：

$().attr({

"属性名1": 值,

"属性名2": 值,

…

})

#### 2.3.3 修改样式

* 修改时都要修改内联样式：

$().css("css属性"[,值]) .style.css属性

* 获取样式时要获得计算后的完整样式：

$().css("css属性") .getComputedStyle(elem)

* 同时修改多个样式值：

$().css({

"css属性": 值,

…

})

问题：如果批量修改多个属性值，代码会很繁琐

解决：用class批量操作样式

$().addClass("class1 class2…")

$().removeClass("class1 class2")

$().hasClass("class")

//在有与没有"class"两种状态间切换

$().toggleClass("class")

【TIPS】class2中的各样式会覆盖class1中同属性的样式。

### 2.4 添加/删除/替换/克隆

#### 2.4.1 添加

步骤：

> 用$("html片段")创建新元素，HTML片段中已经包含了子元素和属性的定义

> 将新元素添加到DOM树上

- $parent.append($新元素) //return parent

.prepend($新元素)

- $新元素.appendTo(parent) //return 新元素，可继续进行链式操作

.prependTo(parent)

- $现有元素.before($新元素) //return 现有元素

.after($新元素)

- $新元素.insertBefore($现有元素) //return 新元素

.insertAfter($现有元素)

简写：如果将代码片段添加到父元素后，不需要后续操作，可直接$parent.append("html片段")/$("html片段").appendTo(parent);

#### 2.4.2 删除

$().remove 删除自身

#### 2.4.3 替换

$旧元素.replaceWith($新元素) //return 旧元素

$新元素.replaceAll(旧元素) //return 新元素

#### 2.4.4 克隆

var $elem2=$elem.clone();

默认：.clone()是浅克隆，只赋值样式和属性，不复制行为

修改：.clone(true)是深克隆，即复制样式和属性，又复制行为

## 三、jQuery事件

### 3.1 事件绑定方式

#### 3.1.1 常规事件绑定方式

$elem.on(事件名,function(){

var $elem=$(this);

…

})

##### 【笔试题】jQuery中共有几种事件绑定的方式

* + .bind("事件名",function(){…})代替.addEventListener()

.unbind("事件名",fun)代替.removeEventListener()

.unbind("事件名") 移除该事件上所有的处理函数

.unbind()移除该元素上所有事件的所有处理函数

* .once("事件名"，function(){…})绑定后只能触发一次，触发后自动解绑
* .live()/.die()——已废弃
* 事件委托：其实就是利用冒泡

//绑定

.delegate("selector","事件名",function(){

this->e.target

不用if($tar.is("selector"))而将选择器写到参数里

})

//解绑

$parent.undelegate(…)

* 用on统一了bind和delegate

> 普通绑定

$elem.on("事件名",function(){})代替bind()

> 利用冒泡

//绑定

$parent.on("事件名","selector",function(){

this->e.target

不用写if判断了

})

//解绑

.off(…)

#### 3.1.2 简化事件绑定

对21种常用的事件，提供了简化版绑定的API：

$elem.事件名(function(){…})

**!注意：**该简化版本没有重载，故不支持delegate简化版冒泡。

### 3.2 页面加载后执行

* 仅DOM内容加载完就提前触发：仅等所有HTML和JS

DOMContentLoaded

何时：只要不依赖于CSS和图片的代码，都要在DOM内容加载后，就提前执行。用户可早一步用上。

比如：所有事件绑定代码，都要放在DOMContentLoaded中。

问题：DOMContentLoaded有兼容性问题

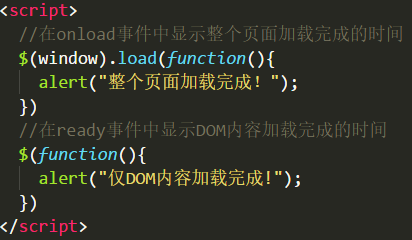
解决：$(document.ready(function(){}))

简化为 --> $().ready(function(){…}) --> $(function(){…})

**!注意：**绝大多数jQuery代码都要放在$(function(){…})中

* 整个页面内容加载完才触发：等HTML，CSS，JS，图片

何时：必须依赖于CSS和图片的代码，只能放在window.onload()中



页面会先弹出“仅DOM内容加载完成!”，而后再弹出“整个页面加载完成!”

### 3.3 鼠标事件

mouseover mouseout

问题：频繁冒泡

解决：都用mouseenter mouseleave代替over和out

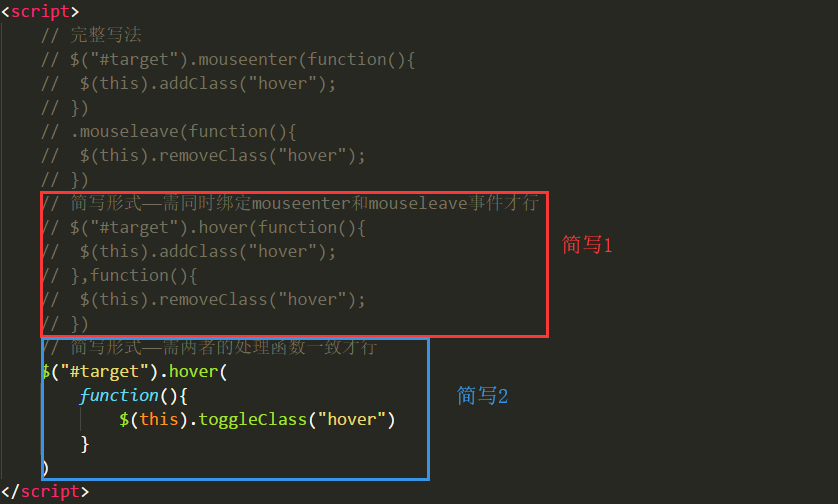
优点：反复进出子元素，也不会触发父元素的事件

简写1：如果同时绑定mouseenter和mouseleave可简写为-->

hover(function(){…},function(){…})

简写2：如果两个处理函数可合并为一个处理函数-->

hover(toggleClass("hover"))



### 3.4 模拟触发

定义：所谓模拟操作，就是比如用户点击按钮触发某种效果，而当页面加载完毕后模拟用户点击按钮，直接触发某种效果的方式。即，没有点击按钮，也可以执行按钮的操作。

如何：$elem.trigger("事件名")

简写：$elem.事件名()

### 【笔试题】jQuery中$的原理

$其实就是创建jQuery类型子对象的意思，包括以下四种情况：

* 如果传入选择器，$会查找DOM元素，并保存进jQuery对象中

> 优化：如果传入的选择器简单，则调用按HTML查找的API，如果传入的选择器复杂，则调用按选择器查找的API

* 如果传入DOM元素，$会直接封装DOM元素，封装进jQuery对象中
* 如果传入HTML片段，$会创建新元素，封装进jQuery对象中
* 如果传入函数，$会绑定DOM内容加载后执行的事件

## 四、jQuery动画

### 4.1 简单动画

效果固定的三组API

#### 4.1.1 显示隐藏

.show([ms]) .hide([ms]) .toggle([ms])

> 默认：封装的是.style.display="block"|"none"

> 设置时间参数，可增加动画效果

#### 4.1.2 上滑下滑

.slideUp([ms]) .slideDown([ms]) .toggleSlide()

#### 4.1.3 淡入淡出

.fadeIn([ms]) .fadeOut([ms]) .fadeToggle()

问题：

> jQuery动画API都是用js定时器实现的，效率不如css transition

> 效果是写死的，不可维护

【总结】不推荐使用jQuery简化版动画API，但是.show()/.hide()/.toggle()不加参数的情况是推荐的

### 4.2 自定义(万能)动画

#### 4.2.1 普通自定义动画

可对大多数css属性自定义动画效果

// 可让当前元素从当前状态，经过指定事件，过渡到目标样式

$().animate({

…:…

},ms)

问题：只支持单个数值的属性，甚至不支持颜色动画(颜色由rgb三组值构成)

#### 4.2.2 排队和并发

* 并发：多个css属性同时变化

> 如何：放在同一个animate内的多个css属性是并发变化

* 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，多个animate是不排队执行

* 动画结束后自动执行：

其实每个动画API都有最后一个回调函数参数。回调函数会在动画结束后自动执行。

* 停止动画：$().stop()

问题：默认只停止当前正在播放的一个动画，队列中其他动画继续执行

解决：$().stop(true) 停止当前动画，并清空队列

> 选择正在播放动画的元素：$().is(":animated")

## 五、jQuery类数组对象操作

jQuery为类数组对象，提供了简化版的API

### 5.1 遍历

#### 5.1.1 遍历当前jQuery对象

$().each(function(i,elem){

// i获得当前遍历到的位置

// elem 获得当前正在遍历的DOM元素

// each中的this->elem，所以参数elem可省略

})

#### 5.1.2 遍历所有类数组对象

$().each(类数组对象,function(i,elem){…}) //静态方法

EX:

*function* add(){

//arguments

*var* sum = 0;

$.each(arguments,(*i*,*elem*)*=>*{

sum += elem;

})

return sum;

}

*console*.log(add(1,2,3));

*console*.log(add(1,2,3,4,5));

### 5.2 查找

语法：$().index()

使用方法举例：

//获得当前单击的li在所有li中的位置i,i及其执行的都变为红色，i之后的都变为白色

$("ul.score").on("click","li",*function*(){

*var* i = $("ul.score>li").index(this);

$(`ul.score>li:lt(${i+1})`).css("background","red")

$(`ul.score>li:gt(${i})`).css("background","white")

})

分析：第四行代码中：找到当前元素(this)指向的li元素在"ul.score>li"找出来的数组中存在位置的下标。

## 六、添加自定义API

### 6.1 定义自定义API

在专门的.js文件中，定义函数并添加到Object.fn中：

jQuery.fn.自定义API = function(){

//this->将来调用该API的.前的jQuery对象

//因为已经是jQuery对象了，所以不用再$(this)

}

### 6.2 调用自定义API

在网页中，先引入jquery.js，再引入自定义.js文件，之后再编写自定义脚本，调用自定义API：

$().自定义API()

## 七、插件 ★

定义：拥有独立HTML内容，CSS样式和JS功能的网页部分。

作用：网页中有些功能，可能反复使用。今后，在遇到此类情况时，均使用插件，降低代码的重用。

### 7.1 插件的使用

#### 7.1.1 jQuery官方插件(jQuery UI)

* 使用步骤：

> 从官方下载

> 按插件的要求，定义HTML内容

> 先引入插件的css文件

> 引入jquery.js和插件js

> 在自定义脚本中，找到插件的父元素，调用插件API

$("插件父元素").插件API()

**!注意：**插件API和插件名相同

* 特点：

> 侵入性，即插件根据自身的需要，为元素自动添加样式和属性

> 优点：简单！

> 缺点：不便于维护！

#### 7.1.2 第三方插件

根据官方手册查阅用法

#### 7.1.3 自定义插件

前提：已经用HTML、css、js实现了插件效果。

自定义插件有两种风格：

##### 7.1.3.1 jQuery UI式(侵入式)

详细参见7.2.1

##### 7.1.3.2 Bootstrap式(DIY式)

详细参见7.2.2

### 7.2 封装自定义插件

#### 7.2.1 封装jQuery UI式插件

何时：小范围自用，不考虑过多变化时——简单

* 封装：

> 将插件的css提取到一个专门的.css文件中

> 在专门的.js文件中为jQuery类型添加自定义的API

- 在自定义API中自动查找子元素，侵入class和自定义扩展属性

- 事件绑定

**!注意：**一个插件内的css选择器以统一的父元素选择器开头。

* 使用：

> 按插件要求编写HTML，不用带class

> 引入插件的css

> 引入jQuery.js和插件.js

> 查找插件元素的父元素，调用插件函数

$("my-accordion").accordion()

#### 7.2.2 封装Bootstrap式插件

何时：大范围公开使用时，必须考虑各种变化——可维护

* 封装：

**>** 将插件的css提取到一个专门的.css文件中

**>** 在插件.js中自动查找带自定义扩展属性的元素，自动绑定事件

**!注意：**一个插件内的css选择器以统一的父元素选择器开头。

* 使用：

> 按插件要求编写HTML，手动在HTML元素上按需定义class

> 引入插件CSS

> 引入jqurey.js和插件.js

## 八、AJAX和跨域

### 8.1 各版本jQuery中的ajax用法

#### 8.1.1 3.x版本以下

$.ajax({

url:"…",

type:"get"|"post",

data:"变量=值&…"|{变量:值,…},

dataType:"json"

success:function() //在响应成功结束后自动执行

//res就是响应的结果

},

error(){//当请求出错时，自动触发},

complete(){//无论成功还是错误，只要请求结束就执行}

})

#### 8.1.2 3.x版本

.then(function(res){

…

})

### 8.2 跨域概述

定义：一个域名下的网站，请求另一个域名下的网站的资源

问题：不允许xhr对象发送ajax跨域请求！

### 8.3 禁止发送ajax请求的情况

以下情况不允许发送ajax请求：

> http://www.tedu.com ajax -> <http://www.tmooc>

> http://oa.tedu.com ajax -> <http://hr.tedu.com>

> http://localhost:3000 ajax -> <http://localhost:8000>

> https://baidu.com ajax -> <http://baidu.com>

> <http://localhost> ajax -> http://127.0.0.1

**!注意：**端口号或协议不同、本机域名访问访问本机ip也算跨域(协议不同本质上就是端口号不同)

### 8.4 跨域访问方式 ★

共有两种方式用以跨域请求：

#### 8.4.1 JSONP

JSONP，需要客户端与服务器端配合

定义：填充式JSON——JSON with Padding(填充)

何时：今后跨域首选JSONP

使用JSONP跨域的方法：

> 用其他可以跨域请求的元素，代替ajax

> 程序中用<script>元素代替ajax发送请求(script的src属性可以跨域)

##### 8.4.1.1 方案一

**>** 服务器端：将要返回的数据，填充进一条字符串格式的js语句中，组成一条正确的可执行的js语句，再返回。

**>** 客户端：添加<script src="服务器端地址">

结果：script可跨域请求到服务器端返回的js语句，并在客户端立即执行。

问题：服务器端返回的js语句是写死的，众口难调。

##### 8.4.1.2 方案二

**>** 服务器端：返回一条自定义函数的调用语句

要求客户端必须执行指定名称的函数

**>** 客户端：提前定义一个与服务器端同名的函数

函数有一个参数可接受服务器端的数据

问题：函数名是写死的，极容易发生冲突！

##### 8.4.1.3 方案三

**>** 服务器端：接受一个函数名参数，将客户端发来的函数名参数拼接到js语句的开头

**>** 客户端：<script src="地址?参数名=本地参数名">

问题：script是写死的！只能在页面加载时请求一次

##### 8.4.1.4 方案四

**>** 服务器端：保持不变

**>** 客户端：动态创建script元素

**!注意：**script只要被加载到页面上，就立刻自动请求

问题：script不断累积

解决：在回调函数中自动删除script

**!注意：**HTML规定script内不能再嵌套成对的script标签

##### 8.4.1.5 方案五

$.ajax也支持jsonp

$("#btn").click(function(){

$.ajax({

url:"http://localhost:8080",

type:"get",

dataType:"jsonp", //服务器端必须支持jsonp

// <script ?callback=随机函数名>

success:function(res){

alert(res);

}

})

})

**>** 服务器端：保持不变

**>** 客户端：使用jQuery封装好的ajax对象，将其dataType改为jsonp。(原理同方案四)

**!注意：**此方案中的<script src="…">是添加在head标签的第一个子元素位置上，而方案四中的<script src="…">是添加在body标签的最后一个子元素位置上。

【强调】dataType:"jsonp"需要服务器端配合才能实现

【总结】总的来说dataType:"jsonp"的原理和方案四一致：

> 也是通过在head中动态添加<script>发送请求

> 也是通过?callback=函数名，发送函数名到服务器端

> 函数名其实也是$.ajax自动为success函数生成的随机函数名

#### 8.4.2 CORS

CORS，只需要服务器允许即可，不需要客户端的配合。

在服务器端添加响应头：

// node中：

res.writeHead(200,{

"Content-Type":"application/json;charset=utf-8",

"Access-Control-Allow-Origin":"指定来源域名或所有(\*)"

})

问题：指定来源为所有(\*)会导致访问的安全问题，而指定源头又会很麻烦。

## 九、Session

### 9.1 session定义及原理

session出现的原因：http协议是无状态的！前后两次请求之间，无法共享数据

定义：服务器端内容中保存用户私密信息的存储空间

何时：希望在多个请求之间长时间保存一个共享数据时

原理：

**>** 当首次请求服务器时，服务器为当前客户端创建一个专门的内存存储空间，称为session并分配随机的session\_id，返回给客户端，并保存在客户端的cookie中

**>** 每次客户端请求服务器时，都自动携带cookie和session\_id到服务器端。可打开session，取出session中保存的数据使用。

### 9.2 session的销毁

**>** 如果客户端长时间未请求服务器，服务器自动删除该客户端对应的session

**>** 关闭浏览器，会导致存储在客户端cookie中的session\_id丢失；当再次打开浏览器访问服务器时，服务器会分配新的session和session\_id。原session即使没有释放，也访问不到。

### 9.3 使用session

**>** 项目本地安装并引入express-session模块

**>** 配置session模块中间件

- 在请求时，会将session中的数据取出，放在req的session属性中

**>** 读写session内容

- req.session["变量名"]

【小工具】postman--->post发送请求的测试工具

### 9.4 常见问题

问题：发送ajax请求session空

解决：在发送请求时设置以下标红部分的代码

$.post({

url: "...",

data: {

appId: appId

},

dataType: "json",

timeout:300000,

xhrFields:{withCredentials: true},

success: function (result) {

},

error: function () {

}

});

# 第十四章 VueJS

## 一、VueJS框架概述

### 1.1 原生API vs 函数库 vs 框架

DOM jQuery Vue

* 原生API：浏览器/平台已经实现的，可以直接使用的API，比如DOM、BOM。

> 优点：万能

> 缺点：繁琐

* 函数库(library)：基于原生的API，进一步封装的，更简化的一组函数的集合，比如jQuery。

> 优点：每个API都极简化

> 缺点：没有改变做事的方式，比如依然需要增删改查和事件绑定。

* 框架(framework)：前人将多次成功的项目经验总结起来，形成的半成品项目。

> 优点：无需重读劳动，后人只需要继续开发新功能即可。

> 缺点：彻底改变了做事的方法！需要适应

### 1.2 VueJS概述

#### 1.2.1 基本概念

定义：一个基于MVVM设计模式，用于构建用户界面的渐进式JavaScript框架。

渐进式(progressive)：虽然提供了很多功能，但不强迫全部使用。可逐渐使用，逐渐扩大使用范围。

MVVM设计模式：待续

何时使用Vue：适用于以数据操作为主的项目

特点：

> 响应式编程

> 组件化

> 模块化

> 简洁

**!注意：**Vue 不支持 IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。但它支持所有兼容 ECMAScript 5 的浏览器。

#### 1.2.2 环境搭建

VueJS提供两种方案进行VueJS的环境搭建：

##### 1.2.2.1 直接用<script>引入

直接下载并用 <script> 标签引入，Vue 会被注册为一个全局变量(Window.Vue)。



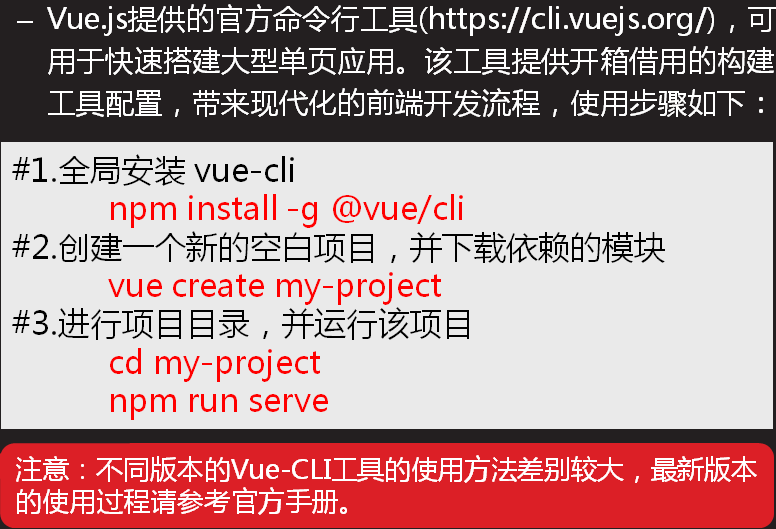
vue.js下载地址：<https://cn.vuejs.org/v2/guide/installation.html>

**!注意：**现目前的VueJS最新版本为：2.5.x

**!注意：**vue是一个纯前端框架，所以不需要nodejs、Apache等服务器端软件就可使用。

##### 1.2.2.2 VUE-CLI命令行工具

用VUE-CLI命令行工具生成项目脚手架代码，并填充内容(待续)



### 1.3 MVVM设计模式

#### 1.3.1 引入MVVM设计模式

传统网页组成：3部分

> HTML：定义网页内容

> CSS：定义网页样式

> JS：添加交互行为

问题：JS肩负的任务太多太杂了，产生了大量重复的劳动(比如频繁的增删改查、事件绑定等)

解决：重新划分3部分----->MVVM模式

> Model模型数据

> Vue视图：包含网页内容(HTML)和样式(CSS)

> ViewModel：视图模型/控制器：代替之前的DOM/JQUERY操作，自动把模型中的数据，绑定到View中的指定位置。(其实就是 new Vue();)

#### 1.3.2 什么是MVVM设计模式

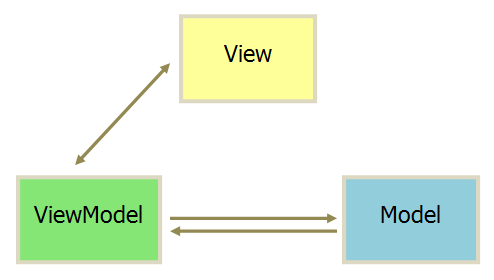
MVVM是一种设计思想，M表示Model，V表示视图View，VM表示数据与模型。

VM：首先要理解binding，可以参考WPF中的Binding，一个控件与后台数据模型（VM）进行绑定，后台数据模型也可以与M进行交互。VM就是将V->M或M-V的数据进行转换或者封装，主要思想如下：

> 当前台View发生变化时，View与VM进行了绑定，VM又与M进行交互，从而使M得到了改变。

> 当M变化时，以C#为例，M中的DO实现了INotifyChanged，当检测到变化时，M通知VM，VM与V进行了绑定，然后实现M->V。

**!注意：**区别MVC设计模式中，C是用于跳转至哪个页面，而MVVM模式中的VM只是将数据进行转换或者封装。



#### 1.3.3 创建ViewModel

引入Vue.js时，等于在全局引入了一种新类型Vue，包含：

> 构造函数：创建该类型的子对象

> 原型对象：包含该类型子对象共用的成员

new Vue()创建Vue类型的一个实例，其实是创建一个用于同步Model和View的ViewModel对象。

new Vue({

el:"选择器", // 告诉Vue监听哪个父元素的内容

data:{ //作为Model，保存页面所需要的数据

模型数据：值

}

})

#### 1.3.4 new Vue(); 两大子系统

##### 1.3.4.1 响应系统

new Vue()将data中的每个模型数据提升为整个Vue对象的访问器属性。在访问器属性的set方法中，只要修改模型数据的值，就触发通知。通知Vue框架修改页面。

##### 1.3.4.2 虚拟DOM树

问题：真实DOM树包含的内容太多！极其不便于比较和遍历

解决：用js程序模拟的一棵简化版的DOM子树，称为虚拟DOM树

如何生成虚拟DOM树：new Vue({el: "选择器"})会查找指定父元素，并遍历其中所有的内容，仅将可能发生变化的元素和属性用js对象结构集中存储在内存中；

何时使用虚拟DOM树：当修改数据模型，触发响应系统时，响应系统通知Vue要修改DOM树。Vue框架就遍历虚拟的DOM树，找到发生变化的节点，利用底层的DOM API仅修改发生变化的DOM元素。

虚拟DOM树的优点：

> 节点个数和属性个数都比真实DOM树，少得多，及其便于遍历和比较；

> 仅修改受影响的DOM元素，不变的元素不用修改；

### 1.4 现阶段使用Vue框架的步骤

* 定义HTML和CSS界面，只不过用特殊标识标记可能改变的位置；
* 在内存中定义模型数据对象data，其中，页面有几处变化，data中就要有几个模型变量与之对应；
* 创建new Vue()实例对象，充当ViewModel：

> new Vue()中定义el: "选择器"，指向要监视的父元素；

> new Vue()中还要将data对象包含进来作为model。

## 二、VueJS指令

### 2.1 数据绑定语法

{{}}双括号语法，外号大胡子语法{MUSTACHE}，学名：补缺。

什么是数据绑定：选择特定的Model变量，绑定到View试图中指定位置，使视图中指定位置的值随Model变量同步变化。

何时：只要希望用Model中的一个变量填补View中的一个动态显示位置时。

如何：{{模型变量名}}

执行时，会用模型变量的当前值，代替{{}}位置；

{{}}中可包含以下几种情况的内容：

> {{模型变量名}} ex：{{uname}}

> {{运算}} 算术/比较/逻辑/三目运算。ex：{{price\*count}}

> {{方法调用}} ex：{{uname.toUpperCase()}}

> {{对象.属性}} ex：{{address.city}}

> {{数组[i]}}

**!注意：**不能写if else for等程序结构

问题：{{}}只能绑定innerHTML,不能绑定属性和事件。

解决：使用指令。

### 2.2 指令

定义：Vue.js提供的专门用于增强HTML功能的特殊属性

因为HTML只有希望动态绑定HTML属性时，都用指令。包括下述13个。

#### 2.2.1 v-bind

作用：专门绑定属性值

语法：<ANY v-bind:属性="模型变量">，不用加{{}}

简写：v-bind可省略--->:属性="模型变量"

ex:

<a :href="url">查看详情</a>

**!注意：**通过":"绑定的属性值可以写成任何形式的js表达式

**!注意：**多个class属性可以同时存在，最后会合并为一个；有且仅有class属性有次特权。

#### 2.2.2 v-for

作用：根据数组或集合内容，反复生成多个相同的HTML元素

语法：<ANY v-for="(元素,下标) in 数组|对象|字符串" >

> 根据in后的数组/集合中元素的个数，反复生成当前HTML元素

> 每次in都会取出数组/集合中当前元素值和位置下标

> 如果<ANY>下还有子元素，则子元素可用for获得元素值和下标继续绑定内容

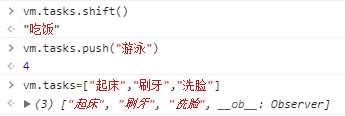
**!注意：**v-for必须放在要反复生成的HTML元素上，不能放在父元素上。

简写：

> 如果不关心下标，只关心内容：v-for="变量 in 数组"

> in可换为of(为了迎合一些人的习惯写法，给in起了个别名of，效果两者完全一致)

增删改查操作：



ex：

<ul>

<li v-for="(task,i) in tasks">

<span>{{i+1}}</span>-

<b>{{task}}</b>

</li>

</ul>

<script>

var vm = new Vue({

el:"ul",

data:{

tasks:["吃饭","睡觉","打亮亮","跑步"]

}

})

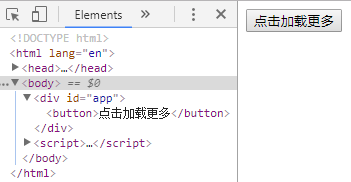
</script>

#### 2.2.3 v-if

作用：根据一个bool类型的模型变量值作为条件，控制是否生成该元素的DOM树(控制一个元素的显示和隐藏)。

语法:<ANY v-if="bool类型的模型变量">

ex:



<div id="app">

<button v-if="hasMore">

点击加载更多

</button>

</div>

<script>

var vm = new Vue({

el:"#app",

data:{

hasMore:true

}

})

</script>

hasMore:false; -----> <div id="app"><!----></div>

【TIPS】如何控制多个元素中选其一显示？

<ANY v-if="条件1">

<ANY v-else-if="条件2">

…

<ANY v-else>

依次判断每个条件，只要条件为true，就生成该元素，其余元素不生成(隐藏)

