# 前言

学习分五个阶段:

阶段一：10天，DB+PHP 8’

阶段二：20天，HTML+CSS+JS+AJAX 5’

阶段三：20天，JS+DOM+jQuery 7’

阶段四：20天，HTML5+Bootstrap+Vue.js 6’

阶段五：20天，Ionic+NG（angularJS）+React 9’

阶段六：10天，微信+Node.js+项目管理（gulp、webpack、git） 6’

编程思维——机器思维

编程学习的方法：由人的思维方式转变为机器思维方式：

（1） 理解思路/算法/逻辑

（2） 仿写别人的代码

（3） 默写之前写过的代码

（4） 重复（1），（2），（3）

总结：勤思考，多练习

## 程序员常用的快捷键 ：

Win+e 调出我的电脑（资源管理器 Explorer）

Win+r 调出（“运行”窗口） cmd/calc/mspaint

Win+d 显示或隐藏桌面

Alt+tab 切换窗口 从前往后

Alt+shift+tab 切换窗口（从后往前）

Alt+f4 关闭当前页面

缩略词补全展开功能

Div+tab 补全为<div></div>

H1#bt2 补全为<h1 id="bt2"></h1>

H1.bt2+tab 补全为<h1 class="bt2"></h1>

H1>span>a+tab  补全为<h1><span><a href=""></a></span></h1>

#box+tab 补全为<div id="box"></div>

.box+tab 补全为<div class="box"></div>

Span\*5+tab 补全为<span></span><span></span><span></span>

Lorem+tab 补全为一段随机生成的假文

Ctrl+alt+↓： 复制当前行

Alt+↑/↓： 移动当前行

Ctrl+d ： 删除当前行

Ctrl+/： 注释当前行

## 软件工程和开发流程——贯穿到整个课程

### 1.软件工程的定义：

软件开发方法：

重量级开发方法：需要大量的文档来控制软件开发过程

ISO9000，CMM/CMMI

轻量级开发方法：没有对大量正式文档的要求

极限开发

敏捷流程

### \*软件的生命周期：

三个时期八个阶段

**软件定义期**

（1）可行性研究阶段

（2）需求分析阶段

**软件开发期**

（3）概要设计阶段 架构师

（4）详细设计阶段 设计师

（5）开发实现阶段 前端/后台/UI工程师

（6）测试阶段 测试工程师

**软件维护期**

（7）部署阶段

（8）维护阶段

开发人员在自己的电脑上安装的服务器，一般仅在开发阶段供测试，调试使用，称为：“Development Mode（开发模式）”

Web服务器：让别人访问自己编写的网页

DB服务器：存储项目中的数据（用户、产品、订单…）

辅助服务器：帮助开发人员调试、维护其他的服务器

XAMPP(www.xampp.org)套装软件：是一个服务器软件的集合，包含常用的各种服务器，如Apache，MySQL，….

Xampp.org

在客户端浏览器中输入：http://176.116.110.187:80

查询本机的ip地址

Cmd ----->ipconfig

遇到问题/BUG如何解决：

自己分析，尝试解决 =》 同桌 =》 项目经理 =》 讲师 =》 文华老师 =》

# 数据库

## 1.什么是数据库

数据库的概述

Database Server：数据库服务器，专用于存储网页中的数据。

关系型数据库：MySQL

非关系型数据库：MongoDB（？）

关系型数据库管理系统分为两部分：

1. 服务器端：负责永久存储数据、维护数据，人不能直接观看

服务器上数据的逻辑结构：Server > Database > Table > Row > Column

Oracle MySQL: MySQL的分支

MariaDB:MySQL的分支

LAMP：Linux+Apache+MySQl+PHP

2. 客户端：用于向服务器发起“增删改查”改命令

2.如何操作数据库：使用MySQL服务器的步骤

（1）服务器端：安装MySQL服务器端软件

C:\xampp\mysql\bin\ mysqld.exe

D:deamon:精灵，守护者，守护程序，服务器程序

（2）服务器端：启动服务器端软件

计算机 Port端口号：1~65535（从1到65535）

（3）客户端：安装一款MySQL客户端软件

C:\xampp\mysql\bin\ mysql.exe

（4）客户端：运行客户端程序连接到远程的服务器

C:\xampp\mysql\bin\ mysql.exe –uroot -p;

(默认密码为空，不用填)

或者在XAMPP Shell中敲入如下命令：

#mysql -u -p;

## 2.MySQL常用管理命令

提示：

（1） MySQL命令大小写都可以！推荐在编写关键字的时候用大写字母，非关键字可以小写。

（2） 所有命令都必须以“ ; ”结束（注：英文分号）

SHOW DATABASES; 显示服务器中当前所有的数据库名

USE 库名; 进入指定的数据库中

SHOW TABLES; 显示当前数据库中有哪些数据表

DESC 表名; 描述指定表的结构（有哪些列）

网页模式：localhost/phpmyadmin

(或者点击XAMPP 中mysql行的admin)

## 3.SQL语言

结构化查询语言，专用于操作（增删改查）数据库服务器中的数据。是一门国际标准化语言，被各大数据库厂家所支持。

单行注释：#show databases; “#”注释当前行的sql语句

多行注释：/\*。。。\*/ 注释所包含的所有sql语句

set names utf8;//因为shell是gbk编码，所要设置该项

#进入指定的数据库

USE database\_name;

### 1.常用的SQL语句：

#### 1.DDL :(数据定义语言)-------定义数据的结构

**Create**/**drop**/alter/truncate

#创建新的数据库，其中的字符使用指定的编码UTF-8

CREATE DATABASE 库名 CHARSET=UTF8 ;

#丢弃一个已有的数据库(如果存在的话)

DROP DATABASE IF EXISTS 库名 ;

#### 2.DML:(数据操作语言)-------------操作数据行

**Insert /delete /update**

添加数据:

INSERT INTO table\_name VALUES (…);

删除数据:

DELETE FROM table\_name ; #删除所有的记录行,**慎用!**

DELETE FROM table\_name WHERE 列=值; #删除满足条件的行

修改数据:

UPDATE table\_name SET 列=值 , 列=值 , 列=值 ;

#这样会改变所有的行,**慎用!**

UPDATE table\_name SET 列=值 , 列=值 WHERE 列=值 ;

#修改满足条件的行

#### 3.DQL:(数据查询语言)----------对数据没有影响

**Select**

查询数据:

SELECT \* FROM table\_name ;

#### 4.DCL:(数据控制语言)------------控制用户权限

Grant/revoke

### 2.列类型和列约束

#### 2.1 MySQL中列的类型

##### (1)数字类型

1. TINYINT 占用1个字节存储空间,范围**(-128~127)**,最省空间
2. SMALLINT 占用2个字节存储空间,范围**(-32768~32767)**
3. INT 占用4个字节存储空间,范围**(-2147483648~2147483647)**
4. BIGINT 占用8个字节存储空间,……很大很大,有的系统只占4个字节
5. FLOAT(M,D) 单精度浮点型,占用4个字节 FLOAT(4,2) --99.99
6. DOUBLE(M,D) 双精度浮点型,占用8个字节
7. DECIMAL(M,D) 严格定点数,用于精确运算,如货币金额

M表示总有效位数,D表示小数点后面允许的有效位数

##### (2)字符串类型

1. CHAR(M) 固定长度字符串,长度最大为255个字符
2. VARCHAR(M) 长度可变的字符串,长度最大为65535个字符
3. TEXT(M) 不能超过2^32,即40亿个字符,(即4G空间)

##### (3)日期时间类型

1. DATE : 日期类型,范围:’1000-01-01’到’9999-12-31’ 必须用引号括起来,采用’yyyy-MM-dd’格式
2. TIME : 时间类型,必须用引号括起来,采用’hh:mm:ss’格式
3. DATETIME: 日期时间类型,必须用引号括起来,采用

’yyyy-MM-dd hh:mm:ss’格式

##### (4)布尔类型

BOOL/BOOLEAN

取值只能为TRUE(等价于1)或者FALSE(等价于0)

等价于tinyint(1),等价于长度为1的数字,插入0视为false,插入非0数字,即为true

**#(0)设置下面所有的SQL命令中的字符所用的编码方案,**

**#上来就写,写在sql文件最上方**

**SET NAMES UTF8;**

#### 2.2 MySQL中列上的约束

Constraint:约束,列上的值往往是有限制的,如:

性别:只能男或女

政治面貌:只能取党员,团员,群众之一

高考成绩:FLOAT(4,1) 999.9 取值有规则

手机号码:有格式要求

用户名:必须唯一

密码:不能为null字符串,长度有限制(6~12),长度不能少于N位

员工所在部门:可取值必须在部门表中存在

……..

##### (1)主键约束(PRIMARY KEY)

**语法:列名 类型 PRIMARY KEY;**

声明为”主键”的列上不能出现NULL值,且不能重复,如商品编号/id

表中所有的记录行会自动按主键列上的值进行排序

**一个表至多只能声明一个主键**

##### (2)唯一约束(UNIQUE)

**语法:列名 类型 UNIQUE;**

声明为”唯一”约束的列上不能出现重复值,但可以出现多个NULL

如:用户名不能重复

##### (3)非空约束(NOT NULL)

**语法:列名 类型 NOT NULL;**

声明为”非空”约束的列上不能出现NULL,但可以重复,如:密码

##### (4)(MySQL无效)检查约束(MySQL无效)

Create table user(

…

Sex char(1) check(sex=’男’ or sex=’女’)

…

);

##### (5)默认值约束(DEFAULT)

**语法: 列名 类型 DEFAULT ’值’;**

声明了默认值的列若未指定值,则使用默认值,若指定了特定的值,则使用指定的值

##### (6)外键约束

**FOREIGN KEY (foreign key)**

可重复可为空,外键列上出现的值必须在另外一个表的主键列上出现过

**语法 列名 类型;**

**FOREIGN KEY(列名) REFERENCES 表名(列名);**

**Foreign key(列名) references 表名(列名);**

**提示:如果为员工表添加了一个参考部门表的外键列,增加/修改 员工都要查询一下部门表,所以增加/修改 效率降低.**

**老版本MySQL默认不支持此约束,新版本MySQL支持**

##### 主键和外键的区别:

主键(PRIMARY KEY), 唯一且非空约束,是排序依据

外键:FOREIGN KEY, 可重复可为空,外键列上出现的值必须在另外一个表的主键列上出现过

##### 自增列

MySQL 数据库中专有的’自增列’

AUTO\_INCREMENT ,只有mysql支持,用于实现自增

语法:

CREATE TABLE 表名(

主键列名 INT PRIMARY KEY AUTO\_INCREMENT,

……

提示:自增列只能用于主键列,且必须是整数型主键.自增列无需手工指定特定的值,只需要赋值为null,mysql服务器会自动查询当前已有的最大整数,在此基础上+1.

### 3.简单查询

#### (1)只查看特定的列

**语法:** **SELECT 列名,列名,…列名 FROM 表名;**

示例:

查询所有部门名称

Select dname from dept;

查询所有员工的工资,生日,姓名

Select salary, birthday, ename from emp;

#### (2)查询所有的列

**语法: Select \* from 表名**

提示: ‘ \* ’ 指代 ‘ 所有列 ’

#### (3)给列取别名:以简化或更好的说明

**语法: select 列名 as 别名, ….. ,列名 as 别名 from 表名;**

**语法: select 列名 别名, ….. ,列名 别名 from 表名;**

Select ename as 姓名,ename e form emp;

示例:

查询所有员工birthday(生日),salary(月薪),ename(姓名)

Select birthday 生日,salary 月薪,ename 姓名 from emp;

#### (4)只显示列上不同的值(即合并相同的值)

**语法:**

**select distinct列名,….,列名 from 表名;**

**SELECT DISTINCT列名,….,列名 FROM 表名;**

Distinct : 明显的, 独特的, 清楚地, 有区别的

示例:

查看哪些部门有员工

Select distinct dept\_id from emp ;

#### (5)按照指定列上的值排序

**语法:**

**SELECT …. From 表名 order by 列名 ASC;**

**(升序排列,默认,可不写 [asc])**

**SELECT …. From 表名 order by 列名 DESC;**

**(降序排列)**

**SELECT …. From 表名 order by 列名1,列名2 DESC;**

(在列名1升序排列时,再按照列名2的降序排列)

Order: 排序,命令,规则

示例:查询员工的所有信息,记录行按工资的从小到大排序

Select \* from emp order by salary ;

Select \* from emp order by salary DESC ;

查询员工的所有信息,按照员工部门编号升序排序,若部门编号相同,再按生日排序

#### (6)查询时进行运算

**语法:**

**select 列名运算符值 ,…., from 表名;**

示例:查询员工的姓名及年薪

Select ename,salary\*12 from emp;

练习:公司给每人加薪500,查询出每个员工姓名,月薪,年薪

Select ename , salary+500 , (salary+500)\*12 from emp ;

Select ename 姓名, salary+500 月薪, (salary+500)\*12 年薪 from emp ;

#### (7)查询出满足特定条件的记录行

**语法:**

**select …. From …. Where 条件;**

其中的”条件”可以是多种表达式,如:

= , > , < , >= , <= , !=

示例:查询出30号部门的员工的所有信息

Select \* from emp where dept\_id = 30;

练习:查询出工资在18000及以上的员工的所有信息

Select \* from emp where salary >= 18000 order by salary;

查询出在1990-10-05日以后出生的员工所有信息

Select \* from emp where birthday < ‘1990-10-05’;

直接复制会报(Invalid gbk character string:’05\xA1’;)错

查询部门不在20部门的员工所有信息

Select \* from emp where dept\_id != 20;

Select \* from emp where dept\_id <> 20;

**其中的条件”条件”可以是多种表达式的组合,如**

**AND(并且) OR(或者) NOT(非)**

**示例:**

**查询出工资在10000~20000的员工的所有信息**

**Select \* from emp where (salary>=10000) and (salary<=20000);**

练习:

查询出1990年10月出生的员工;

Select \* from emp where birthday<’1990-10-31’ and birthday>’1990-10-01’;

Select \* from emp where birthday between ‘1990-10-01’ and ‘1990-10-31’;

查询出20号和30号部门的员工的所有信息;

Select \* from emp where dept\_id=20 or dept\_id=30;

**查询出20号,30号,40号,80号部门的员工的所有信息;**

Select \* from emp where dept\_id=20 or dept\_id=30 or dept\_id=40 or dept\_id=80;

Select \* from emp where dept\_id IN (20,30,40,80);

**IN:在某个范围内**

查询出不在20号,30号,40号,80号部门的员工的所有信息;

Select \* from emp where dept\_id!=20 and dept\_id!=30 and dept\_id!=40 and dept\_id!=80;

**Select \* from emp where dept\_id not in (20,30,40,80);**

#### (8)分页查询: …limit start , count; limit----mysql专属

**语法:**

**select … from …. Where …. Order by … limit start , count;**

start :表示从满足条件的哪一行记录开始读取,,第一行称为0

count:此次最多读取多少行记录

假设每一页显示5条记录:

第一页:…limit 0,5;

第二页:…limit 5,5;

第三页:…limit 10,5;

第四页:…limit 15,5;

第N页:…limit (N-1)\*5,5;

分页查询出10部门所有员工的员工信息,只要第一页/每页5条

Select \* from emp where dept\_id=10 limit 0,5;

分页查询出10部门所有员工的员工信息,只要第二页

Select \* from emp where dept\_id=10 limit 5,5;

#### (9)模糊查询 ------LIKE

**语法:**

**SELECT .. FROM .. WHERE … LIKE ‘%..%’;**

查询出姓名包含字母e的所有员工信息

Select \* from emp where ename=’e’ ; //错误

Select \* from emp where ename=’%e%’ ; //错误

Select \* from emp where ename LIKE ’%e%’ ;

**Sql查询中, % 代表任意多个任意字符 ; \_ 代表任意一个字符;**

查询出姓名中以m结尾的员工的所有信息

Select \* from emp where ename like ‘%m’;

查询出姓名中倒数第二个字符是i的员工的所有信息

Select \* from emp where ename like ‘%i\_’;

(不能直接复制,标点符号是中文的)

#### \*补充:

**如何查询部门编号为NULL的员工所有信息**

**NULL 不能用 = 和 != 进行判定!**

**Select \* from where dept\_id IS NULL;**

**查询部门编号不是NULL的员工所有信息**

**Select \* from where dept\_id IS NOT NULL;**

### 4.复杂查询

#### (1)分组函数

Count(列名/\*):查询出满足条件的记录行

Max(列名):查询指定列上的最大值

Min(列名):查询指定列上的最小值

Sum(列名):查询指定列上的所有数据的总和

AVG(列名):查询指定列上的平均值

语法

Select count(列名/\*) from ….; 查询出满足条件的记录行数

示例:查询所有员工的数量

Select count(\*) from emp ;

查询出10号部门的员工数量

Select count(\*) from emp where dept\_id=10;

查询出大内公司部门的数量

Select count(\*) from dept;

Group by 分组,按小组划分

查询出各部门的员工数量

Select dept\_id,count(\*) from emp group by dept\_id ;

查询出所有员工工资的最大值

Select max(salary) from emp;

查询出所有员工的工资的最小值

Select min(salary) from emp;

查询出所有员工的工资的平均值

Select avg(salary) from emp;

查询出每个部门编号及该部门的工资最大值,最小值

Select dept\_id,max(salary),min(salary),avg(salary) from emp group by dept\_id;

查询出员工工资最大的,姓名及工资

Select ename,salary from emp where salary=(select max(salary) from emp);

查询出所有员工的工资总和

Select sum(\*) from emp;

查询出每个部门的编号及部门所有员工的工资总和

Select dept\_id,sum(salary) sum from emp group by dept\_id;

#### (2)子查询

在一条查询语句中在嵌入另一条查询语句

嵌套的查询语句称为子查询

示例:查询出market部门的所有员工的所有信息

1. 根据部门名称查询其编号

Select did from dept where dname=’market’;

2.根据部门编号查询该部门所有员工的信息

Select \* from emp where dept\_id=10;

合并:

Select \* from emp where dept\_id=(select did from dept where dname=’market’);

(不能直接复制,’ ’ 中文的)

查询出工资比tiger所有信息

Select \* from emp where salary<(select salary from emp where ename=’tiger’);

查询出年龄比kate大的员工所有信息

Select \* from emp where birthday<(select birthday from emp where ename=’kate’);

(不能直接复制,’ ’ 中文的)

#### (3)跨表查询

跨表/多表查询,一次查询得到的结果集中的列来自于不同的表;跨表查询一定要防止出现笛卡尔积

示例:查询出每一个员工的姓名及其所在部门的名称

Select ename,dname from emp,dept;//错误 笛卡尔积

=====================================================

==================== sql-92=========================

Select ename,dept\_id,dname,did from emp dept where emp.dept\_id=dept.did;//防止出现笛卡尔积

提示:sql-92标准中的跨表查询无法显示连接条件为null的记录行,如员工的部门编号为null,该记录无法显示

================sql-99======================================

##### 内外连接

sql-99中:(1)内连接------作用等同于sql-92中的跨表查询

select ename,dname from emp inner join dept on emp.dept\_id=dept.id;

sql-99中:(2)左外连接: 会显示出(关键字…join)左侧表中所有的记录

select ename,dname from emp left outer join dept on emp.dept\_id=dept.id;

select ename,dname from emp left join dept on emp.dept\_id=dept.id;

sql-99中:(3)右外连接: 会显示出(关键字…join)右侧表中所有的记录

select ename,dname from emp right outer join dept on emp.dept\_id=dept.id;

select ename,dname from emp right join dept on emp.dept\_id=dept.id;

sql-99中:(4)全连接:会显示出两侧表中所有的记录

select ename,dname from emp full join dept on emp.dept\_id=dept.id;

//mysql 不支持

提示:使用结果集的合并操作,可以类似的实现’全连接’的效果,

但是有BUG;

#### (4)结果集合并

Union:用于将两个查询结果集合并为一个大的结果集

合并结果集,重复数据仅显示一遍:

Select…from…

Union

Select…from…;

合并结果集,允许出现重复数据

Select…from…

Union all

Select…from…;

## 面试题:DELETE 和 TRUNCATE 的区别是什么

# PHP

## 1.web服务器

web服务器的作用:相当于商店里的”售货员”,功能是:

1. 接受并理解客户端的请求
2. 查找客户端需要的资源(文件+数据)
3. 发送资源给客户端浏览器

Web服务器分为两类:

1. 静态web服务器;

提供的内容任何人在任何时间访问都是相同的

静态服务器软件:Apache httpd , MS IIS , NginX

工作原理:服务器上特定的目录下保存文件,让客户端请求该目录下的内容-----web服务器不会特殊处理文件内容.

1. 动态web服务器;

提供的内容不同人在不同时间访问都可能不同

动态网页中的内容是可变的,往往需要数据库访问,其他服务器访问才能实现,只有一些动态的编程语言才能实现,主流的动态网页编程语言:

.jsp 使用java编写网页

.Asp.net 使用c#.net编写网页

.PHP 使用php编写网页

.Node.js 使用服务器端JS编写网页

网络小知识:

1. 如何查看本机ip地址:ipconfig
2. 所有计算机都有一个特殊IP地址:120.0.0.1,永远指代当前计算机
3. 所有计算机都有默认的域名:localhost,永远指代当前计算机

## 2.PHP:

PHP: personal home page 后改为 PHP: Hypertext Preprocessor

<?php

echo "hellp &nbsp; world";

echo "<br>";

?>

## 面试题:如何自学一门编程语言

1. **了解背景:历史,特点,应用场合**
2. **搭建一个开发环境,输出 Hello World**
3. **变量和常量**
4. **数据类型**
5. **运算符**
6. **逻辑结构**
7. **通用小程序**
8. **函数和对象**
9. **常用组件,工具,框架**

**实用小项目**

## 3.PHP中的变量和常量

### 1.PHP中的变量

variable: 变量,指值可能发生改变的量.