#### 2.2.4 v-show

作用：和v-if几乎完全一样！

语法：<ANY v-show="bool类型的模型变量">

##### 【笔试题】v-show和v-if的区别

* v-show：不改变DOM树，通过display:none/block控制显示隐藏

> 优点：效率高

> 缺点：只适合控制一个元素的显示隐藏。如果控制多个元素的显示隐藏时，代码会很繁琐。

* v-if：通过添加/删除DOM树上元素节点的方式控制显示隐藏

> 缺点：效率低

> 优点：专门控制多个元素选择其一显示，代码精简

**总结：**如果一个元素频繁显示隐藏切换，首选v-show；如果多个元素控制选其一显示，首选v-if,v-else-if,v-else。



#### 2.2.5 v-on

作用：专门为元素绑定事件处理函数

语法：

> 在new View()实例对象内添加事件处理函数

var vm = new Vue({

el:"选择器",

data:{ 模型变量:值，…},

methods:{

处理函数1(参数列表){

//访问当前Vue内自己的模型变量

this.变量名

},

处理函数2(参数列表){…}

}

})

> 在HTML元素上绑定事件处理函数

<ANY v-on:事件名="处理函数()">

> 获得事件对象e：

<ANY v-on:事件名="处理函数($event)">

**!注意：**$event是Vue内置的事件对象，不可更改

简写：

> v-on:被@替代<ANY @事件名="处理函数()">

> 事件修饰符：对常用事件操作的简写

- 取消冒泡：e.stopPropagation()

<ANY @事件名.stop="处理函数()">

- 阻止默认行为：e.preventDefault()

<ANY @事件名.prevent="处理函数名()">

- 取消冒泡且阻止默认行为：

<ANY @事件名.prevent.stop="处理函数名()">

> 键盘事件：<ANY @事件名.keyCode值="处理函数名()">

#### 2.2.6 v-html和v-text

v-html 使用模型变量替换当前元素的innerHTML

> 何时：如果绑定一段HTML代码片段

v-text 使用模型变量替换当前元素的textContent

> 何时：如果绑定一段纯文本内容

##### 【笔试题】v-html vs v-text vs {{}}

* 是否编译内容

v-html，专门用于绑定HTML片段，绑定的内容会被浏览器解析为正文

v-text和{{}}，专门用于绑定不需要解析的纯文本内容，保持内容的原样不变。

问题：当请求响应慢时，用户可能短暂看到{{}}语法。

解决：用v-text和v-html代替{{}}；当响应慢时v-text和v-html暂时显示为空白。

#### 2.2.7 v-cloak

作用：在vue实例加载完之前，暂时隐藏需要绑定的元素；避免让用户看到，增强用户体验。

语法：

> 自定义v-cloak属性选择器，定义display:none

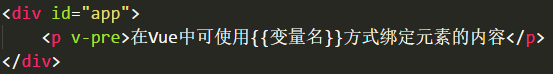
> 在要绑定的元素上，添加v-cloak

原理：new Vue()实例加载完成后，会自动找到v-cloak属性，并移除。

#### 2.2.8 v-pre

作用：保护内容中的{{}}不被编译，保持原样

当正文中刚好也有{{}}时，不想被Vue识别并使用



#### 2.2.9 v-once

作用：让当前元素只在首次加载时，绑定一次。之后，即使模型数据发生变化，页面也不改变。

原理：标有v-once的元素在首次加载后，就从监视队列中移除了。不再受监视。

问题：以上十种指令加上{{}}一共是一种都是单项绑定，不能将视图中的数据反向更新回内存的模型数据中。

解决：采用v-model进行双向绑定

#### 2.2.10 v-model ★

单项绑定：仅能将内存中模型数据的修改更新到视图中(M->V)。

双向绑定：既能将内存中模型数据的修改更新到视图中，又能将视图中的更改，反向更新回内存中的模型数据(V->M)

何时：只要页面上内容修改，也想反向更新回内存中的模型数据时——专门用于绑定表单元素。

<表单元素 v-model:value="模型数据">

简写：可以省略:value，<表单元素 v-model="模型数据">

##### 2.2.10.1 自定义监听函数

监听函数：当模型数据发生变化时自动执行的函数

何时：只要希望模型数据一发生变化就立刻执行一项任务——搜索帮助/表单验证。

如何：见下述代码中蓝色部分

<div id="app">

<input type="text" v-model:value="kword" @keyup.13="search()">

<button @click="search()">百度一下</button>

</div>

<script>

var vm = new Vue({

el:"#app",

data:{ kword:"" },

methods:{

search(){

console.log("查询包含关键词:"+this.kword+"的网页");

}

},

watch:{  
 模型变量(){

//只要同名模型变量发生变化就自动执行此函数

}

}

})

</script>

##### 2.2.10.2 各表单元素的绑定

**>** 文本框/文本域：v-model="模型变量"

**>** 单选按钮：

<input type="radio" name="组名" value="值 1" v-model="变量">

<input type="radio" name="组名" value="值 2" v-model="变量">

其实v-model绑定的是checked属性

checked=(变量==value)

**>** 复选框：v-model绑定的也是checked属性

**>** 选择框：v-model绑定selected=(变量==option的value)

### 2.3 绑定class和style属性

#### 2.3.1 绑定class属性

* 用字符串方式绑定

<ANY class="固定不变的class" :class="pno==i?'active':''">

* 用对象方式绑定

<ANY class="固定不变的class" :class=" xxxClassObj ">

new Vue({

data:{

xxxClassObj:{

fade: true/false

in: true/false

active: true/false

}

}

})

结果: :class中的类会和class中的类合并

:class中只有值为true的类名才能出现在class中

【总结】

> 在多个元素之间切换一个class，首选用字符串方式

> 控制一个元素的class，有与没有切换，首选用对象方式

#### 2.3.2 绑定style属性

同class属性一样，有以下两种方式进行style属性的绑定:

* 作为字符串属性绑定

<div id="mask" :class="mask\_hide" :style="`top:${top}px;left:${left}px`">

* 用对象方式

<ANY :style="styleObj">

new Vue({

el:

data:{

styleObj:{

css属性: 属性值,

…:…

}

}

})

【总结】建议今后所有style都采用对象语法绑定。

### 2.4 计算属性

#### 2.4.1 计算属性概述

问题：不是所有属性的值都可直接获得

解决：采用计算属性

定义：不实际存储属性值，而是根据其他属性的值动态计算获得。

何时：只要一个值不能直接获得，需要根据其他值计算获得时。

#### 2.4.2 计算属性的使用

* 定义

new Vue({

el:"",

data:{},

methods:{},

watch:{},

computed:{

属性名(){ return 计算后的值}

}

})

* 绑定：和普通属性的绑定完全一样！

{{属性名}}

**!注意：**不要加()

#### 【笔试题】计算属性 vs 函数

* 计算属性的值可被Vue缓存并重复使用，只要计算属性依赖的其它属性不改变，就不用重复计算！
* 函数只要调用，就重新计算！

### 2.5 自定义指令(了解)

Vue.js中除了13中预定义指定外，还需要增强HTML的其他功能，就需要添加自定义指令。

#### 2.5.1 创建自定义指令

Vue.directive("指令名",{ //创建一个自定义指令

inserted(elem){ //当当前元素被挂载到DOM树后自动执行

//elem 当前所在元素的DOM对象

//elem 可调用一切DOM的API

}

})

**!注意：**在定义指令时，指令名不要加v-前缀!

#### 2.5.2 使用自定义指令

<ANY v-指令名>

### 2.6 过滤器

#### 2.6.1 过滤器概述

过滤器(filter)，接收原始数据后，执行再加工，才显示。

何时：原始数据不能直接使用时，都要用过滤器

比如：性别(1,0 -> 男，女)

钱(345.50 -> ￥345.50)

时间(ms -> 字符串格式)

状态(10,20,30,40 -> 未付款，已付款，已发货，已签收)

#### 2.6.2 使用过滤器

Vue官方没有提供任何预定义的过滤器，都要自己定义

##### 2.6.2.1 创建过滤器

Vue.filter("过滤器名",function(value,形参1, 形参2,…){

// value接收的是原始值

return 加工后的新值

})

##### 2.6.2.2 使用过滤器

只要绑定数据，都可用过滤器：模型变量|过滤器名|过滤器名…

**!注意：**后一个过滤器获得的原始值，是前一个过滤器加工后的值

**!注意：**v-text不能使用过滤器

ex:

<script>

Vue.filter("genderConvert",function(val){

return val==1?"男":"女";

})

</script>

<div id="app">

性别:<span>{{gender|genderConvert}}</span>

</div>

<script>

new Vue({

el:"#app",

data:{

gender:1,

}

})

</script>

### 2.7 axios ★

#### 2.7.1 axios概述

定义：一个基于Promise的支持HTTP请求响应的函数库

为什么：浏览器中向服务器端发送ajax请求，共有4种方法

> 使用原生的xhr对象——繁琐

> 使用jQuery中的$.ajax()——大材小用

> 旧版Vue中，Vue官方自带了Vue-resource组件，专门发送xhr请求——新版本中已经不推荐使用

> axios，官方推荐的代替Vue-resource的，专门发送ajax请求的函数库

何时：只要在Vue中发送ajax请求，都用axios

如何：axios是一个单独的js文件，与平台无关，既可用在浏览器中，也可运行在node.js中。

#### 2.7.2 使用axios

在浏览器中：

> 下载并引入axios.min.js文件，辅助下载并引入qs.min.js

> 调用axios函数，发起异步请求

* get请求:

axios.get("url",{

params:{

请求参数1:值,

…:…

}

}).then(function(res){

//res 不是直接的响应结果

//res.data才是!

})

await和then选其一即可

* post:

var res =

await axios.post("url",Qs.stringify({

请求参数1:值,

…:…

}))

//.then()

await和then选其一即可

## 三、组件 ★

### 3.1 组件概述

定义：组件(Component)，指拥有专属的HTML，CSS和数据的，可重用的页面独立区域。

为什么：重用！分工协作!松耦合！

何时：只要发现独立的区域，都要封装为组件；今后的一个HTML页面，一定是由多个人一起完成的

### 3.2 使用组件

#### 3.2.1 创建组件

Vue.component("组件名",{

template:"选择器", //代替了new Vue中的el

data:function(){ //代替了new Vue中的data:{模型变量}

// 每次都要调用函数，为本次组件创建专属的数据模型

return:{ //{}--->动词，创建新对象

模型变量:值,

…:…

}

},

// 其余同new Vue()

// … methods:{}

})

问题：组件专属的HTML放在哪？

解决：HTML5中，采用<template>HTML片段</template>

template是专门为程序临时保存一段隐藏HTML片段的元素

何时：专门用于为组件保存HTML片段

规定：template中只能且必须有唯一的一个父元素包裹组件的片段

如何：

> 定义<template>元素包裹组件的HTML片段，还需要添加一个唯一的父元素

> Vue.component

#### 3.2.2 在视图中使用组件

一个组件其实就是一个可重用的自定义HTML标准

<组件名></组件名>

执行时，会用组件的template替换<组件名></组件名>

### 3.3 Vue实例/组件的生命周期

#### 3.3.1 生命周期概述

##### 【笔试题】Vue实例/组件的生命周期共几个阶段，每个阶段做什么。

* 创建组件对象和模型数据对象

new Vue( {data:{} })

Vue.component("xxx",{ data:function(){ return { …} }})

* 挂载阶段

扫描真实DOM树，创建虚拟DOM树，并绑定模型中的数据到视图中

new Vue( {data:{},el:"#app"})

Vue.component("xxx",{

data:function(){

return {…}

},

template:"#template"

})

* 更新阶段：

只要模型数据发生变化，就更新组件的内容和DOM树

* 销毁阶段：

主动调用了$destroy()函数，销毁组件时。

#### 3.3.2 生命周期钩子函数

定义：在组件的每个阶段都会自动执行的函数

何时：如果希望在组件的不同阶段执行一项任务时

包括：

beforeCreate() $el:undefined data:undefined

created() $el:undefined data:{…}

beforeMount() $el:空DOM对象 data:{…}

mounted() $el:完整DOM树 data:{…}

beforeUpdate() $el:

updated()

beforeDestroy()

destroyed()

**!注意：**请求数据(axios.get())可放在created或mounted中均可；但是，如果有DOM操作，就必须放在mounted中

### 3.4 组件化开发 ★

将来拿到网页，先划分组件。

组件化开发：将大的网页划分为若干组件区域，每个组件区域都有专门的HTML，CSS和数据。

#### 3.4.1 自定义组件

自定义组件一共分为以下3种：

* 根组件：

new Vue({

el:"父元素选择器",

data:{模型变量}

})

* 全局组件：

Vue.component("组件名",{

template:"template元素的选择器",

data:function(){

return { 模型变量}

}

})

* 局部组件(子组件)：只能用于特定的父组件内

何时：只要规定一个组件只能出现在指定的父组件内

如何：将全局组件降级

var 子组件对象 = {

template:"template元素的选择器",

data:function(){

return { 模型变量}

}

}

在父组件中，添加components属性，包含子组件对象，并为子组件起名！

var 父组件 = {

… …

components:{

"子组件名":子组件对象

}

}

#### 3.4.2 组件间通信 ★

分为三大类：父->子、子-> 父、兄弟间传参

##### 3.4.2.1 属性下行——父->子

* 子：定义props属性(可被外部访问，并赋值的特殊模型数据)

> 在组件内的用法和data中模型变量完全一样!

> 可被其他组件发现并赋值

var 子组件 = {

… …

props:["模型变量"…]

… …

}

* 父组件的template中：<子组件 :子组件props属性="父组件模型变量">

【结果】父组件将自己的模型变量赋值给子组件内的props属性变量。

ex：

<template id="tplTodoList">

<div>

<ul>

<li v-for="(task,i) in tasks">

<xz-todo-item :task="task" :i="i"></xz-todo-item>

</li>

</ul>

</div>

</template>

<template id="tplTodoItem">

<div>

<span>{{i+1}}</span> - <span>{{task}}</span> <button>×</button>

</div>

</template>

<script>

*var* xzTodoItem = {

template:"#tplTodoItem",

props:["i","task"]

}

*var* xzTodoList = {

template:"#tplTodoList",

props:["tasks"], //data:{ tasks }

components:{

// Vue框架可将：驼峰 --> 横线分割

xzTodoItem //"xz-todo-item":xzTodoItem

}

}

##### 3.4.2.2 事件上行——子->父

* 父组件中：

<template>

<子组件 @自定义事件名="处理函数~~(参数)~~">

<!-- 当事件处理函数没有参数时可省略() -->

</template>

var 父组件 = {

methods:{

处理函数(参数){ //此处的参数是this.$emit传过来的

// 根据参数修改this.模型变量

}

}

}

* 子组件任意位置都可触发父组件中绑定的自定义事件

data:{"task":""}

click(){

this.$emit("自定义事件名",参数值)

}

会触发父组件template中，提前绑定的@自定义事件，并将参数传递给父组件中的处理函数。

##### 3.4.2.3 兄弟间传参

采用先给父组件，再由父组件绑定到另一个子组件。

#### 3.4.3 SPA应用

SPA:Single Page Application 单页面应用

什么是：整个应用程序只有一个完整的HTML文件，所谓的“页面”，其实只是不同组件片段而已。在浏览器中时，按需加载组件片段内容，替换HTML文件中指定位置。

##### 3.4.3.1 多页面应用 vs 单页面应用

* 请求次数：

> 多页面应用，会造成反复请求；

> 单页面应用，之请求一次，就拿到所有的内容

* 页面跳转的本质：

> 多页面应用，删除整棵DOM树重建

> 单页面应用，只替换DOM树中局部节点对象即可

* 共用资源：

> 多页面应用，多个页面共用的资源，也需要反复请求下载

> 单页面应用，只在首次加载HTML文件时，请求下载一次

* 页面切换动画：

> 多页面应用，不可能实现页面切换动画

> 单页面应用，很容易实现页面切换动画(页面都在本地，且DOM树是不释放的)

问题：首屏加载慢！

解决：将来Vue项目中可设置“懒加载”——按需加载

##### 3.4.3.2 单页面应用实现的原理

* 定义多个template
* 定义一个唯一的完整的HTML文件

> HTML中要有一个特殊的元素占位，表示将来要被不同template替换的位置

* 定义路由字典

> 路由：一个请求地址和一个组件名的对应关系

> 路由字典：包含一组路由键值对的集合

var routes = {

{#index,tplIndex}

{#products,tplProducts}

}

* 根据当前url中相对路径或请求参数的不同，用不同的template代替占位符的位置

> 获得相对路径或请求参数

> 遍历路由列表，找到对应的template，并读取其中内容

> 用template内容代替占位符的位置

#### 3.4.4 Vue-router组件

Vue-router：是Vue中专门实现单页面应用的组件

##### 3.4.4.1 使用Vue-router组件

Vue核心功能中不包含Vue-router，需要下载后才能使用

* 下载并引入vue-router.js

<script src="js/vue.js"></script>

<script src="js/vue-router.js"></script>

* 将页面区域都定义在template中
* <div id="#app">中添加占位元素<router-view></router-view>

<router-view></router-view>表示将来要被替换的区域

* 所有template，都要创建对应的组件对象

共用的template，创建全局组件

替换的router-view的组件，创建局部子组件

* 建立路由字典：

var routes = [

{path:"/",component:默认组件}

{path:"/index",component:组件对象},

… …

{path:"/\*",component:xzNotFound组件}

]

* 创建路由器对象

var router = new VueRouter({

routes //routes:"routes"

})

* 创建Vue实例，并载入路由器对象

new Vue({

el:"选择器",

router

})

##### 3.4.4.2 路由跳转

* 在HTML中写死：

<router-link to="/相对路径">文本</router-link>

【原理】router-link用法和a几乎完全一样，因为router-link运行时会被自动翻译为a标记

**!注意：**路由字典中path中写的是什么上述中的to就是什么；to也可以进行属性绑定

* 在Vue中程序跳转

this.$router.push("/相对路径");

##### 3.4.4.3 路由传参

**>** 在接收参数的组件中：

props:["参数变量"]

**>** 配置路由地址：

{path:"/相对路径/:变量名",component:组件对象,props:true,children:[

{path:"/相对路径",component:组件对象},

… …

]}

**>** 跳转时：地址——>/相对路径/参数值

**!注意：**此时的参数值不加":"

**!注意：**一旦给路由地址给了/:变量名(参数名)，不传参则会报错

问题：当路由传递两个及以上的参数时，如"/login/index/tmgj"后面的"/index/tmgj"为一个整的参数，但第二个"/"会使Vue错认为传递了两个参数，因此会找不到对应的路由地址，弹出"404 Not Found"页面。

解决：使用encodeURIComponent()和decodeURIComponent()将路由中的参数进行转化。

##### 【笔试题】this.$router vs this.$route

$router 代表路由器，专门执行跳转任务

$route 封装当前路径信息的对象

##### 3.4.4.4 嵌套路由

如果一个router-view内的部分内容，也需要根据二级路径的不同，动态替换时，就用嵌套路由。

如何：

> 定义模板，封装子组件对象

> 在父组件中添加<router-view>

> 在路由字典中：

{path:"/上级路径",component:父组件对象,children:[

{path:"/上级路径/二级路径",component:子组件对象},

… …

]}

> 跳转：/上级路径/二级路径,其中

/上级路径决定根组件中router-view的内容

/二级路径决定父组件中router-view的内容

## 四、脚手架

### 4.1 脚手架概述

定义：具有核心功能的半成品代码

我们只要向其中添加个性化自定义内容即可

作用：简化和标准化项目结构

今后只要开发Vue项目，都是用脚手架生成

### 4.2 使用脚手架

* 下载Vue-CLI脚手架命令行工具

> Vue-CLI专门用于通过命令生成脚手架代码

> 先安装命令行工具：npm i -g @vue/cli

* 用命令行工具创建项目代码

> vue create 项目名

> Manually select features手动选择功能

- 选Babel:专门自动将ES6代码转换为ES5的代码

比如在ES6中抛出模块用export default{…}

- 选Router：Vue-Router组件

- 选Vuex：客户端状态管理——下一个老师讲

**!注：**以上选项，按上下键切换选项，空格决定是否选中，回去确定选择

> Use history mode for router 选Y 采用不带#的路由地址

> 选 In package.json 将所有子工具的配置，都集中放在一个配置文件中(package.json)

> Save this as a preset for future projects?

是否将当前项目配置保存为默认项目配置 选N

> Use <http://registry.npm.taobao.org> for faster installation?

是否使用淘宝的镜像NPM库进行更快的安装？选NO

* 运行测试项目

> cd 项目文件夹

> npm run serve 启动临时开发服务器，编译并宿主Vue生成后的文件

* 安装必要组件：axios,bootstrap,jquery

> npm i -save axios

> npm i -save querystring

### 4.3 脚手架项目结构

#### 4.3.1 public

专门保存不需要编译的项目静态资源的文件夹

比如可赋值学子商城项目的img文件夹到public下

还保存作者唯一的静态页面:index.html

|public

|css

|bootstrap.css

|base.css

|img

|js

|jquery3.2.1.js

|bootstrap.js

|qs.min.js

#### 4.3.2 src

包含项目所有要编译的源代码文件

assets:需要编译打包的静态资源，比如共用的css和共用的js

views:所有直接出现在路由字典中的组件

components:所有全局组件

app.vue:整个项目中唯一根组件

main.js：是根组件app.vue的主程序

router.js:整个项目的路由器和路由字典

store.js：VUEX的组件(现在不用)

### 4.4 每个组件的内容

每个组件都是一个.vue文件

包含三部分内容：

<template>

<div>

<!--原template中的HTML片段-->

<h3>This is an Header page</h3>

</div>

</template>

<script>

export default { //创建一个子组件对象，相当于 var header={...}

// 原组件对象中的属性:data,mounted,

}

</script>

<style scoped> <!--scoped表示<style>里的样式仅归当前组件使用-->

<!--当前组件自用的css-->

</style>

### 4.5 axios的使用

在main.js中引入axios模块，并将axios添加到Vue类型的原型对象中

//ES6引入模块

import axios from 'axios'

//相当于const axios = require("axios");

Vue.prototype.axios=axios;

问题：axios请求默认不带cookie！导致无法使用服务器端session

解决：配置axios，让其携带cookie

axios.default.withCredentials = true;

问题：如果服务器端采用CORS跨域，默认无法接受验证消息

解决：

> 服务器端：安装cors模块 npm i -save cors

express主模块app.js中，配置cors中间件，允许接受身份认证信息。

> 客户端：

app.use(cors({

origin:'http://localhost:8080',

credentials:true

}))

### 4.6 调用axios发送请求

在任意.vue中:

this.axios.get/post( ... )

# 第十五章 VUE UI

程涛 taonetwork2018

## 一、VUE UI概述

### 1.1 阶段课程概述

vue移动端UI组件库/项目 [mint-ui/mui] 6天(重点)

HTML5新特性(游戏+特效)/项目 7天(重点)

微信公众号+微信小程序 6天

混编(phonegap) 1天

### 1.2 VUE UI概述

vue是当下很火的前端框架，但是有一定问题：

问题：vue主要针对pc用户，vue对移动端用户支持不是很友好。

解决：针对移动端用户，使用第三方组件开发

> 饿了么：基于vue框架移动UI(User Interface)组件 Mint-UI

网址：<https://mint-ui.github.io/#!/zh-cn>

> MUI：开源团队

> 饿了么：Element UI 针对PC端

网址：<http://element.eleme.io/#/zh-CN>

### 1.3 服务器端知识点

* 服务器种类

> node.js 支持高校高并发服务器

> Apache 稳定服务器(php)

> nginx 高并发服务器

**!注意：**以上三种服务器是相互补充的关系，并非替代关系。

* 复习数据库

存储大量数据

> 关系型数据库：MySQL(免费；一秒25个并发量);Oracle;SQL Server…

> 非关系型数据库：Redis(并发量高：一秒2W个并发量)

### 1.4 Vue-cli脚手架

#### 1.4.1 脚手架作用

vue-cli 脚手架创建vue项目的作用：

> 自动创建项目目录结构

> 本地调试

> 热加载

> 代码部署

#### [1.4.2 安装和配置](#_4.2_使用脚手架)

* 安装脚手架工具

> npm i vue-cli -g #全局环境下安装脚手架

i #参数安全

g #全局安装，可在命令行输出指令

**!注意：**需要预先安装基础软件node.js

* 初始化项目(创建项目目录；安装项目依赖第三方模块)

> vue init <template> <app-name>

init #初始化

template #模板名称(webpack)

app-name #项目名称

ex:vue init webpack xz

* 回答问题完成项目配置

> 项目名称 Project name(xz-app) 回车

> 安装vue-router n

> 安装ES-lint 严格模式检查工具 n

> 以下安装检测工具

* 安装成功启动项目

> cd xz-app

> npm run dev

* 浏览器直接访问

> <http://localhost:3001/#/>

#### 1.4.3 项目目录结构

xz-app-00

node\_modules 保存第三方模块

package.json 启动项目指令

src 项目中所有代码

assets 静态图片

components 项目中自己开发组件保存在此处理

lib 移动端组件mui

main.js 入口文件

router.js 路由模块

App.vue 根组件

#### 1.4.4 简单开发流程

* 在main.js加载使用模块；加载路由；加载ajax
* 在components下的自定义路径中创建自定义组件

<template>

<div class="app-home">

// 组件的HTML片段

</div>

</template>

<script>

export default {

data(){ return {

// 模型数据

}}

}

</script>

<style>

// 组件的CSS样式

</style>

* 在router.js配置组件(路由；组件名)

> 引入组件

import 组件对象名称 from "./components/Test.vue"

> 配置组件访问

{path:"/路径",component:组件对象名称}

【示例】[testContainer.vue](../VUE%20UI/day01/vue_app_00/vue_app_00/src/components/TestContainer.vue)

【练习】创建组件 listContainer.vue 循环创建用户信息列表

编号;名称;年龄

## 二、Mint-UI组件库

### 2.1 Mint-UI安装与与配置

> 学习环境安装

下载配置js/css/font

> 生产环境(脚手架)

npm i mint-ui

### 2.2 Mint-UI组件分类

> JS Component js组件

> CSS Component css组件

> Form Component form组件

### 2.2 Mint-UI使用

下载安装基本组件：

> mint-ui css 文件

> mint-ui js 文件

> vue.js 文件

**!注意：**下载完后在main.js文件中引入，加载文件有顺序，先加载 vue.js 再加 index.js

#### 2.2.1 按需引入组件

/\*main.js文件\*/

// 按需引入指定组件

import {Header,Swipe,SwipeItem,Button,Lazyload} from "mint-ui";

// 注册到当前项目中

Vue.component(Header.name,Header);

Vue.component(Swipe.name,Swipe);

Vue.component(SwipeItem.name,SwipeItem);

Vue.component(Button.name,Button);

Vue.use(Lazyload);

#### 2.2.2 引入完整Mint-UI组件库

/\*main.js文件\*/

// 完整引入mint-ui

import MintUI from "mint-ui";

Vue.use(MintUI);

### 2.3 Mint-UI组件

#### 2.3.1 消息提示框 Toast

简短消息提示框(alert("\*"))支持自定位置

##### 2.3.1.1 Vue中的用法

* + 如何使用

this.$toast("提示消息");

* + 常用参数

|  |  |
| --- | --- |
| 参数名 | 解释 |
| message | 提示性文字 |
| postion | top:middle;bottom |
| duration | 持续时间(毫秒)，-1不会自动消失 |
| className | 添加样式 |
| iconClass | 图标的类名 |
| close() | 关闭提示框 |

##### 2.3.1.2 Mint-UI中的用法

* 引入：

import {Toast} from "mint-ui"

* 使用：

Toast("评论内容不能为空")

#### 2.3.2 懒加载 Lazyload

在开发中，当程序中需要利用的资源时，在程序启动的时候不加载，只有在运行当需要这些资源的时候再去加载这些资源。

<div id="app">

<ul>

<li><img lazy="loading"></li>

</ul>

</div>

/\*设置属性选择器\*/

#app ul img[lazy=loading]{

width: 100%;

height: 100px; /\*显示加载图片，高度100px\*/

margin: auto;

}

#### 2.3.3 轮播图 Swipe

* 引组件

import {Swipe,SwipeItem} from "mint-ui";

* 注册

Vue.component(Swipe.name,Swipe);

Vue.component(SwipeItem.name,SwipeItem);