数学中的变量:

PHP程序中声明变量的格式:

$变量名=值;

**变量名中可以包含数字,字母,'\_',但不能以数字开头,不能出现空白字符.若由多个单词组成,推荐使用"下划线命名法","小驼峰命名法","大驼峰命名法"**

**下划线命名法 $my\_office\_addr**

**驼峰命名法 小: $myOfficeAddr 大: $MyOfficeAddr**

### 2.PHP中的常量

constant: 常量,指一旦赋值就不能再改变的量

声明常量的格式:

const 常量名 = 值;

常量名中可以包含数字,字母,'\_',但不能以数字开头,不能出现空白字符.推荐使用全大写.习惯上,常量名都使用纯大写字母,若包含多个单词,使用"下划线命名法",如: MY\_FIRST\_CONSTANT

常量只能赋值一次,不能被改变

## 4. PHP中的数据类型----重点!!

php中声明变量或常量无需指定类型,但底层所有数据都是有数据类型的. var\_dump()函数可以输出变量的类型和值

PHP中的数据类型分为:"三大类八小种"

### (1)标量类型/值类型:

1.boolean 布尔类型

2.int/integer 整数

3.float/double 浮点型 php中的float等同于double

4.string 字符串

$a=020; //八进制,由0开头,即为八进制数

$b=0x20; //十六进制,由0x开头,即为十六进制数

转义字符

\" =>"

**单引号和双引号的区别:**

**双引号能识别字符串中的变量,并自动替换为变量的值**

### (2)复合类型

#### 1)array 数组类型

语法:

**$变量名 = [值1,值2,...]**

访问第i 个元素:

**$变量名[$i];**

获得数组中元素的个数

**count($数组名);**

给数组中添加一个新元素, 下标为count($arrScores)

**$数组名[]=新的元素值**

**注意:**

**(1)数组和普通变量不同**

**$x = 100; //int(100)**

**$y = [100]; //array()**

**(2)数组变量不能使用echo进行输出!(var\_dump()可以)**

**(3)保存在数组变量中的每一个值都需要有一个下标(序号),需要使用**

**count($数组名);返回指定数组中元素的个数**

##### (1)索引数组(Indexed Array)

每个元素的下标都是一个数字,形如0,1,2,3...N ,数组的总长度为N+1

**声明方式: $arr=[90,20,32]**

**访问元素: echo$arr[$i];]**

**添加元素: $arr[]=80;**

**获取长度: echo count($arr);**

##### (2)关联数组(Assocation Array):

每个元素的下标都是一个自定义的字符串,系统不会自动生成下标

**声明语法:** **$arr['lid' => 8 , 'title' => "戴尔燃7000" , 'price' =>5388 , 'isOnsale' => true];**

**元素数量 : echo count($arr);**

**获取元素 : echo $arr['title'];**

**添加元素 : $arr['shelfTime'] =123123213122;**

**注意:关联数组每个元素的下标都是自定义的字符串,不是0/1/2/3..**

**不能用for循环进行遍历**

#### 2)object 对象类型

### (3)其他类型

resource 资源类型

null/NULL 空类型

## 5.PHP中的运算符

### (1)算数运算符

+ - \* / %(取余) ++ --

**$n++是"先赋值再自增",++$n是"先自增再赋值"**

### (2)比较运算符

> < >= <= == ===(全等) != !==

== //不同类型,会自动先进行隐试转换

=== //全等 ,不会进行隐试转换,若类型不同,则返回false

### (3)关系运算符

&&(与/并) ||(或) !(非) (都是短路的)

### (4)赋值运算符

= , +=, -=, \*=, /= , %= $str .= "123"; 结果是计算结果

### (5)位运算符

将数字以二进制形式进行计算

<< 左移 >>右移

### (6)字符串拼接运算符

echo "字符串1" . "字符串2";

### (7)三目运算符

语法：表达式1？表达式2：表达式3；

若 表达式1 的结果为true 则整个表达式的值为 表达式2 的值

否则，（表达式的结果为false）整个表达式的值为 表达式3的值；

## 6.PHP中的逻辑结构

### (1)顺序执行

### (2)选择执行

#### 选择结构——if..else

语法:

**if(表达式){**

**要执行的语句;**

**}**

若表达式的值为true，则执行要执行的语句。

若表达式的值不是bool值，则会隐试转换

if(0) , if(null) , if("") , if(0.0)

都会相当于 if(false)

语法：

**if( 表达式 ){**

**条件为真时执行的语句1;**

**}else{**

**条件不满足时执行的语句2;**

**}**

含义：若“表达式”的值为true，则执行“**条件为真时执行的语句1;**”，否则执行“**条件不满足时执行的语句2;**”

语法：

**if( 表达式1 ){**

**条件满足表达式1时执行的语句1;**

**}else if( 表达式2 ){**

**条件满足表达式2时执行的语句2;**

**} else if( 表达式3 ){**

**条件满足表达式3时执行的语句3;**

**}else{**

**条件不满足所有表达式时执行的语句;**

**}**

#### 选择结构----switch..case..

语法:

**switch(变量){**

**case 值1: //变量名===值1**

**满足场景1时执行的语句;**

**break;**

**case 值2: //变量名===值2**

**满足场景1时执行的语句;**

**break;**

**.....**

**default: //变量名不等上述值**

**上述场景都不满足时执行的语句;**

**}**

**提示:if..else能完成switch..case所有的功能;只是在判定变量全等的情形下,后者看上去更简单些**

### (3)循环执行

#### 1.循环结构------while循环

语法:

**while(循环条件){**

**循环体;**

**}**

**while(true){**

**if(条件){**

**break;**

**}**

**}**

含义:当"循环条件"为真的时候,执行循环体,当"循环条件"为假时,退出循环

#### 2.循环结构 ------do...while 循环

语法:

**do{**

**循环体;**

**}while(循环条件);**

含义:执行循环体 , 再判定循环条件是否满足 , 若满足则再次执行循环主体 ,否则就退出循环体

**区别:do..while.. 循环主体至少执行一次; while..循环主体则可能一次都不执行**

#### 3.循环结构------------for循环

语法:

**for(表达式1 ; 表达式2 ; 表达式3){**

**循环体;**

**}**

**括号里面的 两个分号,一个都不能少**

含义:首先执行且仅执行一次表达式1 ------一般用于给计数器赋初始值,再执行表达式2,若为true,则执行循环主体;循环主体执行完后,再执行表达式3-----一般是计数器的改变;然后再次执行表达式2的判定,若为true,则执行循环主体.....

#### 4.循环结构----foreach

foreach:对于数组中的每一对元素,都看做一个下标遍历指向一个值变量, 对每一对元素都执行一次循环主体

**语法:**  **$key:键(下标) $value:值**

**foreach($数组名 as $key=>$value){**

**循环主体;**

**}**

**foreach($laptop as $value){**

**循环主体;**

**}**

**含义**:一次把数组中的每一对下标=>值,赋值为指定的两个变量($key和$value).执行一遍循环体.

**提示**:foreach循环可以遍历关联数组,也可以遍历索引数组

但关联数组只能用foreach遍历

**作用之一**:遍历数组(关联数组,索引数组)

练习:创建一个关联数组,保存一个商品信息,包括编号,名称,单价,上架时间,使用foreach遍历其中的每个元素

## 7. 通用小程序

九九乘法表

水仙花数

打星星

质数判定

闰年判定

## 8.函数和对象

### 1.函数------了解

**定义:**是一段预定义好,并可以反复执行(使用)的代码块,其中可以包含多条可执行语句

**语法:**

**function 函数名(参数列表){**

**函数执行体;//可以有多条语句**

**}**

**调用: 函数名(参数值 , ......);**

**函数名中可以包含字母,数字,下划线,但不能以数字和下划线开头,**

**使用"下划线命名法","驼峰命名法"**

**函数声明时,参数列表中可以声明0~N个参数,用逗号分隔开**

**函数声明时,可以声明一个返回值 return**

**return后面的语句无法执行,代表方法结束**

### 函数

#### 为什么要用函数:

封装"反复多次执行的代码" / "不易编写的代码"

#### 如何声明函数:

function 函数名(..){要执行的代码}

#### 如何调用函数

函数名(....);

#### 什么是形式参数(形参):

function f1($n1,$n2,$n3){...} // $n1,2,3就是形参

#### 什么是实际参数(实参):

f1(1,2,5); //1,2,5,就是实参

#### 什么是返回值:

function f1(..){.... return 值;}

$rtnValue = f1();

## 9.常用组件,工具,框架

## 10.实用小项目

## 使用PHP预定义的函数连接MySQL服务器

早期PHP提供了一套函数操作MySQL:mysql\_xxx()

后又提供了一套性能更加优秀的操作MySQL的函数库:

**mysqli\_XXX();**

**使用MySQLi函数操作连接MySQL服务器的步骤:**

**提示:操作过程与"命令行客户端连接MySQL服务器",过程是一样的**

**(1)连接到MySQL服务器**

**(2)向MySQL服务器发送SQL命令**

**(3)查看执行结果**

**(4)断开到MySQL服务器的连接---------可以省略**

即:总共只涉及到4个函数

### #1.连接到MySQL服务器,返回值是一个resource类型

**$conn=**

**mysqli\_connect('localhost','root',"","danei",3306);**//主机名,用户名,密码,要连接的数据库,端口号

//var\_dump($conn);

### #2.向MySQL服务器发送SQL语句

**$sql = "...";**

**$result=mysqli\_query($conn,$sql);**//在哪一个连接上,执行哪一条语句

//var\_dump($result);

**$result = mysqli\_query($conn,$sql);**

**上述语句的返回值**

##### (1)对于DML(insert/delete/update)语句

**执行失败,就返回false**

**执行成功,就返回true**

##### (2)对于DQL(select)语句

**执行失败,就返回false**

**执行成功,就返回 查询结果的描述对象**

### #3.查看执行结果

**if($result){**

**echo "成功";**

**}else{**

**echo "失败";**

**}**

### #4.关闭连接 可以不写 mysqli\_close($conn);

//mysql\_close($conn);

== //不同类型,会自动先进行隐试转换

=== //全等 ,不会进行隐试转换,若类型不同,则返回false

## 函数(/预定义变量):

### 0.echo "新用户的自增编号为:"; echo mysqli\_insert\_id($conn);

### 1.mysqli\_affected\_rows($conn);

**获得指定连接上刚刚执行的SQL语句影响的行数**

if($result){

echo "delete succ<hr>";

//读取DML(insert/delete/updata/)语句操作影响的记录数

echo "删除操作影响的行数".**mysqli\_affected\_rows($conn);**

}else{

echo "delete err<hr>";

echo $sql;

}

### 2.require('init.php');

#在当前位置包含指定的文件

**require(...);在项目中常用于包含多个页面的公共部分**

**init.php //(初始化php)**

**/\***

**require:包含**

**request:请求**

**\*/**

有1.php

$num1 =10;

echo $num1;

有2.php

$num2 =10;

echo $num2;

#echo $num1; #错误! 变量不能跨页面(文件)使用

**require(...);**

**echo** $num1;//正确

### 3.@,,,,,,, die

**#@用于屏蔽/压制 当前行代码产生的错误消息**

**@ $n=$\_REQUEST["num"];**

**if($n===null){//客户端未提交必须的请求数据**

**//向客户端输出一个提示性的消息,然后终止执行**

**die('未接收到客户端提交的num变量!');**

**}**

### 4.$\_REQUEST — HTTP Request 变量

该数组中保存了客户端请求中提交的数据(关联数组)

PHP预定义的变量------无需声明可以直接使用

### 5从查询结果集中抓取记录

#### 1.试着从查询结果集中抓取一行记录

**$row = mysqli\_fetch\_row($result);//返回索引数组**

**$row = mysqli\_fetch\_assoc($result);//返回关联数组**

#### 2.试着从查询结果集中抓取所有的记录行

**$rowList = mysqli\_fetch\_all($result);**

**$rowList = mysqli\_fetch\_all($result, MYSQLI\_ASSOC);****//返回的子数组是关联数组**

**var\_dump( MYSQLI\_ASSOC ); //int(1)**

**http://127.0.0.1/html/day13\_day03/login.php?uname=wnhua.li&upwd=fanbinigbing**

**? : 表示的是后面的数据是要提交给服务器处理的数据**

**key=value : key是提交给服务器的名称, value提交给服务器的值**

**& : 表示后续还有继续要提交的数据**

WEB的组成

1.WEB设计----jpg,psd

2.WEB前端----网页(html)

1.HTML-----网页实现

2.CSS--------修饰和美化网页

3.JS-----------JavaScript,

3.WEB后端(服务器)---php,jsp,db

# HTML5Basic

## 1.HTML快速入门

### 1.1.WEB基础

#### 1.1.1. Internet:网

是一个全球性的计算机互联网络

中文名称称为 "因特网", "国际互联网"

主要服务:

1. Telnet

2. Email

3. WWW

4. BBS - 电子公告板(论坛)

5. FTP - 文件传输协议

基本实现技术:

1.分组交换原理

2. TCP/IP协议簇

#### 1.1.2 WEB与Internet

WEB 是Internet上最流行的应用(网页应用)

WEB : 万维网 或 环球网 即 WWW(World Wide Web)

web的规范是由 W3C 来定义的

W3C : World Wide Web Consortium---万维网联盟

WEB主要是将各类型信息以及服务进行无缝连接

ex:百度贴吧,天涯论坛....

QQ邮箱,网易邮箱

WEB:

从广义的角度上讲是所有的网页集合

从狭义的角度上讲是一个具体的网页

#### 1.1.3 WEB的工作原理

WEB程序是基于 浏览器/服务器(B/S) 模式的应用

由 WEB服务器 ,浏览器 , 通信协议 三部分组成

通信协议:在web中采用的是HTTP通信协议

HTTP: Hyper Text Transfer Protocol

超级 文本 传输 协议

作用: 规范了数据是如何打包以及如何传递的

##### (一)web服务器:

1.主要功能

存储web上的内容信息(图片,文字,音视频)

能够响应浏览器的请求

提供最基本的安全功能

2.服务器产品

1.TOMCAT

2.IIS

3.APACHE

3.实现技术

1.php

2.JSP

3.ASP.NET

##### (二)WEB浏览器

1 .主要功能

1.代替用户向服务器发送请求(UA-User Agent)

2.作为HTML的解释器和内嵌脚本(JS)的执行器

3.能够用图形化的方式显示HTML

2.浏览器产品

1. Microsoft IE

2. Mozilla Firefox(火狐)

3. Google Chrome

4. Opera Opera(欧朋)

5. Apple Safari

#### 1.1.4.HTML概述

HTML 是编写网页(web)的一款语言,以.html或.htm作为结尾 的一个文件.

HTML : Hyper Text Markup Language, 超级文本标记语言

Hyper Text : 超文本,具备特殊功能的文本

普通文本a: 普通字符

超文本 a: 超链接

Markup : 标签,标记,规范了超文本的组成形式

<a></a>

Language : 语言,语法

#### 1.1.5. HTML基础语法

##### 1.标记

标记,也称为元素,在HTML中,用于描述功能的符号.

比如:<a>,<b>,<div>,<p>

标记在使用时需要用<>括起来

(1)封闭类型标记

也称为"双标记",必须成对出现

语法 : <标记></标记>

ex:

<div></div>

<a></a>

.... .....

(2)非封闭类型标记

也称为"单标记"或"空标记",可以没有结束

语法 : <标记>

或者

<标记/>

ex :

<img> 或者 <img/> 图片

<br> 或者 <br/> 换行

<hr> 或者 <hr/> 水平线

<input> 或者 <input /> 输入控件

##### 2.嵌套

在一对标记中,允许出现另一对标记,以此类推

注意:

1.注意嵌套顺序,一定要完整嵌套

ex:

<a>

<b>百度</b>

</a>

<a>

<b>

<u>百度</u>

</b>

</a>

2.被嵌套的标记,要用缩进(Tab)来表示

ex:

<a>

<b>百度</b>

</a>

不好的结构

1.<a><b><u>百度</u></b></a>

2. <a>

<b>

</b>

</a>

##### 3.属性 和 值

###### 1.什么是属性

属性是用来修饰元素的

###### 2.语法

2.1 属性必须声明在开始标记中

2.2 属性名称与标记名称间用 空格 隔开

2.3 属相名称 和 值 之间 , 用 " = " 连接

属性=值

<标记 属性=值></标记>

<标记 属性='值'></标记>

<标记 属性="值"></标记> 企业用的居多

2.4 多属性之间用 空格 隔开 , 排名不分先后

<标记 属性="值" 属性="值"></标记>

ex :

<p align="center">这是一个段落</p>

###### 3.四大基本属性

HTML 中任何标记都会具备的属性

1.id

定义元素在页面中独一无二的标识

2.title

当鼠标移入到元素上时所提示的文本

ex:<p title="让你移你就移啊">移上来试试</p>

3.class

类,定义元素要引用的类选择器(CSS中使用)

4.style

定义元素的 内联(行内)样式(CSS中使用)

ex:

定义一对div元素,定义id的属性值为container,定义

鼠标移入上来时候,提示为"容器".并让文本水平居中

对齐

<div id="container" title="容器"

style="text-align:center"></div>

###### 4.注释

允许编写在HTML源码中,但不想被浏览器所解释的内容,

可以放在注释中

<!-- 注释内容 -->

注意:

1.注释不能放在标记中

**错误!** <div <!-- --->></div>

2. 注释不能嵌套

**错误!** <!--

<!--

-->

-->

#### 1.1.6. HTML文档结构

##### 1.基本文档的组成

1.文档类型声明

作用: 告诉浏览器当前采用的是第几代的HTML版本

<!doctype html>

2.HTML页面

<html></html>

##### 2. html页面元素

html包含两个直接子元素

1.页面头部

<head></head>

作用:

定义网页的说明信息, 如 : 标题 , 编码 , CSS ....

2.页面主体

<body></body>

作用:

定义给用户去看的信息, 如 : 文字,图片,超链接....

3.页面头部 - head

允许包含的内容如下:

1.网页标题 :

<title>标题<title>

2.定义页面编码格式

<meta charset="utf-8"/>

UTF-8 : 8bit Unicode Transfermation Format

注意 :

要保证网页中的文本也是utf-8 的编码格式

才可以

## 2.文本标记

### 1.作用

改变文本在网页中的默认效果

### 2.特殊字符 & 文本标记

#### 1.特殊字符

在HTML页面中,若干个空格和回车,最终只会被解释成一个空格

1. &nbsp; -- 一个空格

2. &lt; -- < (less than)

3. &gt; -- > (greater than)

4. &copy; -- © (copyright)

5. &yen; -- ¥

6. &times; -- ×

#### 2.文本标记

##### 1.文本样式标记

<b></b> : 加粗显示文本

<i></i> : 斜体显示文本

<u></u> : 下划线显示文本

<s></s> : 删除线显示文本

<sup></sup> : 以上标的形式显示文本

<sub></sub> : 以下标的形式显示文本

特点:

多个标记与其他文本会在一行内显示

##### 2.标题元素

<hn></hn> n : 1~6

特点:

1. 一级标题最大 , 六级标题最小

2. 每个标题独占一行

3. 每个标题都会加粗

4. 每个标题都具备上下垂直的空白距离

5. 每个标题都具备 align 属性, 取值为:left,center,right

##### 3.段落元素

语法 : <p></p>

特点 :

1. 每个段落独占一块

2. 每个段落都有垂直的空白间距

3. 每个段落都具备 align 属性

##### 4. 换行元素

语法 : <br> 或 <br/>

##### 5.分割线元素(水平线)

语法 : <hr> 或 <hr/>

属性 :

1. size :

尺寸 , 以px 为单位的数值(类似于高度)

2.width :

宽度 , 以px或 % 为单位的数值

3.align :

水平线的水平对齐方式,取值为 left,center,right中

一个 默认为:center?

4.color :

水平线的颜色,取值为计算机中颜色表现方式

##### 6.预格式化

作用 : 保留源文档中的格式 ( 回车 和 空格 )

语法 : <pre></pre>

**慎用!!** (估计用不上)

##### 7.分区元素

1.块分区元素

作用 : 用于实现页面的布局

语法 : <div></div>

特点 :

1. 独占一块

2. 会具备align属性

2.行分区元素

作用 : 用于包裹文本并且为文本施加样式

语法 : <span></span>

##### 8.块级元素 与 行内元素

1.块级元素 :

独占一行 或 独占一列的元素都是块级元素

<h1>~<h6>

<p></p>

<hr>

<div></div>

<pre><pre>

特点:

1.都具备 align 属性

2.用于实现页面的布局

2.行内元素

多个元素会在一行内显示

<span>

<i>,<b>,<s>,<u>,<sub>,<sup>,<a>

特点:

1.会在一行内显示

2.主要用于处理文本

##### 9.注意 !!!!

**1. p标记中不能嵌套块级元素**

**2. 行内元素中不要嵌套块级元素 !!!!**

## 3.图像和连接

### 1.URL

#### 1.目录结构

目录 : WEB站点中的文件夹

目录结构 : 文件夹 与 文件夹 或 文件 之间的关系

#### 2.URL

URL(Uniform Resource Locator)

统一资源定位器,用来表示网络中任何一个资源的位置

URL(路径)在WEB中有三种表现形式

##### 1.绝对路径

1.什么是绝对路径

从资源文件所在的最高级目录下开始的完整路径表现形式

1.获取网络资源

由

协议名

http 或 https

域名(IP) /

www.baidu.com

路径目录 /

img

文件名 /

bd\_logo1.png

组成

https://www.baidu.com/img/bd\_logo1.png

2.获取本机资源

从文件所在盘符位置处开始查找

2.相对路径

1.什么是相对路径

从当前文件所在的位置处开始查找资源文件所经过的路径

2.三种表现形式

1. 同目录 , 直接找

2. 子目录 , 先进入

3. 父目录 , 先后退 ../ : 返回上一级目录

../../ : 返回上两级目录

3.根相对路径

永远都是从服务器所在的根目录处查找资源文件

语法 : 以 / 作为开始的路径表现形式

ex : /JD/images/Koala.jpg

注意 :

1.URL是严格区分 大小写 的

2.URL 中不允许出现中文

### 2.图像

#### 1.web中支持的网页格式

1. **jpeg**

\*.jpg , 普通图像标准格式

2. **gif**

\*.gif , 动图

3.**png**

\*.png , 支持透明色

#### 2.图像元素

语法 : <img>

属性 :

1 . src

要显示的图像的URL

2 . width

以px或%为单位的数值

3 . height

以px或%为单位的数值

注意:

如果只指定width 和 height 中的一个属性值的话,

那么另一个值也会跟着等比缩放

<h3>相对路径显示图片</h3>

<img src="img/Koala.jpg" alt="" width="300px">

<h3>根相对路径显示图片</h3>

<img src="/day11/img/Koala.jpg" alt="" width="300px">

### 3.链接

#### 1.什么是链接

链接,又称为超链接, 当用户点击时,能完成页面跳转的操作,就是链接

#### 2.语法

标记 :<a></a>

属性 :

1 . href

要跳转到的页面的URL

2 . target

目标 , 指打开新网页的形式

取值 :

(1) **\_blank**

在新标签页中打开网页

(2) **\_self**

(默认值)

在自身标签页中打开新网页

#### 3.链接的表现形式

1.**目标为资源文件的下载**

链接地址为 \* . rar / \* . zip的时候就为资源下载

2.电子邮件的链接

<a href="mailto:xxx@tedu.cn">联系我们</a>

**3.返回页面顶部的空链接**

**<a href="#">TOP</a>**

4.链接到JavaScript

<a href="JavaScript:JS脚本代码">执行</a>

#### 4.锚点(mao(同矛) dian)

在网页中的摸个位置处做记号 , 可以在任何位置处跳转到记号位置处

使用方式

##### 1.定义锚点

1.通过 a 标记的 name 属性 定义锚点

<a name="NO1">第一章</a>

2. **通过 任意标记 的 id 属性定义锚点**

<div id="NO2">第二章</div>

##### 2.链接到锚点

**<a href="#锚点名称"></a>**

**<a href="页面URL#锚点名称"></a>**

## 4.表格

### 1.什么是表格

表格是一个用来按照一定结构显示数据的元素

表格是由一些被称为"单元格"按照从上到下,从左到右的顺序排列在一起而形成的

概念: 表格, 行 , 单元格(列)

### 2.表格的语法

创建表格 : <table></table>

创建表行 : <tr></tr> ----- Table Row

创建单元格 : <td></td> ---- Table Data

一个表格(table)中允许包含若干行(tr),一行(tr)中允许包含若干单元格(td)

注意 :

1.在HTML中,默认情况下,每行中的列数都是相同的

2.表格中的td 可以 被 th(行标题 或 列标题) (Table Head) 所取代

### 3.表格的属性

#### 1.表格(table)的属性

1.width

宽度 , 以 px 或 % 为单位的数值

2.height

高度,以px为单位的数值

3.align

设置表格的对齐方式

取值 : left / center / right

4.border

设置表格的边框宽度

以px为单位的数值,默认值为0

5.cellpadding

单元格内边距:单元格内容 与 单元格边框 之间的 距离

以 px 为单位的数值

6.cellspacing

单元格外边距 : 单元格 与 单元格 之间的距离

以 px 为单位的数值

7.bgcolor

背景颜色

#### 2.表行(tr)的属性

1.align

设置当前行的内容的水平对齐方式

取值 : left / center / right

2.valign

设置当前行的内容的垂直对齐方式

取值 : top / middle / bottom

3.bgcolor

设置当前行的背景颜色

#### 3.单元格(列)(td/th)的属性

1.align

设置当前列的水平对齐方式

2.valign

设置当前列的垂直对齐方式

3.width

设置当前列的宽度

4.height

设置当前列的高度

5.colspan

设置单元格跨列

6.rowspan

设置单元格跨行

#### 4.表格的标题

标记 : <caption></caption>

注意 :

1.该表即要书写在<table>开始标记之下

2.该标记最多只能有一个

#### 5.表格的复杂应用

##### 1.行分组

表格可分为3个部分:

1.表头行分组

允许将表格的第一行被分为表头行分组

<thead>></thead> (里面包含table的第一行<tr>..</tr>)

2.表主体行分组

表格的内容可以被分为若干个表主体行分组

每个表主体行分组中,允许包含若干的连续行

<tbody></tbody>

3.表尾行分组

允许将表格的最后一行被分为表尾行分组

<tfoot></tfoot>

注意:如果表格中的行不显示指定分组的话,那么默认都分

在tbody里面

##### 2.不规则表格(重点)

创建的表格每行的列数可能是不相同的,称之为不规则表格

1.通过td的colspan 实现"跨列"操作

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,要删除出去

2.通过td的rowspan 实现"跨行"操作

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,要删除出去

##### 3.表格的嵌套

允许在一个表格中再嵌入另一个表格

被嵌套的表格,必须放在td中

#### 6.display:table

表格的表现形式,所以表格不是块级元素,也不是行内元素,就称为"table元素"

特点:

1.表格的尺寸(宽度)是以内容为准

2.每个表格独占一行

## 5.列表

### 1.什么是列表

默认的列表,是按照从上到下的顺序进行数据排列的一种结构

### 2.列表的组成

列表是由 "列表类型" 和 "列表项" 两部分组成

列表类型 : 分为 "有序列表" 和 "无序列表"

列表项 : 显示在列表中的若干项内容,默认情况下,每个列表项前,

都会有"列表项的标志"

### 3.列表的语法

#### 1.有序列表

列表类型 : <ol></ol> -- Order List

列表项 : <li></li> -- List Item

**属性** :

1.type

类型,列表项标志的类型

取值:

1. 1 , 默认值,按数字方式显示

2. a , 按小写字母的方式显示

3. A , 按大写字母的方式显示

4. i , 按小写罗马数字的方式显示

5. I , 按大写罗马数字的方式显示

2. start

列表项的起始编号,是从 几 开始

#### 2.无序列表

列表类型 : <ul></ul> ---- Unorder List

列表项 : <li></li>

**属性**:

1.type

列表项前的标志的类型

取值:

1.disc , 默认值,实心圆

2.circle , 空心圆

3.square , 实心矩形

4.none , 不显示列表项标志

### 4.列表的嵌套

在一个列表中又嵌套另一个列表

被嵌套的列表只能放在li中

### 5.定义列表

1.什么是定义列表

往往用于要给出一类事物的定义的情形

2.语法

<dl></dl> : 来表示一个定义列表

在<dl>中 允许包含以下内容:

<dt></dt> : 来表示定义列表中的标题(待解释的名词)

<dd></dd> : 来表示定义列表中的内容(名词对应的解释)

3.使用场合

"图文混排" 时使用

## 3.结构标记

### 1.结构标记的作用

用于提升标记的"语义性"

### 2.常用结构标记

#### 1.<header>元素

作用 : 表示页面 或 某部分内容的头部

语法 : <header></header>

#### 2.<nav>元素

作用 : 表示页面中的导航链接区域

语法 : <nav></nav>

#### 3.<section>元素

作用 : 表示主体内容中的小节,可以当做主体内容

语法 : <section></section>

#### 4.<article>元素

作用 : 表示页面中的 文字描述性 居多的内容, 如:新闻,

帖子信息,微博条目,博客内容

语法 : <article></article>

#### 5.<aside>元素

作用 : 用于定义页面中边栏的信息内容

语法 : <aside></aside>

#### 6.<footer>元素

作用 : 用于定义页面中的底部信息内容

语法 : <footer></footer>

## 4.表单(重难点)

### 1.表单的作用

表单用于收集用户信息,并提交信息到服务器

在HTML中表单的组成

1.表单元素

整理用户最后输入好的信息,并且提交给服务器

2.表单控件

用于与用户交互的控件元素(如 : 文本框 , 按钮...)