* 在自己组件使用 mintui 调用图片轮播

把自己组件中的template内容里加上mintui中图片轮播的HTML片段

<mt-swipe :auto="4000" :show-indicatory="true">

<mt-swipe-item>1</mt-swipe-item>

<mt-swipe-item>2</mt-swipe-item>

</mt-swipe>

#### 2.3.4 导航条

##### 2.3.4.1 顶部导航

* 在main.js中引入所需组件

import {Header} from "mint-ui";

* 在main.js中注册引入的组件

Vue.component(Header.name,Header);

**!注意：**今后所有的此类操作都应在Vue实例化之前执行

* 在自定义组件中使用mintui组件

<mt-header fixed title="学子商城"></mt-header>

##### 2.3.4.2 底部分页导航

**!注意：**此后所有操作若无说明，前两步操作同上，仅贴出该组件必要的HTML片段

<nav class="mui-bar mui-bar-tab">

<a class="mui-tab-item mui-active" href="#tabbar">

<span class="mui-icon mui-icon-home"></span>

<span class="mui-tab-label">首页</span>

</a>

<a>…</a>

</nav>

##### 2.3.4.3 将导航应用到每个页面

修改根组件App.vue 顶部导航栏和底部导航栏 tarbar mui



<template>

<div class="app-container">

<!-- 顶部导航条 -->

<mt-header fixed title="学子商城"></mt-header>

<!-- 中间替换的内容组件 -->

<router-view></router-view>

<!-- 底部导航栏 -->

<nav class="mui-bar mui-bar-tab">

<a class="mui-tab-item mui-active" href="#tabbar">

<span class="mui-icon mui-icon-home"></span>

<span class="mui-tab-label">首页</span></a>

<a>…</a></nav></div>

</template>

#### 2.3.5 按钮 Button

* 在main.js中引入所需组件

import {Header,Button} from "mint-ui";

* 在main.js中注册引入的组件

Vue.component(Button.name,Button);

**!注意：**今后所有的此类操作都应在Vue实例化之前执行

* 在自定义组件中使用mintui组件

<mt-button type="primary" size="large">按钮1</mt-button>

## 三、MUI组件库

MUI:最接近原生APP移动端UI组件库

### 3.1 MUI安装与配置

下载 js/css/fonts 保存在src/lib下

在main.js中：

import "./lib/mui/css/mui.css";

import "./lib/mui/css/icons-extra.css";

### 3.2 MUI使用

#### 3.2.1 六宫格

在自定义组件<template></template>的所需位置处写入以下MUI的六宫格HTML片段即可完成六宫格布局：

<!-- 六宫格 -->

<ul class="mui-table-view mui-grid-view mui-grid-9">

<li class="mui-table-view-cell mui-media mui-col-xs-4 mui-col-sm-3">

<router-link to="/home/newslist">

<img src="../../img/menu1.png" alt="">

<div class="mui-media-body">新闻资讯</div>

</router-link>

</li>

<li>…</li>

<li>…</li>

</ul>

#### 3.2.2 数字输入框 NumberBox

* 额外再引入mui.js，增加动态操作

import mui from "../../lib/mui/js/mui.js"

* 复制以下HTML片段到<template>标签中的所需位置

<!--限定最小值和最大值(1~9)-->

<div class="mui-numbox" data-numbox-min='1' data-numbox-max='9'>

<button class="mui-btn mui-btn-numbox-minus" type="button">- </button>

<input id="test" class="mui-input-numbox" type="number" value="5" />

<button class="mui-btn mui-btn-numbox-plus" type="button">+</button>

</div>

问题：当引入mui.js出错误

Uncaught TypeError: 'caller', 'callee', and 'arguments' properties may not be accessed on strict mode functions

解决：

> 安装第三方模块禁止脚手架严格模式

babel-plugin-transform-remove-strict-mode

> 不用mui.js文件,自己写触发事件

#### 3.2.3 卡片视图 Card

* 复制以下HTML片段到<template>标签中的所需位置

<div class="mui-card">

<div class="mui-card-header">页眉</div>

<div class="mui-card-content">

<div class="mui-card-content-inner">

包含页眉页脚的卡片，页眉常用来显示面板标题，页脚用来显示额外信 息或支持的操作（比如点赞、评论等）

</div>

</div>

<div class="mui-card-footer">页脚</div>

</div>

## 四、Element UI

### 3.1 Element UI安装与配置

> 加载elementUI样式文件

> 加载vue.js

> 加载element.js

**!注意：**加载上述后两个文件有顺序，先加载vue.js，后加载element.js

### 3.2 图标显示问题

element UI中的图标依赖于字体图标文件，如需使用图标需下载并引入字体图标文件：

> 将下载好的font文件夹放入css文件夹下，与e.css同级

> font文件夹中一般是以下两种文件：  
 element-icons.ttf/element-icons.woff

### 3.3 定制样式

创建class添加元素，使定制样式的选择器优先级高即可

通常组件都是less；sass动态样式编写内容

# 阶段项目：学子商城移动端(Vue版本)

## 一、首页HomeContainer.vue

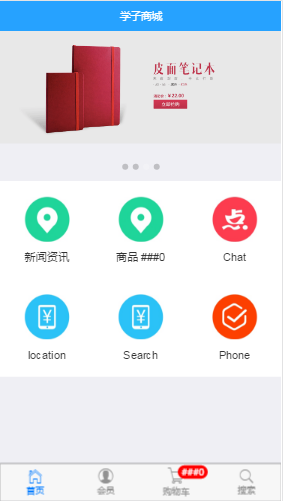
### 1.1 创建空组件

路径：src/components/tabbar/HomeContainer.vue

创建空组件

配置访问路径 /home

### 1.2 界面组件引入



从上图可见，首页需要Mint-UI的顶部和底部导航条，轮播图Swiper；MUI的六宫格组件。

参照第十五章中[2.3节](#_2.3_Mint-UI组件)及[3.2节](#_3.2_MUI使用)内容进行各组件的引入，完成界面的创建。

### 1.3 轮播图片获取及组件封装

**!注意：**图片保存在服务器目录下，img图片不能保存在数据库，数据可以保存图片“路径”。所以需要从脚手架端获取服务器端的图片地址等相关数据。

#### 1.3.1 使用vue-resource获取图片列表

##### 1.3.1.1获取服务器端数据的方法

* 常用的获取服务器端数据的方法有以下两种：

> 新方法：axios

> 旧方法：vue-resource

**!注意：**以上两种方法之间不是互斥关系，两者均可使用。

##### 1.3.1.2 下载及配置

* 将vue-resource 安装脚手架

npm i vue-resource -D

* 引入当前项目-main.js

import VueResource from "vue-resource"

* 注册-main.js

Vue.use(VueResource)

* 客户端使用:当前vue实例对象添加属性 $http.get()/.post()

this.$http.get(url).then(result=>{})

问题：- No 'Access-Control-Allow-Origin'

解决：跨域访问->当前两个程序相互访问时:域名不同;端口不同;协议不同就属于跨域访问，ajax默认禁止跨域访问。

* 描述:脚手架 HomeContainer.vue->ajax->nodejs /imagelist

3001 3000

* 如何解决现在跨域问题

> 修改node.js 服务器允许脚手架3001端口所有访问

**-**安装第三方组件 cors node.js/node\_modules

**-** app.js 配置

加载跨域访问模块

const cors = require("cors")

配置允许哪个程序哪个端口访问

app.use(cors({

origin: ["http://127.0.0.1:3001","http://localhost:3001"],

credentials:true

}))

#### 1.3.2 组件封装

若在一个项目中要重用同一个组件很多次，又不想每次都贴一大段HTML片段，可将这个组件进行自定义封装，之后导出以供重用。

* 封装子组件(以轮播图swiper为例)，创建子组件如下：

<template>

<div class="app-swiper">

<h3>图片轮播</h3>

<mt-swipe :auto="2000">

<mt-swipe-item v-for="item in list" :key="item.id">

<img :src="item.img\_url">

</mt-swipe-item>

</mt-swipe>

</div>

</template>

<script>

export default {

data(){ return { }},

props:["list"] // 子组件接收父组件数据

}

</script>

<style>

.app-swiper .mint-swipe{

height: 150px;

}

.app-swiper .mint-swipe img{

width: 100%;

}

</style>

router.js中引入上述子组件

import swiper from "./components/sub/swiper.vue"

在需要使用该子组件的组件中将其注册

export default {

data(){return {}

},

components:{

"swiper-box":swiper

}

}

在其他组件的<template></template>使用

<!--调用子组件 -->

<swiper-box :list="imglist"></swiper-box>

### 1.4 启动显示优化

问题：当脚手架项目启动不会自动显示Home组件

解决：将router.js里配置的以下路径

{path:"/",component:"HelloContainer.vue"}-->{path:"/",redirect:"/home"}其中redirect中填写想要首屏显示的组件的对应路径。

## 二、新闻页NewslistContainer.vue/NewInfoContainer.vue

### 2.1 新闻列表页

#### 2.1.1 组件创建

> 创建空组件 news/newsListContainer.vue

> 配置组件访问路径 /home/newslist

> mui media-list.html

> 数据表 新闻表

问题：Error: ER\_SP\_UNDECLARED\_VAR: Undeclared variable: NaN

sql语法执行时出错误有一个参数NaN

解决:没有正确发送参数pno pageSize

【课后作业】

1)在新闻列表中添加按钮 mint-ui button

2)在新闻列表最底顶添按钮并且点击事件

#### 2.1.2 过滤器格式化日期

使用过滤器对原始数据进行改写后返回并显示。

* 在main.js 中创建过滤器

Vue.filter("过滤器名称",function(val){ val | 前日期

return "";

})

* 组件使用{{item.ctime | 过滤器名称}}
* 过滤器代码：yyyy-mm-dd h:m:s

Vue.filter("dataFilter",function(val){

var data = new Date(val);

var y = data.getFullYear();

var m = data.getMonth()+1;

var d = data.getDate();

var h = data.getHours();

var min = data.getMinutes();

var s = data.getSeconds();

m<10&&(m="0"+m); //短路逻辑

d<10&&(d="0"+d);

h<10&&(h="0"+h);

min<10&&(min="0"+min);

s<10&&(s="0"+s);

return `${y}-${m}-${d} ${h}:${min}:${s}`;

})

#### 2.1.3 加载更多 ★

* 原理：点击加载更多按钮后，向服务器请求下一页数据(原理同分页)，并使用concat拼接在接收数据的数组后，自动刷新显示。
* 使用：下载配置使用组件

> npm i mint-ui -D 下载vue\_app\_00 己经存在

> main.js文件

import {Button} from "mint-ui"

Vue.component(Button.name,Button);

> 自定义使用

<mt-button type="primary" size="large">加载更多</mt-button>

size属性使按钮宽度占满父元素宽度，相当于width=100%

> 加载更多含义原先新闻保留新内容添加原有新闻后面

data(){return list:[]} concat

新数组=第一页数组[].concat(第二页数组)

> 如果己经是最后一页不要再送ajax 请求,或者提示用户己经是最后一页

#ajax 发请求查询数据库

添加属性 hasMore:true 是否有下一页数据

if(!hasMore)retrun;

### 2.2 新闻详情页

#### 2.2.1 组件创建

操作流程当用户点击新闻列表中某一条新闻，然后按要求跳转，在此组件中显示详细内容，包含评论。

> 创建空组件 newInfoContainer.vue

> 访问路径 /home/newInfo

> 参数传递 newList.vue 参数nid newInfo.vue 接收

> 获取数据创建新闻详细

#### 2.2.2 跳转参数传递

* 标签式参数传递

**>** 传递参数

<router-link :to="'组件名称?id='+item.id">

新闻 1

<router-link>

**>** 获取参数

this.$route.query.id; (接收参数)

this.$route.query.age; 在created()

**!注意:**

# 在vue 接收参数统一使用 this.$route

# 编译跳到某个组件 this.$router

【练习】HomeContainer.vue 点击 “新闻资讯”按钮跳转 NewsList.vue

* 编程式跳转参数传递

**>** 传递参数

this.$router.push("组件名称");

this.$router.push("组件名称?id="+id)

this.$router.push("组件名称/19")

**!注意：**使用/传递参数，需要额外做一些处理

# 在组件的<script>标签中进行带参跳转：this.$router.push("/home/shop/19")

# 在router.js的指定路由中绑定要传递的参数：{path:"/home/shop/:id"...}

**>** 获取参数

this.$route.params.id

**!注意：**只有"/19"方式传参获取时要使用上述方法获取参数，其余均用query。

【示例】this.$router.push("/home/newlist")

【练习】用编程式跳转编写首页中的商品页跳转，主要步骤如下：

> 创建空组件 ShopContainer.vue 商品组件

> 位置 /tabbar/ShopContainer.vue

> 访问 /home/shop

#### 2.2.3 跳转传参小结

* 跳转组件二种方式:标签;编程

> 编程更为灵活

* 参数:查询字符串?id=10 参数 /19

> 第一种常见查询字符中

> 第二种参数稍少一些

* @click.stop.prevent="jumpShop()"

> stop 停止事件冒泡

> prevent 停止事件默认行为 a;submit;button

### 2.3 新闻评论子组件

#### 2.3.1 新闻评论子组件概述

新闻详情页中主要包含：新闻标签、新闻内容、评论(评论列表;发表评论)

在新闻详细组件发送ajax请求获取新闻信息并且显示指定位置

问题：什么时候发送ajax显示数据

解决：当组件创建成功发送请求，即created()

#### 2.3.2 子组件创建

**!注意：**子组件是其它组件中组成部分，子组件不用配置路由地址

* 创建空子组件(评论子组件)

> 路径：components/sub/comment.vue

* 在新闻详情组件加载，评论子组件

> 引入子组件

import comment from "../sub/comment.vue"

> 注册子组件并且给组件起一个标签名

components:{

"comment-box": comment

}

> 在模块使用标签名调用子组件

<comment-box></comment-box>

#### 2.3.3 父子组件间传参

* 将新闻编号传递子组件(父组件传递参数子组件)

> 父组件: newInfoContainer.vue

> 子组件: comment.vue

* 父组件传递

> 父组件传递参数子组件,参数名称id

<comment-box :id="this.id"></comment-box>

> 子组件接收

props:["id"]

this.id 父组件传递数据 {{id}}

【课后练习】创建评论子组件

> 发表评论

> 评论列表{加载更多}

> 创建数据表创建服务器程序

#### 2.3.4 评论页功能编写

##### 2.3.4.1 发表评论

需求：允许用户对某一条新闻发表评论，成功提示“评论成功”，在评论列表立即在第一楼显示刚发表的内容。

* 服务器端：node.js，接收发表内容，添加数据库中并且提示。

**>** 服务器端发送的请求方式一定是post，请求路径/postComment

**>** 用户参数msg

**>** sql

# 添加数据；追加数据；插入数据INSERT

INSERT INTO xz\_comment VALUES(null,msg,'匿名',now(),nid);

# 添加sql添加哪列写哪列

INSERT INTO xz\_comment(content,user\_name,ctime,nid) VAULES(?,'匿名',now(),?)

**>** json{code:1,msg:"添加成功"}

**!注意：**再次区分POST和GET请求

|  |  |  |
| --- | --- | --- |
| 网络请求 | POST | GET |
| 传递数据量 | 没限制 | ?msg=11 1kb(几十个汉字) |
| 语义 | 发送服务器数据 | 获取服务器数据 |
| 安全 | 不安全 | 不安全 |

* 脚手架：vue-cli，用户点击发表评论按钮，先检查用户评论是否为空。

若不为空，则发送ajax请求将用户评论内容发送到node.js服务器端。

* 提示错误信息

> main.js 引入组件(项目大多组件都要此功能)

> 当前组件直接引入

- import {Toast} from "mint-ui"

- Toast("评论内容不能为空")

##### 2.3.4.2 显示评论列表

* 服务器编写路由 GET /getComment nid
* 脚手架发送ajax 请求
* 遇到问题发送AJAX请求路径长

/\*main.js \*/

Vue.http.options.root = "http://127.0.0.1:3000/";

Vue.http.options.emulateJSON = true;

#VueResource 配置选项指定服务器地址

#emulateJSON 发送 post修改请求头信息

#发送内容格格改为 application/x-www-form-urlencoded

/\*comment.vue\*/

this.$http.get("/comment");

#### 2.3.5 常见错误

* 服务器报错 NaN 参数pno=undefined parseInt(pno) NaN

参数错误:F12->Console->Network

* 中文处理

> main.js Vue.http.options.emulateJSON = true;

> bodyParse

> req.body.msg

问题：添加评论后直接显示在评论列表，显示为第一行

解决：点击发表按钮之后自动触发事件进行界面重绘

> 服务器

SELECT id,content,ctime FROM xz\_comment

WHERE nid = 1

ORDER BY id DESC

LIMIT 0,5;

> 发表评论 脚手架

> 立即查询新评论内容 排序 1页

## 三、顶部导航和底部导航栏

详细见第十五章[2.3.4章节](#_2.3.4_导航条)

## 四、商品页

goods/GoodList.vue 商品列表

path:"/home/goodslist"

goods/GoodInfo.vue 商品详情

【练习】从HOME组件点击商品按钮跳转goodlist.vue

【课后作业】商品列表 发送ajax请求获取一组数据 5商品信息

图片、标签、价格 旧 新、剩 3

【课后作业】商品详细

### 4.1 商品列表页 GoodListContainer.vue

* 依据 list列表创建创建模板中商品
* 点击商品图片跳转GoodInfo.vue 传递参数

商品编号(建议采用编程跳转)

### 4.2 商品详情页 GoodInfoContainer.vue

#### 4.2.1 步骤

* 商品详细--图片轮播

服务器:node.js /imagelist

脚手架:mintui 图片轮播

(建议Home组件图片轮播参数)

#### 4.2.2 加减按钮组件

### 4.3 购物车

#### 4.3.1 购物车组件创建

组件名称：tabbar/ShopContainer.vue

组件访问路径：/shop

#### 4.3.2 组件界面实现

点击tabbar 购物车显示购物车组件

轮播图、商品列表

卡片组件：mui number-box 小计

* 卡片布局方式 mui/card.html
* 图片轮播创建子组件

> sub/swiper.vue 图片轮播子组件

> goodsInfo.vue 父组件

> <swiper-box :list="..."></swiper-box>

## 五、补充知识点及错误点

### 5.1 像素

像素二种:物理像素 逻辑像素

设计师:创建效果图 ps 物理像素

程序: 代码 逻辑像素

比值: iphone6 2 iphone6-plus 3

设计师图片物理像素 120px/2 = 60px

### 5.2 图片路径

* 图片路径：

> 绝对路径 <http://163/image/banner1.png>(大多数)

> 相对路径 ../../img/banner1.png(少用)

**!注意：**所有商品图片保存在服务器端

### 5.3 启动Vue项目命令

东哥 npm run serve 启动脚手架

程涛 npm run dev 启动脚手架

npm run dev 脚本名称与脚本名称对应指令 package.json

### 5.4 v-for报错问题

* vue 推荐项目 <v-for :key="item.id">

vue 循环对数据排序操作，提升效率使不重复属性

### 5.5 命令行常见报错

* 服务器失败停止工作

ERR\_CONNECTION\_REFUSED

* 端口冲突

Error: listen EADDRINUSE :::3000

解析:有二个程序都占用 3000端口

解决:vscode 点击“终止终端”

* 语法错误

ERROR Failed to compile with 1 errors

\* ../img in

解析:脚手架对程序要求非常严格

* 配置出错/ajax url地址出错

http://127.0.0.0.1:3001

* 模块引入出错

Error: Cannot find module 'express'

解析：没有找到指定模块 express

### 5.6 常见问题及解决

问题1：如何修改脚手架启动端口 3001

解决1：

/\*package.json文件\*/

"scripts": {

"dev": "vue-cli-service serve --port 3001 --host 127.0.0.1",

}

问题2：数据库乱码问题

解决2：

status; 查看数据库当前状态

//操作流程

USE xz;

status;

[Db characterset: utf8] #当前学子商城库utf8没问题

示例：

CREATE DATABASE xz CHARSET=UTF8

**!注意：**mysql数据库特性地方 编码名称不是UTF-8是UTF8

问题3：Unknown custom element: <swiper-box> - did you register the

解决3：没有注册 swiper-box 组件

问题4：网络控制面板没有发现请求信息

解决4：调用方:-->style 选择器错误

### 5.7 使用脚手架创建项目小结

* 最好使用现有脚手架
* node\_modules/package.json

> 全局安装脚手架指令 npm i vue-cli -g

> 初始化 vue init webpack xz\_app

# 第十六章 HTML5新特性

## 一、新的语义标签

header,footer,section…

## 二、增强型表单(表单2.0)

### 2.1 新的input type

<input type="?">

H4：text,password,button,checkbox,radio,submit,file,reset

H5：email,url,number,search,range,color,date,month,week

### 2.2 新的表单元素

H4：input,button,textarea,select/option

H5：datalist,progress,meter,output

#### 2.2.1 建议列表 datalist

* 作用：datalist本身不可见，但当用户在绑定了list的input框中输入数据时，既可以从datalist列表中选取一个数据，也可以手工输入
* 使用：

<datalist id="list1">

<option>xxxx</option>

<option>yyyy</option>

</datalist>

<input type="text" list="list1" />

#### 2.2.2 进度条 progress

显示一个进度条有两种形式

<!--左右晃动进度条-->

<progress></progress>

<!--具有指定进度之的进度条(0-1)-->

<progress value="0.7" />

#### 2.2.3 刻度尺 meter

此元素会随着当前值不同改变颜色[红；黄；绿]

如果当前值与最佳值非常接近[绿]，反之[红]

**!注意：**low和high是决定颜色变化的临界值

<meter min="最小值" max="最大值" low="下限" high="上限" optimum = "最佳值" value="当前值">

</meter>

#### 2.2.4 输出output

output：输出 语义标签，没有任何外观样式，样式同span

<h4>4.输出</h4>

月薪：100000

几个人： <input type="number" value="1" />

需要交五险一金费用：<output>45000</output>

### 2.3 新的表单属性

> placeholder 占位符

> autofocus 自动获取焦点

> multiple 允许输入多个值(用逗号分隔)

> form 用于把一个输入域放置在form外部，最后一起提交

<form id="f5">

</form>

<input type="text" form="f5"/>

以下属性和验证相关：

> required 必填项，内容不能为空

> minlength 最小字符串长度

> maxlength 最大字符串长度

> max 最大数值

> min 最小数值

> pattern 正则表达式

## 三、音频和视频

重点：主流应用，兼容性差

多媒体(音频，视频)数据在网页上非常吸引用户

Adobe Flash：音频；视频；动画；游戏；绘图；存储数据

Flash在一定范围内被H5取代

> Flash绘图(AS/Flex) -> Canvas + SVG

> Flash动画(游戏) -> Canvas+第三方游戏平台(微信小游戏)

> Flash视频和音频 -> video/auto(不是完全取代)

> Flash客户端存储 -> H5 WebStorage

### 3.1 视频 video

#### 3.1.1 video标签

H5提供一个新的标签用于播放视频

<video src="x.mp4"></video>

问题：视频有很多种不同格式 .mp4 .avi .flv .webm .ogg，浏览器播放指定视频需要安装特定解码器软件

解决：将视频使用工具：“格式工厂”

<video>

<source src="x.mp4" />

<source src="x.avi" />

<source src="x.flv" />

<source src="x.webm" />

您的浏览器版本太低，请升级喔~

</video>

**!注意：**mac电脑有一些chrome浏览器播放视频必须保存在服务器中

#### 3.1.2 video标签常用成员属性

> autoplay[false] 是否自动播放视频(只要写上就是true) <video autoplay>

> controls[false] 是否显示播放控件 时间长度 播放按钮

> loop[false] 是否循环播放

> muted[false] 是否静音播放

> poster:"" 在播放第一帧画布之前显示海报

> preload 预加载策略

- auto 预加载视频元数据和一定的缓冲时长

- metadata 预加载视频元数据

- none 不预加载任何数据

#### 3.1.3 video.js对象常用属性

volume:1 当前音量

playbackRate:1 回放速率 大于1快放 小于1慢播

paused:false/true 当前视频所处暂停状态

#### 3.1.4 video.js对象常用方法

play() 播放视频

pause() 暂停播放

#### 3.4.5 video.js对象常用事件

onplay() 当视频开始播放时触发的事件

onpause() 当视频暂停播放时触发的事件

【课后作业】创建用户注册表单，将视频作为表单背景元素

提示：video自动播放循环播放静态，绝对定位，z-index为负值

【练习】不使用video自带的controls自定义播放/暂停按钮，鼠标移出视频区按钮隐藏，鼠标移入视频区按钮显示。

代码见[HTML5/day02/01\_video.html](../HTML5/day02/01_video.html)

### 3.2 音频 audio

**!注意：**音频的标签属性、对象属性、对象方法、对象事件均同视频。

【练习】使用复选框控制网页背景音乐

【TIPS】[\*]是否播放背景音乐

cb.onchange = function(){

//判断this.checked == true;

}

【练习】在上述练习基础上，完成淡入播放效果。

【TIPS】采用定时器，每隔1s音量+0.1。

## 四、网页绘图技术

### 4.1 Canvas 绘图概述

Canvas(重点：特效，复杂统计图形，小游戏)

网页中实现绘制：走势图、统计图、网页游戏、应用都要使用绘制技术。主要有以下三种绘图技术：

> SVG绘制 2D矢量图(图片清晰，放大缩小不失真) 2000年 后纳入H5标准

> Canvas绘制 2D位图(图片清晰，放大会失真) H5标准

> WebGl绘制 3D位图 (常用框架three.js) 尚未纳入H5标准

### 4.2 Canvas 绘图

* 难点：坐标系；单词比较多。
* 步骤：

> 创建画布： <canvas id="c3" width="500" height="400"></canvas>

> 获取画布： var c3 = document.getElementById("c3");

> 获取画笔： var ctx = c3.getContext("2d");

> 查找位置： 默认原点(0,0)在界面左上角

> 绘图：见4.2.1及之后

**!注意：**高宽可以在标签属性里定义，也可以在JS中定义，但不能在CSS中定义(画布会变形)。

#### 4.2.1 矩形 rect

**!注意：**矩形定位点在自己的左上角；方法或属性名严格区分大小写

* 属性：

> ctx.lineWidth = 1; //描边宽度

> ctx.strokeStyle = "#fff"; //描边样式

> ctx.fillStyle = "#fff"; //填充样式

* 方法：

> ctx.strokeRect(x,y,w,h); //描边矩形

> ctx.fillRect(x,y,w,h); //填充矩形

> ctx.clearRect(x,y,w,h); //清除矩形范围内所有元素

**!注意：**填充矩形只能设置填充样式，同样描边矩形也只能设置描边样式。

【练习】在画布描边一个可以左右移动的100\*80的矩形

【TIPS】使用定时器，先清除画布上已有的内容，再重新绘制矩形(x不停增加)

【练习】矩形上下移动

【练习】矩形斜线移动(so funny~)

【练习】销售统计图

> 服务器端程序 app.js GET /sales

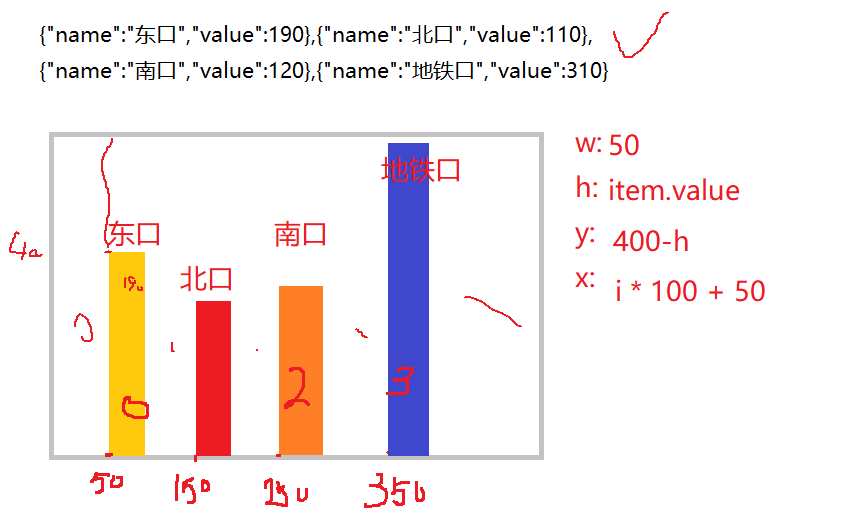
返回销售统计信息

[{name:"万寿路南口",value:120},

{name:"万寿路北口",value:110},

{name:"万寿路地铁口",value:190}]

> 获取数据绘制销售统计图(柱状) jquery



【课后作业】

作业1：药品公司销售统计图

作业2：依据数据在mysql创建对应表格

发发发药品公司

药品编号 药品名称 药品价格 一月 二月 三月..十二月

1001 立特灵 19.00 1000 900 300

1002 六味地黄丸 29.00 2000 1900 300..

1003 五味地黄丸 21.00 1100 111 11

作业3：依据结果创建统计图

#### 4.2.2 文本 text

> ctx.font = "18px sans-serif"; //文本大小和字体

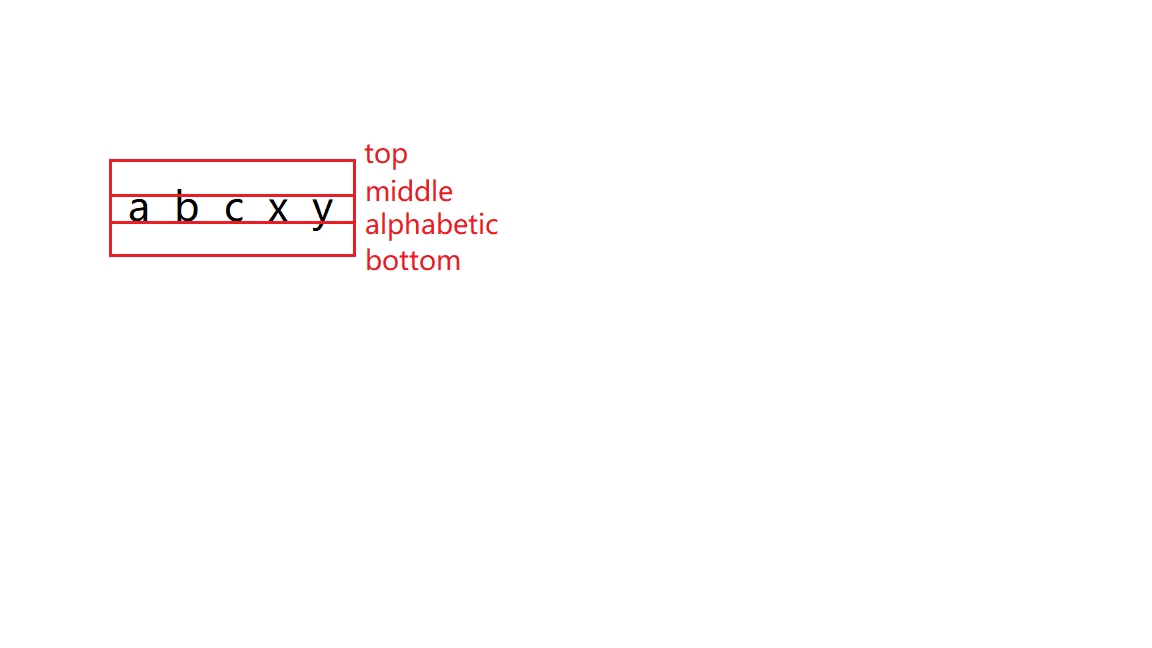
> ctx.textBaseline = "top"/"middle"… //调整字体基线位置

> ctx.fillText(str,x,y) //填充一段文本

> ctx.strokeText(str,x,y) //描边一段文本

> ctx.measureText(str) //测量一段文本宽度(width:x)

**!注意：**ctx.measureText(str)返回的是一个js对象obj，需要用obj.width来获取宽度。



【练习】在画布左上角右上角左下角右下角居中，画5段文本

【练习】在画布绘制左右移动文本

#### 4.2.3 路径 path

path：有时需要从多个坐标组成任意形状，此时路径不可见，但可以进行"描边"，"填充"将路径显示出来。

> ctx.beginPath(); //开始一条新路径

> ctx.closePath(); //闭合当前路径

> ctx.moveTo(x,y); //移动到指定点

> ctx.lineTo(x,y); //从当前点到指定点画一条直线

> ctx.arc(cx,cy,r,start,end); //绘制圆拱形

cx,cy圆心 r半径 start开始弧度 end结束弧度

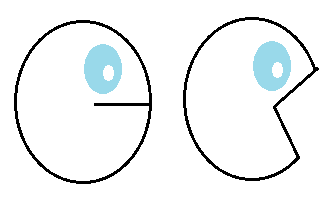
> ctx.stroke() //描边

> ctx.fill() //填充

**!注意：**弧度 = 角度\*Math.PI/180

【练习】使用圆拱形绘制可以前进的圆环进度条

【练习】吃豆子：创建两个函数 openMouth() closeMouth()，每隔一秒交替调用。



#### 4.2.4 图片 image

canvas属于客户端技术，运行在客户端的浏览器上的，而图片都是存储在服务器端的。

canvas绘图必须先下载图片等待图片下载完成，才能绘制。

> 创建图片对象： var p3 = new Image();

> 下载图片： p3.src = "1.jpg";

> 绑定事件，下载成功： p3.onload = function(){

> 将图片绘制到画布上 ： ctx.drawImage(p3,x,y); //原始大小

ctx.drawImage(p3,x,y,w,h) //缩放图片

}

【练习】左下角 右下角 居中绘制小飞机，居中飞机绘制成二倍大小 "小飞机"

【练习】在画布中央绘制，可以随着鼠标移动而移动飞机(c3.onmousemove = function(){ //…})

**!注意：**canvas只能将事件绑定画布上，不能绑定某个图形上

#### 4.2.5 变形 rotate/translate

canvas绘制中也有变形技术，可以针对某一个图片在绘制过程中进行变形操作。

> ctx.rotate(弧度); //旋转画笔对象

> ctx.translate(x,y); //将整个画布原点平移到指定点

> ctx.save(); //保存画笔状态

> ctx.restore() //恢复画笔状态

**!注意：**默认旋转以原点(0,0)，用平移改变原点之后若要清除画布，需要加大清除区域。

**!注意：**后两个方法的使用，是在开始绘制图形之前用ctx.save()保存画笔的原始状态，当当前形状绘制完毕之后，再用ctx.restore()将画笔归为原始状态。

【练习】创建定时器让小飞机不停旋转

【练习】创建定时器让小飞机绕自己为中心旋转

【练习】画布上画二架以自己为中心旋转小飞机

飞机1 飞机2 飞机2旋转速度快2倍飞机1

问题：当画布上出现一个以上有元素,会出相互影响(角度累加;原点平移)

解决：绘画之前先保存，绘画之后恢复

【课后练习】

作业1: 柱子颜色随机; 销售额

作业2: 查询2018年每个月药品销售统计信息

1月 1000 2月1100 ....

#### 4.2.6 渐变对象 linearGradient

* 创建渐变对象

var g=ctx.createLinearGradient(x1,y2,x2,y2);

* 添加颜色点

g.addColorStop(offset,color);

// offset取0-1之间的数，表示用color颜色填充到画布横向的offset位置处

* 赋值给填充样式或描边样式

ctx.fillStyle = g;

ctx.strokeStyle = g;

* 使用画笔绘制图像

ctx.fillRect(x,y,w,h);

ctx.strokeRect(x,y,w,h);

### 4.3 SVG 绘图概述

#### 4.3.1 位图/矢量图

* 位图：由一个一个像素点组件，每个点都有自己的颜色，色彩细腻，但放大失真。
* 矢量图：有一个一个线条组成，每个线条都有自己的颜色、方向，可以无限缩放，但细节不够丰富。

|  |  |  |
| --- | --- | --- |
|  | Canvas | SVG |
| 类型 | 2d位图 | 2d矢量图 |
| 如何绘图 | 使用js代码绘图 | 使用标签绘图 |
| 事件绑定 | 每个图形不是元素，无法绑定 | 每个图形都是元素，可以直接绑定事件监听 |
| 应用场景 | 特效，小游戏 | 统计图，地图 |

#### 4.3.2 SVG使用

SVG技术诞生早2000年，早期作为XML技术扩展应用出现。H5标准把常用SVG标签采纳为标准，但有一些废弃了。

用法：先在HTML文档所需位置创建SVG画布

* 创建SVG画布

<svg width="" height=""></svg>

* 在HTML中创建元素

<rect width="" height="" x="" y="" fill="" fill-opacity="" stroke="" stroke-opacity=""></rect>

【练习】在svg画布上创建30\*300柱子,初始化淡红色(#faa)填充和深红色(#800)边框都是半透明,鼠标悬停变为不透明，离开恢复原样.

【练习】在svg 画布上绘制一个柱子，初始高度为0时使用定时器，不断修改高度到300 停止。

【练习】使用 ajax获取服务器上数据，依据数据绘制统计图

* 动态创建元素

> 字符串拼接

var html = "<rect></rect>"

svg.innerHTML = html;

> 对象创建

var rect =

document.createElementNS("http://www.w3.org/2000/svg","rect");

svg.appendChild(rect);

【练习】在svg画布上随机绘制30个实心圆形,大小随机，位置随机,填充颜色随机，透明度随机。点击某个圆之后它慢慢变大/变淡直至消失从dom删除。

【课后作业】

> 作业1:复习canvas/视频

> 作业2:自己总结东哥 vue 原理概念(背)

> 作业3:把 vue 改版项目

> 作业4:canvas 图形验证码(示例：验证码不是用canvas生成,后台php/java/node.js生成)

### 4.4 SVG 绘图

#### 4.4.1 矩形 rect

<rect width="" height="" x="" y="" fill="" fill-opacity="" stroke="" stroke-opacity=""></rect>

#### 4.4.2 圆形 circle

<circle cx="" cy="" r="" fill="" stroke=""></circle>

#### 4.4.3 椭圆 ellipse

<ellipse cx="" cy="" rx="" ry="" stroke=""></ellipse>

> rx：水平半径

> ry：垂直半径

#### 4.4.4 直线 line

<line x1="" y1="" x2="" y2="" stroke="" stroke-width="">

#### 4.4.5 折线 polyline

一条折线上可以有任意多个连续点

<polyline points="x,y x,y x,y" fill="" stroke=""></polyline>

【练习】绘制五角星

#### 4.4.6 文本 text

SVG画布不允许使用span,p等绘制元素

<text font-size="" fill="" x="" y="">文本内容</text>

#### 4.4.7 图像 image

<image xlink:href="x.jpg" width="" height="" x="" y=""></image>

#### 4.4.8 渐变对象 linearGradient

渐变对象属于一种特效对象，定义特效对象：

<defs>

<linearGradient id="g3" x1 y1 x2 y2>

<stop offset="0" stop-color="">

</linearGradient>

</defs>

<ANY filter="url(#f3)"></ANY>

#### 4.4.9 滤镜 filter

滤镜也属于一种特效对象，所以要先定义特效对象。

> 高斯模糊滤镜

<defs>

<filter id="f3" >

<feGaussinanBlur stdDeviation="3">

</filter>

</defs>

<ANY filter="url(#f3)"></ANY>

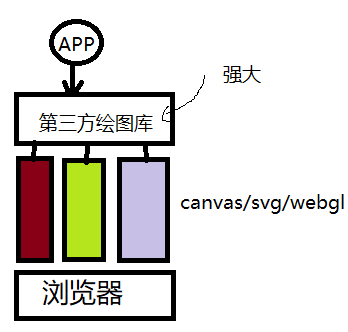
#### 4.4.10 图形组 g

如果几个图形的属性一致，可以用<g></g>包含图形的标签，并将共有属性写在g标签上。

### 4.6 Echarts 第三方绘图库 ★

官网：<http://www.echartsjs.com/index.html>

常用于完成企业中项目功能；Echarts使用前需要下载安装配置(完整版)。



#### 4.6.1 工作流程

* 加载echarts.min.js文件
* 创建容器绘制图形

<div id="main" style="width:500px;height="400px">

* 创建echarts对象并且初始化

var mychart = echarts.init(document.getElementById("main"));

* 创建配置项对象

var option = {};

* 将配置项对象加载到echarts对象即可

mychart.setOption(option);

【练习】药品销售统计echartjs绘制 柱状统计图，发送请求jquery 完成。

#### 4.6.2 图形绘制

* 完整代码：

<head>

…

<script src="js/echarts.min.js"></script>

</head>

<body>

<h1>第三方绘图库——Echarts柱形图</h1>

<div id="main" style="width:500px;height:400px;"></div>

<script>

var main = document.getElementById("main");

var myChart = echarts.init(main);

var option = {

title:{text:"绘制折线统计图"}, //标题

toolbox:{show:true,feature:{ //工具箱

saveAsImage:{show:true}, //将此div内容保存为图片

dataView:{show:true}, //数据视图

restore:{show:true}, //恢复

dataZoom:{show:true}, //显示区域

magicType:{type:["line","bar"]} //切换视图

}},

tooltips:{}, //数据提示框

legend:{data:["销量"]}, //图例

xAxis:{data:["衬衫","羊毛衫","休闲鞋"]}, //x轴内容

yAxis:{}, //y轴空着

series:[{ //显示数据

name:"销量",type:"line",data:[7,11,1,5,4]

},{

name:"库存",type:"bar",data:[100,30,101,1,5]

},{

name:"访问方式",type:"pie",radius:50%,center:[50%,50%], data:[{value:135,name:"搜索引擎"},…]

}]

};

myChart.setOption(option);

</script>

</body>

* 代码图解：



【课后作业】药品销售统计图，柱状，折线，饼图

## 五、地理定位

### 5.1 概述

Geolocation，地理定位：使用js获取浏览器所在地址坐标(经度，纬度，海拔，速度)，用于实现LBS Location Base Service。例如：饿了么，滴滴打车…

* 手机浏览器：

> 首选手机GPS芯片与网络通信获取，定位精度到米。

> 次选手机浏览器通信基站定位获取，定位精度到公里。

* PC浏览器：

> 默认情况下，通过电脑IP地址计算获取位置，定位精度到公里。

* HTML5提供新对象：用于获取当前浏览器定位信息

window.navigator.geolocation{

getCurrentPosition: fn, //获取当前定位信息

watchPosition:fn, //监听定位变化

clearWatch:fn //清除监听

}

**!注意：**上述代码在国内不能使用，因为该方法是由谷歌提供，国内无法访问IP地址库。国内可以使用百度地图和腾讯地图。

### 5.2 百度地图

#### 5.2.1 注册配置及使用

**>** 注册百度地图开发账户

**>** 创建网站，并申请访问密钥(AccessKey)

**>** 在自己的网页中嵌入百度地图API

#### 5.2.2 开发流程

* 创建地图实例

var map = new BMap.Map("container");

* 创建坐标点

var point = new BMap.Pointer(经度,纬度);

* 以坐标点为中心显示

map.centerAndZoom(point,15)

* 开启鼠标滚轮缩放

map.enableScrollWheelZoom(true);

## 六、拖放API

Drag & Drop：拖动和释放

HTML5为实现拖放操作创建了7个事件，分两组：

* 拖动源对象：(会动)可以触发3个事件

> ondragstart 拖动开始

> ondrag 拖动中

> ondragend 拖动结束

整个拖动过程：dragstrart\*1+drag\*n+dragend\*1

* 拖动目标对象: (不会动)可以触发4个事件

> ondragenter 拖动进入

> ondragover 拖动悬停在上方

> ondragleave 拖动离开

> ondrop 拖动释放

拖动过程：dragenter\*1+dragover\*n+dragleave\*1

拖动过程：dragenter\*1+dragover\*n+drop\*1

**!注意：**必须阻止dragover的默认行为

【练习】使用拖动源对象提供事件，实现“可以随鼠标移动小飞机”

【提示】父元素相对定位,子元素绝对定位修改left/top，使飞机移动

【练习】使用拖动事件的源对象和目标对象

【提示】在刚开始拖动(src,ondragstart)记录被拖动源对象id，目标释放(target, ondrop)，根据此id删除对象。

## 七、Web Worker

### 7.1 概述

程序：program 指可以被CPU执行的代码，存储外储

进程：process/task 将程序调用放入内存中并分配空间，随时供CPU执行

线程：thread 线程是进程内部执行代码的单位

* Chrome浏览器中的线程模型：

一个Chrome进程内部至少有6个线程在向web服务器发起请求以获取资源。还有一个线程负责所有内容的绘制并且执行代码(css/js)-UI主线程。

观察如下代码：

<button>按钮一</button>

<script src="x.js">很耗时的js任务</script>

<button>按钮二</button>

问题：在网页执行过程中绘制内容和执行js是同一个线程。有可能先显示按钮后等待一段时间再显示其他内容。

解决：创建新县城执行耗时的JS任务。

<button>按钮一</button>

<script>

//创建一个woker对象，并启动一个新线程来执行x.js代码

new Woker("x.js");

</script>

<button>按钮二</button>

**!注意：**html5 要求如果程序Worker线程，程序中不能出现，任何DOM/BOM 代码。

### 7.2 Worker数据传递

#### 7.2.1 UI线程的发送与接收

<script>

var w = new Worker("js/01.js");

var msg = "这是UI线程要发送的数据";

w.postMessage(msg); //发送数据给worker

w.onmessage = function(e){ //从worker接收数据

console.log("这是Worker线程发过来的数据:"+e.data);

}

</script>

#### 7.2.2 Worker线程的发送与接收

/\* js/01.js文件 \*/

//从worker接收数据

onmessage = function(e){

console.log("这是UI线程发过来的数据:"+e.data);

}

//发送数据给UI

postMessage(msg);

【练习】在HTML中创建input元素(输入数值)，创建button按钮“开始计算累加和”。点击此按钮，创建Worker线程来计算用户输入数字的累加和。计算完成将结果发送UI线程，在button按钮下的<p></p>中显示结果。

【提示】①创建HTML；②创建js计算累加和；③接收发送数据

### 7.3 小结

项目中何时使用Worker：

> 如果js 代码中出现DOM或者BOM代码不能用Worker(jquery)

> Worker适合执行耗时JS任务(复杂计算/统计信息)

## 八、Web Storage

### 8.1 概述

作用：在浏览器中存储当前用户专用数据，如购物车

**!注意：**在客户端存储数据最基本要求，安全性要求低数可以保存浏览器，如：密码，安全性高数据；购物车:安全性低数据。

客户端存储数据：

> Cookie 浏览器兼容性好，但不能超4KB且操作复杂

> Flash 依赖Flash播放器

> Web Storage 操作简单，但不能超8MB,

> IndexDB 存储量大，但尚不标准

### 8.2 session

定义：会话(操作过程)浏览器从打开某个网站的第一个页面开始(会话开始)中间可能打开多个网页(会话中)直到关闭浏览器(会话结束)。整个过程称为:"浏览器与web 服务器的一次会话"。

### 8.3 webStore对象

webstore技术中浏览器为用户提供二个对象

#### 8.3.1 sessionStorage

此对象仅供此次会话所有页面共同使用，一旦关闭浏览器则数据消失

* 获取数据：

var value = sessionStorage.getItem(key)

var value = sessionStorage[key];

* 修改数据

sessionStorage.setItem(key,value)

sessionStorage[key] = value;

* 清除数据

sessionStorage.clear();

sessionStorage.removeItem(key);

* 其它

sessionStorage.length //数据个数

var key = sessionStorage.key(i);

【练习】使用sessionStorage保存用户名：创建index.html右上角显示“请登录”超链接；创建login.html输入用户名，提交按钮；用户点击提交按钮。

【提示】登录成功 3s后跳转首页(setTimeout()+location.href)

【课后作业】使用 echarts 完成销售统计图(柱状)要求指定不同颜色

#### 8.3.2 localStorage

此对象供此次会话以及后续会话使用，即使浏览器关闭数据存在。即永久保存数据。

用法同sessionStorage对象。

## 九、Web Socket

### 9.1 概述

* http 网络协议

> 作用：传输网页资源(网页搬运工) HTML/CSS/JS/IMAGE

> 工作方式：属于“请求-响应”，只有客户端发起请求信息，服务器才会返回响应消息，没有请求就没有响应。

* webSocket 网络协议

> 作用：传输网络数据

> 工作方式：“广播-收听”，客户端连接服务器不再断开，双方都可随时发送数据。

结论：webSocket协议是对http协议的补充。

### 9.2 程序创建

#### 9.2.1 服务器(php/java/node.js)

#### 9.2.2 客户端 html5新对象 webSocket

* 创建webSocket客户端对象并且连接服务器

var socket = new WebSocket("ws://127.0.0.1:9001");

* 向服务器发送数据

socket.send(str);

* 接收服务器数据

socket.onmessage = function(e){ e.data}

* 关闭连接

socket.close();

# 阶段项目：Canvas游戏——大鱼小鱼(待整理)

## 一、项目概述

### 1.1 项目目录结构

game/index.html--程序唯一个html功能调用所资源并显示

src--保存游戏所需图

babyEye0. png babyEye1. png 小鱼眼睛 2

babyFade0. png ..babyFade19. png 小鱼身体20

babyTail0.png .. babyTail7.png 小鱼尾巴8

blue.png fruit.png 两种食物2

background.jpg 大海背景 1

js--保存游戏所有js

commonFunctions.js --游戏函数

main.js --游戏入口

background.js --绘制背景

ane.js --海葵

fruit.js --食物

mom.js --大鱼

baby.js --小鱼

### 1.2 项目工作流程

* index.html

> 加载所有js 文件[mom.js baby.js main.js ..]

> 当页面元素加载完成 调用 main.js game()

* main.js

> 声明所有全局变量

> 声明game() ->init()

->gameloop();

> init() 创建所对象并且调用初始化方法

> gameloop() 创建定时器绘制游戏所有角色

## 二、主要文件内容

### 2.1 game/index.html

## 三、各实物对象

### 3.1 海葵

### 3.2 食物

### 3.3 大鱼

-大鱼面向鼠标移动

监听画布鼠标移动操作 mx my

将鼠标坐标赋值大鱼坐标

-大鱼吃食物

碰撞检测:大鱼是否碰撞到某个食物

公式:计算二点之间距离

创建文件 js/collsion.js 完成碰撞检测

大鱼和所有食物判断如果两点之间距离小于 900

将当前食物状态false

# 第十七章 微信小程序

## 一、小程序概述

### 1.1 微信与微信公众号

微信：通讯工具(9亿用户群)

微信公众号：涉及电视台、报纸、杂志等多方面内容，包括：

> 小程序 [APP 苹果 安卓]

> 小游戏 [跳一跳]

> 电商平台 [微店]

### 1.2 微信小程序概述

特点：微信小程序，简称“小程序”，不需要下载即可使用，用完即走，而通常的APP如“支付宝”，需要下载、安装、配置、删除等步骤。

大小：小程序刚发布时要求软件项目最大不能超过1MB，升级之后最大不能超过2MB。

运行：小程序运行在微信自带的浏览器中。开发者开发完之后可上传至微信平台，在平台中运行。

### 1.3 微信小程序的功能

小程序适合项目：功能简单，效率要低，使用频率低。

如：

> 斗鱼直播：直播实时视频信息

> 支付程序：可以

> 读书程序：可以

### 1.4 小程序开发

#### 1.4.1 注册小程序账户

在微信公众号平台注册小程序账户，获得appId(微信公众号平台分发给开发者的密钥，形如：wxw93ija87a92902)，用于微信平台验证程序合法性。

微信公众平台：<https://mp.weixin.qq.com/>

#### 1.4.2 小程序开发工具

* 下载：

<https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html?t=18111420>

> Windows 64位 / Windows 32位 / Mac OS

> Windows 仅支持 Windows 7 及以上版本

* 使用：

> 开发工具需要使微信扫一扫功能进入工具(下图左)

> 创建项目路径与项目名称(上图右)

#### 1.4.3 项目主体目录结构

pages -- 项目所有组件

index.wxml - xml文件(非常严格)

app.js -- 入口文件

app.json -- 项目全配置文件

app.wxss -- 全局样式文件

**!注意：**json文件不能加注释，内容严格。

**!注意：**wxss文件比之css文件新增两个功能：

> 单位 rpx，相当于响应式px；

> @import "x.wxss"

#### 1.4.4 小程序配置

##### 1.4.4.1 全局配置项app.json

{

"pages": [

"pages/home/home", /\* 放在前面的首先显示 \*/

"pages/index/index",

"pages/logs/logs"

],

"window": {

"backgroundTextStyle": "light",

"navigationBarBackgroundColor": "#ff6610",

"navigationBarTitleText": "本地宝",

"navigationBarTextStyle": "white"

}

}

pages[array] 页面路径列表，默认显示数组第一个组件内容

window[array] 窗口样式(渲染方式)

**!注意：**颜色不能用red等单词表示，会报错。

**!注意：**之前VUE中的tabBar兼容性差

【练习】修改显示组件顺序

> app.json "pages":["pages/index/index"...]

> 哪个组件在第一位则默认显示哪个组件

##### 1.4.4.2 局部配置项(组件的json)

exam01/exam01.json 局部

**!注意：**当全局和局部的配置项冲突时，采用就近原则，即优先局部配置项。

#### 1.4.5 小程序数据显示

##### 1.4.5.1 数据显示方式

小程序动态获取对应组件data值,使用方式双花括号

{{msg}}

#data 属性保存初始化数据

**!注意：**控制属性加在双括号之内

##### 1.4.5.2 控制属性

* if语句：

<view wx:if="{{condition}}"></view>

condition:值为true显示view中内容；值为false不显示

* else-if语句：

<view wx:elif="{{}}"></view>

* else语句：

<view wx:else></view>

* <block></block>

括号开始括号结束

**!注意:**如果判断内容多个组件block包含里来

* for语句：

<view wx:for="{{数组对象}}"></view>

**!注意：**默认情况循环：当前元素item；下标 index，但可修改

<view wx:for-index="idx" wx:for-item="obj"></view>

**!注意：**小程循环元素之前为提高效率有一个排序操作,提供不重复列,没有属性报警告。

> 添加wx:key

<block>

<swiper-item wx:for="{{banners}}" wx:key="item.id">

<image src="{{item.img\_url}}" mode="aspecfill" lazy-load> </image>

</swiper-item>

</block>

【练习】九九乘法表

## 二、page 组件

要新建一个页面需要在pages文件夹目录下先新建一个新目录test，然后在test下新建page，通常page名称同外层目录。此时新建的这一个page会创建四个文件：

test.js test.json test.wxml text.wxss

### 2.1 wxml 内容文件

#### 2.1.1 wxml 文件语法简述

小程序的wxml(WeiXin Markup Language)文件是一种xml文件，用于描述网页结构。xml-格式要求非常严格(不是html)。

> 一个xml文件必须有且只有一个根元素。

> 所有标签有开始必须有结束

> 所有属性值必须双引号 <button name="btn" >

#### 2.1.2 wxml 常用标签

<view></view> 相当于 <div></div>

<text></text> 相当于 <span><span>

<image></image> 相当于 <img src="" />

<navigator></navigator> 相当于 <a>

<form></form>

<block></block> 容器，将内部多个带逻辑语句的标签包含起来

【练习】首页创建 pages/test 目录 ->test page[组件]

test.js test.wxml test.wxss test.json

##### 2.1.2.1 <image> 图片

* 用法：

<image src="x.jpg" mode="aspectFill " lazy-load></image>

* 常见属性：

> 图片路径src

- 本地绝对路径 /pages/img/blue.png ok

- 相对路路径 ../img/blue.png error

- 网络绝对路径 <http://127.0.0.1:3000/blue.png> ok

> 图片懒加载lazy-load

- 默认值false(即不写lazy-load的情况下)

- 如果当前图片是通过 ajax异步内容 false -> true(直接在标签里添加即可)

<image lazy-load></image>

> 图片宽度和高度mode

- 默认宽高：300px\*225px

- aspectFill 保证图片充满整个容器；方式：选取宽高缩放值里较小的值来缩放；代价：一部分图片内容被截断。

- aspectFit 保证图片完整地装进容器；方式：选取宽高缩放值里较大的值来缩放；代价：容器无法装满，出现留白。

##### 2.1.2.2 <text> 文本

相当于HTML中的<span><span>，只有text元素内的内容用户才能进行全选(复制)

* 用法：

<view class='head1'>

<text decode>&nbsp;&lt;jscsshtml</text> /\* <jscsshtml\*/

</view>

<view class='head2'>

<view>&nbsp;&lt;jscsshtml</view> /\*&nbsp;&lt;jscsshtml\*/

</view>

* 常见属性：

:selectable 文本是否可以选中，默认：false

:decode 文本是否解码，默认false，如：&lt; &gt;

##### 2.1.2.3 <navigator>跳转

* 用法：

<navigator url="当前小程序跳转链接"></navigator>

##### 2.1.2.4 <swiper> 轮播图

* 用法：

<swiper>

<swiper-item>

<image src="" >

</swiper-item>

</swiper>

* 常见属性：

> autoplay 自动播放

> interval 间隔时间

> indicator-dots 指示点

> vertical 方向

### 2.2 wxss 样式文件

#### 2.2.1 像素换算

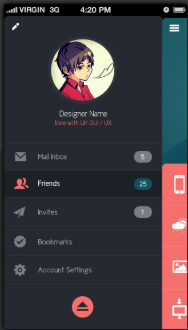
小程序对css文件封装操作 wxss。

开发项目中的px与rpx：

> 设计师ps 标准图片(效果图) px [物理像素]

> 开发 网页(图片) px (逻辑像素)

**!注意：**默认样式文件和js文件自动加载wxml，不用手工引入(微信小程序平台自动操作)。



* 示例：

> iphone 6 设计图宽度(物理像素 750) 2x(2倍)

- ex:(逻辑像素 375px) (逻辑像素 750rpx)

> iphone 6 plus 3x

**!注意：**widht:height 使用 rpx;font-size:12px;

#### 2.2.2 样式引入

@import "引入其它样式文件"

小程序样式布局:弹性布局

### 2.3 js 文件

#### 2.3.1 小程序中的js语法

**>** ECMAScript

- 基本数据类型 number;string;boolean;null;undefined

- 引入 Object;Number;Date;RegExp;...