### 2.表单元素(重点)

#### 1.语法 : <form></form>

注意 : 在<form>中的表单控件的数据是可以提交给服务器的,其

他的不行

#### 2.属性 :

##### 1.action

作用:定义表单提交时发生的动作,实际上定义的是服务器

端处理程序的地址

ex:

<form action="login.php">

默认是提交给自己

##### 2.method

作用 : 定义表单数据的提交方式

取值 :

**1.get(默认值)**

1.以明文的方式将数据提交给服务器

2.安全性 很低

3.大小限制为 最大2KB

4.在向服务器要数据时,使用get提交方式(如:搜索等)

**2.post**

1.以隐式的方式将数据提交给服务器

2.安全性 很高

3.没有提交大小限制

4.需要将数据传递给服务器进行处理的时候,用post

提交方式(如 : 登录 , 注册 , 文件提交 )

**3.put**

... ....

**4.delete**

... ....

##### 3.enctype

作用 : 指定表单数据进行编码的方式(允许将什么

样的数据提交给服务器)

取值 :

**1.application/x-www-form-urlencoded**

默认值,允许将普通数据,特殊字符,都能提交给服务器

**2.multipart/form-data**

允许将 文件 提交给服务器

3.text/plain

只能将普通数据提交给服务器

(特殊字符,文件则无法提交)

**get提交方式,地址栏的体现**

**http://127.0.0.1/html/day12\_day02/login.php?uname=sad**

**post提交方式,地址栏的体现**

**http://127.0.0.1/html/day12\_day02/login.php**

### 3.表单控件

#### 1.表单控件分类

1. input元素(重难点)

2.textarea元素

3.select 和 option 元素

4.其他元素

#### 2.input 元素(重点)

##### 1.作用

收集用户信息,允许让用户输入 或 选择数据

##### 2.语法

###### 1.标记

<input> 或 <input/>

注意 :

input可以有各种各样的表现形式

(如,文本框,密码框,单选按钮,复选按钮)

无论表现成什么样子,都会具备一些相同的属性

###### 2.属性

1.type

根据不同的type值 , 可以创建不同的控件

2.name

定义控件的名称,提供给服务器使用

3.value

控件的值,提供给服务器使用

4.disabled

禁用控件,无值属性,只需要有属性即可

###### 3.input详解

1.文本框 和 密码框

文本框 : type="text"

密码框 : type="password"

特殊属性 :

1.maxlength

最大长度,限制输入的字符数

2.readonly

只读,无值的属性,只要属性出现在元素中即可

disabled 与 readonly 的异同点

1.相同点

只能看,不能改

2.不同点

1.disabled

是没有办法提交给服务器的

2.readonly

可以提交给服务器使用

3.placeholder

占位符,默认显示在控件上的文字,无法提交给服

务器(不是值(value))

2.按钮

1.提交按钮 : <input type="submit">

作用 : 将表单中的控件的数据提交给服务器

2.重置按钮 : <input type="reset">

作用 : 将表单中的控件的数据恢复到初始化的状态

3.普通按钮 : <input type="button">

作用 : 没有功能,用于执行客户端脚本(Javascript)

属性 :

1.value

可以作为按钮上显示的文本

4.<button></button>

作用: 表示为一个按钮

属性:

type : submit / reset / button

3.单选按钮 和 复选框

单选按钮 : <input type="radio">

复选框 : <input type ="checkbox">

属性:

1.name

一组的单选按钮 或 复选按钮,name属性值要一致

2.checked

设置默认被选中,无值属性

4.隐藏域 和 文件选择框

1.隐藏域

1.作用

想提交给服务器,但不想给用户看的数据要放在

隐藏域中

<input type="hidden">

2.文件选择框

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

注意 :

1.该标记不需要给标记

2.**form的method属性必须为 post**

3.**form的enctype属性必须为 multipart/form-data**

#### 3.textarea元素

多行文本域/文本框

1.语法 :

<textarea>文本</textarea>

属性:

1.name

当前控件的名称,提供给服务器使用

2.cols

指定文本域的列数

3.rows

指定文本域的行数

4.readonly

只读

#### 4.select 和 option 元素

选项框

1.语法

1.标记 & 属性

1.<select></select>

作用 : 提供最基本的选项框,里面允许有若干个"选项"

属性 :

1.name

控件的名称,

2.size

选项框中默认显示选项的数量

如果取值大于1,则为滚动列表

否则的话,则为下拉选项框

3.multiple

设置多选,

滚动列表允许实现多选

(配合Shift 或 Ctrl键位一起使用)

无值属性

2.<option></option>

1.作用

选项框中的选项,在一对<select>中,

允许出现若干的<option>

2.语法

<option>显示的内容</option>

属性:

1.value

选项的值,提供给服务器使用

2.selected

预选中,无值属性

#### 5.其他元素

1.label元素

1.作用

关联文本与表单控件的,关联成功后,

点击文本的时候就如同点击表单控件一样

2.语法

<label>文本<label>

属性:

1.for

要关联的表单控件的ID

2.fieldset 和 legend 元素

1.作用

为控件分组,提供分组的显示效果

2.语法

<fieldset>

<legend>分组的标题</legend>

...............

</fieldset>

#### 6.iframe 浮动框架

1.作用

在一个浏览器中显示多个网页的内容

相当于是在一个网页中,嵌入另一个网页

2.语法

<iframe></iframe>

属性

1.src

要嵌入的网页的url

2.width

指定浮动框架(嵌入的网页)的宽度

3、height

指定浮动框架(嵌入的网页)的高度

4. frameborder

指定浮动框架的边框，

不想要边框的话将该属性设置为0

### 4.新表单控件(新input元素)

#### 1.电子邮件类型

功能:

要求输入的数据必须符合Email的规范,否则不允许提交表单

<input type="email">

#### 2搜索类型

功能:

在基本的文本框基础上提供了一个快速清除的操作

<input type="search">

#### 3.URL类型

功能:

输入的数据必须满足URL(网址)的规范

<input type="url">

#### 4.电话号码类型

功能:

主要针对移动端的设备,pc机无效,一但元素被激活时,

就会在设备中显示出一个拨号键盘

<input type="tel">

#### 5.数字类型

功能 : 提供了非常方便的数字选择和验证功能

<input type="number">

属性:

1.min

当前控件允许接收的最小值

2.max

当前控件允许接收的最大值

3.step

点击按钮更换数字时的变化长度,默认为1

#### 6.范围类型

功能:

提供一个滑块控件,允许用户去选择数据,

多数要配合JS一起用

语法 :

<input type="range">

属性:

1.min

2.max

3.step

4.value

#### 7.颜色类型

功能:

提供最基本的颜色拾取控件

<input type="color">

#### 8.日期类型

功能 :

提供一个日历让用户选择

<input type="date">

#### 9.周类型

功能 :

与date类似, 但只能选择周

<input type="week">

#### 10.月类型

功能 :

与date类似, 但只能选择月

<input type="month">

# JavaScript 基础

## 1.JavaScript 概述

### 1.什么是JavaScript(JS)

JS是一种运行于JS解释器/引擎中的解释型脚本语言

JS运行环境 : 解释器 / 引擎

1.独立安装的JS解释器 , 如 NodeJS

2.嵌入在浏览器内核中的JS解释器

解释型脚本语言

1.纯文本组成

2.执行时,逐条解释运行,不用经过编译

应用场景:

PC,平板电脑,手机,机顶盒,......

### 2.JS的发展史

1.1992年Nombas 为自己的软件 CEnvi开发了一款脚本语言

ScriptEase, 可以嵌入在网页中运行

2.1995年 Netscape(网景)为自己的Navigator2.0开发了

一种脚本语言LivesScript,后来更名为JavaScript

3.1996年 Microsoft 为了进军浏览器市场,在IE3.0开发了一个

JavaScript的克隆版JScript

4.1997年 网景公司将JavaScript1.1 版本提交给了

ECMA(欧洲计算机制造商联合会)组织.

核心也送给了ECMA,称之为ECMAScript(简称为ES)

完整的JS组成的三部分

1.核心(ECMAScript)

提供了最基本 语法(如: 变量, 函数, 循环)

2.文档对象模型( DOM, Document Object Model)

提供了 与 HTML元素进行交互的操作

3.浏览器对象模型( BOM, Browser Object Model)

提供了 与 浏览器进行交互的操作

### 3.JS的特点

1.JS可以由任意文本编辑工具进行编写

2.无需编译,直接由JS解释器运行

3.弱类型的语言

4.JS也是面向对象的编程语言

### 4.JS的用途

#### 1.运行在客户端的JS

1.客户端的数据计算

2.表单的输入验证

3.浏览器事件的触发和处理

4.网页的特效制作

5.服务器的异步数据提交

#### 2.运行在服务器端的JS

1.分布式运算

2.实时服务器

3.网络应用

.... .....

## 2.JavaScript 基础语法

### 1.浏览器的内核

内核的作用 : 负责整个页面的渲染,由两部分组成

1.内容排版引擎 - 解析 HTML / CSS

2.脚本解释引擎 - 解析 JavaScript

|  |  |  |  |
| --- | --- | --- | --- |
| 浏览器 | 内核 | 排版引擎 | 脚本引擎 |
| IE | Trident | - | Chakra |
| FireFox | Gecko | - | Monkey系列 |
| Safari | Webkit | Webcore | Nitro |
| Chrome | Webkit | Webcore | V8 |
| Opera | Presto | - | Carakan |
| Opera(2013) | Webkit | Webcore | V8 |

### 2.搭建JS运行环境

#### 1.独立安装的JS解释器 - NodeJs

console.log("Hello World");

在控制台中,打印输出一句话(Hello World)

#### 2.嵌入在浏览器内核中的JS解释器

##### 2.1.直接在浏览器的控制台中输入脚本并执行

##### 2.2.将JS脚本嵌入在HTML页面中执行

###### 1).html元素事件

###### 2).<script>元素中嵌入JS脚本

<script>

**console.log("打印在控制台上的信息");**

**document.write("显示在网页上的一句话");**

**window.alert("以弹出框的方式显示一句话")**

... ...

</script>

###### 3).在外部脚本文件中嵌入JS 代码

1.创建js 脚本文件(\*\*\*.js)

2.在脚本文件中编写JS脚本代码

3.在页面中对脚本文件进行引用

<script src="\*\*\*.js"></script>

注意:

在引入的js文件的<script>中不允许

再编写JS脚本代码

### 3.JS调试

#### JS的代码错误

JS是解释型脚本语言,碰到错误,解释器则终止执行.

但不会影响后续"块"的代码执行

一对<script></script> 是一个"块"

### 4.语法规范

1.语句 - 能被JS引擎所解释的一句代码

1.由表达式,关键字,运算符组成

2.严格区分大小写

name,Name是两个东西

3.每一条语句都是由 ; 作为结尾

4.语句中所有的符号都是 英文

5.通常情况下,每条语句独占一行

2.注释

1.单行注释 : //

2.多行注释 : /\* .... \*/

## 3.JavaScript 变量 和 常量

### 1.变量

#### 1.什么是变量

变量(Variable)

一个允许被反复使用并且可以变化的数据

#### 2.变量的声明

##### 1.声明变量

var 变量名;

##### 2.为变量赋值

变量名=值;

注意:

1.变量最好被赋值后再使用,否则的话,值为undefined

2.声明变量时,最好加上var关键字

##### 3.变量声明的简便写法

var 变量名 = 值 ;

ex :

1.声明变量保存用户姓名

var uname="张无忌";

2.声明变量保存用户年龄

var uage=18;

##### 4.一条语句中声明多个变量

var 变量名1,变量名2,....;

var 变量名1=值1,变量名2=值2,变量名3,变量名4=值4,...;

##### 5.变量的使用

console.log(变量名);

document.write(变量名);

window.alert(变量名);

以上代码是将变量值放在不同的作用中

注意 : 千万不能加 " " 或 ' '

字符串的拼接 : 使用 " + "

##### 6.变量名的命名规范

1.可以包含字母,数字,下划线以及 $

2.不能以数字开头

3.尽量见名知意

4.尽量采用驼峰命名法

##### 7.变量的复杂使用

###### 1.未经初始化的变量

1.定义变量 但 未赋值

var userName;

console.log(userName);

结果为 : undefined

2.直接使用未被声明过的变量

console.log(userAge);

结果为 : 语法错误

###### 2.对变量进行赋值的操作 - SET操作

只要变量出现在 = 的左边,

一律都是赋值操作

var age = 18;

var newAge = age //将age的值赋给newAge

//针对newAge是赋值操作,针对age是取值操作

###### 3.对变量进行取值的操作 - GET操作

只要变量没出现在 = 的左边 ,一律都是取值操作

var age = 18

var newAge = age //将age的值赋给newAge

//针对newAge是赋值操作,针对age是取值操作

age = age +1

= 右边的age,是取值操作(GET)

= 左边的age,是赋值操作(SET)

### 2.常量

#### 1.什么是常量

一经声明,就不允许被修改的数据叫常量

#### 2.声明常量

const 常量名 = 值 ;

注意:

1.常量名称一般采用全大写形式

2.常量声明时必须赋值

## 4.JavaScript 数据类型

### 1.什么是数据类型

数据在内存中的类型是什么样的,

解决了数据在内存中占据空间大小的问题

### 2.JS的数据类型分为 :

#### 1.基本数据类型 的值

又称为"原始数据类型"

**值直接保存在变量本地的数据类型**

##### 1.number 数字类型

变量中保存的值是数字形式的,就是数字类型,

并且在使用过程中,不能用" "或' '括起来的数字,是数字类型

var age = 32; //数字类型

var myage = "32"; //字符串

可以存储32位的整数,以及64位的浮点数

8bit = 1byte (8位 = 1字节)

1024byte = 1KB

整数占 4个字节 , 浮点数(小数)占8个字节

有浮点数参与到运算的话,

就容易造成精度丢失的问题

**var result = 数字/变量.toFixed(保留的位数);**

###### 表示整数 :

1.十进制整数

由0~9 十个数字组成

逢十进一

2.八进制整数

由0~7八个数字组成

逢八进一

在JS中,八进制数字是以0作为开始的

var n1 = 010 ;

console.log(n1); //输出 8 ,

console.log()里统一用十进制输出

通过n1.toString(8) ,

函数将n1转化为8进制(字符串?)

3.十六进制整数

由0~9,A~F来组成

逢十六进一

在JS中,十六进制数字是以0x作为开始的

var n1 = 0xA;

###### 表示浮点数

1.由小数点计数法表示

var n1 = 36.85 ;

2.由指数方式表示

var n2 = 36.85E5 //36.85\*(10的5次方)

##### 2.string 字符串类型

1.什么是字符串

表示的是一段信息, 在使用时用"" 或 ' '标示出来

由 Unicode 字符 , 数字,标点来组成

JS中的每个字符也都有一个Unicode码来进行表示

ex:

"张"的二进制表示法

"张".charCodeAt() : 10进制表示

"张".charCodeAt().toString(2) : 2进制表示

"张".charCodeAt().toString(16) : 16进制表示

将Unicode码转换为JS字符

var str = "\u5f20"; //输出"张"//

ex :

中文汉字的Unicode范围

从4e00 开始 到 9fa5 结束

**在JS中的特殊字符需要通过转义字符来表示**

如

**\" : 表现为 "**

**\' : 表现为 '**

**\\ : 表现为 \**

**\n : 表现为 换行**

**\t : 表现为 缩进 (tab)**

##### 3.boolean 布尔类型

1.作用

通常用于表示条件的结果

2.取值

true 或 false

在做+-\*/%运算的时候,

true可以当做1运算,

false可以当做0运算

##### 4.null 空类型

表示的是对象为 空 (释放对象的空间)

##### 5.undefined

语义: "数据不存在"

在声明变量但为赋值的时候,

或访问的对象的属性不存在时也是undefined

#### 2.引用数据类型 的对象

**值无法保存在变量本地的复杂数据类型**

如果一个复杂的数据结构，同时保存多个数据，就不能直接保存在变量中

存储在window之外的一块独立存储空间

独立存储空间中可同时存储多个数据，拥有唯一的地址

变量名仅保存存储空间的地址

### 3. 数据类型的转换

#### 1.如何查看变量或数据的数据类型

在JS中允许使用typeof()查看变量或数据的数据类型

语法 : var result = typeof(变量或数据);

#### 2.隐试转换

不同类型的数据之间会进行自动转换来保证类型统一

数字 + 字符串 : 将数字转换成字符串

console.log(123+"abc"); //123abc

数字 + 布尔值 : 将布尔值转换成数字

true 转换为 1

false 转换为 0

字符串 + 布尔值 : 将布尔值 转换 为 字符串

true 转换为 字符串 "true"

false 转换为 字符串 "false"

布尔值 + 布尔值 : 转换为数字类型

true 转换为 1

false 转换为 0

true + false = 1;

#### 3.显示转换 - 由转换函数完成的转换操作(强制转换)

##### 1.toString()

将任意类型的数据转换为string

语法:

var result = 变量.toSting();

##### 2.parseInt()

将任意类型的数据转换为 整数

语法 :

var result = parseInt(带转换的数据);

result 是最终转换完成的整数结果

注意:

1.碰见第一个非数字字符,则停止转换

var result = parseInt("35abc");//35

var result = parseInt("35.34");//35

2.如果第一个字符就是非数字字符,结果为**NaN**(Not a Number)

var result = parseInt("a32bc");//NaN

##### 3.parseFloat()

将任意类型的数据转换为 浮点类型

var result = parseInt("35.563");//35.563

var result = parseInt("A35.563");//NaN

var result = parseInt(".563");//

##### 4.Number()

将任意类型的数据转换为 number类型

注意 :

如果转换的数据类型中包含非法字符,则为NaN

语法 :

var result = Number(待转换的数据);

ex:

var result = Number ("35.563");//35.563

var result = Number ("35.563A");//NaN

## 5.JavaScript 运算符 和 表达式

### 1.表达式

a = 5 ; // a 的值 为5, 表达式的值 也为5

### 2运算符

#### 1.算术运算符

+ , - , \* , / , % , ++ , --;

##### 注意:

###### 1.自增 和 自减 , 如果单独使用的话,

无论在 前缀 还是 后缀,结果都是相同的

var a=5; a++; console.log(a); //6

var a=5; ++a; console.log(a); //6

###### 2.配合其他语句一起使用, ++在后,先使用变量的值在自增

var a=5; console.log(a++); //打印5, 打印完后 a= 6

###### 3.配合其他语句一起使用, ++在前,先使用变量的值在自增

注意:

- , \* , / , % , ++ , --;

两边的操作数如果不是数字类型,

解释器会隐试的转换为数字类型,

转换成功则正常运算,

否则结果为NaN

ex :

"5" \* "3" = 15 ; "a"\*"b"=NaN

#### 2.关系运算符

##### 1.什么是关系运算符

判断数据之间大小 和 相等关系的,

一般是组成条件判断的 重要运算符之一

##### 2.运算符

> , < , >= , <= , == , != , ===,!==

###### 注意 :

1.如果运算符两边都是数字类型,则比较数字大小

2.如果运算符两边有一个边是数字的话,

则试图将另外一个数据 隐试转换 为数字,

再进行比较数字的大小

3.如果运算符两边都是字符串的话,

则比较每位字符的Unicode码,Unicode码大的为大

###### 注意 :

Q: 如何判断一个数据是否为数字

通过 isNaN() 判断一个数据 是否为 非数字

var result = isNaN(数据);

如果 数据 是数字的话, result 为 false

如果 数据 不是数字的话, result 为 true

#### 3.逻辑运算符

##### 1. 运算符

&& , || , !

##### 2. 作用

关联多个条件的

true && false : false

false && false : false

true || false : true

!true : false

##### 3.短路逻辑

###### 1.短路与

条件 && 操作

###### 2.短路或

操作1 || 操作2

var input = prompt("输入一段数据:");

var result = input || "这个家伙很懒...";

注意:

null , "", false , undefined , 0 ,0.0

以上内容,在条件中,全部当假处理

#### 4.位运算符

##### 1.什么是位运算符

将数字转换为二进制后,再进行运算

位运算会以整数的方式进行运算

##### 2.位运算符

###### 1.按位与 &

语法 : a & b

语义 :

1.将两个数字都转换为2 进制数 , 然后逐位比较

ex :

5 & 3

5 : 101

3 : 011

2.每位上对应的两个数字都为 1 的时候 ,

那么该位的结果为1, 否则 ,该位结果为 0

ex :

5 & 3

5 : 101

3 : 011

======================

001

结果 为 1;

作用 : 快速判断是否为奇数

1 & 1 = 1 ; 2 &1 =0 ; 3 & 1 = 1; 4 & 1=0; 5 & 1=1;

###### 2.按位或 |

语法 : a | b

语义 :

1. 将两个数字都转化为2进制数,然后逐位比较

2.每位上对应的两个数字,至少有一个为1的话 ,

那么该位的结果为1,否则,该位结果为0

ex :

5 & 3

5 : 101

3 : 011

======================

111

结果 为 7;

作用 : 快速取整(将小数取整时使用)

num | 0 ; ex : 59.234 | 0 = 59

###### 3.按位 异或 ^

语法 : a^b

语义 :

1. 将两个数字都转化为2进制数,然后逐位比较

2. 每位上对应的两个数字,相同时该位的结果为0,

不同时,该位的结果 为 1

ex :

5 & 3

5 : 101

3 : 011

======================

110

结果 为 6;

作用 : 快速 交换位置(只能做数字的交换)

var a = 5;

var b = 3;

a = a ^ b;

console.log(a);

b = b ^ a;

console.log(b); // b = 5

a = a ^ b;

console.log(a); // a = 3

#### 5.赋值运算符

##### 1. 赋值运算符 =

将右边的表达式 计算出结果 赋值 给 左边的变量

var a = b = c = 50;

##### 2. 扩展的赋值运算符

+= , -=, \*=, /= , %= , &= , |= , ^=

a += b; ==> a=a+b;

#### 6.条件(三目)运算符

单目运算符 : 只有一个操作数的运算符

++ -- !

双目运算符 : 有两个操作数的运算符

+ , - , \* , / , %

&& , ||

> , < , >= , <= , == , != , ===,!==

& , | , ^ , << ,>>

**三目运算符 : 有三个操作数**

? :

条件? 结果1 : 结果2

#### 7.使用 "+" 进行字符串的拼接

+ 用于 两个数字/布尔 ,表示 算术运算

+ 用于字符串 , 表示的是 字符串 的拼接

#### 8.运算符优先级

var num = 7;

var result = num & 1 == 1;

console.log(result); // 1

**以下优先级 , 由高到低**

1.() , [] , . (小括号,中括号,点)

2. ++ , -- ,!

3. \* , / , %

4. + , -

5. << , >>

6. < , <= , > , >=

7.== , != , === , !==

8. &

9. ^

10. |

11. &&

12. ||

13 . ? :

14. =, += , -=, \*=, /= , %= , &= , |= , ^= ...

## 6.JavaScript 函数

### 1.什么是函数

函数 ( function ) , 也称为方法

是一段预定义好,并可以被反复使用的代码块

1.预定义好:

提前声明,但不会马上被执行的内容

2.反复使用:

可以被多个元素,多次调用

3.代码块

函数中允许包含多条可执行的语句

函数的本质上也是一个功能完整的对象

### 2.函数的 定义 和 调用

#### 1.普通函数的声明 和 调用

语法 : function 函数名(){ 若干条可执行语句; }

ex :

function sayHello(){

console.log("欢迎来到王者荣耀");

console.log("敌人还有5秒钟到达战场");

}

函数的调用

在任何JS的合法位置处, 都允许通过

函数名();

来实现函数的调用

#### 2.带参数的函数的 声明 和 调用

##### 1.什么是参数

函数在运行中,需要外部传递进来的辅助运算的数据

参数, 可以让函数变得更加灵活

##### 2.语法

function 函数名(参数列表){

若干条可执行语句;

}

##### 3.带返回值函数的声明

1.什么是返回值

函数体内计算出一个数据带给函数的外部

2.语法

function 函数名(参数列表){

语句1;

语句2;

return 数值

}

调用

var result = 函数名();

//result 中保存的就是函数的返回值

用过的带参数的返回值

var result = parseInt(35.3);

#### 3.变量 的作用域

##### 1. 什么是作用域

作用域指的是变量或函数的可访问范围

在 JS 中 作用域 分为两种 :

1.函数作用域

只能在当前函数被访问的内容

2.全局作用域

一经定义, 在代码的任何地方都能访问

##### 2.函数作用域中的变量

又称为 " 局部变量", 只能在定义的函数中被访问到,

离开定义的函数就无法使用

注意:

所有的参数 , 都是局部变量

##### 3.全局作用域中的变量

又称为 "全局变量"

一经定义好,在代码的任意位置处都能访问

如何声明全局变量

1.直接在<script>标记中声明的变量

2.声明 变量 不使用 var

注意: 该方式尽量少用,容易出问题

##### 4.声明提前

JavaScript 程序在 正式执行前 , 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端,但对变量的赋值还保留到原位置

function fun3(){ console.log(n); var n=1;console.log(n)} //undefined ,1

**注意: 所有的变量声明,尽量放在所在作用域的最顶端**

### 3.函数

#### 1.值传递

基本类型的数据在做参数的时候,

都是按值传递的方式进行传递的

引用类型的数据 在做参数的时候,都是按引用的方式进行传递的

1.什么是值传递

基本数据类型在做参数传递时,传递的并不是值本身,

而是值的一个副本

注意 :

以后在做基本数据传递参数时,尽量不要在函数体中修改值,

因为改完后对实际数值是没有影响的

#### 2.函数的作用域

也分为

全局作用域的函数 - 全局函数

以及

函数作用域的函数 - 局部函数

1.全局函数

在最外层<script>中定义的函数就是全局函数

全局函数一经定义,任何位置都能使用

2.局部函数

在一个函数体内定义的函数就是局部函数

局部函数只能在定义的函数体内被访问,

出了范围后就无法访问

测试 :

1.声明一个函数 fun1 --内容自定

2.在fun1的内部 ,再声明一个函数 inner

在 inner中向控制台输出一句话 : 内部函数

3.尝试着在最外层调用 inner 函数 - 看结果

4.尝试在fun1 中调用inner函数 -查看结果

#### 3.ECMAScript 提供的全局函数

ES提供的全局函数不用编写,在JS引擎中直接就自带的函数,

直接调用即可

##### 1.parseInt()

##### 2.parseFloat()

##### 3.isNaN()

##### 4.isFinite() : 判断指定数据是否有有限值

##### 5.encodeURI() :

对统一资源标识符(URI)进行编码,并返回编码后的结果

##### 6.decodeURI()

对已编码的一资源标识符(URI)进行解码,返回解码后的形式

##### 7.encodeURIComponent()

对统一资源标识符的有效组件进行编码,并返回编码后的结果

实际上encodeURIComponent() 允许将 :// 也一同进行编码

##### 8. decodeURIComponent ()

对已编码统一资源标识符的有效组件进行解码,

并返回解码后的结果

##### 9.eval()

执行 以字符串方式传递进来的js代码

#### 4.递归 - 算法

1.问题

1.用一个函数实现 5 !

5! = 5\*4\*3\*2\*1;

5! = 5\*4!

4! = 4\*3!

3! = 3 \*2!

2! = 2\*1

2.求指定数字的阶乘

n! = n \* (n-1)!

(n-1)! = (n-1) \*(n-2)!

##### 1.什么是递归

在一个函数中,**直接**或**间接**的**再调用自己**的过程就是**递归**

##### 2递归的实现

###### 1.边界条件

决定递归何时向上返回

###### 2.递归前进

只要没满足递归的边界条件,就一直向下调用自己

###### 3.递归返回

当满足边界条件时,向上返回

(将所有计算出来的值,依次返回给调用者)

### 4.匿名函数 :

是一段没有名称只有功能的函数,

只为某一个操作而存在,不能被其他函数调用

## 7.JavaScript 逻辑结构

### 流程控制结构

1.顺序结构

2.分支结构 (选择结构)

根据条件选择一段代码去执行

3.循环结构

根据条件重复的执行一段代码

### 1.顺序结构

### 2.分支结构

#### 1.if

根据条件选择某块代码去执行

语法 :

if(条件){

满足条件要执行的语句

}

注意 :

1. if后的条件尽量是boolean类型,

如果不是布尔类型的话,会自动转换

以下数值,会自动转换为false

if(0){} if(0.0){} if(NaN){}

if(""){} if(null){} if(undefined){}

2.if后的{}是可以省略的,如果省略的话,

只执行第一条语句

#### 2.if...else...结构

语法:

if(条件){

满足条件要执行的语句

}else{

不满足条件时要执行的内容

}

#### 3.else.....if....结构(多重if结构)

语法:

if(条件1){

满足条件1要执行的语句块1

}else if(条件2){

不满足条件1但满足条件2时要执行的内容

}... ...else{不满足上述所有条件时,要执行的内容}

#### 4.switch...case

##### 1.使用场合

等值判断

##### 2.语法结构

switch(变量){

case 常量1:

语句块1;

break;

case 常量2;

语句块2;

break;

......

default:

语句块n;

}

注意:

switch 后的变量与各个case后的常量是采用"==="判断时候相等的,所以数值以及数据类型完全相等时,才能匹配成功

拓展用法

switch(变量){

case 1:

case 2:

case 3:

console.log("...........");

break;

.......

}

### 3.循环结构

#### 1.循环的特点(要素)

1.循环条件 - 循环的次数

2.循环操作 - 反复执行的相同操作是什么

#### 2.while

语法:

while(条件){

循环操作

}

注意:

1.在循环操作中,要更新循环条件,否则就是死循环

**Math.random()**

**生成并返回一个0~1之间的小数.**

#### 3.循环的流程控制语句

1.break

作用 : 提前结束整个循环

2.continue

作用 : 终止本次循环,继续执行下次循环

#### 4.do...while()

1.语法:

do{

循环操作

}while(条件);

2.do while 与 while 的区别

1.do while

先执行循环体,再判断条件

至少执行一次循环体

2.while

先判断,再执行循环体,(可能一次都不执行)

#### 5.for

##### 1.语法

for(表达式1;表达式2;表达式3){

循环操作;

}

表达式1: 循环条件的声明

表达式2: 循环条件的判断

表达式3: 更新循环条件

2.**注意:**

在确定循环次数的情况下,优先使用for

不确定循环次数的情况下,优先使用while/do..while

##### 2.for循环三个表达式特殊用法

###### 1.三个表达式都可以被省略

for(;;){}

1.省略第一个表达式

缺失变量,可以省略,但尽量在循环外部将声明补齐

2.省略第二个表达式

死循环

3.省略第三个表达式

死循环

###### 2.表达式1 和 表达式3 的多样化

表达式1 和 表达式3 的位置处 , 分别可放置多个表达式,中间用 " , " 隔开即可

#### 6.循环的嵌套

在一个循环体内,再出现一个循环

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

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

}

}

注意:

外层循环走一次,内层循环走一轮

## 8.JavaScript 数组

### 程序 = 算法 + 数据结构

### 1.什么是数组

数组,就是数据的组合,即一个变量中允许存放多个值

数组中的元素,是按照线性顺序来排列的

也是典型的"栈"式结构

线性结构特点:

除第一个元素外,每个元素都有一个"前驱"元素

除最后一个元素外,每个元素都有一个"后继"元素

通过元素所在位置的"下标"来访问元素**(从0开始 , 到 长度-1 结束)**

### 2.声明数组(一维)

#### 1.声明一个空数组变量

var 数组名 = [];

#### 2.声明数组并赋值(初始化)

var 数组名 = [数据1,数据2,数据3,....]

#### 3.声明一个空数组变量

var 数组名 = new Array();

#### 4.声明数组并赋值(初始化)

var 数组名 = new Array(数据1,数据2,数据3);

### 3.数组是引用类型的对象

数据是存放在"堆"中的

在"栈"中会对"堆"中数据的地址进行引用

1.将数组赋值给变量的时候,实际上赋值的是地址

2.数组做为参数的时候,也是按值传递的,

但传递的是数组的地址的副本.

3.null

表示 不再指向任何地址 即 释放对象的空间

null 和 undefined 的区别

1.相同点 :

都是原始类型(基本数据类型),数据都保证在"栈"中

2.不同点 :

1.undefined

表示变量声明过但为赋值,

一般为未赋值变量的默认值,

从不主动使用

2.null

表示变量不再指向任何地址.

一般用于主动释放指向对象的引用

(手动释放空间)

### 4.数组的使用

#### 1.访问数组元素

使用的是 下标(即索引)

var arr = ["张无忌","赵敏"];

arr[5] = "金毛狮王";

console.log(arr[3]);//undefined

总共6个元素

#### 2.length属性

作用 : 获取数组的长度(元素的个数)

1.允许将 数组的length 属性设置为0,

来完成数组元素的清空操作

2.配合循环做 循环遍历 数组中的每一个元素

3.配合着数组名称,找到最新元素要插入的位置

#### 3.使用 for...in 遍历数组元素

允许使用 **for...in** 遍历数组中每个元素的下标

语法 :

**for(var 变量 in 数组){**

//变量:表示的是数组中的每个元素的索引

**}**

#### 4.关联数组

用字符串做下标

var array = []; array["xiyouji"] = "孙悟空";

关联数组的内容, 只能通过 字符串 做下标来取值

允许通过 for..in 来循环遍历关联数组

#### 5.练习

##### 冒泡排序:

对一组数字进行排序,原理: 两两数字相比较,小的往前排

特点:

1.外层循环 : 控制比较的轮数,总共比较了 length -1 轮

2.内层循环 : 控制每轮中比较的次数,

随着轮数的增加,次数反而减少

比较两个数的大小,小的靠前,大的靠后

#### 6.数组的常用API方法

**API : Application Programming Interface**

应用程序 编程 接口

##### 1.字符串拼接

###### 1.toString()

把数组转换为字符串,并返回结果

var arr = [2,10,8];

var str = arr.toSpring();

###### 2.join(分隔符)

将数组的元素,用指定的 分隔符 连接到一起

##### 2.连接数组 ( concat() )

函数 : **concat()**

作用 : 拼接两个或更多的数组,并返回拼接后的结果

语法 : **arr1.concat(arr2,arr3,.....)**

注意 :

1.该函数不改变现有数组,仅仅是返回拼接后的数组

2.每个参数可以是具体的数值,也可以是数组

##### 3.获取子数组( slice() )

从某数组中,截取几个连续的元素组成新的数组

函数 : slice()

语法 : **arr.slice(start , [end])** // end 可以省略

start : 从哪个下标出开始选取 ,

**取值为正,从前向后算,**

**取值为负,从后往前算**

0 1 2 3

var arr = ["中国" , "美国" , "日本" , "英国"]

-4 -3 -2 -1

end : 指定结束处的下标(不包含);可以省略,

如果省略的话,相当于从start一直截取到结束

注意 :

1.该函数不会改变现有数组,会**返回**全新的子数组

##### 4.修改数组( splice() )

允许从数组中删除一部分元素,同时在添加另一部分元素

函数 : **splice()**

语法 : arr.splice(start,count,e1,e2,....)

**start** : 指定添加,或删除元素的起始位置

**count** : 要删除的元素个数,如果取值为0,则表示不删除

**e1,e2,e3**... : 在start位置处要添加的新内容

**返回值** : 被删除元素的数组

注意 :

splice 会改变现有的数组

##### 5.数组反转( reverse() )

颠倒数组中元素的顺序

语法 : **array.reverse()**;

注意 :

该函数会改变当前数组的内容

练习 : 倒叙输出一个数组

##### 6.数组排序( sort() ) (重点)

函数 : **sort**

语法 : **array.sort();**

作用 :

默认情况下,按照元素的Unicode码的大小按升序排序

注意 :

直接改变现有的数组内容

**特殊** :

允许自己指定排序函数,从而实现对数字的升序 或 降序

**array.sort(排序函数);**

//排序函数

**function sortAsc(a,b){ return a - b; }**

**array.sort(sortAsc);**

**原理** :

1.指定 排序函数 sortAsc,

定义a 和 b 两个参数,

数组会自动传递进去,如果返回值是 >0 的数 ,

则交换两个数字的位置,否则不变

2.将排序函数指定给sort , 数组会自动的两两比较

#### 7.数组的进出栈

JS是按照标准的"栈式操作"访问数组的

讲究的是 "入栈" 和 "出栈"

所有的"栈式操作"的特点是"后进先出"

##### 1.出入栈函数

###### 1.push()

入栈,在栈顶(数组的尾部)添加指定的元素,并**返回**新的长度

var arr = [10,20];

//向栈顶增加新数据30(将30保存在数组的尾部)

var length = arr.push(30);

//length 保存的是 30 入栈后 数组的长度

###### 2.pop()

出栈,**删除**并**返回**栈顶(数组尾部)的元素

var arr = [10,20,30];

var r1 = arr.pop();

var r2 = arr.pop();

###### 3.shift()

作用 : 删除数组头部的元素并**返回**

**4.uns**h**ift()**

作用: 在数组的头部(第一个元素)增加新元素

#### 8.二维数组

##### 1.什么是二维数组

一个数组中的元素又是一个数组,即数组中的数组

var arr = [ [10,20] , [30,40] , [50,60] ]

##### 2.访问二维数组

###### 1.访问数组中的单个元素

arr[0][1] : 20

arr[2][1] : 60

###### 2.循环遍历二维数组 - 嵌套的循环结构

### 垃圾回收

#### 垃圾

不再被任何变量使用的对象

#### 垃圾回收

程序会自动释放不再被任何变量使用的垃圾对象的内存空间

#### 为什么

内存中的空间都是有限的，内存占用越多，程序运行越慢

#### 垃圾回收器

js引擎中，专门监控，并释放垃圾的小程序

#### 如何

垃圾回收器伴随主程序在后台并行执行

垃圾回收器会记录每个对象被几个变量使用着

只要发现一个对象，不再被任何变量使用，就释放该对象所占内存

#### 建议:

只要一个变量不再使用，都要主动赋值为null

## 9.JavaScript DOM操作

DOM : Document Object Model 文档对象模型

让JS 有能力操作页面的元素

使用 JS 动态操作页面

### 1.使用JS获取页面上的某个元素

1.元素要增加ID属性

2.通过ID获取页面指定元素

**var elem =** **document.getElementById("元素ID");**

### 2.修改 标记内的 文本内容

<input type="text" id="input1">

<div id="d1">HELLO WORLD</div>

**属性**: **innerHTML**

**作用 : 获取** 或 **设置** 指定**元素标记**内的**HTML内容**

**//1.获取页面元素**

**var d1 = document.getElementById("d1");**

**//2.获取 d1 中的HTML内容**

**var str = d1.innerHTML ;**

**//3.设置 d1 中的HTML 内容**

**d1.innerHTML = "更新的内容";**

### 3.获取 或 设置 表单控件的数据

**表单控件** 主要通过 **value** 属性 来**设置** 或 **获取**数据

**<input type="text" id="uname">**

**//1.获取uname的值**

**var** **uname = document.getElementById("uname ");**

**//2.获取 uname 的值**

**var v =** **uname.value;**

**//3.设置 uname 的值**

**uname.value = "通过DOM操作input";**

### 4.文本框 和 密码框 - onblur

onblur : 失去焦点

<input onblur="函数名()">

# AJAX

## 1.页面的访问方式

### 1.同步访问

在访问服务器时,只能等待服务器的响应,不能做其他事情

相当于,在同一个时间只能干一件事

代表场合 :

1. 输入网页访问网址

2. a标记的默认跳转

3. submit 按钮的表单提交

### 2.异步访问

在向服务器发送请求时,不耽误用户在网页上做其他操作.

相当于,在同一个时间上,用户能做多个事情

代表场合 :

1. 用户名的重复性验证

2. 聊天室

3. 股票走势图

4. 搜索建议

## 2.什么是AJAX

Asynchronous JavaScript And Xml

异步的 JS 和 Xml

本质, 使用JS中的 **XMLHttpRequest(XHR)** **对象** 异步的

----向服务器发送请求

服务器响应回来的是 **部分数据** 而不是完整的页面,

并且可以以 **无刷新** 的效果来更改页面中的局部内容

## 3.获取 AJAX 对象 - XMLHttpRequest

在主流浏览器中 : new **XMLHttpRequest()**;

注意 : 在IE8 以下的浏览器中 不支持 XMLHttpRequest

在 IE8以下浏览器中 :

new ActiveXObject("Microsoft.XMLHttp");

测试 :

console.log(window.XMLHttpRequest);

如果浏览器**不支持**XMLHttpRequest,以上会打印 **null**

如何创建AJAX对象

**var xhr;**

**if(window.XMLHttpRequest){**

**xhr = new XMLHttpRequest();**

**}else{**

**xhr = new ActiveXObject("Microsoft.XMLHttp")**

**}**

**封装 :**

**function getXhr(){**

**var xhr;**

**if(window.XMLHttpRequest){**

**xhr = new XMLHttpRequest();**

**}else{**

**xhr = new ActiveXObject("Microsoft.XMLHttp")**

**}**

**return xhr;**

**}**

## 4.XHR的常用 属性 和 方法

### 1.open() - 方法

作用 : **创建请求**

语法 : **open(method , url , isAsyn)**

1.**method**

请求方式 : POST 或 GET

2.**url**

请求地址

3.**isAsyn**

使用 **同步(false)** 还是 **异步(true)** 的方式发送请求

练习 :

1.在页面中创建一个按钮,点击时,实现以下操作

1.创建 / 获取 xhr 对象

2.创建一个请求 , 采用异步的方式,get 的提交方法,

向respone.php 发送一个请求

### 2.readyState - 属性

作用 : 表示 Xhr 对象 的请求**状态**

**值 :**

**0 : 请求消息还未发送**

**1 : 已经打开到 web 服务器的连接**

**2 : xhr对象已经接收响应消息的头部**

**3 : xhr对象已经接收响应消息的主体**

**4 : xhr已经接收完成响应消息**

**注意 :**

**当readyState 为 4 的时候 ,表示所有的响应消息都接收完毕**

### 3.status - 属性

作用 : 表示 **服务器** 响应状态码

值 :

1 . **200**

**正常接收请求以及给出响应**

2. **404**

要请求的资源不存在

3. **500**

服务器内部错误

**注意** :

当**status** 的值是 **200** 的时候 ,

表示**服务器**已经**正确**的给出所有的**响应**

### 4.onreadystatechange - 事件 / 属性

当**xhr** 的 **readyState** 发生**改变时**,要自动激发的操作

用于 **监听** **readyState** **状态改变**的

**xhr.onreadystatechange = function(){**

**//每当readyState状态码发生改变的时候 要执行的操作**

**//注意: 判断条件不能换位,**

**//因为readyState 为 4时, status 才能为200**

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

**//当xhr 已经完全接收响应了,**

**//并且服务器也正常给出响应的时候要做的事**

**var txt = xhr.responseText; // 服务器端响应回来的数据**

**}**

**}**

### 5.send() - 方法

作用 : **发送请求**

语法 : **xhr.send(body)**

**body** : 发送的请求数据. 如果是**get**提交方式,此处为**null**

如果是**post** 提交方式 ,此处就是提交的**数据**

## 5.发送异步请求的步骤

### 1.获取 AJAX 对象 : xhr

### 2.创建请求 : xhr.open();

### 3.设置onreadystatuschange

判断 **readyState** 以及 **status** 并接收响应

### 4.发送请求 : xhr.send();

### 5.汇总

ex:

**/\*\***

**\* 验证登录信息**

**\*/**

**function checkLogin(){**

**//1、获取用户名 和 密码**

**var uname = $("uname").value;**

**var upwd = $("upwd").value;**

**//2、拼请求地址url(带请求参数的)**

**var url="00-homework.php?uname="+uname+"&upwd="+upwd;**

**console.log(url);**

**//3、实现AJAX请求**

**//3.1 获取 xhr**

**var xhr = getXhr();**

**//3.2 创建请求**

**xhr.open("get",url,true);**

**//3.3 设置回调函数**

**xhr.onreadystatechange=function(){**

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

**console.log("已经成功!!!");**

**var resText = xhr.responseText;**

**//将 resText 放在 msg-show中**

**$("msg-show").innerHTML=resText;**

**}**

**}**

**//3.4 发送请求**

**xhr.send(null);**

**}**

## 6使用GET方式 , 发送请求数据

在请求地址后, 拼接请求参数

... ...

**xhr.open("get","response.php?uname=zhang",true)**;

... ...

## 7.使用AJAX发送POST请求

### 1.在请求地址后不能拼请求参数

### 2.请求参数要放在send()方法中

**xhr.send("key=value&key=value")**

**ex: xhr.send("uname=zs&upwd=123'')**

### 3.在发送请求之前,需要将请求消息头改掉

**xhr.setRequestHeader("Content-Type",**

**"application/x-www-form-urlencoded")**

**Content-Type** ; 告诉服务器数据如何编码

等同于form表单中的encType

在AJAX中 , **Content-Typ**会 被修改为 text/plain,

所以数据是无法提交的.