**>** 小程序没有 BOM/BOM概念

- widow;document;alert();....

**>** 小程序有一个顶级js对象 wx(类似window)

- 查询文档:小程序文档->API

#### 2.3.2 js文件结构

小程序一个组件对应.js中 默认的一个对象。js文件由一个page对象组成，page对象内可以设置初始数据、编写生命周期和事件触发函数的方法。

* Page:当前组件对象

> data：页面的初始数据

> onLoad/onReady/onShow/onHide/onUnload：生命周期函数

> onPullDownRefresh/onReachBottom/onShareAppMessage：组件事件处理函数

Page({

/\* 页面的初始数据\*/

data:{},

/\* 生命周期函数--监听页面加载 \*/

onLoad: function (options) {

},

/\* 生命周期函数--监听页面初次渲染完成 \*/

onReady: function () {

},

/\* 生命周期函数--监听页面显示 \*/

onShow: function () {

},

/\* 生命周期函数--监听页面隐藏 \*/

onHide: function () {

},

/\* 生命周期函数--监听页面卸载 \*/

onUnload: function () {

},

/\* 页面相关事件处理函数--监听用户下拉动作 \*/

onPullDownRefresh: function () {

},

/\* 页面上拉触底事件的处理函数 \*/

onReachBottom: function () {

},

/\* 用户点击右上角分享 \*/

onShareAppMessage: function () {

}

})

#### 2.3.3 小程序生命周期

##### 2.3.3.1 生命周期概述

生命周期：一个组件从创建到销毁过程。

* 全局生命周期:app.js

- onLaunch 当小程序运行只执行一次

* 局部生命周期(组件生命周期)

- onLoad 组件加载时触发一个组件只会调用一次

##### 2.3.3.2 生命周期函数

* onReady

页面初次渲染完成触发,一个页面只会调用一次,代表页面己渲染结束可以进入交互操作

* onShow

页面显示或者切换前台时触发

* onHide

页面隐藏/切换后台触发

* onUnload

页面卸载时触发.通过程序跳转其它组件，即redirectTo();navigateBack()触发。

#### 2.3.4 组件自带事件处理函数

* onPullDownRefresh 监听用户下拉操作

> 默认小程序禁止用户下拉操作，如若需要触发该事件，可在在全局文件配置"window"：

enablePullDownRefresh:false

* onReachBottom 监听用户上拉触底
* onPageScroll({}) 监听用户滑动页面事件

> scrollTop 页面在垂直方向己经滚动距离

## 三、事件、组件跳转传参及AJAX

### 3.1 事件概述

#### 3.1.1 事件绑定

* 事件：组件表示层(wxml)与业务逻辑层(js)交互手段
* 事件绑定：

<view id="tapTest" data-id="10" bindtap="函数">Click Me</view>

* 事件类型：

tap 手指触摸后马上离开

longtap 手指触摸后，超过350ms再离开(长按钮) 旧(1.5.0之前)

longpress 手指触摸后，超过350ms再离开 新(1.5.0之后)

* 绑定方式：

在相应的Page对象定义事件处理函数：

Page({

函数名:function(event){

console.log(event);

},

…

})

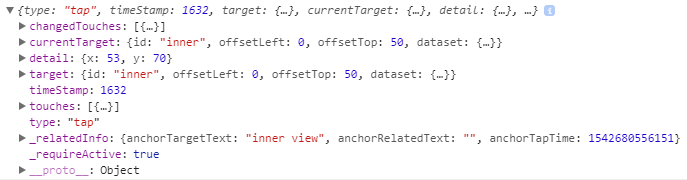
问题："longtap" event is deprecated. Please use "longpress" event instead.

如果当前元素指定 longpress并绑定函数tap事件有可能不会触发。

解决：deprecated:"过期"，过期即指不建议再使用，如果一定要用也可以正常使用。但建议采用longpress。

#### 3.1.2 事件对象

* 事件对象信息：



type:"tap" 触发事件名称

target:{id:"tapTest",dataset:{"idx":19}} 触发事件元素

currentTarget{} 当前元素

timeStamp 页面打开到触发事件所经过毫秒数

#### 3.1.3 事件传播方式

* 冒泡事件:当一个组件上的事件被触发后，该事件会向父节点传递
* 非冒泡事件:一个组件上的事件被触发后，该事件不向父节点传递

小程序区别事件处理是否使用冒泡方式看前缀：

> bind 冒泡事件 bindtap

> catch 非冒泡事件 catchtap

<view id="outer" bindtap="handleTap1">

outer view

<view id="middle" bindtap="handleTap2">

middle view

<view id="inner" catchtap="handleTap3">

inner view

</view>

</view>

</view>

### 3.2 事件参数传递

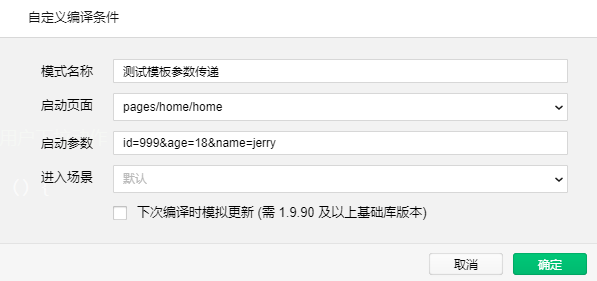
#### 3.2.1 编译时传参

编译前将要传递的参数设置好，在组件的js文件中接收(js文件中各生命周期函数都有一个形参options用于接收数据)。

* 参数传递路径：

> 自定义编译条件 ——> 组件生命周期函数形参options

* 原理：使用查询字符串传参
* 编译设置：微信开发工具的工具菜单-->编译配置-->新增编译条件



* 接收(js文件)：options.id/options.age/options.name

**【TIPS】**在js文件中接收到数据，通常可在此事件中发送ajax请求

#### 3.2.2 事件处理函数传参

问题：小程序不推荐使用参数传递数值方式来将参数在函数中进行传递。

解决：使用自定义属性

* 参数传递路径：

> wxml文件中标签的自定义属性的值 ——> 对应组件js文件对应绑定事件形参

* 传递：

在标签中绑定自定义属性，并将值传递，可绑定多个自定义属性传递多个值

<view id="tapName" data-idx="39" bindtap="tapName>Click Me!</view>

* 获取：

在处理函数内部用event接住结果并打印

Page({

tapName:function(event){

console.log(event.target.dataset.idx);

}

})

#### 3.2.3 传递给前端页面

js文件中获取到的值(ajax等)想要传递回页面，更改页面对应标签显示的数据，需要将此值“交给”组件的data对象，由data对象进行[数据更改显示](#_1.4.5.1_数据显示方式)。

* 参数传递路径：

> js文件中要传递的值 ——> wxml文件对应标签需要更改显示数据的值

【TIPS】由于1.4.5.1节已经说明过数据显示方式，在此不再赘述，仅对数据添加到data进行说明。

问题:获取参数值不能将数值显示模板

解决:

> 在低版本小程序 this.data.id = id; 正确 2017 <

> 新版本小程序 使用新方法解决 setData

setData({data属性名:数值})

setData将数据异步更新视图wxml同时改变 this.data值

### 3.3 组件间跳转

小程序不同组件之间可以进行自定义跳转

#### 3.3.1 标签跳转

<navigator url="组件路径"></navigator>

其中，路径可采用

> 绝对路径：/pages/exam06/exam06

> 相对路径：../exam05/exam05

**!注意：**使用标签方式跳转还可后退

#### 3.3.2 编程跳转

##### 3.3.2.1 navigateTo

* 用法

wx.navigateTo({

url:"组件路径",

…

})

* 说明

> 保留当前组件，跳转到应用内其他组件

**!注意：**不能跳转写在tabbar里的内容

##### 3.3.2.2 navigateBack

* 用法

wx.navigateBack({

url:"组件路径",

…

})

* 说明

> 关闭当前页组件返回上一个组件或其他组件

【练习】exam06组件添加一个按钮点击按钮跳转 exam07传参数?id=101

##### 3.3.2.3 redirectTo

* 用法

wx.redirectTo({

url:"组件路径",

})

* 说明

> 关闭当前组件跳转到应用内其它个组件

> 不允许跳转tabbar

> 卸载当前组件(在卸载之前触发 onUnload)

##### 3.3.2.4 reLaunch

* 用法

wx.reLaunch({

url:"组件路径",

})

* 说明

> 关闭所有组件，打开到应用内某个组件

##### 3.3.2.5 switchTab

* 用法

wx.switchTab({

url:"组件路径",

})

* 说明

> 跳转到tabBar组件，并关闭所有非tabbar组件

##### 3.3.2.6 小结

> 不同组件之间跳转保留优先 wx.navigateTo()

> 如果当前组件定义在tabbar wx.reLaunch()

### 3.4 AJAX请求

小程序中显示数据依赖 AJAX获取服务器上数据。

保存当前data属性在模板显示

* 语法：

// 小程序发送异步请求方法

wx.request({})

* 常用属性和方法

> url 请求服务器程序地址

> data string/object/array 请求参数

> method 请求方法

> header 请求header appliation/json

> success 请求成功回调函数

> fail 请求失败回调函数

> complete 无论成功或失败都执行

* 常见错误

问题1：发送ajax请求时，协议必须为https才能访问

解决1：工具菜单->项目详细->[\*] 不校验合法域名

问题2：this.setData is not a function;

原因2：在请求处理函数this 指定当前 wx对象 wx没有 setData

解决2：将this指定从 wx 修改 Page ----> 使用箭头函数

问题3：参数传递失败 id;age

解决3:打开网络控制面板 network

## 四、小程序特性

### 4.1 交互

#### 4.1.1 提示框

* 用法：

wx.showToast({

title: "…", //提示的内容，为了兼容性考虑，最好少于7个字

icon: "…", //图标：success/loading/none

duration: 1000 //持续时间 ms

})

* 效果图：

success loading none

#### 4.1.2 模态框

* 用法：

wx.showModal({

title: "是否删除",

content: "是否删除指定文件",

success: (res)=>{

//res:errMsg;cancel:true/false;confirm:true/false

if(res.confirm){

this.setData({isSave:"收藏"});

}else{

this.setData({isSave:"取消收藏"});

}

}

})

* 效果图：



【练习】以下view文字切换效果 "收藏" "取消收藏"

<view>收藏</view>

点击view 提示模态框"是否收藏文章"

是:<view>取消收藏</view>

否:<view>收藏</view>

#### 4.1.3 操作菜单

* 用法：

wx.showActionSheet({

itemList: ["","",""],

itemColor: "#fff",

success: (res)=>{

console.log(res); //res:errMsg;tapIndex

console.log(res.tapIndex);

}

})

* 效果图：



### 4.2 存储

#### 4.2.1 局部数据存储

局部，指当前组件的生命周期内，采用当前组件的data:{}存储数据。

#### 4.2.2 全局数据存储

全局，指小程序的生命周期内，采用如下方法进行数据存储：

* 定义：

在app.js中定义全局共享数据:

APP({

globalData:{LoginName:"",list:[]}

})

* 使用：

// 1.加载全局组件

const app = getApp();

// 2.读取数据

var uname = app.globalData.LoginName;

// 3.操作数据

app.globalData.LoginName = "tom";

### 4.3 音频

小程序后台播放器播放音乐，对于微信客户端来说，只能同时有一个后台音乐在播放，当其他小程序占用音乐播放器，原有小程序音乐立即停止播放。

#### 4.3.1 方法

* 播放背景音乐

wx.playBackgroundAudio({})

> dataUrl 音乐链接

- 目前支持的音频格式：mp3;wav;aac;m4a

> title 音乐标题

> convertImgUrl 音乐封面

* 暂停背景音乐

wx.pauseBackgroundAudio({})

* 停止背景音乐

wx.stopBackgroundAudio({})

* 开发顺序：

> 将音频文件保存在服务器 node.js/public/bg.mp3

> 启动服务器

> 打开浏览器访问bg.mp3

> index组件

* 效果图：

#### 4.3.2 监听事件

* 监听音乐播放事件

wx.onBackgroundAudioPlay((callback))

* 监听音乐暂停事件

wx.onBackgroundAudioPause

* 监听音乐停止播放事件

wx.onBackgroundAudioStop

### 4.4 视频

* 播放方式一：通过video组件显示视频

<video src="视频路径"></video>

默认视频宽高为：300px 225px；

常见属性：

> autoplay 自动播放

> loop 循环

> muted 静音

> poster 广告图片

> controls 控件

* 播放方式二：通过video组件显示视频(选择视频 相册 照相机)

wx.chooseVideo({

sourceType: ["album","camera"], //相册相机

maxDuration: 60, //视频长度上限

camera: ["front","back"], //相机可以用前/后置摄像头

success: (result)=>{

result.tempFilePath; //视频路径

}

})

### 4.5 模板

当组件中有可重复使用的功能，可以将其封装到模板中，其他组件如果需要可以直接调用。类似于vue的子组件。

* 创建目录：

item.template.wxml

item.template.wxss

* 添加代码

<template name="item\_name">

<view class="item">{{user.name}}</view>

</template>

.item{color:red;}

* 其他组件如何调用(引入模板；调用模板)

// 引入模板wxml

<import src="item/item\_template" />

// 引入样式文件

@import "post\_item/post\_item\_template.wxss"

* 调用模板

<template is="模板名称" data="{{user}}" />

【常见错误】

> th "..." not found from "./pages/post/post.wxml"

原因:模板路径

> Template "post-item1" not found.

原因:名称不同正确

【练习】将轮播图功能创建模板保存在post调用

### 4.6 tabBar

小程序主页底层导航条，该导航条是通过配置文件创建

/\* app.json \*/

pages:[], //组件访问路径

windows:{}, //组件配置

tabBar:{

"position": "bottom", //tabBar位置

"color": "#000",

"selectedColor": "#fc0",

"list":[{ //第一组

"pagePath": "pages/home/home", //组件路径

"text": "首页", //显示在tabBar上的组件名

"iconPath": "iconPath", //显示tabBar上的图标

"selectedIconPath": "selectedIconPath" //被选中图标样式

},{ //第二组(至少两组)

"pagePath": "pages/profile/profile",

…

}]

}

**!注意：**tabBar里的图标图片存在本地即可；list中至少有两个对象；要在tabBar中显示地组件必须在pages中正确配置。

## 五、phonegap--混编

### 5.1 phonegap概述

问题：移动端技术[bootstrap;vue;小程序…]不能直接调用移动端设备底层硬件，如：加速传感器、录音、录视频…

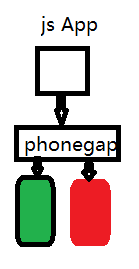
解决：实现对底层硬件调用

> 方式一：学习新语言，苹果端：用Object-C；安卓端：Java

> 方式二：类似于通用平台的软件，如phonegap

js程序 ——> phonegap ——> 调用底层硬件

phonegap是一种使用前端技术(js)创建的移动平台，该平台封装了很多JS API，可以使用前端JS调用底层设备。



#### 5.2 创建phonegap环境

#### 5.2.1 服务器

上传程序模板

点击phonegap启动服务器+查找程序模板位置

Open existiong PhoneGap project

启动服务器(会给出服务器地址，如：<http://172.16.10.1:3000>)

**!注意：**如果状态图为绿色代表启动服务器成功，为黄色表示服务器冲突

#### 5.2.2 手机模拟器

夜神模拟器

上传andriod app

点击app连接phonegap

### 5.3 编写phonegap程序

#### 5.3.1 device硬件设备

> device.platform 当前平台

> deivice.uid 设备编号

> device.version 设备版本

#### 5.3.2 相机

* navigator.camera.getPicture(fn(url),fn,{quality:50});

> fn 拍照成功 url相片地址

> fn 拍照失败

> quality：50 延迟时间

EX：

/\* 拍摄照片并显示在页面的img标签上 \*/

function setAvatar(){

navigator.camera.getPicture(

function(url){

var pic = document.getElementById("myImg");

pic.src = url;

},function(){},{quality:50}

)}

#### 5.3.3 提示蜂鸣震动

> navigator.notification.alert() 提示框

> navigator.notification.confirm("内容",function(e){}) 确认框

> navigator.notification.beep(次数) 蜂鸣器

> navigator.notification.vibrate(时长-毫秒数) 震动

function getNotification(){

navigator.notification.alert("注意");

navigator.notification.confirm("是否关机",function(e){});

//连续3声蜂鸣

navigator.notification.beep(3);

//连续震动1秒

navigator.notification.vibrate(1000);

}

#### 5.3.4 录音

* navigator.device.capture.captureAudio(fn(list),fn,{limit:2});

> fn(list) 成功录音结束 list音频文件路径

> fn 录音失败

> {limit:2} 一次录两段音频

EX:

function record(){

navigator.device.capture.captureAudio(

function(list){

alert(list.length);

alert(list[0].name); //音频文件默认名

},

function(){},

{limit:2}

)}

#### 5.3.5 加速传感器

* navigator.accelerometer.getCurrentAcceleration(fn,fn);

> fn 获取成功——>返回坐标参数x;y;z

> fn 获取失败

## 六、微信公众号

### 6.1 微信订阅号概述

#### 6.1.1 微信小游戏

在微信底层的基础之上，加了高级的游戏引擎制作而成。

游戏引擎：Cocos Creator;Egret;Layout

#### 6.1.2 微信订阅号

概念：订阅号是一个个人媒体平台，每天可向粉丝发送一条消息(包括：文章、音频、视频文件)。

订阅号分类：八卦号、历史讲解、揭示内幕、小众(白骨精…)、热门("咪蒙情感"…)

创建个人订阅号：注册邮箱，填写身份证号。

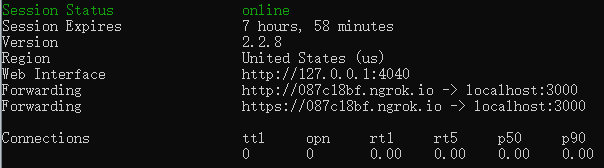
### 6.2 订阅号测试账户开发(问答工具)

开发者工具 ——> 公众平台测试账户，主要步骤如下：

* 创建服务器并且有合法域名[内网穿透工具]

> cmd到ngrok目录下输入命令：ngrok.exe http 3000

> 申请到的合法域名：<https://743f7eb4.ngrok.io>



* 下载第三方微信聊天模块

> node\_modules/wechat

* 使用聊天模块调用模块方法完成聊天操作

let config = {

appid:"wx31c2f70c58017a68 ",

token:"weixin", //令牌

encodingAESKey:""

};

//所有微信发送的请求都是post请求

app.post("/",wechat(config,(req,res)=>{

req.weixin.Content 服务器发送文字

res.reply(返回文字);

}));

> token，令牌(一般为字符串)，与微信公众平台的接口配置信息一致即可。



* 在测试帐户中添加域名和主机地址(如上图)

> 接口配置信息

- url:[合法域名] https://743f7eb4.ngrok.io

- token:[weixin] 在开发配置项 token一致

> JS接口安全域名修改

- url:[合法域名] 743f7eb4.ngrok.io

* 邀请其它用户测试

【TIPS】下载地址：

> 内网穿透工具：<https://ngrok.com/>

> 第三方微信聊天模块：npm i wechat

【TIPS】聊天工具流程图：



【常见错误】

> 在公众号平台上配置不成功

在第4步之前启动 node.js 服务器，测试连接发送给node.js get 请求

app.get("/"); 测试连接

app.post("/"); 聊天

故：在发送post请求进行聊天之前需要先用get测试连接。

> 服务器故障

node.js 服务器停止或者代码出错

ngrok.exe 重新申请域名

# 阶段项目：微信小程序——学子商城(待整理)

## 一、首页home

### 1.1 顶部导航栏

颜色值：#ff6610

> 修改window下的navigationBarBackgroundColor属性。

**!注意：**所有组件的顶部导航栏颜色相同，故在app.json文件中进行配置。

### 1.2 轮播图

swiper

<view class="slider"> /\* 自定义样式 \*/

<swiper autoplay interval='3500' indicator-dots>

<block>

<swiper-item wx:for="{{banners}}" wx:key="{{item.id}}">

<image src="{{item.img\_url}}" mode="aspecfill"

lazy-load></image>

</swiper-item>

</block>

</swiper>

</view>

### 1.3 九宫格

<view class="index-nav"> /\* 自定义样式 \*/

<block>

<view class="nav-item" wx:for="{{navs}}" wx:key="{{item.id}}">

<view wx:if="{{item.nav\_url !== ''}}">

<image src="{{item.nav\_url}}" mode="aspecfill"

lazy-load></image>

</view>

<text>{{item.nav\_name}}</text>

</view>

</block>

</view>

### 1.4 两个按钮

就是两张图片，添加<image>标签，再用ajax从服务器请求图片地址即可。

### 1.5 常见问题

问题1：手机上浏览不到小程序中图片

解决1：修改小程序工具配置不验证域名

> 工具菜单->项目详情->勾选中[\*]不校验合法域名

问题2：手机上预览图片不显示

解决2：修改请求图片地址不能 127.0.0.1

data:{

list:[{img\_url:"http://127.0.0.1/1.jpg"}]

}

> 查询本机IP地址：按钮键盘[window+r]->cmd->ipconfig

data:{

list:[{img\_url:"http://172.163.100.145/1.jpg"}]

}

> 自己手机开一个热点,笔记本连到热点

ipconfig 无线局域网适配器 ..

> 重新编译预览上小程序

【课后作业】home组件添加按二个按钮 "购物" "订单"；商品新闻列表。

## 二、美食页food

### 2.1 美食列表组件

* 标签及数据内容：日期;图版;标题;内容
* 要求及步骤：

> 创建空组件

> 点击home 组件中"美食"按钮跳转新闻列表组件

> 添加回退功能

### 2.2 发送ajax及模拟数据返回

> 在服务器创建模拟3条数据 /newslist

[{ctime:"",img\_url:"",title:"",desc:""}...]

> 在food组件接收返回数据

【课后作业】小程序学子商城新闻列表样式；小程序学子商城新闻详细

## 三、商品页

### 3.1 商品注册

--注册商品编号和商品价格 undefined

-(发送参数失败)请示头参数

network->请求头

-(响应数据失败)

### 3.2 商品列表(分页)

-组件页面结构;wxss

-模拟数据

-如何完分页

# 阶段总结：Vue框架总结：

## 1.Vue生态圈(技术栈)

* vue + vueRouter + vuex + axios + Element/Mint

> vuex官方的插件，集中式的管理数据：

- weex 通过vue的语法来实现原生开发nativeApp

- vux 移动端的UI库(类似Mint)

<http://github.com/vuejs/awesome-vue>

## 2.Vue

### 2.1 Vue组件的创建和使用

Vue.component('',{})

Vue.component(''，{

components:{

//局部组件

}

})

### 2.2 基本语法

{{}}

v-for/v-if/v-bind :/v-on @/v-model

data/methods/watch/computed/props

### 2.3 组件间通信

* 父——>子：属性下行props down

<son SonName='zhangsan'></son>

props:['sonName']

this.sonName

* 子——>父：事件上行events up

绑定：handleEvent(msg){}

<son @myEvent='handleEvent'></son>

触发：this.$emit('myEvent','123');

* 兄弟：bus

var bus = new Vue()

bus.$on('myEvent',(msg)=>{})

bus.$emit('myEvent',123)

### 2.4 与远程服务器端的通信工作

axios:非官方，先下载安装再使用 npm i axios

Axios.get()/post().then(()=>{})

### 2.5 路由模块

本质：建立起url和组件之间的映射关系

#### 2.5.1 基本用法

指定容器 router-view

创建组件 \*\*.vue

配置路由词典 const myRoutes = [

{path:'',component:\*\*}，{}

]

配置路由器 myRouter = new VueRouter({

routes:myRoutes

})

调用路由器 new Vue({

router:myRouter

})

#### 2.5.2 跳转和传参

this.$router.push()

router-link to=''

/detail --->/detail/:id

this.$route.params.id

#### 2.5.3 嵌套

一个路由对象嵌套了其它的路由对象

> 给A组件制定一个容器 router-view

> 配置子路由：

{

component:Acomponent,

children:[

{},{}

]

}

#### 2.5.4 路由守卫

* 全局守卫

router.beforeEach((to,from,next)=>{

next(true/false) //true->放行；false->禁止访问

})

* 路由独享守卫

{

beforeEnter: ()

}

### 2.6 vuex

在vue推出的集中式的以状态方式来管理数据的插件

state/getters/mutations/actions/modules

存储数据state：

在state中数据基础之上，派生新的数据

修改数据：

异步操作

将数据仓库进一步的模块化

张众磊 [bjzhangzl@tedu.cn](mailto:bjzhangzl@tedu.cn)

**课程大纲：**

angular以及ionic

reactjs以及ReactNative

python以及django

# 第十八章 Angular

## 一、Angular概述

### 1.1 推荐网站

官方网站：angular.io/angular.cn

推荐网站：

* 基于angular的项目集合:

> <https://github.com/gdi2290/awesome-angular>

* ng-cli所支持创建的类:

> <https://github.com/angular/angular-cli/wiki/generate>

* 基于vue的过滤器的项目:

> <https://github.com/freearhey/vue2-filters>

* 货币符号的国际标准:

> <https://en.wikipedia.org/wiki/ISO_4217>

* weex 原生应用

> <http://weex.apache.org/cn/guide/>

* angular组件间通讯的官方说明

> <https://angular.cn/guide/component-interaction>

* angular路由守卫补充

> <https://angular.cn/guide/router>

* reactjs的著名的第三方工具集合

> <https://github.com/enaqx/awesome-react>

* babel

> <https://www.babeljs.cn>

* RN官网

> <http://facebook.github.io/react-native/>

* RN资源整理

> <https://github.com/jondot/awesome-react-native>

* ReactNavigation官网

> <https://reactnavigation.org/>

* python手册：

> <https://docs.python.org/3.6/tutorial/index.html>

> <https://docs.python.org/3.6/library/index.html>

### 1.2 Angular是什么

angular是一套框架，适用于多种平台 移动端&桌面端

在学完了Angular框架的使用之后，可以使用NativeScript做跨平台开发，使用Ionic做移动端的开发，用material或者ngZorro来实现PC端的开发

### 1.3 为什么使用Angular

ts(typeScript)是es6的超集，是一种强类型约束的语言。它主要的优点有：

> 支持组件化

> 更好的可读性和可维护性

> 更好的社区支持和文档支持

> 体积更小，即加载快，用户体验更好

> 强大的命令行支持cli(command line interface)