必须手动修改为**application/x-www-form-urlencoded**

## 8.AJAX的数据传输格式

AJAX

异步的JavaScript 和 XML

XML 现在被新的数据格式所替代 JSON

### 1.什么是JSON

JavaScript Object Notation

JavaScript 对象 表示法

JSON 是仿照JS对象的格式来编写出来的一个字符串

#### 1.JavaScript Object

JS对象 , 主要由 **属性** 和 **方法** 来组成

##### 1.编写JS对象

###### 1.单个对象

由{} 来表示一个对象

**var person ={**

**属性名称1 : 属性值1,**

**属性名称2 : 属性值2,**

**属性名称3 : 属性值3**

**};**

###### 2.访问对象数据 - 访问属性

**对象名.属性**

###### 3.使用数组保存对象 - 多个对象

**var array = [**

**{对象1的属性和值} ,**

**{对象2的属性和值},**

**{对象3的属性和值}**

**];**

#### 2.JSON 字符串

JSON的规范

1.对象的属性名,必须使用**" "**引起来

2.JSON 的字符串 , 用**' '**引起来

1.JSON 对象的字符串表示

**var person ='** **{**

**"属性名称1 " : "属性值1 ",**

**"属性名称2" : "属性值2",**

**"属性名称3" : "属性值3"**

**}' ;**

2.JSON 数组的字符串表示

**var array ='** **[**

**{"属性名称1 " : "属性值1 "},**

**{"属性名称2" : "属性值2"},**

**{"属性名称3" : "属性值3"}**

**]' ;**

#### 3.将JSON 字符串 转换成 JS对象

**var p = '{"name":"张三丰","gender":"男","age":86 }';**

1.使用eval()函数

**var obj** = **eval( "(" + p + ")" );**

2.使用 **JSON.parse()** 来解析JSON 字符串得到JS对象

**var obj** = **JSON.parse(p)**

3.使用 **JSON.stringify()** **将**JS**对象 转化** 为JSON格式**字符串**

**var obj = {name:"张三丰",gender:"男",age:86 }**

**var str = JSON.stringify(obj);**

#### 4.在php中, 可以直接将数组(一维,二维,关联)直接转换成JSON格式字符串

通过 **json\_encode()** 将数组转换为字符串

# XML

## 1.什么是XML

eXtensible Markup Language

可扩展的 标记 语言

XML的标记没有被预定义,需要自行定义的

XML的宗旨是做数据传递的 , 而非数据展示

## 2.XML的语法

XML可以独立保存成 \*\*\*.xml 文件也可以以字符串的形式出现

### 1.XML的最顶端是 XML 的声明

**<?xml ?>**

属性 :

1. **version** : 指定XML的版本, 需要将其值设定为 **1.0**

2.**encoding** : 指定当前文档的编码,通常指定为 **utf-8**

### 2.XML的标记语法

1.XML都是由标记组成(与HTML相同),但所有元素都必须有显示的关闭标记

**<person>张三丰</person>** 正确

2.XML 中的标记是严格区分大小写, 开始和结束的标记必须一致

**<Person>张三丰</Person>(一般全用小写)**

3.标记是允许嵌套的,但必须是正确嵌套

<person>**<name>张三丰</name><age>100</age>**</person>

4.每个XML标记允许有若干自定义属性,格式等同于HTML,

但属性必须有引号

**<Person id="1001" pid="10">张三丰</Person>**

5.每个XML文档中有且只有一个根元素

### 3.使用AJAX请求XML文档

#### 1.要遵循AJAX 的请求步骤

1.创建 / 获取 xhr

2.创建请求

3.设置回调函数

4.发送请求

#### 2.创建请求时,需要将请求地址修改为 xml 文件的地址即可

xhr.open("get","student.xml",true);

#### 3.在回调函数中,处理响应数据

使用 xhr.responseXML 来获取响应数据,返回的是XML文档对象

#### 4.解析XML内容

1.核心 : **getElementsByTagName("tagName");**

返回值 :返回一个包含指定元素们的**"数组"**

2.获取某元素中的HTML文本

elem.innerHTML 属性即可

#### 5.在PHP中返回 XML 格式的字符串

1.按照XML的语法结构,拼字符串即可,然后再响应给客户端

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

**$xml .="<person></person>"**

2.必须增加响应消息头

**header("Content-Type:application/xml")**

# CSS

## 1.了解CSS

### 1.什么是CSS

Cascading Style Sheets 层叠样式表 或 级联样式表

或 简称 为 样式表

### 2.问题

1.网页中所有的文本都变成红色

2.网页中所有div的文本都变为蓝色

(**过时**) 标记 :font

属性 :color

3.网页中所有div的文本都变为绿色

HTML的缺点 :

1.不同标记中实现相同的效果,使用的是不同的属性

2.没有有效的提升代码的可重用性

### 3.CSS 与 HTML

1.HTML : 构建网页结构

2.CSS : 构建网页样式

**W3C 建议尽量使用CSS样式来取代 HTML 属性**

## 2.CSS语法(重点)

### 1.CSS样式表的使用方式(重点)

#### 1.内联方式

又称为 "行内样式"

将样式内容定义在HTML元素的style 属性中

语法 : (ANY指代任意一个标签)

<ANY style="**样式声明**1;样式声明2;.....样式声明n;">

##### 样式声明:

1.由**样式名称** 和 **样式属性值** 共同组成

2.样式名称(属性) 和 样式值 用 : 连接

3.多个样式声明之间用 ; 分隔

##### 常用样式属性 和 值 :

|  |  |
| --- | --- |
| 属性 | 值 |
| color | 合法的颜色值 |
| background-color | 合法的颜色值 |
| font-size | px 或 pt为单位的数值 |

#### 2.内部样式表

##### 1.什么是内部样式表

将"样式规则" 声明在HTML文档<head>中的<style>中

<head>

<style>

样式规则1

样式规则2

....

</style>

</head>

##### 2.样式规则的语法

样式规则由 选择器 和 若干样式声明 来主成

1. **选择器**

作用 : 规范了页面中哪些元素能够使用声明好的样式

元素选择器:由元素名称作为选择器,

表示要匹配页面中指定的所有元素

2.**若干的样式声明**

位于 选择器之后 ,使用{}括起来,

里面可以包含若干的样式声明

#### 3.外部样式表

以独立样式表文件(\*.css)的方式存在

样式表文件中,只能包含"样式规则"

使用外部样式表步骤

1.创建样式表文件

2.在要使用的网页上,引入样式表文件

<head>

<link rel="stylesheet" href="url">

</head>

### 2.CSS样式表的特征

#### 1.继承性 :

具备父子关系,子元素 可以 继承父元素中的一部分效果

<div style="color:red"><span>文本是红色的</span></div>

#### 2.层叠性

允许为一个元素定义多个样式规则

样式不冲突时,可以合并成一个

#### 3.优先级

为元素定义多个规则时,如果样式声明冲突,

则按照各自的优先接来应用样式

优先级,由低到高

1.浏览器缺省设置

2.外部 或 内部样式表

就近原则 : 以后定义这为准

3.内联方式

#### 4. !important 规则

**!important** 用于显示调整优先级

选择器{

属性 : **值 !important;**

}

#### 5.错误提示

##### 1.Unknown property name

属性名 写错了!!!

##### 2.invalid property value

属性值 写错了!!!

### 3.CSS基础选择器(重难点)

#### 1.通用选择器

语法 : \*{}

作用 : 匹配页面中所有元素

ex :

\*{ font-size:12px }

(尽量少用)

#### 2.元素选择器

别名 : 标签选择器, 标签样式 ...

作用 : 匹配页面中指定元素的所有标记

ex :

body{ font-size : 12px; }

#### 3.类选择器

作用 : 允许任意元素通过 class 属性 来引用的选择器

语法 :

##### 1.定义: .类名{ ... }

强调

1. **.**  在声明时不能省略

2. 类名:字母,数字,下划线(\_),连字符(-),数字不能开头

##### 2.引用:

<ANY **class**="类名">

###### 多类选择器的引用方式

作用 : 让一个元素同时引用多个类选择器

语法 : class 属性值 是一个用 **空格** **隔开**的选择器列表

###### 分类选择器的"定义"方式

将**类选择器** 和 **元素选择器**结合起来使用,

从而实现对某个元素的**不同样式**的**细分控制**

语法 : **元素选择器.类选择器**{...}

ex :

div.redColor{ ... }

匹配 : class 为redColor 的div 元素

#### 4.ID选择器

作用 : 专属定制,只匹配页面上指定ID值的元素

语法 : **#ID值**{....}

ex:

#container{ font-size:16px; }

#### 5.群组选择器

特点 : 选择器的声明是一个 以 **,** 隔开的选择器列表

作用 : 将多个 样式效果**相同**的选择器,放在一起定义

语法 : **选择器1,选择器2,....,选择器n**{.....}

#### 6.后代选择器

**后代** : 不限制层级的子/孙子 元素们都是后代

语法 : 选择器1 选择器2{....} (用空格隔开)

1.匹配id为container中所有的div元素

#container div{...}

#### 7.子代选择器

**子代** : 只有一级层级关系的子元素是子代

语法 : 选择器1**>**选择器2{...}

1.匹配id为container中下一级的div元素

#container>div{

}

#### 8.伪类选择器

##### 1.作用

匹配 某元素 的不同状态的选择器

##### 2.分类

1.链接伪类

2.动态伪类

3.目标伪类

4.元素状态伪类

5.结构伪类

6.否定伪类

##### 3.语法:

**: 伪类**{....}

**选择器 : 伪类**{...}(多用这种)

链接伪类:

1. **:link** , 匹配 未被访问的超链接的状态

2. **:visited** , 匹配被访问过的超链接的状态

动态伪类:

1. **:hover**,匹配鼠标悬停在元素上时的状态

2. **:active**,匹配元素被激活时的状态

3. **:focus**, 匹配元素获取焦点是的状态

练习 :

1.页面中创建一个 超链接,点击时链接到www.baidu.com

2.默认效果下,a标记显示为 颜色为 黑色,不显示下划线

text-decoration:none;/\*不显示下划线\*/

3.鼠标悬停时, a标记显示为红色,显示下划线

text-decoration:underline;

#### 9.选择器的优先级

|  |  |
| --- | --- |
| **选择器类型** | **权值** |
| 元素选择器 | 0,0,0,1 |
| 类选择器 | 0,0,1,0 |
| 伪类选择器 | 0,0,1,0 |
| ID选择器 | 0,1,0,0 |
| 内联样式 | 1,0,0,0 (只要有内联,就以他为准) |

**(自己有的,用自己的,自己没有的,用继承的) 有自定义的,就会优先于继承的**

## 3.尺寸

### 1.单位

#FF0000 -> #f00

#aabbcc -> #abc

color:#DD5044

### 2.尺寸属性

1.作用:

用户设置元素的尺寸(宽度 和 高度)

单位 : 通常为 % 或 px

2.尺寸的属性 :

1.宽度

width

max-width

min-width

2.高度

height

max-height

min-height

测试:

1.创建一个div,使用 开发者工具 观察div的尺寸(宽度,高度)

2.为div设置 宽度500px,高度300px,

再使用 开发者工具 观察div的尺寸

### 3.页面中允许设置尺寸的元素

1.所有**块级元素** 都能设置尺寸属性

2.所有的**行内块元素** 也能设置尺寸属性

表单控件

3.html元素本身就具备 width 和 height 属性的元素,

是可以设置的

table,img

除 **img** 以外的 **行内元素** ,不能设置尺寸

### 4.溢出处理

使用尺寸属性限制元素大小时,如果内容所需的空间大于元素本身的话,会导致内容**溢出**

#### 溢出处理属性:

##### 1.属性

overflow : 横向 和 纵向 都能溢出处理

overflow-x : 横向溢出处理

overflow-y : 纵向溢出处理

##### 2.取值

1.visible

可见的,默认值

2.hidden

隐藏

3.scroll

滚动

4.auto

自动,**溢出时**才**显示**滚动条,(不溢出不显示)

## 4.边框属性

### 1.边框

#### 1.简写属性 - 四个方向边框特征

border : width style color;

width : 宽度

style : 样式

solid,dotted,dashed

color : 颜色

合法的颜色值

transparent : 透明

注意 : border:none; 或 border:0; //无边框??

#### 2.单边定义 - 某个边的边框特征

border - 方向 : width style color

方向 : top / right / bottom / left

#### 3.单属性定义 - 设置四个边框的某一具体属性

border - 属性 : 值

属性 : width / style / color

#### 4.单边单属性定义 - 某一边的某一具体属性

border - 方向 - 属性 : 值

方向 : top / right / bottom / left

属性 : width / style / color

### 2.边框倒角

将元素的四个角 , 由指教倒换成圆角

属性 : border-radius

取值 : 1个数值(最多4个),表示4个角的倒角圆的半径,

可以为 具体数值 或百分比

单角定义 :

border-top-left-radius : 左上角倒角半径

border-top-right-radius : 右上角倒角半径

border-bottom-left-radius : 左下角倒角半径

border-bottom-right-radius : 右下角倒角半径

### 3.边框阴影

属性 : box - shadow

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

h-shadow : 阴影的水平偏移距离(必须的)

取值为正,向右偏移

取值为负,向左偏移

v-shadow : 阴影的垂直偏移距离(必须的)

取值为正,向下偏移

取值为负,向上偏移

blur : 模糊距离(px)

spread : 阴影的尺寸(px)

color : 阴影的颜色

insert : 值,将默认的外阴影,改为内阴影

### 4.轮廓

#### 1.什么是轮廓

在元素外围的一条线 , 位于边框之外

#### 2.属性

1.outline : width style color;

outline : none; 或者 outline : 0;

2.单独定义

outline-width : 轮廓的宽度

outline-style : 轮廓的样式

outline-color : 轮廓的颜色

## 5框模型(重难点)

距离产生美

### 1.什么是框模型

一切元素皆为"框"

框模型: (Box Model) , 也称为"盒子模型",

定义了元素框处理元素内容,边框,**内边距** 和 **外边距**的一种方式

当框模型介入到元素中,元素的实际尺寸会发生改变

元素实际宽度 : 左右外边距+左右边框+左右内边距+width

元素实际高度 : 上下外边距+上下边框+上下内边距+height

### 2.外边距

#### 1.什么是外边距

围绕在元素周围的空白区域,位于边框(边缘)之外

#### 2.语法

margin : value; /\* 四个方向外边距值 \*/

margin-top : value; /\* 上外边距 \*/

margin-right : value;

margin-bottom : value;

margin-left : value;

取值:

1.以px为单位的绝对值

2.以 % 为单位的相对值

3.可以取值为 负数

margin-top : 10px 元素向下移动10px,体现上外边距

margin-top : **-10px** 元素向上移动10px

margin-left : 10px 元素向右移动,体现左外边距

margin-left : **-10px** 元素向左移动

4.可以取值为 auto

1.正常情况下,只对左右外边距有效

2.对块级元素有效

为块级元素指定宽度后, 再将左右外边距设置为auto

则该元素谁平居中

简介写法 :

1.margin : value

四个方向外边距

2.margin : v1 v2;

v1 : 上下外边距

v2 : 左右外边距

3.margin : v1 v2 v3;

v1 : 上外边距

v2 : **左右**外边距

v3 : 下外边距

4.margin : v1 v2 v3 v4;

v1 : **上**外边距

v2 : **右**外边距

v3 : **下**外边距

v4 : **左**外边距

#### 3.具备默认外边距的元素

body , h1~h6 , p , dl , dd (左外边距), ul , ol , pre

通过 CSS Reset(CSS 重置) 将具备默认外边距的元素的外边距设置为0

body,h1,h2,h3,h4,h5,h6,dl,dd,p,ul,ol,pre{

margin:0;

}

#### 4.特殊效果

##### 1.外边距的合并

当两个垂直外边距相遇时 , 他们将合并成一个,

最终的距离值 , 以大值为准

##### 2.上外边距的溢出

在某些特殊的条件下 , 在为子元素设置上外边距时,

有可能会作用在父元素上

特殊条件 :

1.父元素不能有上边框

2.为第一个子元素设置上外边距时

解决方案 :

1.为父元素增加上边框

会影响父元素的高度

2.使用父元素的上内边距来取代子元素的上外边距

也会影响父元素的高度

**3.在子元素上,增加一对 空 <table>元素即可(不需要tr,td)**

<div id="d2">

<table></table>

<div></div>

</div>

4. **#d1:before{**

**content:"";**

**display:table;**

**}**

3.为行内元素 , 行内块元素 设置垂直外边距时的特殊效果

1.**行内元素(除<img>外)** , 垂直外边距无效

2.**行内块元素** ,设置垂直外边距时,整行的内容都跟着发生改变

### 3.内边距

#### 1.什么是内边距

内容区域和边框(边缘之间的空间)

内边距会扩大元素边框的占用区域

#### 2.语法:

属性 :

padding : value;

padding-top:value;

padding-right:value;

padding-bottom:value;

padding-left:value;

取值:

1.以px为单位具体数值

2.以 % 为单位相对数值

padding 的简洁写法 : 根 margin 一样

1. padding : value

四个方向内边距

2. padding : v1 v2;

v1 : 上下内边距

v2 : 左右内边距

3. padding : v1 v2 v3;

v1 : 上内边距

v2 : **左右**内边距

v3 : 下内边距

4. padding : v1 v2 v3 v4;

v1 : **上**内边距

v2 : **右**内边距

v3 : **下**内边距

v4 : **左**内边距

### 4.box-sizing 属性

作用 : 指定元素边框内的尺寸计算方式

取值 :

1.container-box

默认值,边框内的尺寸 = 左右边框 + 左右内边距 + width;

2.border - box

边框宽度 以及 内边距的距离 , 都会算在width 内

## 6.背景属性

### 1.背景色

属性 : background-color

取值 : 合法的颜色值

注意 :

背景色是从边框位置处开始绘制的

### 2.背景图片

属性 : background-image

取值 : url();

### 3.背景平铺

属性 : background-repeat

取值 :

1.repeat : 默认值

2.no-repeat : 没有平铺

3.repeat-x : 水平平铺

4.repeat-y : 垂直平铺

### 4.背景图片的尺寸

属性 : background-size

取值 :

1.width height

2.width% height%

3.cover

将背景图等比放大,

直到背景图覆盖到元素的所有区域为止

4.container

将背景图等比放大

直到 右边 或 下边 覆盖到元素为止

### 5.背景图片的固定

属性 : background-attachment

取值 :

1.scroll : 滚动

2.fixed : 固定

推荐 :

1.将背景图设置在body

2.增加背景图片固定

### 6.背景图片的定位

改变背景图片在元素中的位置

属性 : background-position

取值 :

1.x y

x : 背景图片的水平偏移位置

x取值为正 ,图片右偏移

x取值为负 ,图片左偏移

y : 背景图片的垂直偏移位置

y取值为正 ,图片下偏移

y取值为正 ,图片上偏移

2.x% y%

0% 0% : 左上角

100% 100% : 右下角

50% 50% : 显示在正中间的区域

3.关键字

x : left / center / right

y : top / center / bottom

### CSS Sprites

**将若干幅不经常变换的图片封装到一副大图中**

**配合 背景图片 以及 背景图片定位 将 想看的图片显示出来**

1.在页面中创建一个 与 要显示区域 大小一样的元素

2.将 "雪碧图(精灵图)" 作为元素的背景图

3.改变图片在元素中的位置(左上移动)

## 7.渐变

### 1.什么是渐变

多种颜色逐渐变化的一个效果

### 2.渐变的要素

1.填充方向

从上到下,从下到上,从左到右,从右到左,........,

2.色标

包含颜色 及其 出现的位置

每个色标表示的是一种颜色,渐变中,至少包含两种色标

### 3.渐变的分类

1.线性渐变

按照直线方向,填充渐变色

2.径向渐变

按照圆形的方式填充出来的渐变效果

3.重复渐变

将 线性渐变 或 径向渐变 重复的实现多次

### 4.线性渐变

属性 : **background-image**

取值 : **linear-gradient(angle,color-point,color-point,.....,)**;

#### 1.angle

填充方向 或 角度

1.关键字

1. **to top** : 从下向上填充渐变色

对应 : 0deg

2. **to right** : 从左向右

对应 : 90deg

3. **to bottom** : 从上向下

对应 : 180deg

4. **to left** : 从右向左

对应 : 270deg

2.角度

取值 : **0deg ~ 360deg**

#### 2.color-point

(配合填充方向,决定起始位置)

色标 : 颜色 , 出现的位置

1. red 0%

开始的时候 , 颜色为 红色

2. green 50px

### 5.浏览器兼容性

由于 渐变 是 CSS3 新特性 ,

老版本的浏览器(不支持CSS3)就无法正常显示渐变了!

对于不支持的版本 , 可以添加浏览器前缀

Firefox : -moz-

Chrome 和 Safari : -webkit-

Opera : -o-

#container{

background-image:linear-gradient(...);

background-image:**-moz-**linear-gradient(...);

background-image:**-webkit-**linear-gradient(...);

background-image:**-o-**linear-gradient(...);

}

### 6.径向渐变

属性 : background-image

取值 : radial-gradient(size at position,color-point,.....)

size at position :

size : 半径距离

position : 圆心距离

## 8.文本格式化

### 1.字体属性

#### 1.指定字体

属性 : **font-family**

取值 : value1,value2,value3

ex :

font - family : "微软雅黑","宋体","黑体";

font - family : "microsoft yahei","simsun","simhei"

#### 2.文字大小

font-size : value;

#### 3.字体加粗

属性 : **font - weight**

取值 :

1.**normal**

2.**bold** - 粗体

3.value

400 : normal

900 : bold

#### 4.字体样式

属性 : **font - style**

取值 :

1. **normal**

2. **italic** - 斜体

#### 5.小型大写字母

将所有的小写字母变为大写字母,但个头跟小写字母一样

属性 : font - variant

取值 :

1. normal

2. small-caps

#### 6.字体的简写方式

属性 : font

取值 : **style variant weight size family**

注意 : 一定要将 **family**的值写出来 , 否则无效

font : 24px; /\*无效\*/

font : 24px "simhei"; /\*有效\*/

### 2.文本属性

#### 1.文本颜色

属性 : **color**

取值 : 合法颜色值

#### 2.文本排列方式(水平对齐方式)

属性 : **text-align**

取值 : left / center / right / justify /\*justify :两端对齐\*/

#### 3.文字的修饰(线条修饰)

属性 : **text-decoration**

取值 :

1.none

2.**underline** - 下划线

3.**line-through** - 删除线

4.**overline** - 上划线

#### 4.行高

作用 : 一行数据所占的高度是多少

场合 :

1.设置行间距(模拟)

2.让一行文本在指定高度范围内垂直居中显示

属性 : **line-height**

取值 : 以px为单位 或 无单位数值

#d1{

font-size : 18px;

line-height : 2 ; /\*行高是字体大小的2倍\*/

}

#### 5.首行文本缩进

属性 : **text-indent**

取值 : 以**px**为单位的具体缩进距离值

#### 6.文本阴影

属性 : **text-shadow**

取值 : h-shadow v-shadow blur color;

## 9.表格

### 1.表格常用属性

#### 1.边距属性 : padding

#### 2.尺寸属性 : width , height

#### 3.文本格式化属性 : text- , font-

#### 4.边框属性 : border

table{

border : 1px solid #000

} /\*只适用于表格, 不适用于td\*/

td{

border : 1px solid #000

} /\*只适用于td, 不适用于table\*/

#### 5.背景属性 : background

#### 6.垂直方向对齐属性

属性 : **vertical-align**

取值 : **top / middle / bottom**

### 2.表格特有属性

#### 1.边框合并

属性 : **border-collapse**

取值 :

1.**separate**

默认值 , 即 分离 边框模式

2.**collapse**

边框合并

#### 2边框边距

属性 : border-spacing

取值 :

1.取一个值

同时设置 垂直 和 水平边距

2.取两个值

第一个值 : 水平间距

第二个值 : 垂直间距

两个值用 **空格** 隔开

前提 : 必须处于边框分离模式才有效\

(border-collapse : separate;)

#### 3.标题位置

属性 : caption - side

取值 :

1. top

2. bottom

#### 4.显示的规则

作用 : 告诉 浏览器 如何布局一张表,

即 指定表格 行,列 的 算法规则

属性 : **table - layout**

取值 :

1. auto 自动表格布局 , 列的尺寸由内容来决定

2. fixed 固定表格布局 , 列的尺寸由设定的值来决定的

自动表格布局 VS 固定表格布局

1. 自动表格布局

1.单元格的大小会适应内容的大小

2.表格复杂时 , 加载速度较慢(缺点)

3.但表格比较灵活(优点)

4.适用于不确定每列大小时使用

2. 固定表格布局

1.单元格的尺寸是由设定的值为准,与内用无关

2.会加速显示表格 , 效率较高(优点)

3.但表格不够灵活(缺点)

|  |  |
| --- | --- |
| **自动表格布局** | **固定表格布局** |
| 单元格的大小会适应内容的大小 | 单元格的尺寸是由设定的值为准,与内用无关 |
| 表格复杂时 , 加载速度慢(缺点) | 会加速显示表格 , 效率较高(优点) |
| 但表格比较灵活(优点) | 但表格不够灵活(缺点) |
| 适用于不确定每列大小时使用 | --------------------------------------------- |

## 10.浮动

### 1.定位

改变元素在页面上的默认位置

### 2定位的分类

1.普通流定位

2.浮动定位

3.相对定位

4.绝对定位

5.固定定位

### 3.普通流定位

又称为"文档流定位" , 即页面中默认定位方式

普通流定位 的 特点 :

1.所有元素默认都是从其父元素左上角开始出现的

2.页面中所有的块级元素,都是按照从上到下的方式来排列

每个元素独占一行

3.页面中所有的行内元素, 按照 从左到右 的方式来排列

多个元素能在一行内显示,显示不下则换行

流式定位 : 一行内显示不下所有元素 就 换行 显示

问题 : 在默认的定位方式下 ,

如何能让多个块级元素在一行内显示

解决 : 浮动定位

### 4.浮动定位

#### 1.什么是浮动定位 & 浮动的特点

1. 浮动定位的元素 会 "脱离文档流",

那么元素将不再占据页面空间(元素可以在页面中显示)

后续未浮动元素要上前占位

2. 浮动元素会停靠在 其父元素的 **左**边 或 **右**边,

或者其他已浮动的元素的边缘上

3. 浮动元素只能在 当前行 浮动

4. 浮动元素处理的问题 - 让多个块级元素能够在一行内显示

#### 2.语法

属性 : **float**

取值 :

1.**none**

默认值 , 没有任何浮动效果

2.**left**

左浮动 , 让元素停靠在父元素的左边,

或者左侧已有的浮动元素边缘上

3.**right**

右浮动 , 让元素停靠在父元素的右边,

或者右侧已有的浮动元素边缘

#### 3.浮动引发的特殊效果

1.如果一行中显示不下所有的已浮动元素,那么最后一个将换行.

但有**可能被卡住**

2.元素一但浮动起来之后 , 那么宽度将变得以内容为准

(不指定宽度情况下)

3.元素一但浮动起来之后, 那么将变成块级元素

行内元素 : 不能设置尺寸

块级元素 : 允许设置尺寸

将**行内元素**浮动起来后 , **允许**修改尺寸

4.图片,文字,行内元素,行内块元素

采用的是 "环绕"的方式来排列的,

是不能被浮动元素压在底下的

### 5.清除浮动

#### 1.清除浮动所带来的影响

元素一旦浮动起来之后就会对后续的元素带来一定的位置影响,

如果后续元素不想被影响的话,就可以通过

清除浮动所带来的影响 来解决位置问题

**属性 : clear**

取值 :

1.**none**

默认值,即不做任何清除操作

2.**left**

清除当前元素 **前面**的元素 **左浮动**带来的影响

3.**right**

清除当前元素 **前面**的元素 **右浮动**带来的影响

4.**both**

清除当前元素 **前面**的元素 **任意浮动**带来的影响

#### 2.浮动元素为父元素带来的影响

由于浮动元素会脱离文档流,所以不占据页面空间,

那么就会对父元素的高度带来一定影响

解决父元素高度的影响的方案:

1.直接设置父元素高度

弊端 : 必须要知道父元素的高度

2.设置父元素也浮动

弊端 : 不是任何时候父元素都需要浮动,

并且会对后续元素带来位置的影响

3.为父元素设置 **overflow** 属性

取值为 : **hidden** 或 **auto**

弊端 : 如果有需要溢出显示的东西 , 也一同被隐藏了

4.在父元素中,追加一个**空的块级**子元素,并设置其**clear : both**

5. **.clear::after**{ /\***建议写法**\*/

content:"";

display:block;

clear:both;

}

## 11.显示

### 1.显示方式

决定了元素是 "行内元素" 还是 "块级元素" 还是"行内块元素"....

属性 : **display**

取值 :

1. **none**

不显示元素 元素会脱离文档流(主要用于隐藏)

2. **block**

按 块级元素 的方式显示元素

显示,独占一行,允许修改尺寸 以及 外边距

3.**inline**

按 行内元素 的方式显示元素

显示, 多个元素在一行,不允许修改尺寸 以及 垂直外边距

4. **inline-block**

按 行内块元素 的方式显示元素

显示,多个元素在一行,允许修改尺寸

5.**table**

按 表格 的方式显示元素

显示,每个元素独占一行,允许修改尺寸

### 2.显示效果

#### 1. visibility 属性

作用 : 指定元素的可见性

取值 :

1. **visible** , 默认值 , 元素可见

2. **hidden** , 隐藏 , 依然占据页面空间

3. **collapse** , 用在表格元素 ,

删除一行或一列时不影响表格的整体布局

#### 2. opacity 属性

作用 : 透明度

取值 : **0.0**(完全透明) ~ **1.0**(完全不透明,即显示)

#### 3. vertical-align 属性

1.应用在 表格 中

控制表格中的数据的垂直对齐方式

取值 : top/ middle / bottom

2.应用在 图片 和 行内块元素上

控制 图片 或 行内块 元素 两边的文本

相对于他们的 对齐方式

取值:

1.**top**

2.**bottom**

3.**middle**

4.**baseline** - 基线对齐

1.行内块

是在最后一行文本的底部

2.图片

图片的底部向下3px位置处

在项目中 : **img{ vertical-align:bottom }**

#### 4. cursor 属性

鼠标悬停在元素上时,鼠标的显示效果

取值 :

1. **default**

2. **pointer** : 小手(同<a>的效果)

3. **crosshair** : **+**

4. **text** : **I**

5. **wait** : 等待

6. **help** : 帮助

## 12.列表属性

### 1.列表项的标志

属性 : **list-style-type**

取值 :

1. **none** : 不显示标志

### 2.列表项图像

属性 : **list-style-image**

取值 : **url**(图像路径);

### 3.列表项的位置

属性 : **list-style-position**

取值 :

1. **outside**

默认值,将标志放置于 li 的外边,列表的padding 区域内

2. **inside**

将标志放置于 li 的里边 ,属于li的内容区域的一部分

### 4.列表属性

属性 : **list-style**

取值 : **type url() position;**

常用方式 :

**list-style:none;**

## 13.定位属性

### 1.定位属性

#### 1.定位方式属性

属性 : **position**

取值 : **static / relative / absolute / fixed**

注意 : 如果将元素的定位方式指定为 **relative / absolute / fixed** 中任何一种,

则该元素被称为**"已定位元素"**

#### 2.偏移属性

作用 : 为了修改 **"已定位元素"** 的位置

属性 :

1.**top** 以元素的**上**边为基准边,改变元素的偏移距离

2.**right** 以元素的**右**边为基准边,改变元素的偏移距离

3.**bottom** 以元素的**下**边为基准边,改变元素的偏移距离

4.**left** 以元素的**左**边为基准边,改变元素的偏移距离

#### 3.堆叠顺序

作用 : 修改 **"已定位元素"** 们之间的堆叠效果

属性 : **z-index**

取值 : 无单位的数字

### 2.定位方式 - 相对定位( relative )

#### 1.什么是相对定位

元素会相对于**它自己**原来的位置 , 偏移某个距离.

相对定位的元素,偏移完成之后,原来的位置还会被保留,

不会被其他元素占据

#### 2.语法

属性 : **position**

取值 : **relative**

配合着 **偏移属性** 实现位置的改变

#### 3.使用场合

1.元素的位置微调

### 3.定位方式 - 绝对定位(absolute)

#### 1.什么是绝对定位 & 绝对定位特点

1.绝对定位元素会**脱离**文档流 - **不占据**页面**空间**

2.绝对定位的元素会相对于**离他最近**的 ,

**已定位**的**祖先元素**来实现位置的初始化

3.如果元素没有**已定位**的**祖先元素**的话,

那么它的位置就相对于**body**去实现初始化

#### 2.语法

属性 : **position**

取值 : **absolute**

配合着偏移属性 实现位置的改变

#### 3.小窍门

一般情况 ,绝对定位的元素一定会有一个**已定位**的**祖先元素**,

多数都是由**直接父元素**来充当这个角色

#### 4.绝对定位对元素的影响

1.脱离文档流 , 允许压在其它元素的上方

2.绝对定位的元素 会变成**块级**

3.**margin** 可以正常使用 , 但是**margin** 的左右**auto值**会**失效**

4.绝对定位的元素 ,

将 **left , right , bottom , top** 四个值都设置为**0**的情况下 ,

**margin : auto** 会让整个元素变得**绝对居中**

### 4.堆叠顺序

属性 : **z-index**

取值 : 无单位的数值 , 值 越大 ,越 靠上

注意 :

1. 只能调整**"已定位元素"**的堆叠效果

2. 如果 **z-Index 相同**时 , 那么 后来者居上

3. 父子关系是无法调整堆叠顺序,永远都是子压在父上

### 5.固定定位

1.什么是固定定位

将元素的内容固定在页面的某个位置处,

会一直显示在用户的可视化区域内

2.语法 :

**position : fixed;**

配合 偏移属性 实现位置的摆放

3.注意

1.固定定位的元素**永远**都是相对于**body**做位置的初始化

2.固定定位元素会脱离文档流 , 不占页面空间

3.固定定位元素会变为块级元素

# CSS3高级

## 1.复杂选择器

### 1.兄弟选择器

兄弟 : 通过 **同级** 元素来匹配指定元素**(只能向后找?)**

1.相邻兄弟选择器

作用 : 匹配 某元素**后** **紧紧挨着**它的 指定元素

语法 :

**选择器1+选择器2**{ ... }

ex :

1. #d1**+**div{ /\*匹配出来的 是 #d2\*/ }

2.通用兄弟选择器

作用 : 匹配 某元素**后** **所有** 的指定兄弟元素

语法 :

**选择器1~选择器2**{...}

ex:

#d1**~**div{ /\*匹配 #d2 #d3\*/ }

<div id="d1"></div>

<div id="d2"></div>

<div id="d3"></div>

### 2.属性选择器

允许使用元素所附带的 **属性** 及其 **值** , 来匹配页面元素

<a href="" title="" id="" class="" target="\_blank"></a>

语法 :

核心 : **[ ]**

#### 1.[attr]

**attr** : 表示 任意的属性名称

匹配 附带 **attr** 属性的元素

ex :

**[id]** : 匹配有 **id** 属性的元素

**[title]** : 匹配有 **title** 属性的元素

#### 2.elem[attr]

**elem** : 表示页面中**任意**元素

ex :

1.匹配页面中 ,附带class属性的p元素

**p[class]**{ ... }

2.匹配页面中, 附带class属性 和 id 属性的div元素

**div[class][id]**{ ... }

3. **.container[title]**

#### 3.[attr=value]

**value** : 可以表示任意的属性值

1. **div[id=container]** <==> #container

2.匹配页面中所有的文本框,边框设置为 1px solid #000

**input[type=text]**{ border : 1px solid #000 }

**input[type='text']**{ border : 1px solid #000 }

**input[type="text"]**{ border : 1px solid #000 }

#### 4.[class~=value](非重点)

<p class="c1 c2 c3 c3"></p>

匹配class的属性值 , 是一个值列表 ,

并且value 是该列表中的一个独立选择器的元素

ex :

1.匹配**class**列表中**包含c3选择器**的p元素

**p[class~=c3]**

#### 5.[attr^=value]

匹配 **attr** **属性值** 是以**value 字符**作为**开始**的元素

<div class="col-md-1"></div>

<div class="col-sm-1"></div>

<div class="col-xs-1"></div>

<div class="col-lg-1"></div>

<div class="col-md-2"></div>

<div class="col-sm-2"></div>

<div class="col-xs-2"></div>

<div class="col-lg-2"></div>

ex :

1.匹配所有**class属性值** 是以 **"col"** 作为**开始**的元素

**[class^="col"]**

#### 6.[attr$=value]

匹配 **attr** **属性值** 是以**value 字符**作为**结尾**的元素

ex :

1.匹配所有**class属性值**以**"-1"**作为**结尾**的元素

**[class$="-1"]**

#### 7.[attr\*=value]

匹配 **attr** **属性值**中 **包含** **value 字符** 的元素

ex :

1.匹配所有**class属性值**中**包含** **"md"** 的元素

**[class\*="md"]**

### 3.伪类选择器

#### 1.目标伪类

作用 : 突出显示活动的HTML锚元素 ,

用于匹配当前活动的目标元素

语法 :

**: target** { ... }

#### 2.结构伪类

##### 1. :first-child

匹配属于**其父元素**中的**首个**子元素

**div:first-child**{ /\* 匹配 两个"c1" 和 第一个"d1" \*/}

<body>

<div id="d1">

<div class="c1"></div>

<div class="c2"></div>

<div class="c3"></div>

</div>

<div id="d1">

<div class="c1"></div>

<div class="c2"></div>

<div class="c3"></div>

</div>

</body>

##### 2. :last-child

匹配属于**其父元素**中的**最后**一个子元素

**div:last-child**{ .... }

##### 3. :nth-child(n)

匹配属于**其父元素**中的**第n**个子元素

**div:nth-child(2)**{ .... }

**div:nth-child(1)**<==> **div:first-child**

##### 4. :empty

匹配**没有**子元素的元素

(没有子元素,没有文本,没有回车或空格)

##### 5. :only-child

匹配属于**其父元素**的**唯一**子元素

#### 3.否定伪类

作用 : 将指定选择器匹配上的元素排除出去

语法 : **not(selector)**

ex :

**td:not(:first-child)**{ .. }/\*匹配表格除第一行外的所有行\*/

### 4.伪元素选择器

#### 1. :first-letter 或 ::first-letter

匹配指定元素的首字符

#### 2. :first-line 或者 ::first-line

匹配指定元素的首行

#### 3. ::selection

匹配被用户选取的内容部分(只能改背景色和文字颜色?)

#### 4. :before 或 ::before

作用 : 匹配到 某元素的内容区域 之前

<p>**(:before)**这是一个p</p>

#### 5. :after 或 ::after

作用 : 匹配到 某元素的内容区域之后

<p>这是一个p**(:after)**</p>

## 2.内容生成

### 1.什么是内容生产

通过css的方式 , 向指定元素的内容区域中 , 增加一部分内容

### 2.伪元素选择器

#### 1. :before 或 ::before

作用 : 匹配到 某元素的内容区域 之前

<p>**(:before)**这是一个p</p>

#### 2. :after 或 ::after

作用 : 匹配到 某元素的内容区域之后

<p>这是一个p**(:after)**</p>

### 3.属性

属性 : **content**

作用 : 向指定的区域(**:before** / **:after**)插入生成的文本 或 图片

取值 :

1.字符串 :**""** 表示要生成的**纯文本**

**ex : p:before{content:"**子曰:**"}**

2.图片 : **url(图片路径)**

### 4.解决问题

#### 1.浮动元素父元素的高度问题

<div id="d1">

<div class="c1"></div>

<div class="c2"></div>

<div class="c3"></div>

<div style="clear:both"></div>

</div>

特点 :

1.在父元素的内容区域之后

2.增加一个"空"元素

3.必须是"块级元素"

4.必须设置"clear:both"

解决 :

匹配父元素的选择器::after{

content:"";

display:block;

clear:both;

}

**.clear::after**{ /\***建议写法**\*/

content:"";

display:block;

clear:both;

}

#### 2.外边距的溢出问题

#d1:**before**{

content:"";

display:**table**;

}

## 3.弹性布局(难点)

### 1.什么是弹性布局

设置某元素内的子元素们的布局方式

弹性布局的容器 : 简称 "**容器**" ,一般指的是父元素

弹性布局的项目 : 简称 "**项目**" , 想实现布局效果的元素

**主轴** : 元素排列方向的一根轴 , 默认是横轴(x轴)

**交叉轴** : 与主轴对应的轴 ,

如果主轴为**横轴**的话,那么交叉轴就是**纵轴**

如果主轴为**纵轴**的话,那么交叉轴就是**横轴**

使用弹性布局:

为容器元素增加属性 **display** ,

设置完成后,子元素自动变成弹性布局的项目

属性 : **display**

取值 :

1. **flex**

将 **块级元素** 变成弹性布局的容器

2. **inline-flex**

将 **行内元素** 变成 弹性布局的容器

注意 :

1.容器 中的**项目**们 ,自动会变为**块级元素** , 允许修改尺寸

2.**项目**们的 **float , clear , vertical-align** 属性全部**失效**

3.**容器** 的 **text-align** 属性 也会**失效**

### 2.容器的属性

#### 1. flex-direction

作用 :

决定主轴 , 以及在主轴的排列方向

取值 :

##### 1.row

默认值 , 主轴为 **横轴** ,起点在**左端**(项目从左向右排列)

##### 2.row-reverse

主轴为 **横轴** ,起点在**右端**(项目从右向左排列)

##### 3.column

主轴为 **纵轴** , 起点在**顶端** (项目从上到下排列)

##### 4.column-reverse

主轴为 **纵轴** , 起点在**底端** (项目从下到上排列)

#### 2. flex-wrap

作用 : 在一根轴上排列不下所有项目时,如何处理换行

取值 :

##### 1.nowrap

默认值 , 不换行 , 但是项目会缩小

##### 2.wrap

换行

##### 3.wrap-reverse

换行 , 第一行在最下方

#### 3. flex-flow

作用 : 是 **flex-direction** 和 **flex-wrap** 的简写属性

取值 :

1.**row nowrap** (默认值)

2.**direction wrap**

#### 4. justify-content

作用 : 指定 **项目** 在主轴上的对齐方式

取值 :

##### 1.flex-start

在轴的**起点**对齐

##### 2.flex-end

在轴的**终点**对齐

##### 3.center

居中对齐

##### 4.space-between

两端对齐,项目之间的间隔都是相等的

##### 5.space-around

每个 **项目** 两边的空白间距是相等的

#### 5. align-items

作用 : 定义项目在**交叉轴**上的对齐方式

取值 :

##### 1.flex-start

在**交叉轴**的**起点**对齐

##### 2.flex-end

在**交叉轴**的**终点**对齐

##### 3.center

在**交叉轴**的**中间**对齐

##### 4.baseline

基线对齐

##### 5.stretch

**默认值**,如果项目未设置高度时,

那么项目将占满整个容器的高度

#### 6. align-content

作用 :

当项目 有 多根主轴时,

指定项目们在交叉轴上的对齐方式以及项目们的边距

取值 :

##### 1.flex-start

在**交叉轴**的**起点**对齐

##### 2.flex-end

在**交叉轴**的**终点**对齐

##### 3.center

在**交叉轴**的**中间**对齐

##### 4. space-between

在**交叉轴**的两端对齐

##### 5.space-around

每个 **轴线** 两侧 的间隔是相等的

### 3.项目属性(共6个)

#### 1.order

作用 : 定义项目的排列顺序,值越小,越靠前,默认值为0

取值 : 整数数字

#### 2.flex-grow

作用 : 定义项目的放大比例,主轴有剩余空间时有效,

默认值为0, 即不放大

取值 : 整数数字

#### 3.flex-shrink

作用 : 定义项目的缩小比例,默认值为1.

当主轴空间不足时,项目们如何缩小的

取值 : 整数数字, 取值为0 , 则不缩小

#### 4.flex-basis

作用 : 定义项目在主轴上的空间大小

取值 :

1. auto , 默认值 , 项目本身大小

2. length , 自定义大小 , 等同于width 或 height

#### 5.flex

作用 : grow , shrink , basis 简写版本

取值 :

1 . auto : 即 1 1 auto

2 . none : 即 0 0 auto

3 . grow shrink basis

#### 6.align-self

作用 : 定义 当前项目 与其它项目不一样的交叉轴对齐方式

取值 :

1.flex-start

2.flex-end

3.center

4.baseline

5.stretch

6.auto - 默认值 ,(即不覆盖)

## 4.CSSHack

解决浏览器兼容性的问题

### 1.CSS Hack的原理

根据不同的浏览器 , 编写不同的CSS代码

靠 CSS的优先级 来解决浏览器兼容性

### 2.CSSHack的实现方式

1.CSS 类内部Hack

在CSS样式属性名称前或者属性值的后,

添加浏览器能够识别的标识,来实现兼容性

2.选择器Hack

在选择器前加前缀识别不同的浏览器

3.HTML头部引用Hack

使用 HTML中的条件注释, 来判断浏览器的类型

语法 :

**<!--[if 条件 IE 版本]>**

满足条件时,要执行的内容

**<![endif]-->**

版本 :

指的是IE的版本号 6~10;

如果省略版本号的话,则判断是否为IE浏览器

条件 :

1. **gt**

判断当前浏览器是否大于指定版本

<!--[if gt IE 6]>

这段内容只能在IE6以上的浏览器中显示

<![endif]-->

2. **gte**

判断当前浏览器是否大于等于指定版本

3. **lt**

判断当前浏览器是否小于指定版本

4. **lte**

判断当前浏览器是否小于等于指定版本

5. **!**

判断当前浏览器是否不等于指定版本

## 5.转换

### 1.什么是转换

改变元素在页面上的 **大小**, **位置** , **形状** ,以及**角度**的一种方式

可以向元素应用 2D 以及 3D 的转换效果

### 2.转换属性

#### 1.转换效果

属性 : **transform**

取值 :

1. **none** : 默认值 , 即没有任何转换效果

2. **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

#### 2.转换的原点

属性 : **transform-origin**

取值 :

1. **两个值** : 表示 原点 在 **x轴** 和 **y轴**的位置

2. **三个值** : 表示 原点 在 **x轴** 和 **y轴** 和 **z轴** 的位置

可以取数值 / 百分比 / 关键字

多个值 中间用 **空格** 隔开

#### 3.2D转换

##### 1.位移

改变元素在页面上的位置

属性 : **transform**

取值 :

1.**translate(x)**

x取值为正 , 向右移动

x取值为负 , 向左移动

2.**translate(x,y)**

x取值为正 , 向右移动

x取值为负 , 向左移动

y取值为正 , 向下移动

y取值为负 , 向上移动

3.**translateX(x)**

4.**translateY(y)**

##### 2.缩放

属性 : **transform**

取值 :

1. **scale(value)**

value表示的是**x轴** 和 **y轴** 的缩放倍率

2.**scale(x,y)**

3.**scaleX(x)**

4.**scaleY(y)**

**x , y** 的取值

默认值 为 **1**

放大 : 大于1的数值

缩小 : 0 - 1 之间的数字

##### 3.旋转

改变元素在页面上的角度

属性 : **transform**

取值 : **rotate(ndeg)**

**n** 取值 为**正** ,**顺**时针旋转

**n** 取值 为**负** ,**逆**时针旋转

注意 :

1.**转换原点**会影响旋转效果

2.旋转操作时,**连同坐标轴**一同都跟着转

##### 4.倾斜

改变元素在页面中的形状

属性 : **transform**

取值 :

1.**skewX(ndeg)**

让元素向横向倾斜

n取值 为**正** , y轴**逆**时针倾斜(**左**倾斜)

n取值 为**负** , y轴**顺**时针倾斜(**右**倾斜)

2.**skewY(ndeg)**

让元素向纵向倾斜

n取值 为**正** , **x**轴**顺**时针倾斜(**左**边高)

n取值 为**负** , **x**轴**逆**时针倾斜(**右**边高)

3.**skew(x)**

等同于 **skewX(ndeg)**

4.**skew(x,y)**

#### 4.3D转换

##### 1.属性

属性 : **perspective**

作用 : 模拟 人眼睛 到 3D转换物体的距离,

取值越大,表示离物体越远,取值越小,表示离物体越近

取值 : 以 px为单位的数值

注意 : 该属性要加在 3D转换元素的 父元素上

##### 2.3D旋转

属性 : **transform**

取值 :

**rotateX(xdeg)**

以**x轴**为中心轴,旋转元素

**rotateY(ydeg)**

以**y轴**为中心轴,旋转元素

**rotateZ(zdeg)**

以**z轴**为中心轴,旋转元素

**rotate3D(x,y,z,ndeg)**

**x,y,z** 取值 大于0的话 , 则该轴参与旋转

**x,y,z** 取值 为0的话 , 则该轴不参与旋转

rotate3D(1,0,0,45deg) ==> rotatex(45deg)