### 1.4 什么时候使用Angular

spa(single page application) 单一页面应用程序，适合于中大型项目。

### 1.5 搭建Angular开发环境

* 搭建开发环境的命令行：

npm install -g @angular/cli

ng new my-app

cd my-app

ng serve --open

### 1.6 准备工作

将模板项目放到指定文件夹C:\xampp\htdocs\framework\1\_ng\day01；解压缩到当前文件夹，通过vscode打开这个工程

> 启动Angular项目： npm start

> 访问路径： localhost:4200

* 启动流程

main.ts 启动的根模块是app.module.ts导出的模块；在app.module.ts所指定的根组件app.component.ts，在这个组件模板内容app.component.html

## 二、Angular基本使用

* 明确：Angular的开发是以模块为基本单位，模块中可以包含组件、指令、管道…

### 2.1 angular相关概念

* 定义：组件是可被反复使用的，带有特定功能的视图

#### 2.1.1 angular使用配置

> 环境变量路径

打开我的电脑点击鼠标右键--》属性--》高级系统设置--》环境变量--》path-->编辑==》新建，添加路径为：

C:\xampp\htdocs\framework\1\_ng\day01\myngapp\node\_modules\.bin

**!注意：**要记得重新启动vscode!

> 修改默认shell

"terminal.integrated.shell.windows": "C:\\Windows\\System32\\cmd.exe"

> 在命令行创建组件

在命令行中输入命令，创建组件

#### 2.1.2 组件、装饰器、元数据定义

* 组件：是可被反复使用的，带有特定功能的视图
* 装饰器：是为了描述当前的类是一个什么样的类

@Component @NgModule @Directive …

* 元数据：是为了告诉Angular如何来处理当前这个类

### 2.2 组件的创建和使用

* 命令行创建组件：

ng generate(g) component demo01

(创建了组件、到根组件声明)

* demo01.component.ts

import {Component} from '@angular/core'

//装饰器

@Component({

//metaData 元数据

selector:'app-demo01', //组件名

templateUrl:'',

styleUrls:['']

})

// 导出组件模块以供使用

export class Demo01Component{}

* 组件的使用

直接将组件类中所指定的selector属性对应的值，作为标签来使用

### 2.3 angular基本语法

#### 2.3.1 {{}}

将表达式expression执行的结果输出到标签中显示：

<any>{{expression}}</any>

#### 2.3.2 循环

<any \*ngFor="let tmp of list;let myIndex=index"> </any>

#### 2.3.3 条件

<any \*ngIf="expression"></any>

ex:<p \*ngIf="isMember"> 这是会员专区 </p>

【练习】

> 步骤1：创建一个组件demo02-lianxi

ng generate component demo02-lianxi

> 步骤2：在根组件的模板中调用新建的组件

> 步骤3：在demo02的组件类中，创建一个由3个学生对象(sName sScore)构成的数组，在模板中去循环创建多个p，在p中显示学生的姓名

问题：循环指令和条件判断指令不可以同时使用

解决：使用ng-container，它是一个容器，不会影响DOM结构

#### 2.3.4 绑定

##### 2.3.4.1 事件

<any (EventName)="handleEvent()"></any>

##### 2.3.4.2 属性

<any [myAttr]="expression"></any>

##### 2.3.4.3 双向数据绑定

* 准备工作

让核心模块 来调用表单模块，表单模块中的东西就可以在核心模块中调用！

> 找到app.module.ts,引入FormsModule

import {FormsModule} from '@angular/forms'

> 修改根模块的元数据

@NgModule({

imports:[FormsModules]

})

* 双向绑定：

<表单元素 [(ngModel)]="someData" (ngModelChange)=""></表单元素>

**!注意：**在绑定ngModelChange事件要注意先后顺序，即要先双向绑定数据，才能将数据经过事件处理函数ngModelChange的参数传回进行相关处理。

【练习】

> 步骤1：完成demo04-lianxi的组件的创建和调用

> 步骤2：组件的模板内容:input/button/ul

> 步骤3：点击button时(事件绑定)，将输入框的值(双向数据绑定)，添加到ul的li中显示

### 2.4 指令 direct

组件和指令的关系：组件就是一个带有模板内容的指令。

// vue:

Vue.directive('',{})

#### 2.4.1 指令的创建及调用

* 创建：

ng g directive my-change

* 调用：

找到指令类的selector中括号中的值：appMyChange

<any appMyChange></any>

#### 2.4.2 获取调用指令的元素

import { ElementRef } from '@angular/core'

constructor(private eleRef:ElementRef){

}

this.eleRef.nativeElement

#### 2.4.3 指令传参及其获取

* 发送：

<any appMyChange="123"></any>

* 接收：

import {Input,OnInit} from '@angular/core'

export class MyChangeDirective implements OnInit{

@Input() appMyChange=""

ngOnInit(){

//this.appMyChange

}

}

【练习】

> 步骤1:创建一个自定义的指令change-bg

> 步骤2:创建一个组件demo06-lianxi，在此组价中写h2 调用自定义指令，并传值red

> 步骤3：在自定义的指令内部接收传递的数据，修改当前调用指令的元素的背景色

### 2.5 管道 pipe

* vue：

vue2.0之后不再内置过滤器，交给计算属性

> 语法:

<any>{{expression | myFilter(1,2) | myFilter2}}</any>

Vue.filter('',(msg,arg1,arg2)=>{

//数据处理

return 处理后的结果

})

* Angular：

管道的本质是一个有参数有返回值的方法，意义是为了对要显示的数据执行筛选、过滤、格式化...

#### 2.5.1 内置管道

* 语法：

<any>{{expression | myPipe:arg1:arg2 | myPipe2}}</any>

* 具体：

> uppercase

> lowercase

> slice:start[:end]

> currency:'CNY/EUR'

> number

> percent

> json

#### 2.5.2 自定义管道

* 创建：

ng g pipe test

* 配置：

test.pipe.ts

在transform方法中实现管道具体的需求

* 使用：

<any>{{expression | test}}</any>

* 补充:

自定义管道接收参数时，支持剩余参数的指定

transform(value:any,...args:Array<any>){

//args

}

【练习】

> 步骤1:创建一个自定义的管道sex

> 步骤2:在组件demo07-pipe中写h2 调用自定义管道，传递1-->男 0-->女

### 2.6 服务 service

服务类:将各个组件要复用的数据或者方法封装在此，以方便不同的组件内部的复用

#### 2.6.1 创建服务

ng g service test

**!注意：**装饰器是 @Injectable

#### 2.6.2 配置服务

在服务类中 封装要复用的数据和方法

#### 2.6.3 使用服务

组件是服务最大的消费者

* 引入

import {TestService} from '../test.service'

* 实例化

constructor(private myService:TestService){}

* 调用

this.myService.\*\*

【练习】

> 步骤1：创建一个服务 heart-beat

在此服务中，封装两个方法，startBeat/stopBeat

开始心跳：启动一个周期性定时器，在终端输出'正在心跳...'

结束心跳: 结束定时器的运行

> 步骤2：在demo08-service组件中，写两个按钮，能够在按下时，调用服务中封装的两个方法

### 2.7 生命周期钩子函数

* angular组件的lifeCycle:

> ngOnInit 负责初始化

> ngOnDestroy 负责清理工作

> ngOnChanges 负责在数据变化时，做逻辑判断和数据处理的

## 三、组件间通信及网络通信

### 3.1 父-->子 props down

* 原理：在调用子组件时，给子组件绑定自定义属性，并将要传给子组件的值给该属性作为属性值传递给子组件；子组件中创建相应变量，接收该属性值即可。

> 父组件中调用子组件，并绑定属性将值传递给子组件：

<son [sonName]="michael"></son>

> 子组件中创建sonName(变量名必须和父组件中绑定的属性名一致)变量，并通过其获得传过来的值：

import { Input } from '@angular/core'

@Input() sonName = ""

console.log(this.sonName);

### 3.2 子-->父 events up

* 原理：父组件在调用子组件时，给子组件绑定自定义事件，并将此事件"导出"给子组件中某个触发事件，当用户触发了子组件的这个事件，即将子组件的值通过自定义事件传递给父组件。

> 绑定事件(父组件调用子组件，给子组件绑定事件)

<son (myEvent)="handleEvent($event)"></son>

handleEvent(msg){

//msg就是事件触发是传递来的数据

}

> 触发事件(子组件内部)

import { Output } from '@angular/core'

@Output() myEvent = new EventEmitter()

this.myEvent.emit(123/\*要传递回父组件的数据\*/);

【练习】

> 步骤1：创建两个组件类

demo10-user: label+input+button

demo10-chatroom :ul+user组件

> 步骤2：在根组件中调用chatroom组件，在chatroom组件中调用user组件

> 步骤3：调用user组件，通过属性传值（lucy），让user组件的label显示

> 步骤4：点击user组件的按钮时，将input的值，传递到父组件，父组件接收到值，保存在数组，将数组和ul的li绑定(\*ngFor)

### 3.3 父组件主动从子组件获取数据

#### 3.3.1 local variable

上述章节中可以通过子组件中的指定事件触发向父组件"发送"数据，但是父组件处于"被动"的状态，以下讲解父组件主动从子组件中获取数据的方法。

* 给子组件指定一个本地变量(local variable)

/\* demo01.parent.component.html \*/

<son #mySon></son>

#### 3.3.2 ViewChild

* 在父组件类中，通过viewchild得到组件组的实例对象mySon，进而做一些操作

/\* demo01.son.component.ts \*/

sonName = "Michael"

/\* demo01.parent.component.ts \*/

import { ViewChild } from '@angular/core'

@ViewChild('mySon') demo01

console.log(this.mySon.sonName);

上述ViewChild的使用只是其中一种：

> 如果找子组件有本地变量：

@ViewChild('mySon') demo11

> 如果找子组件没有指定本地变量

import {Demo11SonComponent} from '../\*\*\*'

@ViewChild(Demo11SonComponent) demo11

**!注意：**只能在模板中使用；适合父主动的从子获取数据。

### 3.4 service(?)

单个实例模式(单例模式)，在一个app中，不同的组件去实例化同一个服务，对应的都是一个对象

目标：

在a组件初始化 将服务的实例数据操作

在b组件 放按钮，点按钮，去服务的示例的数据

### 3.5 网络通信概述

常用网络通信方法：ajax，$.get...，Axios，fetch，wx.request

异步操作数据的方式：callback,promise,aysnc/await/RxJS(RxJS是使用Observables的响应式编程库，它使编写异步或基于回调的代码更容易，且比promise的兼容性)

设计原则：单一责任，高内聚 低耦合，最少知识法则，开闭原则

设计模式：（观察者模式、单例模式、工厂模式）《大话设计模式》

> 创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

> 结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

> 行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

### 3.6 HttpClient

* 让根模块依赖于HttpClientModule

import {HttpClientModule} from '@angular/common/http'

@NgModule({

imports:[HttpClientModule]

})

* 在组件中引入HttpClient类，实例化 ，执行网络请求

import { HttpClient } from '@angular/common/http'

constructor(private myHttp:HttpClient){}

this.myHttp.get/post().subscribe(()=>{})

【练习】

> 步骤1：创建两个组件类

demo13-login

demo13-header

> 步骤2：

在根组件中调用login，在login中调用header、button

> 步骤3：

在login中调用header完成标题的通讯

> 步骤4：

在login中点击按钮，向服务器发起请求

## 四、路由模块

本质：建立url和组件之间的映射关系

SPA:在一个app，有一个完整的页面，在该页面中指定一个容器，在访问不同的页面，根据url从路由词典中匹配对应的组件，加载显示在容器中

### 4.1 基本用法(配置路由词典)

* 两种方案：

> 方案1：把所有的路由设置都放在根模块 （适合组件比较少）

> 方案2(推荐)：将路由设置单独的放在一个模块 （适合大型的项目）

#### 4.1.1 路由配置及调用

* 创建一个用来配置路由的自定义的模块

ng g module my-router --routing

* 指定一个盛放组件的容器

<router-outlet></router-outlet>

* 创建组件

ng g component demo14-login

ng g component demo14-register

* 路由词典：一个多个路由对象构成的数组

在新建的模块中my-route.routing.module.ts

const routes:Routes = [

{path:"",component:\*\*}

]

* 调用路由词典

在新建的模块中my-route.routing.module.ts

@NgModule({

imports:[RouterModule.forRoot(routes)]

})

* 让根模块调用刚配置过的自定义模块

import {MyRouterRoutingModule} from '\*\*'

app.module.ts

@NgModule({

imports:[MyRouterRoutingModule]

})

* 测试

localhost:4200

localhost:4200/login

localhost:4200/register

【练习】

> 步骤1：两个组件

demo15-list

demo15-detail

> 步骤2：给两个组件指定不同的路由地址

list/detail

**!注意：**路由地址不能以/开头，否则是会报错的

#### 4.1.2 路由匹配规则

> 路由地址为空

{path:'',component:LoginComponent}

> 某一个具体的路由地址

{path:'login',component:LoginComponent}

> 异常情况

{path:'\*\*',component:IndexComponent}

> 设置属性

path/component/redirectTo/children/canActivate

### 4.2 跳转 Router routerLink

Vue:

this.$router.push/go/..

<router-link to='/login'></router-link>

Angular:

* 方式1：编程式跳转

import {Router} from '@angular/router'

constructor(private myRouter:Router){}

this.myRouter.navigateByUrl("main17/"+this.uname);

* 方式2：标签式跳转

<any routerLink="/desPath"></any>

### 4.3 传参 ActivatedRoute

#### 4.3.1 vue回顾

* vue传参基本步骤：

> 配置接收方的路由地址

/detail --> /detail/:id

> 传参

/detail/10

> 接收参数

this.$route.params.id

#### 4.3.2 angular传参

* Angular中传参：

> 配置接收方的路由地址

/detail --> /detail/:id

> 传参

/detail/10

> 接收参数

import {ActivatedRoute} from '@angular/core'

constructor(private myRoute:ActivatedRoute){}

this.myRoute.params.subscribe((result)=>{

//result.id

})

#### 4.3.3 动态拼接参数

this.myRouter.navigateByUrl('detail/10')

this.myRouter.navigateByUrl('detail/'+this.id)

<any routerLink="/detail/10"></any>

<any [routerLink]="'/detail/'+id">

</any>

【练习】

> 步骤1：准备两个组件并配置路由

demo18-list demo18-detail

demo18List demo18Detail

> 步骤2：在list组件中，在组件初始化的时候，请求用户列表

http://jsonplaceholder.typicode.com/users

将请求到的数组保存起来，显示在无序列表(列表项显示用户的name)

> 步骤3：点击列表项，跳转到demo18Detail

同时将当前用户id传递给详情页

> 步骤4：在详情页 接收路由传递的参数 并渲染在h2

### 4.4 路由嵌套

在一个spa的某一个页面中，嵌套其它的页面(一个路由对象中 可以嵌套其它的路由对象)。

需求：比如A，需要嵌套B或者C

> 步骤1：

在A组件指定一个容器

<router-outlet></router-outlet>

> 步骤2：

给B、C组件指定路由

{

path:'a'

component:A,

children:[

{path:'b',component:B},

{....}

]

}

【练习】

在路由地址为mail对应的组件demo19-mail中，通过子路由inbox/outbox来访问demo19-inbox、demo19-outbox

**!注意：**在访问子路由，得带上父级路由的路由地址，比如: <http://localhost:4200/mail/inbox>

### 4.5 路由守卫

路由守卫，保护指定路由地址的组件是否可被访问

可以去在一个服务中 实现CanActivate类，在方法中通过返回值的true、false来告诉angular是否允许访问

> 步骤1：创建一个服务

ng g service my-guard

> 步骤2：在服务中实现一个CanActivate接口类中的方法canActivate

> 步骤3：在方法canActivate中 执行运算，返回给true（可以访问）/false （禁止访问）

import {CanActivate} from '@angular/router'

@Injectable({})

export class MyGuardService implements CanActivate{

canActivate(){

//计算

return true/false

}

}

> 步骤4：在需要保护的路由对象中添加了属性

canActivate

{

path:'admin',

component:Demo14AdminComponent,

canActivate:[MyGuardService]

}

【练习】

> 步骤1：完成组件类的创建

demo22-todobox

demo22-todoinput

demo22-todolist

demo22-todoitem

> 步骤2：

给demo22-todobox指定一个路由地址 todobox

> 步骤3：实现添加功能

点击todoinput中的add按钮时，将输入框的值 通过events up的方式传递到todobox； (Output EventEmitter)

todobox接收到值，保存在数组，通过props down传递给todolist(Input)

todolist接收到todobox传递来的数据，循环创建多个todoitem，让todoitem显示正确的内容

## 五、angular总结

* 组件：

可被反复使用的 带有特定功能的视图

* 服务：

将不同的组件 需要复用的数据或者方法

放在一个服务类中，方便模块化、提供代码的 复用率

* 依赖注入:Dependency Injection

可以专心的去使用关心的类，无需理会背后是如何出来；

* 装饰器与元数据

装饰器：描述一个类的功能

元数据：指定类的具体的配置信息，告诉angular如何处理当前这个类

* 模板

增强后的html

* 指令

给模板添加逻辑控制或者绑定、

* 数据绑定

() [] [(ngModel)]

\*ngFor \*ngIf...

* 管道

在模板中显示数据时，通过管道来实现筛选

过滤、格式化效果

# 第十九章 Ionic组件库

## 一、Ionic概述

### 1.1 什么是Ionic

Ionic是基于angular的移动端UI组件库。

Ionic = angular + 封装的UI组件库 + icon + phonegap

PC端的UI组件库一般是：MD Zorro <https://material.angular.cn/>

<https://ng.ant.design/>

ionic图标库：<https://ionicons.com/> (可直接根据名字在代码中使用)

### 1.2 Ionic的特点

> 基于angular、phonegap

> 支持ts

> 丰富的组件和icon

> 强大的cli支持

### 1.3 Ionic的使用

#### 1.3.1 环境搭建

npm install -g ionic

ionic start myApp tabs/sidemenu/blank

cd myApp

ionic serve

#### 1.3.2 编程方式

所有的逻辑基于angular

ui来通过ionic中内置的组件和icon

#### 1.3.3 准备工作

> 将myProject.zip拷贝到指定的目录:C:\xampp\htdocs\framework\2\_ionic

> 解压缩到当前文件夹

> 用vscode打开C:\xampp\htdocs\framework\2\_ionic\myProject下的工程

> 启动开发服务器

npm start

#### 1.3.4 Ionic页面的创建和使用

* 步骤1：创建页面

ionic g page demo01

* 步骤2：声明

找到app.module.ts声明一下Demo01Page

import {Demo01Page} from '../pages/demo01/demo01'

@NgModule({

declarations:[Demo01Page],

entryComponents:[Demo01Page]

})

* 步骤3：调用

<page-demo01></page-demo01>

或者

app.component.ts

import {Demo01Page} from '../pages/demo01/demo01'

rootPage:any = Demo01Page

【练习】

> 步骤1：创建一个页面 demo02-lianxi

> 步骤2：在demo02-lianxi中的页面类中准备一个整数数组 myList=[1,2,3,4]；将数组中显示demo02-lianxi的模板的ul

> 步骤3：将demo02-lianxi的页面类设置为默认要显示的页面

### 1.4 Ionic的生命周期函数

* 以下两个钩子函数只会被执行一次:

> ionViewDidLoad 挂载

> ionViewWillUnload 准备卸载

* 以下函数可以执行很多次:

> ionViewCanEnter 守卫：能不能加载

> ionViewWillEnter

> ionViewDidEnter

> ionViewCanLeave 守卫:能不能离开

> ionViewWillLeave

## 二、Ionic组件

### 2.1 button

* button相关的样式如下：

<!-- secondary 绿色 src/theme-->

<!-- block 占满宽度 -->

<!-- round 圆角 -->

<button ion-button block round color="myColor">IONIC按钮(占满宽度 圆角 自定义粉色)</button>

<button ion-button clear>可点击文本(没有按钮边框)</button>

<button ion-button outline>可点击文本(有按钮边框)</button><br />

<button ion-button icon-left>

<ion-icon name="home"></ion-icon>

主页

</button>

<button ion-button icon-right>

设置

<ion-icon name="cog"></ion-icon>

</button>

<br />

<button ion-button block round color="secondary">在线支付</button>

<button>普通的按钮</button>

* 对应效果如下：



* 使用自定义颜色的方法：

> 路径：../src/theme/variables.scss $colors

$colors: (

myColor: #fba,

);

### 2.2 list

#### 2.2.1 图标列表

* 用法示例：

<ion-list>

<ion-item \*ngFor="let tmp of list">

<ion-icon name="home" item-start></ion-icon>

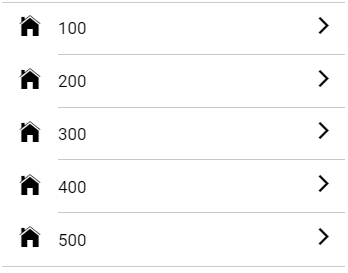
{{tmp}}

<ion-icon name="arrow-forward" item-end></ion-icon>

</ion-item>

</ion-list>

* 对应效果图：



#### 2.2.2 头像列表

常用于头像或者联系人列表：

* 用法示例：

<ion-list>

<ion-item \*ngFor="let tmp of list">

<ion-avatar item-start>

<img src="../../assets/imgs/logo.png" />

</ion-avatar>

{{tmp}}

</ion-item>

</ion-list>

* 对应效果图：



#### 2.2.3 缩略图列表

* 用法示例：

<ion-list>

<ion-item \*ngFor="let tmp of list">

<ion-thumbnail item-start>

<img src="../../assets/imgs/logo.png" />

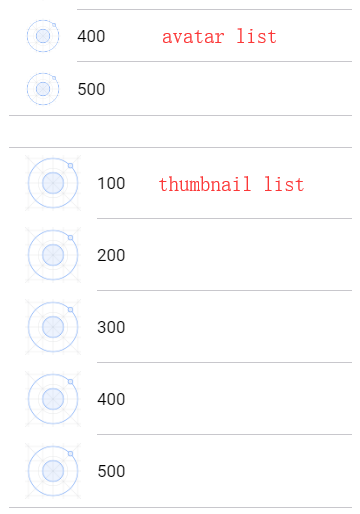
</ion-thumbnail>

{{tmp}}

</ion-item>

</ion-list>

* 对应效果图：



#### 2.2.4 支持侧滑动的列表

* 用法示例：

<ion-list>

<ion-item-sliding \*ngFor="let tmp of list">

<ion-item>{{tmp}}</ion-item>

<ion-item-options side="right"> /\* side默认值为right \*/

<button ion-button color="danger">删除</button>

</ion-item-options>

<ion-item-options side="left">

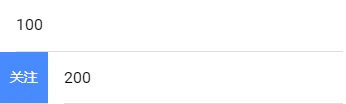
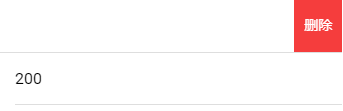
<button ion-button>关注</button>

</ion-item-options>

</ion-item-sliding>

</ion-list>

* 对应效果图：



### 2.3 下拉刷新、上拉加载更多

#### 2.3.1 下拉刷新

> 步骤1：将ionRefresher放在ionContent的第一个子元素的位置

<ion-refresher>

<ion-refresher-content pullingText="" pullingIcon="">

</ion-refresher-content>

</ion-refresher>

> 步骤2：绑定一个特定的事件

<ion-refresher (ionRefresh)="handleRefresh($event)">

</ion-refresher>

> 步骤3：在事件处理函数中，对下拉刷新要执行的业务逻辑进行处理，处理完之后要记得调用complete()

handleRefresh(myRefresher){

//异步操作数据完成之后

//结束刷新动作

myRefresher.complete()

}

#### 2.3.2 上拉加载更多

> 步骤1：将ionInfiniteScroll放在ionContent的最后一个子元素的位置

> 步骤2：绑定一个事件ionInfinite

> 步骤3：在事件处理函数执行加载更多的操作，在异步操作技术之后，通知结束加载动作

handleInfinite(myInfinite){

//异步操作

//结束

myInfinite.complete()

}

### 2.4 自定义布局

ionGrid ionRow ionCol

#### 2.4.1 基本用法

<ion-grid>

<ion-row>

<ion-col></ion-col>

<ion-col></ion-col>

</ion-row>

<ion-row></ion-row>

</ion-grid>

#### 2.4.2 常见设置

> col-\*

> align-items-center/start/end 行中所 有列内容的纵向对齐

> align-self-center/start/end 某一列在行中的纵向对齐方

> justify-content-center/start/end 行中列的横向对齐

> offset-\* 偏移量

> push/pull-\* 右推/左拉

【练习】

> 步骤1：完成demo06-lianxi的创建、声明、调用

> 步骤2：在demo06的组件类中 准备一个数组，数组有20条数据

> 步骤3：渲染在模板中，一行3列

### 2.5 轮播图

#### 2.5.1 基本用法

<ion-slides pager=true autoplay=1000 loop=true effect='flip' speed=2000 paginationType="progress" direction='vertical'>

<ion-slide>123</ion-slide>

<ion-slide>456</ion-slide>

<ion-slide>789</ion-slide>

</ion-slides>

#### 2.5.2 常见设置

> ionSlides

> pager 设置是否显示指示器

> pagingationType 指示器类型 bullets/fraction/progress

> effect 指定动画效果 slide/flip/cube

> speed 动画时间

> direction 滚动的方向

> autoplay 自动播放的时间

> loop 是否要循环播放

### 2.6 卡片

<ion-card>

<ion-card-header></ion-card-header>

<ion-card-content></ion-card-content>

</ion-card>

### 2.7 常见交互窗口

#### 2.7.1 loading

* 作用：表示一个活动正在进行
* 使用：

> 步骤1：引入

import {LoadingController} from 'ionic-angular'

> 步骤2：实例化

constructor(private loadingCtrl:LoadingController){}

> 步骤3：创建和显示

var myLoading = this.loadingCtrl.create({

spinner:'',//ios ios-small bubbles circles crescent dots

content:'',

duration:\*\*\*

})

myLoading.present()/dismiss()/setContent()

#### 2.7.2 actionSheet

* 作用：有多个选择需要用户判断
* 使用：

> 步骤1: 引入

> 步骤2: 实例化

> 步骤3：创建和显示

this.asCtrl.create({

title:'',

buttons:[

{

text:'',

handler:()=>{},

role:''//destructive/cancel

}]

})

【练习】

> 步骤1：在demo08中 添加一个按钮，支付

> 步骤2：点击支付按钮，显示actionSheet (微信支付/支付宝支付/易宝支付);

> 步骤3：点击actionsheet任何按钮，显示loading，3s之后自动结束

#### 2.7.3 ToastController

* 作用：给用户一个操作的结果
* 使用：

> 步骤1：引入

> 步骤2：实例化

> 步骤3：创建和显示

this.toastCtrl.create({

message:'',

position:'top/middle/bottom',

showCloseButton:true/false,

closeButtonText:'',

duration:\*\*\*,

dismissOnPageChange:true/false,

cssClass:\*\*

})

#### 2.7.4 AlertController

* 作用：提供警告、确认、输入提示
* 使用：

> 步骤1：引入

> 步骤2：实例化

> 步骤3：创建和显示

this.alertCtrl.create({

title:'',

message:'',

inputs:[],

buttons:[]

})

【练习】

> 步骤1：完成demo09-lianxi的创建、声明、调用

> 步骤2：在demo09的组件类中 准备一个数组myList=[100,200,300,400];渲染在模板中列表（列表项支持侧滑动，右侧隐藏一个删除的按钮）

> 步骤3：点击删除按钮，显示一个确认窗口，提示：是否删除该数据，点击确认，删除掉；点击取消，不用管

> 步骤4：点击确认，删除成功之后，显示一个通知，提示删除成功

#### 2.7.5 ModalController

* 作用：方便自定义弹窗 显示编辑界面或者选择内容
* 使用：

> 步骤1：引入

> 步骤2：实例化

> 步骤3：创建和显示

this.modalCtrl.create(Demo10WindowPage)

* 进阶使用：

> 关闭模态窗 ViewController

- 引入

- 实例化

- 调用dismiss方法 this.viewCtrl.dismiss()

> 模态窗在关闭时 参数的传递和接收

- 传参：this.viewCtrl.dismiss(1)

- 接收：myModal.onDidDismiss((data)=>{

//data就是传来的数据

})