## 6.过渡

### 1.什么是过渡

使得CSS的属性值 在一段时间内平缓变化的效果

过渡能观察到元素的属性值的变化过程

### 2.过渡四要素(四属性)

#### 1.指定过渡属性

作用 : 指定 哪个 CSS属性值在变化时需要使用过渡的效果

当指定的属性值发生变化时,过渡就会被触发

语法 :

属性 : **transition-property**

取值 :

1. **属性名称**

2. **none : 默认值**

3. **all**

允许设置过渡的属性:

**1.颜色属性**

**2.取值为数字的属性**

**3.转换属性 - transform**

4.渐变属性

5.**visibility**

6.阴影属性

#### 2.指定过渡时长

过渡的效果要在多长时间内完成

属性 : **transition-duration**

取值 : 以 **s** 或 **ms** 为单位的数值

默认值 : 为0 , 意味着不会有过渡效果

#### 3.指定过渡的速度时间曲线函数

属性 : **transition-timing-function**

取值 :

1.**ease**

默认值 ,慢速开始 , 快速变快 , 慢速结束

2.**linear**

匀速

3.**ease-in**

慢速开始 , 加速结束

4.**ease-out**

快速开始 , 减少结束

5.**ease-in-out**

慢速开始和结束 , 中间先加后减

#### 4.指定过渡延迟

当用户激发操作后等待多长时间再显示效果

属性 : **transition-delay**

取值 : 以 **s** 或 **ms** 为单位的数值

#### 5.简写属性

**transition : property duration transition-timing-function delay;**

## 7.动画

### 1.什么是动画

动画指使元素从一种样式逐渐变化为另一种样式的过程

动画是复杂版的过渡效果

本质 : 使用 **"关键帧"** ,来定义动画的每一步

关键帧 : 包含 运行的时间点 以及 动作(样式)

### 2.动画的使用步骤

#### 1.声明动画

指定动画名称 以及 涉及到的关键帧们

#### 2.为元素调用动画

指定元素使用哪个动画效果

### 3.声明动画

样式表中 , 通过 **@keyframes** 规则来声明动画

**@keyframes** 动画名称{

0% | from{

/\*动画开始时,元素的样式\*/

属性 : 值;

属性 : 值;

}

50%{

/\*动画执行到一半时,元素的样式\*/

属性 : 值;

}

100% | to{

/\*动画结束时,元素的样式\*/

属性 : 值;

}

}

(不建议写 form to )

#### 动画声明的浏览器兼容性

Firefox : -moz-

Chrome & Safari : -webkit-

Opera : -o-

@-moz-keyframes 动画名{...}

@-webkit-keyframes 动画名{...}

@-o-keyframes 动画名{...}

### 4.调用动画

#### 1.animation-name

作用 : 要调用的动画名称

#### 2.animation-duration

作用 : 动画完成一个周期需要的时长

取值 : 以 **s** 或 **ms** 为单位的数值

#### 3.animation-timing-function

作用 : 指定动画的速度时间曲线函数

取值 :

**ease** **linear**  **ease-in** **ease-out** **ease-in-out**

#### 4.animation-delay

作用 : 指定动画的播放延迟

取值 : 以 **s** 或 **ms** 为单位的数值

#### 5.animation-iteration-count

作用 : 指定动画的播放次数

取值 :

1.默认值 1 ,表示只播放1次

2.**infinite** ,表示无限次播放

3.自定义数值

#### 6.animation-direction

作用 : 指定动画的播放方向

取值 :

1.**normal** : 正向播放 , 从 0% ~ 100%

2.**reverse** : 逆向播放 , 从 100%~0%

3.**alternate** : 轮流播放 ,奇数次数播放时,正向播

偶数次数播放时,逆向播

#### 7.animation

作用 : 以上 6 个属性 的简写模式

取值 :

**name duration timing-function delay iteration-count direction ;**

#### 8.animation-fill-mode

作用 : 动画的填充模式,

指动画在 播放之前 或 播放之后 的显示效果

取值 :

1. **none** : 默认值 , 无任何效果

2. **forwards** : 当动画播放完成后 ,

元素将保持在**最后一帧**的状态上

3.**backwards** : 动画播放前 ,

在延迟时间内,动画将保持在**第一帧**的状态上

4.**both** : 动画播放前后 , 分别应用在**第一帧**

和**最后一帧**的状态上

#### 9.animation-play-state

作用 : 指定动画处于播放状态 还是 暂停状态

取值 :

1. **pause** : 动画暂停

2. **running** : 动画播放

# JS 核心 / JS CORE

## 1.String

### 1.什么是字符串

一串字符组成的只读字符数组

vs 数组

#### 1.相同 :

1. 下标 ,

2. length :

固定套路 :

1.最后一个元素 : arr[arr.length-1]

倒数第n个元素 : arr[arr.length-n]

2.追加一个新元素 : arr[arr.length] = 值

3.删除最后一个元素: arr.length--

清空数组 : arr.length=0;

3.for遍历

4.slice :

arr.slice(strati,endi+1) 含头不含尾

如果只知道要获取的个数: arr.slice(starti,starti+n)

简写 :

1.负数参数 : 表示倒数第n个位置

2.省略第二个参数 : 一直选取到结尾

3.两个参数都省略 : 完整复制数组

#### 2.不同 : 类型不同! API不通用

### 2.string API

#### 强调 :

**所有API无权修改原字符串,只能返回新字符串!**

#### 1.大小写转换 :

**str.toUpperCase() // 转为大写**

**str.toLowerCase() // 转为小写**

何时 :只要不区分大小写时 , 都要先转换为一致的大小写,再判断

例如 : 验证码 , 用户名 和 邮箱地址

#### 2.获得指定位置的字符

**str.charAt(i)** <==> str[i]

**str.charCodeAt(i)** 获得i位置的字符的unicode 号

**String.fromCharCode(unicode)** 将 unicode 转回字

#### 3.获取子字符串

**str.substring(starti,endi+1)** <==> str.slice(starti,endi+1)

**问题 :**  **str.substring()** **不支持负数** !

**解决** : **str.substring(starti,str.length-endi+1)**;

只知道 截取 个数

**str.substr(starti,n) <=>** str.slice(starti, starti +n)

**强调** :**只有**两个参数**都**是**下标**的函数,**才**考虑含头不含尾

所以 : substr()不用考虑含头不含尾, n不用+1

#### 4.查找关键词 4种

##### 1.查找一个固定的关键词出现的位置

var i = **str.indexOf("关键词",fromi)**

在 str中 ,从**fromi**位置开始, 找**下一个**关键词的**位置**

**返回值** : 本次找到的**一个**关键词的下标

如果找不到 , **返回 -1**

如果直接找**最后一个**关键词

var i = **str.lastIndexOf("关键词")**;

**问题**: 只能查找一种固定的关键词

**解决**: 用**正则表达式**模糊查找多种关键词

##### 2. 判断是否包含符合规则的敏感词:

**var i=str.search(/正则表达式/i);**

**问题**1: 所有正则表达式，默认**区分大小写**！

**解决**: 在第二个/后加i (**ignore**忽略)

**问题**2: 只能找**第一个**，无法找所有！

##### 3. 查找所有关键词的内容:

**var kwords=str.match(/正则/ig)**

**返回值**: 包含所有关键词的**数组**

如果找不到，返回null！

**问题**1: 默认只找第一个

**解决**: 在第二个/后加g (**global**全部)

**问题**2: 只能返回内容，无法返回每个关键词的**位置**

##### 4. 即找每个关键词的内容，又找每个关键词的位置:

#### 5.替换 : 2种

##### 1.简单替换 :

将所有关键词替换为一个统一的值

**如何** : **str =** **str.replace(/正则表达式/ig,"替换值")**

**注意 : 必须用变量接收,因为String的所有API无权修改原字符串**

##### 2.高级替换 :

根据本次找到的关键词不同,动态选择不同的关键词替换

如何 :

**str =** **str.replace(/正则表达式/ig,function(kw){**

**/\* 根据kw(keyword),动态返回不同的替换值 \*/**

**// return kw.length==2 ? "\*\*" : "\*\*\*" ;**

**});**

##### 衍生 :

**删除** : 将替换值 设为""(空字符串);

删除开头的空字符

str.replace(**/^\s+/**,"");

删除结尾的空字符

str.replace(**/\s+$/**,"");

**同时删除开头和结尾的空字符**

str=str.replace(**/^\s+|\s+$/g**,"");

#### 6.切割

**什么是 :** 将字符串,按指定分隔符,切割为多段子字符串

**如何 :** 2 种

##### 1.简单切割 :

切割符是固定的字符串

**var subs** **=** **str.split("切割符");** //返回的是数组

切割后的结果中,不包含切割符

固定套路 : 将字符串打散为字符数组

var chars = str.split("");

##### 2.复杂切割 :

切割符不是固定的

**var subs** **=** **str.split(/正则表达式/);**

## 2.内置对象和包装类型

### 1.什么是内置对象

ES标准中规定的,浏览器已经实现的对象

程序员可以直接使用的对象

**JS中所有内置对象 : 11个**

**String Number Boolean**

**Array Date RegExp Math**

**Error Function Object Global(在浏览器中被window代替)**

### 2.什么是包装类型 对象

专门**封装原始类型**的**值** ,并提供**操作**原始类型值的**API**的**类型对象**

**为什么** : 原始类型的值 , 本身**没有**任何功能 !

**何时** : 试图用原始类型的值调用API时,

程序会自动创建包装类型对象

**特点** : 调用后**自动释放** , **不占用内存空间**

## 3.正则表达式（Regular Expression）:

### 1.什么是:

规定字符串中，字符出现规律的规则

### 2.何时: 2种:

1. 用规则模糊查找多种关键词

2. 验证字符串的格式

### 3.如何:

#### 1. 最简单的规则，是关键词原文

#### 2. 字符集:

**什么是:** 规定1位字符，备选字符列表的集合

**何时:** 只要字符串中1位字符，有多种备选时

**如何: [**备选字符列表**]**

**强调:** 1. 不是数组，**不加逗号**！

2. 无论备选字符列表，有多少字，

一个字符集**只能**匹配**1位**字符！

当备选字符列表部分连续时，可用**-**省略中间的字符

**常见:** 一位数字 **[0-9]**

一位小写字母[a-z]

一位大写字母[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]

#### 3. 预定义字符集: 4个

其实，就是对常用字符集的简写

**\d** 一位数字 [0-9]

**\w** 一位字母，数字或下划线 [0-9A-Za-z\_]

**\s** 一位空字符: 空格 换行 Tab

**.** 除换行外的任意字符

**问题:** 仅使用字符集，无法灵活定义字符的个数

#### 4. 量词:

**什么是:** 专门规定一个字符集，出现次数的规则

**何时:** 只要固定一个字符集出现的次数，就用量词！

**如何:** 字符集量词

**包括: 2大类:**

**1. 有明确数量限制:**

**字符集{m,n}** 至少m个，最多n个

**字符集{m, }** 至少m个，多了不限！

**字符集{m}**  必须m个，不能多也不能少

比如: 手机号: 1[34578]\d{9}

QQ号: \d{4,11}

**2. 没有明确数量限制:**

**字符集?**  可有可无，最多1个

**字符集\***  可有可无，数量不限

**字符集+**  至少一个，多了不限

## 4.RegExp

### 什么是 : 专门封装一条正则表达式,并使用正则表达式,

执行验证和查找功能的对象

### 何时 :

**1.查找**

**2.验证**

### 如何 :

#### 1.创建 : 2种

##### 1.直接量 : var reg = /正则/ig

**何时** : 只要正则表达式是**固定不变**的!

##### 2.用 new :

**var reg = new RegExp("正则","ig");**

**何时** : 如果正则表达式需要**动态生成**

#### 2.验证

验证一个字符串的格式,是否符合正则表达式的要求

**var bool = reg.test(str);** //返回 **bool 值**

**问题** : 默认,正则表达式只要部分匹配 , 就会返回true

**解决** : 只要验证 , 必须**前加 ^** , **后加 $** ,从头到尾完整匹配

#### 3.查找

即找每个关键词的内容,又找每个关键词的位置

**var arr = reg.exec(str);**

在str中 , 找**下一个**符合reg要求的关键词

**返回值 : 数组 :**

**[0:"关键词内容",index:本次找到关键词的位置]**

**如果找不到,返回null**

**原理 :** 每次查找 **3 件事**:

**1.**将本次找到的**关键词**,放入arr中**0**位置

**2.**将本次关键词的**位置**,放入arr中**index**位置

**3.**自动修改**reg**的**lastIndex属性**为**当前位置+关键词长度**

## 5.Math

**什么是:** 专门封装数学计算的常量 , 并提供数学计算的API对象

**何时 :** 只要执行数学计算时

**如何 :**

**创建 :** **不能 new ! 必须用 Math直接调用API**

### 1.取整 : 3 种

#### 1.上取整 :

**只要超过,就取下一个整数**

**Math.ceil(num)**

#### 2.下去整 :

**舍弃小数部分**

**Math.floor(num)**

**vs**

**parseInt(str) //选这个,能接收字符串(去单位)**

#### 3.四舍五入取整:

**Math.round(num)**

**// 缺点 : 只能取整!**

**// 优点 : 返回Number类型数字**

**vs**

**num.toFixed(d)**

**// 优点 : 按任意小数位四舍五入**

**// 缺点 : 返回字符串类型**

**自定义 :**

**function round(num,d){**

**num \*= Math.pow(10,d);**

**num = Math.round(num);**

**return num/Math.pow(10,d);**

**}**

**//需要判断num是不是数字**

### 2.乘方 和 开平方

#### 1.乘方

**Math.pow(底数,幂);**

#### 2.开方

**Math.sqrt(num)**

### 3.最大值 / 最小值

Math.max(值1,num2,num3...)

**Math.max.apply(null,arr)**

Math.min(num1,num2,num3...)

**Math.min.apply(null,arr)**

**问题 :** 不支持获得**数组**中的最大值,最小值

**解决 :** **Math.max/min.apply(null,arr)**

**其中:apply负责将arr数组打散为单个元素,再传给max/min**

### 4.随机数

**Math.random()**

在0~1之间随机生成一个小数

在任意min~max之间,取一个随机整数的公式

**parseInt(Math.random()\*(max-min+1)+min)**

简写 :

从0到任意值

parseInt(Math.random()\*(max+1))

## 6.Date

### 什么是:

**1.封装 - 日期和时间**

**2.提供 - 操作日期和时间的API**

### 何时 :

只要存储日期和时间 , 都用Date类型的对象

### 如何:

#### 创建: 4种

##### 1.获得当前系统时间

**var now = new Date();**

**强调 :** 只能获得**客户端**当前操作系统的时间

##### 2.保存自定义时间 :

**var data = new Date("yyyy/MM/dd hh:mm:ss");**

##### 3.复制一个日期对象

为什么 : 今后,日起对象的计算,

都直接修改原日期对象.计算后,原日期无法保留

何时 : 只要用户希望同时保留计算前后的

开始和结束时间,都要先复制副本.

再用副本计算结束时间

如何 : **var data2 = new Date(data1)**;

##### 4.将毫秒数转化为当地时间

为什么 : 在系统中存储时间,都是使用毫秒数

何时 : 只要将毫秒数转为当地时间显示

如何 : **var data = new Date(ms)**

**补**--**如何获取毫秒数** :

**var ms=date.getTime();**

原理 : 日期对象,中保存的是

1970年1月1日0点0分0秒至今的毫秒数

为什么 : 文字会受时区的影响,而毫秒数不受时区的影响

### API :

#### 1. 八个单位

**FullYear Month Date Day 没有s结尾**

**Hours Minutes Seconds Milliseconds 都有s结尾**

#### 2. 每个单位 都有一对 getXxx()/setXxx()方法

其中 : **getXxx() 负责获取指定单位的数值**

**setXxx() 负责修改指定单位的数值**

**特例 : Day没有set方法**

#### 3. 取值范围

**Month : 0~11** , 计算机中每个月份都比现实**少1**

**读取时 : +1修正**

**修改时 : -1 修正**

**Date : 1~31**

**Day : 0~6**

**Hours : 0~23**

**Minutes / Seconds : 0~60**

### 日期计算 : 2种

#### 1.两日期对象可相减 , 结果是毫秒差

#### 2.对任意单位的数值做加减 : 3步

1.取出单位的值

2.做加减

3.设置回去: date.setXxx(n)

强调 : setXxx()可自动根据n,调整时间进制.

其实可简写:

**date.setXxx(date.getXxx()+n);**

### 日期转字符串

**date.toString()** 转为当地时间的完整格式

**date.tolocaleString()** 转为当地时间的简化版格式

**date.tolocaleDateString();**转换为当地时间的简化版

,仅保留日期部分

**date.tolocaleTimeString();**转换为当地时间的简化版

,仅保留时间部分

**date.toGMTString();** 转为国际标准时间(0时区)

#### 自定义:

**function format(date){**

**var week=["日","一","二","三","四","五","六"];**

**var y = date.getFullYear();**

**var M = date.getMonth()+1;**

**function pad0(val){**

**return val<10?"0"+val:val;**

**}**

**//M<10&&(M="0"+M); //这个也可以**

**M=pad0(M);**

**var d = date.getDate();**

**d=pad0(d);**

**var day = week[date.getDay()];**

**var h=date.getHours();**

**var am = h<12?"上午":"下午";**

**h>12&&(h=h-12);**

**h=pad0(h);**

**var m=date.getMinutes();**

**m=pad0(m);**

**var s=date.getSeconds();**

**s=pad0(s);**

**return y+"年"+M+"月"+d+"日 星期"+day+" "+am+" "+h+":"+m+":"+s+"";**

**}**

## 7.Error

### 什么是:

程序运行过程中,导致程序无法正常执行的问题

程序发生错误后,都会强行中断退出

### 什么是错误处理 :

当程序发生错误时,依然保证程序不中断退出的机制

### 何时 :

只要有可能发生错误的代码,都要提前使用错误处理

### 如何 :

**try{**

**可能出错的正常逻辑代码**

**}catch(err){ //**只有出错才执行 , 不出错不执行

**一但出错,才执行的错误处理代码:**

**提示用户 , 记录日志 , 保存数据**

**}**

**其中 : err : 错误对象 : 专门封装错误信息的对象**

**发生错误时 , 自动创建!**

### 主动抛出错误 :

**throw new Error("错误信息!")**

**何时 :** 函数的作者向函数的调用者**抛出错误**,

**提示**调用者错误的使用了函数

## 8.Function(重点)

### 1.创建 : 3 种

#### 1.声明:

**function 函数名(参数列表){**

**函数体;**

**return 返回值;**

**}**

**问题 : 被声明提前**

**解决 : 直接量声明**

#### 2.直接量※ :

**var 函数名 =** **function (参数列表){**

**函数体;**

**return 返回值;**

**}**

**优点 : 不会被声明提前影响**

**揭示 :**

**函数其实也是一个引用类型的对象**

**function 是创建一个新函数的意思**

**函数名其实就是一个引用函数对象的简单变量**

#### 3.用 new

**var 函数名 = new Function("参数变量",...,"函数体");**

### 2.重载(overload)

#### 什么是:

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,自动选择对应函数执行

#### 为什么:

减少函数的个数 , 减轻调用者的负担

#### 何时 :

只要一项任务 , 根据不同的参数,有不同的实现方式

#### 问题 :

JS 语法 默认不支持重载!

#### 原因 :

JS 中不允许多个同名函数同时存在!

#### 解决 :

用 **arguments 对象**

##### 什么是 arguments 对象:

函数中,**自动创建**的

**自动接收**所有传入函数的参数值的

**类数组对象**(object like array)

相同点 : 1.下标 , 2.length , 3.for遍历

不同点 : 类型不同! API不通用!

##### 何时 :

只要函数的参数个数不确定时,

都可用arguments接住所有参数

##### 问题 :

需不需要参数变量!

##### 参数的作用 :

1. 提醒调用者 , 如何正确使用函数

2. 一般自定义的参数名 , 都是见名知意 , 且都非常简单

##### 所以 :

函数 都要 优先 使用参数变量 接收参数值

### 3.匿名函数

#### 什么是 :

创建函数时,不指定函数名的函数

#### 为什么 :

**节约内存** , **划分临时作用域**

#### 何时 : 2种

1.如果一个函数,只使用一次

2.如果避免使用全局变量,划分临时作用域

#### 如何 : 2种

##### 1.回调 : 将一个函数,交给另一个函数去调用, 3种 :

1.sort比较器

2.str.replace 高级替换

3.事件处理器

##### 2.自调 : 创建函数后 , 立刻调用自己

**为什么** : 避免使用全局变量

**全局变量最大的问题 :** 全局污染

**何时 :** 避免使用全局变量,造成全局污染时

**今后**:**几乎所有**的js程序 , **都**应该放在**匿名函数自调**中

**如何自调 :** **2种**

1. **( function(参数列表){函数体} )() -(红的都是圆括号)**

2. **+function(参数列表){函数体}()**

3. **!function(参数列表){函数体}() -(bootstrap这样用)**

### 4.作用域和作用域链(重难点)

#### 1.作用域(scope)

##### 什么是 :

一个变量的可用范围-----用途

一个**存储变量**的**对象**-----存储/本质

##### 为什么 :

避免不同范围的变量间互相干扰

##### 包括 : 2种

###### 1.全局作用域

不属于任何函数的,外部的范围

**其实 : window**

**全局变量** **:** 反复使用,随处可用!

###### 2.函数作用域

函数内的范围

**其实 :** **活动对象(AO)**

**局部变量** **:** 仅函数内可用 , 不可反复使用!

#### 函数的生命周期

##### 1.开始执行程序前

创建 **执行环境栈(ECS)** : 保存正在调用的函数记录

首先 自动调用浏览器主程序 main()

主程序创建全局作用域对象 **window**

##### 2.定义函数时

在**window**中用函数名**创建全局变量**

**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**score 属性**指回函数来自的作用域对象

##### 3.调用函数时

在**ECS**中**压入**当前函数的**调用记录**

为本次函数调用**创建活动对象AO**

AO中**保存局部变量**

让AO指向函数的scope属性指向的相同**对象window**

变量的使用顺序:

**优先用AO中的局部变量，AO中没有，才去window找**

##### 4.函数调用后

**函数调用的记录**从ECS中**出栈**

**导致:** **AO释放->导致: AO中的局部变量一同释放！**

所以，局部变量仅在函数调用时可用，不可重用！

#### 2.作用域链(scope chain):

由多级作用域连续引用形成的链式结构

**存储:** 所有变量

**控制:** 变量的使用顺序: 先局部，后全局

### 5.闭包(※重难点※)

#### 什么是

即 **重用** 变量，又保护变量不被污染的一种机制

#### 为什么

**全局变量 :** **优点 :** 可重用! **缺点 :** 污染 , 随处可用!

**局部变量 : 优点 :** 仅函数内可用 , 不会污染全局

**缺点 :** 不可重用!

#### 何时 :

只要即重用一个变量，又保护变量不被污染时

#### 如何 : 3步:

1. 用**外层函数**包裹要保护的**变量**和操作变量的**函数**

2. **外层函数**将**内层函数**的对象**返回**到外部

3. 使用者**调用外层函数**，**获得内层函数对象**

#### 原因 : ※

**闭包的形成原因:** 外层函数的作用域对象(AO)无法释放

被内层函数对象引用着

#### 简图: 2步:

1. 找受保护的变量，并确定其最终值

2. 找使用变量的内层函数对象，

只有内层函数才能使用受保护的变量

#### 闭包缺点:

比普通函数占用**更多**内存空间！

#### 解决:

闭包不再使用，要**及时释放**

#### 如何释放:

将引用内层函数对象的变量赋值为**null**

## 9.OOP※

### 什么是面向对象:

程序中都是用对象结构描述现实中一个具体事物

### 什么是对象:

程序中专门描述现实中一个具体事物的程序结构

### 为什么:

现实中，一个数据，必须属于某个具体对象中，才有意义。

### 何时:

今后，只要描述一个事物，

都要将事物的属性和功能集中定义在一个对象中

### 如何: 面向对象三大特点:

封装，继承，多态

### 1.封装

#### 什么是:

创建一个对象，存储一个事物的属性和功能

#### 其中:

##### 事物的属性，会成为对象的属性

属性其实就是存在对象中的变量

##### 事物的功能，会成为对象的方法

方法其实就是存在对象中的函数

功能和属性，统称为成员

#### 为什么:

便于大量维护数据

#### 何时:

只要使用面向对象的方式编程，

都必须先将事物的属性和功能封装在对象中，

再按需使用对象的属性和方法。

#### 如何: 3种:

##### 1. 对象直接量:

var obj=**{**

属性名:值,

... : ... ,

方法名 (){//ES6

...