### 2.8 tabs导航

* 作用：实现底部导航条
* 使用：

> 步骤1：实现底部导航条

<ion-tabs>

<ion-tab [root]="tabIndex" tabTitle="首页"></ion-tab>

<ion-tab [root]="tabCart" tabTitle="购物车"></ion-tab>

<ion-tab [root]="tabUser" tabTitle="用户中心"></ion-tab>

<ion-tab [root]="tabNotFound" tabTitle="不存在"></ion-tab>

</ion-tabs>

> 步骤2：配置tab，给每一个tab指定被选中时要显示的页面类

- 引入tab被选中时想显示的页面类

import {Demo07GridPage} from '\*\*\*'

- 页面类保存

tabIndex = Demo07GridPage

- 绑定到tab的root属性

<ion-tab [root]="tabIndex"></ion-tab>

* 对应效果图：



### 2.9 导航和传参

#### 2.9.1 导航跳转

* 方案1：通过ts

import {NavController} from 'ionic-angular'

import {DestinationPage} from '\*\*\*'

constructor(private navCtrl:NavController){}

this.navCtrl.push(DestinationPage)

* 方案2：通过属性

import {DestinationPage} from '\*\*'

desPage = DestinationPage

<any [navPush]="despage"></any>

#### 2.9.2 跳转传参

* 发送：

this.navCtrl.push(DestinationPage,{id:100})

<any [navPush]="despage" [navParams]='{id:10}'></any>

* 接收：

import {NavParams} from 'ionic-angular'

constructor(private navParams:NavParams){}

this.navParams.get('id')

【练习】

> 步骤1:创建demo13-list demo13-detail

>步骤2：在list中页面加载时候，请求服务器端的数据(HttpClient) <http://jsonplaceholder.typicode.com/users>

将数据保存，渲染ionList中，列表项只显示name

> 步骤3：点击列表项，navPush跳转到detail，并且将用户的id发送过去；detail接收到id显示在h2

### 2.10 表单

在ionic中处理表单是通过ionList和ionItem管理表单的

#### 2.10.1 input框

* input框的常用五种写法：

<ion-list>

<ion-item>

<ion-label inline>用户名：</ion-label> <!--inline默认值-->

<ion-input placeholder="请输入用户名"></ion-input>

</ion-item>

<ion-item>

<ion-label floating>用户名：</ion-label>

<ion-input></ion-input>

</ion-item>

<ion-item>

<ion-label stacked>用户名：</ion-label>

<ion-input placeholder="请输入用户名"></ion-input>

</ion-item>

<ion-item>

<ion-label fixed>用户名：</ion-label>

<ion-input placeholder="请输入用户名"></ion-input>

</ion-item>

<ion-item>

<ion-input placeholder="请输入用户名"></ion-input>

</ion-item>

</ion-list>

* 对应效果图：



#### 2.10.2 单选框

* 使用：

<ion-list radio-group>

<ion-item>

<ion-label>男</ion-label>

<ion-radio value="boy">男</ion-radio>

</ion-item>

<ion-item>

<ion-label>女</ion-label>

<ion-radio value="girl">女</ion-radio>

</ion-item>

</ion-list>

* 对应效果图：



#### 2.10.3 复选框

使用：

<ion-list>

<ion-item>

<ion-label>vue</ion-label>

<ion-checkbox></ion-checkbox>

</ion-item>

<ion-item>

<ion-label>angular</ion-label>

<ion-checkbox></ion-checkbox>

</ion-item>

<ion-item>

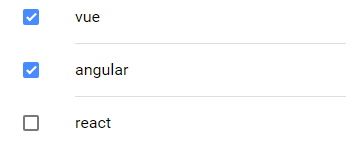
<ion-label>react</ion-label>

<ion-checkbox></ion-checkbox>

</ion-item>

</ion-list>

对应效果图：



#### 2.10.4 滑动开关

* 使用：

<ion-list>

<ion-item>

<ion-label>wifi</ion-label>

<ion-toggle></ion-toggle>

</ion-item>

</ion-list>

* 对应效果图：



#### 2.10.5 滑动组件

* 使用：

<ion-list>

<ion-item>

<ion-range min=0 max=100 pin=true>

<ion-label range-left>0</ion-label>

<ion-label range-right>100</ion-label>

</ion-range>

</ion-item>

</ion-list>

* 对应效果图：



### 2.11 支持滚动的容器

* 使用：

<ion-scroll style="height:200px;white-space:nowrap" scrollX=true>

<ion-card \*ngFor="let tmp of list" style="display:inline-block">

<ion-card-content>{{tmp}}</ion-card-content>

</ion-card>

</ion-scroll>

* 对应效果图：



## 三、项目总结:Angular以及Ionic

### 3.1 采用Angular的技术栈

angular核心库（周边模块：HttpClientModule/FormsModule/RouterModule）

rxjs 异步操作数据流

ui组件库(Ionic/NgZorro/MaterialDesign)

细分：

> 网络请求:HttpClient+Rxjs

> 工具类：service

> 模板的增强:指令、管道

> 样式效果：ionicModule(

ionSlides/ionCart/ionScroll..)

> 数据：在ts中直接在类中进行数据的基本操作(定义、读、写)

# 第二十章 ReactJS（React/React.js）

## 一、ReactJS概述

### 1.1 概述

* 官网描述：

A JavaScript library for building user interfaces。

* 特点：

ReactJS是一个构建UI层(视图)的js库，是用在网页编程上的。

* 生态圈：

react.js(核心库)+Redux+react-router+fetch…

* 使用场景：

数据操作很繁琐的场景

* Facebook团队推出React的原因：

> 浏览器性能瓶颈-->VDOM

> 开发和维护成本过高-->单向数据流

### 1.2 核心概念

* 核心：只关注视图层
* 核心思想：Everything is component
* 核心技术点：

jsx props component state ref

### 1.3 使用

* 引入三个核心js文件：

react.js react-dom.js browser.min.js

**!注意：**必须最先引用react.js核心库

## 二、核心技术

### 2.1 jsx

JavaScriptXML，并不是一种新的语言，是js的扩展

- 允许在js中直接写标签 ReactDOM.render(a,b)

- 可以在花括号中执行运算 {expression}

jsx在react.js开发中不是必须的，但是推荐使用。

**!注意：**要使用jsx需要在script标签内指定type="babel"。babel是一个编译器。

### 2.2 component 组件

#### 2.2.1 组件概述

* 组件：可被反复使用的，带有特定功能的视图
* Vue:

Vue.component(,{

components:{}

})

* Angular:

@Component()

export class Demo01Component{}

#### 2.2.2 组件的创建

var MyComponent = React.createClass({

render:function(){

return \*\*

}

})

#### 2.2.3 组件的使用

做一个标签去调用

<MyComponent></MyComponent>

**!注意：**在Reactjs中，认为标签首字母是小写，会按照普通的html标签来处理；如果首字母是大写，会按照组件来处理

#### 2.2.4 使用组件需要注意的点

> 在渲染时，标签都得有开始、结束标记

> 组件类的名称要遵循全驼峰

> 要同时渲染多个元素，放在一个顶层标签中

> render方法return要渲染的元素，第一个标签不建议换行(可以通过圆括号)

【练习】

> 步骤1：创建一个文件demo04\_lianxi.html

> 步骤2：在该文件中创建两个组件：

MyItem:渲染一个li，指定一个文本内容

MyList:渲染一个由3个MyItem构成的无序列表

> 步骤3：

将MyList作为根组件渲染出来

### 2.3 props 通信

#### 2.3.1 vue/angular通信

vue和angular中 通过props down的方式来实现父组件与子组件的通信工作

* 步骤1：发送

<son sonName='zhangsan'></son>

* 步骤2：接收

> Vue:

{

props:['sonName']//this.sonName

}

> Angular:

@Input() sonName = ""//this.sonName

#### 2.3.2 React 父-->子

* 步骤1：发送

<son sonName='zhangsan'></son>

* 步骤2：接收

this.props.sonName

#### 2.3.3 React 子-->父

通过方法传值：

* 步骤1：在父组件中准备一个有参数的方法

save:function(msg){

//msg就是子组件准备传来的数据

}

* 步骤2：将方法传递给子组件

<Son myFunc={this.save}></Son>

* 步骤3：子组件内部就可以调用父组件传来的方法，而且同时把数据传递过去

this.props.myFunc(123)

#### 2.3.4 this.props.myFunc(123)

props对象中键值对和调用时的键值对是保持一致，额外还多一个this.props.children（组件在被调用时，标签内部的子标签）

React.Children.map(

this.props.children,

(child)=>{

return child

}

)

【练习】使用reactjs实现todobox

> 步骤1：创建demo09\_lianxi.html

> 步骤2：在此文件中准备4组件

ToDoItem ToDoList

ToDoInput ToDoBox

> 步骤3：将ToDoBox作为根组件去渲染

> 步骤4：点击ToDoInput的add，将输入框的值通过子与父通信，传递到ToDoBox

### 2.4 state 状态

#### 2.4.1 state 值的绑定

* 状态：

> 管理数据(定义、读、写)

> 数据绑定

* 初始化状态：

getInitialState:function(){

return {count:0,stu:{},list:[]}

}

* 读：

this.state.count/stu/list

* 写：

this.setState({count:1, /\* stu:{},list:[] \*/ },()=>{

console.log(this.state.count)

})

**!注意：**setState()方法内部实现是一个异步的过程，此方法支持两个参数，第一个参数是要设置的状态对象，第二个参数是写操作成功之后要回调执行的方法。

**!注意：**若有多个状态，需修改其中一个状态对象，需要修改哪个状态就直接修改，其他的保持不变即可(即可以不用写出来)。

#### 2.4.2 state 值绑定样式

var MyComponent = React.createClass({

getInitialState:function(){

return {opacity:0}

},…

render:function(){

return <h1 style={{opacity:this.state.opacity}}>

this is h1

</h1>

}

})

**!注意：**样式不是通过双花括号绑定，而是一样的采用单花括号。内层花括号表示是一个对象。

【练习】

> 步骤1：创建一个组件，组件只有一个h1

> 步骤2：在组件挂载完毕之后，启动一个定时器，修改数据 0 0.1 0.2 ...~1

> 步骤3：将数据绑定到h1的style中opacity

### 2.5 ref

Vue/React中都支持通过ref的方式来在父组件中找到子组件，进而来获取子组件中的数据或者方法。

// reference 引用/参考/参照

<son ref="mySon"></son> // 可把mySon看成是son组件的实例化对象

this.refs.mySon

相当于实例化子组件对象或者是找到元素，用此对象主动获取子组件中的参数。

var Son = React.createClass({

count:0,

getInitialState:function(){

return {list:[1,2,3]}

},

render:function(){

return <h2>这是子组件</h2>

}

})

var Parent = React.createClass({

handleClick:function(){

console.log(this.refs.myInput.value)

// 准备获取子组件中的count对应的数据

var nowCount = this.refs.mySon.count;

console.log("获取到子组件中的count是：",nowCount)

console.log("list的值是：",this.refs.mySon.state.list);

},

render:function(){

return <div>

<h1>这是父组件</h1>

<input type="text" ref="myInput" />

<button onClick={this.handleClick}>clickMe</button>

<Son ref="mySon"></Son>

</div>

}

})

【综合练习】

> 步骤1：创建3个组件MainComponent AComponent BComponent

Main = A(input)+B(button)

> 步骤2：点击B组件的按钮，获取A组件输入框的值

- A组件的输入框一输入，将值传递到Main组件保存起来(onChange )

子 --输入框的值--》父

- Main组件将保存的值（状态）通过属性绑定，传递给B组件

- B组件点击按钮时，读取通过属性传来的数据

### 2.6 React组件生命周期

vue/小程序/angular/ionic…

beforeCreate/ngOnInit/IonViewWillEnter

#### 2.6.1 生命周期钩子函数

* mount:负责初始化

> componentWillMount

> componentDidMount

* update:负责在数据变化时根据业务进行逻辑判断

> componentWillUpdate

> componentDidUpdate

> componentWillReceiveProps(当组件被调用时，通过属性传来的值发生变化时，允许在此方法内修改状态)

* unmount:负责执行清理

> componentWillUnmount

#### 2.6.2 使用钩子函数注意事项

> 通过ref操作元素，必须等到DidMount中或者DidMount之后

> 组件内部的方法，建议将render方法放在类的最后边，将其他react自带的方法放在最前边，中间才是自定义的方法(比如：handleClick)

> 在update相关的钩子函数中，如果执行了状态的写操作，会陷入死循环

> 通过prop传递的值(父-->子)，在组件内部是只读的，不可对它进行修改操作；尝试将props当中的值保存在状态中，状态是允许写操作的

**!注意：**生命周期函数中使用定时器函数，this的指向问题。

> this -> window

componentDidUpdate:function(){

setInterval(function(){

this.state.count --> error:state is not defined

},500)

}



> this -> constructor(组件的构造函数)

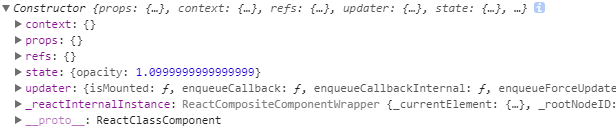
componentDidUpdate:function(){

setInterval(()=>{

this.state.count --> right!

},500)

}



### 2.7 受控表单元素

在react中处理表单，特殊情况：

如果指定了value/checked/selected属性，此表单元素是受React这个库控制的

* 方法一：defaultValue
* 方法二：借助于状态具有绑定的特性

> 步骤1：初始化状态

getInitialState:function(){

return {myAddr:"bj"}

}

> 步骤2：将状态的值绑定到表单元素的value属性

<input onChange={this.handleChange} type="text" value={this.state.myAddr} />

> 步骤3：绑定对应的事件，在事件处理函数中完成状态的写操作

handleChange:function(event){

this.setState({myAddr:event.target.value});

}

<input onChange={this.handleChange} type="text" value={this.state.myAddr} />

## 三、进阶知识

### 3.1 事件处理函数自定义传值

* vue

<button @click="handleClick(1)"></button>

* Angular

<button (click)="handleClick(1)"></button>

* React

<button onClick={()=>this.handleClick(1)}></button>

### 3.2 循环

* vue

v-for

* Angular

\*ngFor

* React

render:function(){

return <ul>

{

this.state.list.map((value,index)=>{

return <li key={index}>{value}</li>

})

}

</ul>

}

### 3.3 条件判断

* vue

v-if

* Angular

\*ngIf

* React

> 方案一：

逻辑与(适合用在判断条件比较简单的场景)

render:function(){

return <div>

{

this.state.isLogin

&&

<h2>欢迎登录</h2>

}

</div>

}

> 方案二：复杂的运算

步骤1：定义一个方法，在此方法中，执行运算，做判断，根据具体的业务需要返回要渲染的元素或者不返回任何东西

isShow:function(){

// 如果scoreList的平均成绩大于80分，就显示h2

var averageScore = 0,totalScore = 0;

for(var i=0;i<this.state.scoreList.length;i++){

totalScore += this.state.scoreList[i]

}

averageScore = totalScore / this.state.scoreList.length;

if(averageScore > 80){

return <h2>优秀</h2>

}

}

步骤2：调用

render:function(){

return <div>

{

this.isShow()

}

</div>

}

## 四、总结

* jsx

> 基础用法：

JavaScript中可以写入 <any></any>

标签及其内部可进行运算 {expression}

> 功能：

建立起了模板内容和数据方法的桥梁

* component

> 创建：

React.createClass({…})

> 功能：

将可复用的视图封装起来，构建UI层

* props

> 父---》子

this.props.\*\*

> 子--》父

基础：属性可以传递方法

this.props.children

* state

> 绑定效果

> 管理数据(初始化、读、写)

* ref

可以通过引用的名称得到一个组件的实例或元素

this.refs.\*\*

* 开发理念：

Learn Once, Write Anywhere

【练习】使用reactjs实现todobox(接P357页练习)

> 步骤2：在此文件中准备4组件

ToDoItem ToDoList

ToDoInput

ToDoBox

> 步骤3：将ToDoBox作为根组件去渲染

> 步骤4：点击ToDoInput的add，将输入框的值通过子与父通信，传递到ToDoBox

> 步骤5：ToDoBox接收传来的数据，保存在状态中(数组)，传递给ToDoList；

> 步骤6：ToDoList接收到数据，循环创建多个ToDoItem，ToDoItem要显示正确数据

> 步骤7：ToDoInput中点击add，添加成功后将输入框的内容清空

> 步骤8：实现删除功能：点击ToDoItem的删除按钮将数据中指定下标传递给ToDoList，再传递给ToDoBox；接收到下标，从数组中删除指定位置的元素。

# 第二十一章 ReactNative

中文官网：<https://reactnative.cn/>

Learn once,write anywhere

学习一次React中核心概念，就可以在其他场景(原生移动端开发ReactNative、ReactVR)进行编程使用。

## 一、ReactNative(RN)概述

简介：使用JavaScript和React编写原生移动应用

竞品：weex(阿里 使用vue) flutter(Google)

### 1.1 移动端开发形式

移动端开发3种常见的开发形式

WebApp：使用前端编程技术，来构建运行在手机浏览器中，提供类似原生app用户体验的APP。优点是具有良好的跨平台性。

NativeApp：原生的app(Andriod/IOS)，使用java/kotlin/swift/oc来进行编程，调用SDK(Software Development Kit软件开发工具)提供的组件来完成原生app的编写。优点在于能够直接调用系统底层硬件。

HybridApp：混合编程，通过web前端技术和原生开发技术，混合起来进行编程而实现的app。

### 1.2 搭建环境

#### 1.2.1 准备PC端的环境

(模板项目、开发服务器)

> myapp.zip拷贝到C:\xampp\htdocs\framework\4\_rn，解压缩到myapp文件夹

> 用vscode打开myapp文件夹

> npm start

#### 1.2.2 准备移动端的环境

(预览)

> 安卓端安装包路径：

C:\xampp\htdocs\framework\4\_rn\myapp\android\app\build\outputs\apk

> 打开myapp

> 摇一摇--》Dev Settings-->Debug Server Host & Port-->输入框

> 查看ip地址：windows+R-->cmd-->ipconfig

172.163.100.145

> 输入主机地址和端口号：

172.163.100.145:8081

> 摇一摇--》reload

**!注意：**第一次reload，如果没有显示，可以试着将app关掉再重新打开

## 二、ReactNative使用

### 2.1 自定义组件的创建和使用

* 创建：

export default class Demo01Component extends Component{

render(){

return <标签></标签>

}

}

* 使用：

> 直接设置为根组件

AppRegistry.registerComponent("myapp",()=>Demo01Component)

> 嵌套调用

<Demo01Component></Demo01Component>

**!注意：**

> 不允许调用html标签

> 组件先引入 再使用

> 类的名称要遵循全驼峰

> 类的导出方式的区别

export default --> 引入时，不需要加花括号；类的名称可以和导出不一致

export class --> 引入时，必须加上花括号，类的名字必须和导出时保持一致

【练习】

创建一个文件demo02\_lianxi.js

> 步骤1：创建一个组件类Demo02Component,指定渲染3个Text --> View

> 步骤2：将Demo02Component 设置为根组件

### 2.2 state的使用

初始化状态和reactjs中不太一样，读写都是一样的

constructor(){

super(); //表示调用父类的构造函数

this.state = {count:0}

}

### 2.3 样式的封装

RN自带的组件都支持样式的封装调用

* 步骤1：引入

import { StyleSheet } from 'react-native'

* 步骤2：封装(在类方法外部)

export default class MainComponent extends Component{…}

const myStyles = StyleSheet.create({

myText:{},

myImage:{}

})

* 步骤3：调用

<Any style={myStyles.myText}></Any>

【练习】

创建一个demo05\_lianxi.js

> 步骤1：完成Demo05Component组件的封装和调用

> 步骤2：初始化一个状态-->list(数组：5个整数构成的数组)

> 步骤3：在组件渲染时，根据list中元素的个数动态创建多个Text（显示数组中的内容）

### 2.4 常用的组件

#### 2.4.1 View

* 作用：实现一个容器
* 使用：

import {View} from 'react-native'

<View></View>

#### 2.4.2 Text

* 作用：展示文本内容的
* 使用：

import {Text} from 'react-native'

<Text></Text>

#### 2.4.3 Button

* 作用：实现一个按钮
* 使用：

import {Button} from 'react-native'

<Button title="刷新" onPress={this.handlePress}></Button>

**!注意：**在事件处理函数中，如果需要操作this，定义成箭头函数的形式

#### 2.4.4 TextInput

* 作用：获取用户输入的信息
* 使用：

import {TextInput} from 'react-native'

handleChangeText=(msg)=>{

//msg就是输入框的数据

}

<TextInput secureTextEntry={true} placeholder="请输入密码"

onChangeText={this.handleChangeText}>

</TextInput>

**!注意：**secureTextEntry={true}意思是输入文本为密码类安全性较高的内容，\*\*\*

#### 2.4.5 Switch

* 作用：实现一个滑动开关（默认就是受控的表单组件）
* 使用：

import {Switch} from 'react-native'

<Switch value={} onValueChange={}></Switch>

#### 2.4.6 FlatList

* 作用：是一个高性能的列表
* 使用：

import {FlatList} from 'react-native'

//有参数有返回值

showItem(info){

//info.item/index

return 显示的列表项

}

<FlatList data={集合} renderItem={this.showItem}></FlatList>

**!注意：**在渲染列表项，遇到key的问题，解决思路:

> 在处理函数中给数据源所对应的数组中每一个对象，添加一个key的属性(值是惟一的，不允许重复)

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

result[i].key = i;

}

// 保存数据，到列表显示

this.setState({list:result})

【综合练习】

demo10\_lianxi.js

> 步骤1：在该文件中封装一个组件Demo10Component

组件的内容:输入框 TextInput\按钮 Button\列表 FlatList

> 步骤2：实现该组件的功能

点击按钮，获取输入框的值(onChangeText)，添加到列表中(状态)去显示

> 步骤3：将组件设置为根组件进行渲染

#### 2.4.7 Image

* 作用：用来显示本地或者网络资源图片
* 使用：

import {Image} from 'react-native'

//显示本地资源图片

<Image source={require("\*\*\*")}></Image>

//显示网络资源图片

<Image source={{uri:""}}></Image>

**!注意：**

> 加载本地资源图片时，不允许做任何的运算(网络资源图片路径允许)

> 加载网络资源图片时，指定宽高，否则即使路径正确，也不会显示

#### 2.4.8 TouchableOpacity

* 作用：是一个容器，放在该容器中的组件，在被按下时，视图效果(透明度)有变化，而且也支持绑定press事件
* 使用：

// 步骤1：引入

import {TouchableOpacity} from 'react-native'

// 步骤2：调用

<TouchableOpacity onPress={}></TouchableOpacity>

#### 2.4.9 ScrollView

##### 2.4.9.1 基本用法

* 作用：实现一个支持滚动的容器，当要显示的元素的高度，超出了屏幕高度，自动添加滚动效果
* 使用：

// 步骤1：引入

import {ScrollView} from 'react-native'

// 步骤2：调用

<ScrollView>...</ScrollView>

##### 2.4.9.2 实现页脚效果

<View>

<ScrollView></ScrollView>

<View>

<Button></Button>

</View>

</View>

#### 2.4.10 ActivityIndicator

* 作用：实现一个组件：加载中的提示
* 使用：

// 步骤1:

import {ActivityIndicator} from 'react-native'

// 步骤2：

<ActivityIndicator size="large" color=""></ActivityIndicator>

【练习】demo14\_lianxi.js

> 步骤1：完成Demo14Component类的创建和调用

> 步骤2：实现一个Text,点击Text("加载更多")时要求有透明度变化的效果，显示一个ActivityIndicator（默认是隐藏，点击Text才显示）

### 2.5 自定义布局

flexDirection、justifyContent、alignItems

render(){

return <View style={{

height:200,backgroundColor:"coral",

alignItems:"center",justifyContent:"center"

}}>

<Text>水平居中</Text>

</View>

}

**!注意：**在RN中默认的主轴是column

### 2.6 网络通讯

var url = "http://jsonplaceholder.typicode.com/users"

fetch(url)

//拿到服务器发回的响应信息response并将其json反序列化

.then((response)=>response.json())

.then((result)=>{

//result就是服务器端所返回的数据在json反序列化之后的数据

})

**!注意：**

> url地址

> 参数的个数和类型以及组织方式

http://www.codeby.com/login?uname=zhangsan&upwd=123456

> 服务器端返回的数据在处理过程要正确的数据类型处理

【练习】demo17\_lianxi.js

> 步骤1：完成Demo17Component类的创建和调用

> 步骤2：组件类要显示：按钮，点击按钮向服务器端发起请求 <http://jsonplaceholder.typicode.com/posts>

将请求到的数据保存显示在列表，列表项显示title到Text

### 2.7 路由与导航

#### 2.7.1 ReactNavigation概述

VueRouter RouterModule NavController

* 路由模块：建立url和组件映射关系

rn中并没有内置路由模块，可以采用第三方的导航模块---》ReactNavigation

* ReactNavigation定位：

Routing and navigation for your React Native apps

针对RN的项目实现路由和导航的功能

* 使用ReactNavigation前提：

npm i react-navigation

#### 2.7.2 ReactNavigation基本使用步骤

* 步骤1：准备一些组件
* 步骤2：创建路由词典

// 引入必须要的工具和类

import {createStackNavigator} from 'react-navigation'

import Demo18Component from '\*\*'

import Demo19Component from '\*\*'

// 设置路由

var MyNavigator = createStackNavigator({

"demo18":{

component:Demo18Component

}

})

* 步骤3：app调用路由词典

AppRegistry.registerComponent("myApp",()=>MyNavigator)

* 步骤4：跳转

this.props.navigation.navigate("目的地的路由地址")

【练习】

* 步骤1：创建组件

在当前项目中，创建组件，一个是Demo20Login,一个是ToDoBox(包含以下组件:ToDoList,ToDoItem,ToDoInput)

* 步骤2：配置路由

给Demo20Login --> demo20

给ToDoBox --> box

默认显示demo20

* 步骤3：组件要求

Demo20Login组件：放上一个按钮，点击跳转到ToDoBox，ToDoBox和之前的要求一样。

> 点击ToDoInput的add按钮，将输入框的值通过子与父通信，传递到ToDoBox中

通过属性传递一个有参数的方法给子组件，子组件就可以调用这个方法，同时将值传递给父组件

> ToDoBox将数据接收到，保存在一个数组(应该是一个状态)，通过props down，传递给ToDoList

> ToDoList接收到数据，动态的创建多个ToDoItem

- 通过读取属性myList传来的数据；

- 遍历；

- 调用ToDoItem时，将数据传递过去，让ToDoItem的Text进行显示

> 实现ToDoItem中的删除功能：点击ToDoItem的删除按钮时，将当前的列表项从列表中移除调(:-11:40)

- ToDoBox中定义一个有参数的方法

参数：要删除的下标

方法体：从数组中删除指定下标的元素

- 将方法传递给ToDoList，再继续传给ToDoItem

- 在ToDoItem调用通过属性传来的方法，将下标传递过去

> 实现ToDoList如果是通过FlatList进行列表处理的：

- 在ToDoList指定一个状态，保存通过属性传来的数组

属性传来的值变化时，保存在状态

componentWillReceiveProps

- FlatList的调用

data/extraData/renderItem

#### 2.7.3 路由传参

* Vue:

修改接收方的路由地址 /detail/:id

this.$router.push('/detail/10')

this.$route.params.id

* Angular:

修改接收方的路由地址 /detail/:id

> 发送

Router router:Router this.router.navigateByUrl('/detail/10')

> 接收

ActivatedRoute myRoute:ActivateRoute

this.myRouter.params.subscribe(()=>{})

* Ionic:

this.navCtrl.push(DetailComponent,{id:1})

this.navParams.get('id')

* ReactNavigation:

> 步骤1：发送

this.props.navigation.navigate("目的地路由地址",{id:10})

> 步骤2：接收

this.props.navigation.getParam('id')

【练习】

> 步骤1：创建两个文件

demo23-order.js -->Demo23Component --》order

demo24-confirm.js --> Demo24Component-->confirm

> 步骤2：demo23

一旦加载，立刻发送请求http://jsonplaceholder.typicode.com/todos

将请求到的数据保存在状态的一个数组中，显示在FlatList,列表项(TouchableOpacity-->Text)显示对象中title；

点击TouchableOpacity跳转到confirm，并且将当前对象的completed值发送

> 步骤3：confirm

接收参数，打印在终端

# 第二十二章 Python

官网：<https://www.python.org/>

Python手册：<https://docs.python.org/3.6/tutorial/index.html>

<https://docs.python.org/3.6/library/index.html>

## 一、概述

### 1.1 学习目标

熟练掌握python基本语法、常见的标准库的使用、面向对象、异常处理

第三方的库Django：ORM、模板语法、基本用法

采用python和Django框架编写出后台的API

### 1.2 python概述

* python是一门编程语言，用来实现：

> Web Development:

- Django, Pyramid, Bottle, Tornado, Flask, web2py

> GUI Development:

- tkInter, PyGObject, PyQt, PySide, Kivy, wxPython

> Scientific and Numeric:

- SciPy, Pandas, IPython

> Software Development:

- Buildbot, Trac, Roundup

> System Administration:

- Ansible, Salt, OpenStack

* 优势：

> 跨平台

> 类库非常丰富

> 可扩展

> 语法简洁

> 开源

* 版本：

> python2.\* --> 遗产

> python3.\* --> 现在和未来

* 环境搭建：

下载安装包 /20\_PYTHON/DAY01/python-3.6.5.exe

或者<https://www.python.org/downloads/windows/>

## 二、基本语法

### 2.1 准备工作

#### 2.1.1 基本用法

* 行注释：#
* 块注释：

"""

块注释 内部可换行(也可使用单引号)

"""

* 输出：print()
* 输入：input() input("提示信息：")
* 获取用户输入：msg = input()
* 缩进：不要随便写，一般使用在控制语句中

#### 2.1.2 命令行用法

* 在命令行直接输入python语句

python(回车)

* 在命令行执行已有的python文件

python demo01.py

**!注意：**为简写文件名，可在输入完demo01后tab键直接补全文件名

* 查询帮助

> 命令行模式下输入help(函数名/keywords)查询函数用法、关键词列表…

### 2.2 变量

score = 80

del score

**!注意：**变量的命名要避免和关键词冲突

* 关键词：

|  |  |  |  |
| --- | --- | --- | --- |
| False | def | if | raise |
| None | del | import | return |
| True | elif | in | try |
| and | else | is | while |
| as | except | lambda | with |
| assert | finally | nonlocal | yield |
| break | for | not |  |
| class | from | or |  |
| continue | global | pass |  |

### 2.3 数据类型

#### 2.3.1 number 数字

# number

score = 80 #int 整数

price = 30.52 #float 浮点数

isSales = False #bool类型

complex = 3+2j #complex复数

print(score,type(score)) 80 <class 'int'>

print(price,type(price)) 30.52 <class 'float'>

print(isSales,type(isSales)) False <class 'bool'>

print(complex,type(complex)) (3+2j) <class 'complex'>

【练习】day01/demo04\_lianxi.py

> 步骤1：定义两个变量，一个单价，一个数量

> 步骤2：输出总价是多少

#### 2.3.2 string 字符串

* 字符串的定义和使用

字符串是可以通过两个单引号、两个双引号、三个单引号、三个双引号进行组织

stuName = 'zhangsan'

stuAdd = "beijing"

stuEmail =''' test@163

.com '''

stuInfo = """ 此学生学习

很认真 """

print(stuName,stuAdd,stuEmail,stuInfo)

tips = "你好！"

print(tips+stuName) #字符串拼接

print(tips\*3) #反复输出 tips 3次

**!注意：**

> 3个单引号、双引号，内部是可以换行的

> 字符串变量乘以整数，意即将字符串反复输出该整数次

#### 2.3.3 list 数组

数组：是由多个元素构成的有序序列

stuList = ["zhangsan","john","Mike"]

print(stuList,type(stuList))

# ['zhangsan', 'john', 'Mike'] <class 'list'>

print(stuList[1]) # john 正序

print(stuList[-1]) # Mike 倒序

stuList[1] = "testName"

print(stuList) # ['zhangsan', 'testName', 'Mike']

#### 2.3.4 tuple 元组

* 元组：带了枷锁的数组，可读，不允许修改

# 定义元组

myTuple = (100,200,300)

# 读取元组中的信息

print(myTuple[1])

# 尝试修改元组中的信息 # TypeError

myTuple[1] = 220

**!注意：**当元组中只有一条数据时，应在该数据后加上","，否则type打印出来该数据会变为它本身的数据类型

myNewTuple = (100)

print(myNewTuple,type(myNewTuple)) # 100 <class 'int'>

myNewTuple = (100,)

print(myNewTuple,type(myNewTuple)) # (100,) <class 'tuple'>

#### 2.3.5 set 无序集合

* 集合：存储不会重复的多个数据构成的无序集合

mySet = set([100,200,300])

print(mySet,type(mySet)) # {200, 100, 300} <class 'set'>

mySet.add(400)

print(mySet) # {200, 100, 400, 300}

mySet.add(450)

print(mySet) # {450, 100, 200, 300, 400}

mySet.remove(300)

print(mySet) # {450, 100, 200, 400}

mySet.remove(500)

print(mySet) # KeyError: 500

**!注意：**

> 使用add()新加入集合的数据的位置是不确定的(无序的)

> 插入重复数据不会报错，但是无任何作用

#### 2.3.6 dict 字典

myDict = {"name":"zhangsan","age":20}

print(myDict,type(myDict))

# {'name': 'zhangsan', 'age': 20} <class 'dict'>

print(myDict['age']) # 20

# print(myDict['ages']) # KeyError: 'ages' 后面的代码不执行

print(myDict.get("age")) # 20

print(myDict.get("ages")) # None

# myDict.get('age') = 22

# SyntaxError: can't assign to function call

myDict['age'] = 22

print(myDict) # {'name': 'zhangsan', 'age': 22}

**!注意：**

> 获取字典中的数据最好采用get()方法，以便于出错时不影响后续代码的执行；

> 字典中定义的键值对的键名必须用引号引起来，不然会报错

> 修改字典中的键值不能采用get()方法，而应采用方括号形式获取值再修改

【练习】day01/demo10\_lianxi.py

> 步骤1：准备一个由3个字符串所构成的数组

> 步骤2：打印数组中下标为2的元素

> 步骤3：获取用户输入的用户名和密码，保存在一个字典中

> 步骤4：打印字典中所存储的用户名

### 2.4 常见API

#### 2.4.1 list 数组API

* 获取数组长度

myList = [100,200,300]

print(len(myList))

* 获取/修改元素

print(myList[0]) # 100

print(myList[-1]) # 300

myList[0] = 150

print(myList) # [150,200,300]

* 增加元素

myList.append(400) # 尾部追加

print(myList) # [100,200,300,400]

myList.insert(0,50) # 向指定的位置插入一条数据

print(myList) # [50,100,200,300]

* 移除元素

myList.pop() # 移除元素，默认是移除尾部元素

print(myList) # [100,200]

myList.pop(1) # 移除下标为1的元素

print(myList) # [100,300]

* 排序

myList.sort() # 默认升序排列

myList.sort(reverse=True) # 降序排列

* 翻转

myList.reverse() # 翻转数组

#### 2.4.2 tuple 元组API

* 获取元组值

myTuple = (100,200,300)

print(myTuple[1]) # 200

* 将数组转换为元组

scoreList = [79,82,64]

myScoreTuple = tuple(scoreList)

print(myScoreTuple,type(myScoreTuple)) # (79,82,64) <class 'tuple'>

* 获取最值

myTuple = (100,200,300)

print(max(myTuple)) # 300

print(min(myTuple)) # 100

#### 2.4.3 dict 字典API

* 获取所有键名keys

print(myDict.keys()) # dict\_keys(['stuName', 'stuSex'])

* 获取所有键值values

print(myDict.values()) # dict\_values(['zhangsan', 'male'])

* 获取所有键值对

print(myDict.items())

# dict\_items([('stuName', 'zhangsan'), ('stuSex', 'male')])

* 获取对应键值value

myDict.get('stuName') # zhangsan

* 修改对应键值value

myDict['stuName'] = 'cch'

print(myDict) # {'stuName':'cch','stuScore':80}

* 增加

myDict['stuSex'] = 'male'

print(myDict)

# {'stuName': 'zhangsan', 'stuScore': 100, 'stuSex': 'male'}

* 删除

myDict.pop('stuScore')

print(myDict)

# {'stuName': 'zhangsan', 'stuSex': 'male'}

【练习】demo13\_lianxi.py

> 步骤1：准备一个空数组

> 步骤2：获取3次用户所输入的成绩信息，将这些信息保存在数组中

> 步骤3：删除下标为2的元素，将数组打印

#### 2.4.4 类型转换

* 显式类型转换

int()/float()/str()/tuple()/list()

nameList = ['zs','sim','michael']

myTuple = tuple(nameList)

print(myTuple,type(myTuple))

# ('zs','sim','michael') <class 'tuple'>

myList = list(myTuple)

print(myList,type(myList))

# ['zs', 'sim', 'michael'] <class 'list'>

myTuple2 = list((100,200,500))

print(myTuple2,type(myTuple2)) # [100, 200, 500] <class 'list'>

* 隐式类型转换

**待续…**

### 2.5 异常处理

#### 2.5.1 错误类型

##### 2.5.1.1 TypeError 类型错误

print("张三的成绩是："+score)

TypeError: must be str, not int

错误原因：不能进行字符串拼接(score不会进行隐式转换)

myTuple = (100,200,300)

myTuple[1] = 220

TypeError: 'tuple' object does not support item assignment

错误原因：元组不允许被修改

##### 2.5.1.2 NameError 变量名错误

del score # 释放变量

print(score)

NameError: name 'score' is not defined

错误原因：变量释放后不能再使用(调用未定义变量也会报此错误)

myDict = {name:"zhangsan","age":20}

NameError: name 'name' is not defined

错误原因：字典中定义的键值对的键名必须用引号引起来

##### 2.5.1.3 SyntaxError 语法错误

myDict.get('age') = 22/print())

# SyntaxError: can't assign to function call

错误原因：不能采用get()方式获取键值后再修改，应采用myDict["age"] = 22

##### 2.5.1.4 IndexError 下标错误

myList = [100,200,300]

print(myList[10])

IndexError: list index out of range

错误原因：所获取的下标值超出了数组长度

##### 2.5.1.5 KeyError 键值错误

myDict={"uname":"Lucy","age":18}

print(myDict["gender"])

KeyError: 'gender'

错误原因：key值错误，不存在该键值对应的键值对

##### 2.5.1.6 ValueError 值类型错误

myDict={"uname":"Lucy","age":18}

int(myDict.get("uname"))

ValueError: invalid literal for int() with base 10: 'Lucy'

错误原因：int()的值不支持文本，即文本无法进行int()类型转换

##### 2.5.1.7 AttributeError 属性错误

age=10

age.append(20)

AttributeError: 'int' object has no attribute 'append'

错误原因：int类型的变量对象没有append方法属性

#### 2.5.2 异常处理

异常不是错误。当未处理错误没有解决时，程序就会崩溃。

异常的处理： try/except

##### 2.5.2.1 基本用法

try:

do sth

except SomeError:

// 错误处理

finally:

// 执行最终要执行的代码

**!注意：**不需要加{}表示块，以缩进表示即可

##### 2.5.2.2 捕获多个错误

* 方式一(各错误各自采用不同的错误处理方式)：

try:

do sth

except Error1:

// 错误处理1

except Error2:

// 错误处理2

finally:

// 执行最终要执行的代码

* 方式二(括号中的两种错采用同样的处理方式)：

try:

do sth

except (Error1,Error2):

// 错误处理

finally:

// 执行最终要执行的代码

* 方式三(所有错误采用同样的处理方式)：

try:

do sth

except Exception:

// 错误处理

finally:

// 执行最终要执行的代码

**【练习】**

**定义一个列表，？？？？**

##### 2.5.2.3 手动触发错误

try:

print('it is a test')

raise(NameError('自定义消息'))

except NameError as msg:

print(msg)

print('已经进入错误处理的流程')

**!注意：**#msg就是触发错误时，所传来的自定义数据

### 2.6 运算符

#### 2.6.1 算术运算

* 运算符：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 加 | 减 | 乘 | 除 | 取余 | 商下取整 | 几次方 |
| + | - | × | / | % | // | \*\* |

**!注意：**

> None和字符串要谨慎和数字放在一起进行算术运算，None值参与算术运算会报TypeError错误

> True值参与运算会隐式转为1

> 字符串\*数值代表将该字符串反复输出该数值次

> 在除法运算中，无论是否整除，结果都是浮点数

> 区分取余和商下取整：

- 10 % 3 = 1

- 10 / 3 = 3

#### 2.6.2 关系运算

* 运算符：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 大于 | 大于等于 | 小于 | 小于等于 | 等于 | 不等于 |
| > | >= | < | <= | == | != |

**!注意：**

>数字、字符串、列表之间、元组之间都可以进行关系运算

> 字符串之间相比较时，从第一字符开始对比，如果字符都一样，比长度(a<b，比较Unicode码值)

> None不支持和任意类型进行比较，会报TypeError类型错误

【练习】demo21\_lianxi.py

> 提示用户分别输入语文成绩、数学成绩、英语成绩

> 计算平均值

> 将平均值和80比较，> True/False

#### 2.6.3赋值运算

* 运算符：

|  |  |  |
| --- | --- | --- |
| 加等于 | 减等于 | 赋值 |
| += | -= | = |

**!注意：**

> 支持同时给多个变量赋值

myList = [10,30]

[num3,num4] = myList #同时给多个变量赋值

print(num3,num4) # 10 30

> 不支持自增、自减

#### 2.6.4 逻辑运算

* 运算符：

|  |  |  |
| --- | --- | --- |
| 逻辑与 | 逻辑或 | 逻辑取反 |
| and | or | not |

* 用法：

myEnglishScore = 80

myMathScore = 90

print(myEnglishScore>90 and myMathScore>90) # False

print(myEnglishScore>60 or myMathScore>60) # True

print(not True) # False

print(not myMathScore>60) # False

#### 2.6.5 位运算

* 运算符：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 与 | 或 | 取反 | 异或 | 左移 | 右移 |
| & | | | ~ | ^ | << | >> |

* 用法：

a = int('00111100',2) # 2表示2进制

b = int('00001101',2)

print(a,b) # 60 13

print("按位与",a&b) # 按位与 12

print('按位或',a|b) # 按位或 61

print('异或',a^b) # 异或 49

print('按位取反',~b) # 按位取反 -14

#### 2.6.6 身份运算

* 运算符：

> in/not in 判断一个数据是否在数组或者元组中

> id 得到一个变量在python所分配的地址

> is 判断前后变量的id是否一致

* 用法：

值类型和引用类型创建时的区别：

num1 = 10

num2 = 10

num3 = num2

print(id(num1),id(num2),id(num3))

print(num1 is num2) #True

print(num3 is num2) #True

myTuple1 = (1,2,3)

myTuple2 = (1,2,3)

myTuple3 = myTuple2

print(id(myTuple1),id(myTuple2),id(myTuple3))

print(myTuple1 is myTuple2) #False

print(myTuple3 is myTuple2) #True

**!注意：**

> 可变数据类型 list dict set；不可变数据类型 number string

> 值类型：数字、字符串；引用类型：list、tuple、dict…

> 引用类型的变量在每次创建时，都会分配一个新的地址

### 2.7 控制流

#### 2.7.1 条件判

* if…else

n = 1

if(n > 10):

print("n是大于10的")

else:

print("n是小于等于10的")

* 多重条件判断 if…elif…else

myScore = 80;

if(myScore >= 90):

print("A")

elif(myScore >= 80):

print("B")

elif(myScore >= 60):

print("C")

else:

print("D")

#### 2.7.2 循环

* 遍历数组的值

myList = [100,200,300,400,500]

for tmp in myList:

print(tmp) # tmp是遍历数组时对应的临时变量

# 100 200 300 400 500

* 遍历获取数组下标

for tmp in range(0,len(myList)):

print("下标为%d的数据是%d"%(tmp,myList[tmp]))

# 下标为0的数据是100

# 下标为1的数据是200

# 下标为2的数据是300

# 下标为3的数据是400

# 下标为4的数据是500

【TIPS】range()函数：range(start,stop[,step])

> 生成一个序列，存储从start开始，到stop结束的数据，步长为step

> 只写一个参数代表count --> range(count)生成从0开始到count截止的序列

for tmp in range(3,10):

print(tmp)

【练习】day02/demo29\_lianxi.py

> 步骤1：实现一个for循环，循环3次，在for循环中获取用户输入的成绩

> 步骤2：将获取到的成绩，计算平均值

> 步骤3：平均值大于80，输出"优秀"，否则输出"一般"

#### 2.7.3 控制

* 用法：continue/break 用法同JS及其他语言

【TIPS】python里随机数的用法：

import random

num = random.choice([1,2,3]) //从[1,2,3]中随机选择一个

【练习】demo32\_lianxi.py

> 创建一个0到100之间的随机数

> 实现一个while循环

提示用户输入一个数字

数字比随机数大：太大了

\*\*\*\*\*\*\*\*\*\*\*小：太小了

相等：猜对了，退出循环

### 2.8 方法

#### 2.8.1 创建方法

def add(num1,num2):

result = num1+num2

return result

#### 2.8.2 调用方法

add(10,20)

**!注意：**参数要和定义的保持一致

#### 2.8.3 参数传递

##### 2.8.3.1 参数概述

* 值传递：修改参数，原始数据不受影响 number/string/tuple
* 引用传递：修改参数，原始数据也会变化 list/set/dict

【练习】demo35\_lianxi.py 求阶乘：#n!=1×2×3×...×n

> 定义一个方法

在方法内，来实现一个阶乘的效果

参数是要求谁的阶乘

返回值是阶乘的结果

> 调用方法，传值，将结果打印

##### 2.8.3.2 关键字参数(调整参数顺序)

* 用法：在方法的调用过程中，是可以通过keyValue指定关键字参数，调整参数顺序

myFunc(arg2=10,arg1=20)

myFunc(10,arg2=20)

myFunc(arg1=10,20) #语法错误

**!注意：**关键字参数必须出现在实际参数后边，或者全部都是关键字参数，否则会报SyntaxError语法错误。

##### 2.8.3.3 可变长参数(参数个数不受限)

* 方式一：

def myFunc(\*args): #args是一个元组类型

myFunc()

myFunc(1,2)

* 方式二：

def myFunc(\*\*args): #args是一个字典类型

myFunc(a=1,b=2)

**!注意：**此时用myFunc(1,2)是会报错的

##### 2.8.3.4 默认值参数(指定默认数据)

* 用法：如果传参，以实际参数为准，如果没有传参，以默认值为调用的数据

def myFunc(arg1,arg2=20):

pass

myFunc(10)

myFunc(10,30)

**!注意：**默认值参数应写在普通的参数之后，否则会报SyntaxError。

【练习】demo39\_lianxi.py(:-17:45)

> 创建一个方法，

默认值参数指的是从哪里开始查询，默认设置为0

indexOf(myList,myItem,默认值参数)

> 在方法中实现功能：

查找一个元素在列表中的位置

找到--》返回下标

没找到--》返回-1

> 调用方法

【练习】demo40\_lianxi.py

> 创建一个方法，传入n个整数，输出其中的最大值和最小值

> 创建一个方法，支持3个参数，调用时，可以调用多次(通过关键字参数，调整参数的顺序）；方法中打印参数内容即可

### 2.9 作用域、闭包

#### 2.9.1 作用域及闭包

* 作用域：Python中也存在全局作用域和函数作用域之分：

count = 10 # 全局作用域

def modifyCount():

global count # 必须先引入全局的count变量才能修改其值

myCount = 0 # 函数作用域

count += 1

print(myCount,count)

modifyCount()

print(count)

* 闭包：重用变量，又保护变量不被篡改

闭包基本原理：

> 外层函数在调用，创建外层函数作用域对象

> 内层函数在创建时，通过自己\_\_closure\_\_属性 引用外层作用域对象

在外层函数执行之后，无法释放，形成一个闭包的结果(变量依然保留在内存中，可重用)

def outer(): # 外层函数

a = 5

b = 0.5

def inner(): # 内层函数

return a + b

return inner

inner = outer()

print(inner)

print(inner())

L(local)>E(embedded)>G(global)>B(block)

**!注意：**

> 在函数内部的作用范围内，如果读全局变量没问题！如果要修改某一个全局变量的值，得必须通过global将此变量引入到函数内部

> 在嵌套函数内部要想修改外层函数中的变量，得通过nonlocal引入到嵌套的内部

【练习】day03/demo44\_lianxi.py

> 定义一个方法getList,在方法:

定义一个数组userList,

定义一个方法login，此方法需要参数(用户名)：向userList数组中头部添加此用户名，在此方法中返回userList

> 调用方法：调用多次login

#### 2.9.2 匿名函数

定义匿名函数，调用lambda来实现lambda num1,num2:num1+num2

def add(num1,num2):

result = num1 + num2

return result

print(add(3,4))

# 此方法同上定义的add()函数

addFunc = lambda num1,num2:num1+num2

print(addFunc(3,5))

#### 2.9.3 装饰器函数

* 定义：装饰器本质是一个方法，它在python经常用来日志处理、性能测试、权限校验等场景，用来处理其他函数。
* 优势：装饰器的强大在于：在不修改原有业务逻辑的情况下，对代码进行扩展，比如权限校验、用户认证、日志处理、性能测试。。。，最大限度的对代码进行复用
* 定义装饰器方法：

> 不带参数：

def checkLogin(func):

def handle():

# 检查用户是否登录，假设登录成功

if False: # False-->不会调用传入的func()

print("检查用户未登录")

func()

return handle

> 带参数：

def deco(func):

def handle(num1,num2):

func(num1,num2)

print("此方法被调用啦")

return handle

@deco

def add(num1,num2):

print("相加的结果是：%d"%(num1+num2))

add(3,5)

* 调用装饰器方法：

@checkLogin # 调用装饰器

def addToCart():

print("添加到购物车成功")

addToCart() # 必须调用购物车方法才能触发装饰器方法

【练习】day03/demo47\_lianxi.py

> 封装一个方法，此方法中有两个参数，方法中执行一个加法运算，输出求和的结果。

> 封装一个装饰器函数，被它装饰的函数在执行的时候，能够在中断输出“此方法被调用了”

## 三、进阶知识

### 3.1模块和包

#### 3.1.1 模块 module

* 模块化：

> 方便团队协作

> 提供代码的复用率

> 方便对业务进行拆分、维护、开发

* 创建自定义模块：

> 一个py文件就是一个模块

* 在一个模块中调用其他模块中的东西：

> 方法一：

import demo48

demo48.\*\*

import demo48 as d

d.\*\*

> 方法二：

from demo48 import test

tese()

from demo48 import \*

test()

【练习】day03/demo50\_user.py

> 创建demo50\_user.py文件，在此文件中封装一些方法:

login logout register (每个方法只需要输出：比如用户正在登录)

> 创建demo50\_cart.py,在此文件中，引入demo50\_user中的login方法，直接调用

#### 3.1.2 包 package

* 定义：包是由多个模块构成的，是对多个模块的一个封装
* 创建一个包：

在python一个文件夹就是一个package，在此文件夹中可以根据业务需要创建多个py文件

* 调用其它的包中的模块：

和之前引入模块语法一致(需要在引入的模块名前 加上包的名字)

import utility.data\_deal

* 示例：

> utility/data\_deal.py

def handleData():

print("正在进行数据处理！")

> demo51\_package.py(也有四种写法)

from utlity.data\_deal import handleData

handleData()

### 3.2 面向对象编程

#### 3.2.1 封装

将现实问题中，有共同特征的实例，把共同特征封装在类中（创建一个类）

* 创建一个类：

class Moester():

def \_\_init\_\_(self,name): # 构造函数(注意是两个"\_")

self.name = name;

def run(self): # 自定义方法

print("…")

* 实例化一个类：

m1 = Monester()

m1.run() # 调用类的方法

**!注意：**

> 类里的构造函数和封装的自定义方法的第一个参数必须是self（类似于this）

> 创建私有内部成员：self.\_\_age = age（构造函数中，同样也是两个下划线）。私有内部成员不可以被实例化之后的对象直接调用，如：m1.\_\_age会报错。

【练习】day03/demo54\_lianxi.py

> 创建一个类Student,

在被实例化时，能够传递学生的姓名、年龄

实现一个方法，输出"张三在学习..."

> 实例化，调用类中的方法

#### 3.2.2 继承

* 定义：在创建一个类时，指定基类(父类)，此类就会拥有父类中的属性和方法
* 使用：

class BaseClass():

class SonClass(BaseClass):

* 示例：

class Monester():

def run(self):

print("正在奔跑...")

class Boss(Monester):

def test(self):

pass # 表示什么都不做

def run(self):

print("正在飞速奔跑...")

b1 = Boss() # 实例化Boss得到一个实例

b1.run()

#### 3.2.3 多态

* 定义：同一个函数，在不同的情况下会有多个状态；在python中主要是通过方法的重写来实现的，如3.2.2中示例中的Boss类中的run()方法。

【练习】day03/demo56\_lianxi.py

> 创建两个类，Car SUV

> Car类在实例化，传递一个参数brand;定义一个方法drive:汽车正在行驶

> SUV类继承自Car，复写drive:汽车正在快速行驶

【练习】

> 实现类Monester：

在创建Monester类的时候，指定hp(默认值参数默认是1000)

定义一个方法whoAmI,输出'我是monester'

> 实现类Boss:

继承自Monester,构造函数希望有两个，一个hp，一个name

复写whoAmI，输出“我是Boss”

> 分别去实例化Monester、Boss

> 分别调用whoAmI方法

### 3.3 python标准库的使用

python有一些官方已封装好的库可供使用，以下列举两个常用的说明使用方法：

# time

import time

print(time.localtime())

# time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=17, tm\_hour=19, tm\_min=30, tm\_sec=40, tm\_wday=0, tm\_yday=351, tm\_isdst=0)

print(time.strftime('%m-%d %H:%M')) # 12-17 19:30

# random

import random

print(random.choice(range(0,10))) # 5

print(random.choice(["john",'lucy','mike'])) # lucy

# math

import math

num = 10.3

print(num)

print(math.ceil(num)) # 向上取整的处理

print(math.floor(num)) # 向下取整的处理

# 和文件、文件夹相关的处理和os的处理

# os:operating system操作系统

import os

print(os.environ) # 查看环境变量

print(os.getlogin) # 得到当前的登录用户

print(os.path.abspath('.')) # 获取当前目录的绝对路径

print(os.path.exists("./demo62\_lib.py")) # 操作文件

### 3.4 数据库

pip install mysql-connector

回顾：node+mysql

在python中 完成mysql数据库服务器中 数据库的CRUD

#### 3.4.1 mysql的使用

* 安装、引入

pip install mysql-connector

import mysql.connector as conn

* 连接指定的数据库服务器

myConnection = conn.connect(

host='localhost',

user='root',

passwd='',

database='xz'

)

* 创建游标对象

# 默认以元组形式出结果，加上dictionary=True表示以字典形式

myCursor = myConnection.cursor(dictionary=True)

* 执行sql语句，处理结果

myCursor.execute(sql)

result = myCursor.fetchall()

**!注意：**如果需要修改数据库中的数据，一定要用调用commit()进行提交：

myConnection.commit()

【练习】demo60\_lianxi.py

> 连接数据库xz

> 获取xz\_index\_product表的数据，将每一行的title整理在一个数组中

#### 3.4.2 连接池的使用

* 引入

import mysql.connector.pooling as pooling

* 创建连接池

config = {

"host":"localhost",

"user":"root",

"passwd":"",

"database":"xz"

}

pool = pooling.MySQLConnectionPool(pool\_size=10,\*\*config)

* 连接

myConn = pool.get\_connection()

* 创建游标对象

myCursor = myConn.cursor()

* 执行sql操作 处理结果

myCursor.execute("select \* from xz\_user")

print(myCursor.fetchall())

# 第二十三章 Django

## 一、概述

### 1.1 django概述

* 官网：<https://www.djangoproject.com/>
* 作用：Django是一个基于python的框架，05年发布，早期是用于做新闻门户网站。此框架本身提供了强大的后台管理系统，可实现网站前后端的所有东西。
* 优缺点：

> 开源框架，有非常棒的文档

> 功能强大，集成了ORM/模板/路由…

> 强大的后台管理

> 耦合度太高

### 1.2 django的安装和使用

pip install django

project和app的使用

命令的基本用法

实现后台api

路由系统

ORM

模板语法