}

**}**

##### 如何访问对象的成员：

**obj.属性名** -> 每个属性的用法和普通变量完全一样！

**obj.方法名()** -> 每个方法的用法和普通函数完全一样！

**问题 :** 对象要访问自己的属性,却找不到

**原因 :** 不加.访问的变量,只能在作用域链中查找,

无法自动进入对象中查找

**解决 :** 错误 : 在属性前加 "对象名." **×**

**在对象内,禁止写死对象名**

因为对象名其实仅是一个普通的变量,随时可能改变

**正确 : 用 this. 属性名** :

**this** : 在函数执行时,自动创建的一个关键词

自动指向正在调用当前函数的.前的对象

**为什么 :** 不受对象名的影响

##### 总结 :

今后,**只要**对象自己的方法,想访问自己的属性,

**必须**用**this.属性名**

##### 2. 用 new : 2步

###### 1.先创建一个空对象

**var obj = new Object();** // new或()可省略,但不能同时省略

###### 2.向空对象中添加新成员 :

**obj.属性名 = 值**;

**obj.方法名 = function(){ .... }**

**说明:** JS中的对象 , 可随时 , 通过强行赋值的方法,添加新成员

**何时:**如果在创建对象时,暂时不知道对象的成员

**揭示:** JS中的对象 , 底层其实就是关联数组

**相同 :** 1.都可用 .或[]随时添加/访问成员

2.访问不存在的成员 , 不报错 , 返回 undefined

3.都可用for in遍历每个成员

**不同 :** 对象比关联数组用法简单

今后 , **都是用对象**来代替**关联数组**使用!

###### 问题 :

一次只能创建一个对象

如果反复创建相同结构的多个对象,相同的代码会很多

###### 解决 : 构造函数

##### 3.用 构造函数 (constructor)

**什么是 :** 描述同一类型的多个对象 , 相同成员结构的函数

**第二个作用 :** 将一个空对象构造为拥有属性和功能的完整对象

**何时 :** 只要 反复创建同一类型的多个对象时,都要先用构造函数描述统一的结构,

**如何 : 2步**

###### 1. 定义构造函数 :

**function 类型名(属性参数, ... ){**

**this.属性名 = 属性参数 ; ...**

**this.方法名 = function(){ ...this.属性名... }**

**}**

###### 2.使用构造函数反复创建对象 :

**var obj = new 类型名( 属性值 , ... ) ;**

调用时的参数值 , 应和定义构造函数时的属性参数保持一致

一致 : 个数 , 顺序

###### new : 4件事 :

**1.创建一个空对象**

**2.** **自动让新的子对象继承构造函数的原型对象**

**3.调用构造函数,将构造函数中的this指向正在创建的新对象.**

**向新的空对象中强行添加新成员**

**4.将新对象地址 返回给变量 保存**

**优点 : 重用结构定义**

**缺点 : 浪费内存**

### 2.继承 :

#### 什么是 :

父对象的成员 , 子对象无需创建即可直接使用!

#### 为什么 :

**代码重用! 节约内存!**

#### 何时 :

所有子对象 都 拥有相同的属性值和方法定义时,

都要 用继承来实现

#### 如何 :

JS中 所有继承 ,都是继承原型对象

##### 什么是原型对象 :

集中存储所有子对象共有成员的父对象

##### 为什么 :

实现继承 , **代码重用! 节约内存!**

##### 何时 :

只要实现继承 , 都要继承原型对象

##### 如何 :

###### 创建 :

**不用手动创建** , 买一赠一

其实创建构造函数同时,都附赠一个空的原型对象

###### 继承 :

**不用手动设置**

用**new创建**新的子对象时,

会自动设置新对象继承构造函数的原型对象

##### 添加共有成员

构造函数.prototype.成员 = 值

#### 总结 :

只要所有子对象共用的成员,都必须集中存储在原型对象中

#### 自有属性和共有属性

##### 自有属性:

直接保存在对象本地的属性

##### 共有属性

保存在原型对象中,所有子对象共有的属性

##### 读取 :

两者完全一样 , 对象.属性

##### 修改 :

###### 自有属性 :

只能用子对象改 , 子对象.自有属性名=值

###### 共有属性 :

只能用原型对象修改 : 构造函数.prototype.属性名 = 值

#### 原型链 : prototype chain

##### 什么是 :

由多级父对象 , 逐级继承 , 形成的链式结构

##### 为什么 :

为了更高级,更大范围的重用

##### 如何 :

所有对象 , 都有 **\_ \_proto\_ \_**属性

原型的 **\_ \_proto\_ \_**指向更上级的父对象

所有对象最终都继承自 **Object.prototype** ----顶级父对象

#### 内置对象的原型链 :

其实每种内置类型 , 都有对应的构造函数和原型对象 , 也最终都继承自 Object.prototype

其中 : 内置类型的构造函数负责**创建**该类型的子对象

内置类型的原型对象 , 负责**保存**该类型所有子对象共有的**API**

**问题 : 旧浏览器不支持新的API**

**解决 : 向旧浏览器中的原型对象中手动添加一个函数**

### 3. 多态:

#### 什么是:

同一个函数在不同情况下表现出不同的状态

#### 重写:

如果子对象觉得父对象的成员不好用，

可在本地定义同名成员，覆盖父对象中继承来的成员

#### 为什么:

体现子对象和父对象之间的差异

#### 何时:

只要子对象觉得父对象的成员不好用，就可以重写！

### 自定义继承

#### 何时 :

只要觉得默认的父对象不好用时,可换父对象

#### 如何 : 3种

##### 1.仅修改一个子对象的父对象

child.\_\_proto\_\_ = father

问题 : \_\_proto\_\_是内部属性

解决 : 获取Object.getPrototypetypeof(Object)

修改 Object.getPrototypetypeof(child,father)

##### 2.修改所有子对象的父对象 :

其实就是改构造函数的原型对象

构造函数.prototype=father

时机:在创建子对象之前就要更换!

##### 3.两种类型间的继承:

###### 何时:

如果多个类型间,包含部分相同的属性结构或方法定义时,

都要再抽出一个父类型

###### 如何: 2步:

**1.抽象父类型:**

构造函数:相同部分的属性定义

原型对象:相同部分的方法定义

**2.让子类型继承父类型**

让子类型的原型对象继承父类型的原型对象

在子类型构造函数中借用父类型构造函数

**问题:**不能直接调用父类型构造函数

**因为:**一个函数不用**new**和**.this**默认指**window**

导致父类型构造函数的属性,流落到全局

**解决:call:**调用函数,

并临时替换函数中的**.this**为任何想要的对象

**何时:**只要函数调用时**.this**不是想要的,就用**call**

**如何: fun.call(obj)**

## ES5※

### 1.严格模式:

#### 什么是:

比普通js运行机制更严格的模式

#### 为什么:

js中存在很多广受诟病的缺陷

**1. 允许给未声明的变量赋值**

**2. 静默失败**

#### 何时:

今后**所有**项目**必须**在严格模式下开发

#### 如何:

在当前代码段的顶部添加**"use strict"**;

在两个范围启用严格模式:

**在全局启用严格模式**: 在全局添加"use strict"

**何时:** 新项目，必须在全局启用

**仅在函数内启用严格模式**: 只在函数内顶部添加"use strict"

**何时:** 旧项目，向严格模式迁移

**要求:**

**1. 禁止给未声明的变量赋值！**

**2. 将所有静默失败升级为错误！**

### 2.保护对象※:

#### 为什么:

传统js中，对象的属性值和结构，可随意更改，

毫无自保和验证能力

#### 何时:

今后几乎所有的对象，都要不给严格规定保护

#### ES5中的变化:

**对象的属性也是一个小的对象:**

**对象的属性，包含四大特性:**

**{**

**value: 属性值,**

**writable:true/false, //控制是否可修改属性**

**enumerable:true/false,//控制是否可用for in遍历到该属性**

**//仅控制for in遍历，用.依然可以访问**

**configurable:true/false //控制:**

**//1. 禁止删除该属性**

**//2. 禁止修改其它特性的值**

**//一旦改为false，不可逆！**

**//今后，只要修改其它特性，都要伴随修改configurable为false——双保险**

**}**

#### 如何:

**获取属性的四大特性:**

**var attrs=Object.getOwnPropertyDescriptor(obj,"属性名")**

**修改属性的四大特性:**

**Object.defineProperty(obj,"属性名",{**

**特性名:值,**

**... : ... ,**

**})**

**问题: defineProperty**每次只能修改一个属性

**解决:** **defineProperties** 一次可**同时修改多个**属性

**Object.defineProperties(obj,{**

**属性名:{**

**特性名:值,**

**... : ...**

**},**

**属性名:{**

**特性名:值,**

**... : ...**

**},**

**})**

**问题:** 普通属性的四大特性，只能提供基本的保护

无法使用自定义规则保护属性

**解决:** 用访问器属性:

### 3.访问器属性:

#### ES5将对象的属性分为:

**数据属性和访问器属性**

**数据属性:** 直接存储属性值的属性

**访问器属性:** 不直接存储数据，仅提供对其它数据属性的保护

#### 为什么:

数据属性只能对自己提供基本的保护，

无法使用自定义逻辑保护自己。

#### 何时:

只要用自定义属性保护数据属性时，就只能用访问器属性

#### 如何: 2步:

**1. 首先定义一个隐藏的实际存储数据的数据属性**

**2. 然后定义访问器属性保护数据属性**

**Object.defineProperties(obj,{**

**\_age:{ //数据属性**

**value:25,**

**writable:true,**

**enumerable:false, //隐藏**

**configurable:false**

**},**

**age:{ //访问器属性**

**get(){ //在试图用访问器属性取值时，自动调用**

**return this.\_age;**

**},**

**set(val){ //在试图给访问器属性赋值时，自动调用**

**//val可自动获得要赋的新值**

**if(val在范围内)**

**this.\_age=val;**

**else 报错！**

**},**

**enumerable:true,**

**configurable:false**

**}**

**})**

**使用访问器属性: 用法和普通数据属性完全一样**！

### 4.防篡改：

**禁止修改对象的结构**

三个级别:

#### 1. 防扩展:

##### 禁止为对象强行添加新属性:

**Object.preventExtensions(obj)**

原理: 对象都有一个内部属性: extensible, 默认true

preventExtensions其实是将extensible改为false！

#### 2. 密封:

##### 在防扩展的同时，再禁止删除所有属性

**Object.seal(obj)**

原理: 将所有属性的configurable特性都改为false

#### 3. 冻结:

##### 在密封的同时，再禁止修改所有属性值

何时: 如果一个对象，连属性值都不让随便改，就要冻结

如何: **Object.freeze(obj)**

原理: 将所有属性的writable都自动改为false

### 5.Object.create:

**三件事: 基于现有父对象，创建一个子对象，再扩展新属性**

**何时: 如果只有父对象，没有构造函数时，也想创建子对象**

**如何:**

**var child=Object.create(father,{**

**//defineProperties**

**属性名:{**

**特性名:值,**

**... : ...**

**},**

**属性名:{**

**特性名:值,**

**... : ...**

**},**

**})**

### 6. call/apply/bind

#### 替换this！

#### 何时:

只要**this**不是想要的，就用call/apply/bind替换

#### call vs apply vs bind

**call/apply : 立刻调用函数，并临时替换函数中的this为指定对象**

**何时:** 如果希望立刻调用函数时

**如何: fun.call(obj,参数值,......)**

**立刻调用fun函数，并临时替换fun中this为obj**

**fun.apply(obj,数组)**

**立刻调用**fun函数，并**临时替换**fun中**this**为obj

同时，**打散数组**参数为单个参数值，再传给fun

**bind:** 基于原函数，**创建一个新函数**，**永久绑定this为指定对象**

**何时:** 如果希望基于原函数创建一个新函数时

或者说，**不是立刻调用**时

**如何: fun.bind(obj,参数值,...)**

**创建一个和fun完全一样的新函数**

**永久绑定fun中的this为obj**

**永久绑定部分参数值**

### 7. 数组API:3组:

#### 1. 判断:

##### every():

判断数组中是否**所有元素**都符合条件

**如何: var bool=arr.every(function(val,i,arr){**

**//val自动获得当前元素值**

**//i获得当前位置**

**//arr 指当前数组对象本身**

**return 判断条件**

})

**原理:** every用function去每个元素上执行一次

只有所有元素都返回true时，整体才返回true

只要有一个返回false，则整体返回false

##### some():

判断数组中**是否包含**符合条件的元素

**如何: var bool=arr.some(function(val,i,arr){**

**同every**

**return 判断条件;**

})

**原理:** some用function去每个元素上执行一次

只要有一个返回true，则整体返回true

除非所有元素都返回false时，整体才返回false

#### 2. 遍历API:

##### forEach():

对原数组中每个元素执行相同的操作

**如何:** **arr.forEach(function(val,i,arr){**

**arr[i]=新值;**

**})**

##### map():

依次取出原数组中的值，执行相同操作后，**放入新数**组返回。

**如何:** **var newArr=arr.map(function(val,i,arr){**

**return新值;**

**})**

#### 3. 过滤和汇总:

##### filter():

选取数组中**符合条件**的元素，**组成新数组**。原数组不变

**如何: var subArr=arr.filter(function(val,i,arr){**

**return判断条件;**

**})**

##### reduce():

遍历数组中每个元素值，**汇总**出一个**最终结果**

**如何: var result=arr.reduce(function(prev,val,i,arr){**

**//prev 截止目前的临时汇总值**

**return prev+val;**

**},prev)**

## ES6:

### 1. let和块作用域

#### 问题1:

hoist，打乱了程序的正常执行顺序

#### 解决:

**今后用let代替var声明变量**

#### 原理:

**let**通过**禁止**在当前**作用域**内**提前使用未声明的变量**，来避免声明提前

#### 强调:

如果已**经用let声明的变量**，不能再**用var重复声明**

#### 问题2:

**js没有块级作用域**, 块内的变量即使不执行，也会影响块外的变量:

**块级作用域:** 程序中的一对儿{}之间，称为代码块

**js vs java: Java** **三级**作用域: **全局，函数，块**

**js**  **两级**作用域: **全局，函数，**没有**块级作用**

#### 解决:

**用let代替var**

#### 规定:

**let**声明的变量仅在当前块内有效！

#### 原理:

自动在块作用域位置创建匿名函数自调，划分临时作用域。

#### 问题3:

原来在for/while/if...内声明的变量，出了结构，就不能使用

#### 解决:

今后，凡是出了结构，还要使用的变量，必须在结构外提前声明。

#### 总结:

**今后所有变量都要用let声明，而不var**

### 2. 箭头函数: 简化回调函数

#### 何时:

今后所有回调函数都可用箭头函数简化！

#### 如何:

##### 1. 去function改箭头

更简化:

##### 2. 如果只有一个参数变量，可省略()

##### 3. 如果函数体只有一句话，可省略{}

**如果仅有一句return，则可省略return**

**箭头函数特点: 内容this通用！**

### 3.参数增强:

#### Default:(默认值)

**function fun(参数1,参数2,参数3=默认值,参数4=默认值){}**

调用时，如果传给参数的值有效，就用传入的值

如果没有传入参数，或传入的参数无效，则用提前指定的默认值

#### REST:(剩余参数列表) 为了代替arguments

##### arguments的问题:

1. 不是真正的数组类型，无法使用数组类型的API

2. 默认只能获得所有参数值，无法仅选取部分参数值

##### 解决: REST

**何时:** 今后，都是**用REST语法代替arguments**

**如何:**

定义函数: **function fun(参数1,参数2,...剩余参数的统称)**

调用时: **...剩余参数的统称 会获得一个数组**

#### SPREAD:(散播) 为了代替apply打散数组类型的参数

##### apply的问题:

1. 必须传入第一个对象参数，代替this

2. 必须将所有参数都放入数组中

##### 解决: spread

**何时:** 今后**只要打散数组类型参数**时，**首选spread**

**如何:**

定义函数时，参数变量是分开定义的

调用函数时，可用fun(值1,...arr)，打散arr数组

### 4. 模板字符串: 简化复杂格式的字符串拼接

#### 何时:

今后**所有字符串拼接**，都用**模板字符串代替+**

#### 如何:

##### 1. 用反引号`包裹整个字符串

##### 2. 在模板字符串内，自动支持换行

##### 3. 在模板字符串内，支持动态生成表达式的值

**要执行的表达式，必须用${}包裹**

### 5. 解构: 简化复杂对象的打散和批量赋值

#### 1. 数组解构: 等号左右两边都是数组格式

解构时，用下标对应值和变量

#### 2. 对象解构: 等号左右两边都是对象格式

解构时，用属性名匹配值和变量

#### 3. 参数解构:

其实，在调用函数时，

如果传入的参数值是数组或对象，也可被解构为单个参数值

用法和数组结构和对象结构完全一样

### 6: class: 为了简化面向对象:

封装，继承，多态，访问器属性，静态方法

**class: 集中定义构造函数和原型对象方法的程序结构**

#### 为什么: 更像封装

#### 何时: 只要创建一种新类型，必须先定义class

#### 如何:

class 类型名{

constructor(...){

this.xxx=xxx;

}

原型对象方法(){//类型名.prototype.xxx()

... this.xxx ...

}

}

#### 继承: 2步:

##### 1. 子类型构造函数中借用父类型构造函数:

super(参数值)

super自动指代父类型的构造函数

##### 2. 让子类型继承父类型

class 子类型 extends 父类型

不用再Object.setPrototypeOf

##### 访问器属性:

class 类型名{

constructor(...){

this.\_age=age

}

get age(){return this.\_age}

set age(val){

... ...

}

}

### 7.Set 集合:

#### 什么是:

值不允许重复的集合

#### 何时:

快速去重复

通过元素值，快速判断是否包含元素时

#### 如何:

var set=new Set(凡是可遍历的，都可打散转为Set)

将set转回数组: var arr=[...set];

### 8.Map 集合:

#### 什么是:

一组键值对的集合

#### 何时:

代替关联数组使用

#### 如何:

var hash=new Map(); //{}

hash.set(key,value); //hash[key]=value

var bool=hash.has(key); //hash[key]!==undefined

### 9.for of : 代替for循环遍历

#### 什么是:

依次获得数组/类数组对象/集合中每个值

#### 何时:

只要**不关心位置**，只关心每个元素值时

#### 如何:

for(var value of 数组|类数组对象|集合){

value: 自动获得当前元素值

}

#### 问题:

1. 无法获得位置

2. 必须从头到尾，逐个执行，不能跳跃或遍历部分

3. 只能从头到尾，顺序遍历，不能反向遍历

### 10.\*\*\*\*Promises

#### 什么是:

解决"callback hell"

#### 为什么:

回调函数，层级很可能很深，可读性极差

#### 何时:

今后，只要多个回调函数，必须保证顺序执行时

#### 如何:

function conn(){

console.log("连接数据库...");

**return new Promise((resolve,reject)=>{**

**//.then .catch**

**//3秒后自动调用callback**

**setTimeout(()=>{**

**var err=Math.random()<0.3?true:false;**

**if(!err)**

**resolve();//query->response**

**else**

**reject("连接出错");//"出错啦"**

**//(err)=>console.log(err);**

**},3000);**

**});**

}

function query(){

console.log("查询数据...");

**return new Promise((resolve,reject)=>{**

**setTimeout(()=>{**

**var err=Math.random()<0.3?true:false;**

**if(!err)**

**resolve();//response**

**else**

**reject("查询出错");//"出错啦"**

**},5000);**

**});**

}

function response(){

console.log("返回响应...");

**return new Promise((resolve,reject)=>{**

**setTimeout(()=>{**

**var err=Math.random()<0.3?true:false;**

**if(!err)**

**resolve();//"成功"**

**else**

**reject("响应出错");//"出错啦"**

**},2000);**

**})**

}

//callback hell

// conn(()=>

// query(()=>

// response(()=>console.log("查询出结果!"))

// )

// );

**conn()//Promise**

**.then(()=>query())//Promise**

**.then(()=>response())//Promise**

**.then(()=>console.log("查询出结果"))**

**.catch((err)=>console.log(err))**

# DOM

## 1. 什么是DOM: Document Object Model

专门操作网页内容的API标准——W3C

### 为什么:

没有DOM前，不同浏览器操作网页内容的API不统一

### 何时:

要想在不同浏览器上都能操作网页内容，

必须用统一的DOM API——几乎100%兼容所有浏览器

### 操作: 五大操作:

查找, 修改, 添加, 删除, 事件绑定

## 2. DOM Tree

### 什么是:

内存中存储所有网页内容的树形结构

### 为什么:

网页内容(元素，文本，属性)都有明显的上下级包含关系。

### 如何:

网页中一切内容，在内存中，都存储在一棵树结构中

document节点是唯一的树根节点，包含所有网页内容

网页中每一项内容(元素，文本，属性)都是一个节点对象

### 节点对象:

#### 类型:

**Node**类型

#### 三大属性:

##### .nodeType: 标识节点类型

**何时:** 只要判断节点类型时，就用nodeType

**包括:**

**document 9**

**元素 1**

**属性 2**

**文本 3**

**问题:** 无法进一步区分元素的名称

##### .nodeName: 标识节点的名称

**何时:** 只要需要进一步区分元素的名称

**包括:**

**document #document**

**元素 全大写的标签名**

**属性 属性名**

**文本 #text**

##### .nodeValue: 标识节点的值

**何时: 几乎不用**

**包括:**

document null

元素 null

属性 属性值

文本 文本内容

## DOM操作:

**查找触发事件的元素**

**->绑定事件**

**->查找要修改的元素**

**->修改元素, 添加新元素, 删除旧元素**

### 1. 查找: 4种:

#### 1. 不需要查找，可直接获得的元素:

**document.documentElement html**

**document.head head**

**document.body body**

#### 2. 按节点间关系查找:

##### 何时:

如果已经获得一个元素，找周围元素时

##### 节点树:

包含所有节点内容的完整树结构

###### 包括: 2大类关系:

1. 父子:

**.parentNode 父节点**

**.childNodes 直接子节点**

**.firstChild 第一个直接子节点**

**.lastChild 最后一个直接子节点**

2. 兄弟:

**.nextSibling 下一个兄弟节点**

**.previousSibling 前一个兄弟节点**

**问题:** 除了**parentNode**外，都受看不见的空字符影响

**解决: 元素树**

##### 元素树: 仅包含元素节点的树结构

**优:** 不受看不见的空字符的影响

**缺:** 1. 连正常的文字，也不包含

2. 兼容问题: IE9+

###### 何时:

只要操作网页内容，首选元素树

###### 包括: 2大类关系:

1. 父子:

**.parentElement 父元素**

**.children 直接子元素**

**.firstElementChild 第一个直接子元素**

**.lastElementChild 最后一个直接子元素**

2. 兄弟:

**.nextElementSibling 下一个兄弟元素**

**.previousElementSibling 前一个兄弟元素**

###### 说明:

元素树不是一棵新树，仅是节点树的一个子集

#### childNodes和children:

都返回动态集合(特殊的类数组对象)

#### 动态集合:

**不实际存储属性值，重复访问集合，都会重新查找DOM树**

**优:** 首次查找速度快！

**问题:** 反复访问集合，会导致反复查找DOM树

**解决:** 遍历时

**错误： for(var i=0;i<children.length;i++){ ... }**

**正确: for(var i=0,len= children.length;i<len;i++){ ... }**

#### 按HTML查找 : 4种

##### 1.按id查找 :

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

###### 强调 :

1.必须用document调用!

2.仅返回一个元素

###### 如果没找到 : 返回null

##### 2.按标签名查找:

###### var elems = parent.getElementsByTagName("标签名")

###### 强调 :

1. 返回动态集合

2. 可在任意父元素上调用!

3. 不但找直接子元素,且在所有后代中查找

###### 如果没找到 : 返回 空 集合

##### 3.按name查找 :

###### var elems = document.getElementsByName("name")

###### 强调 :

1.返回动态集合

2.必须用document调用!

###### 如果没找到 : 返回 空 集合

##### 4.按class属性查找

###### var elems = parent.getElementsByClassName("class")

###### 强调 :

1.返回动态集合

2.可在任意父元素上调用!

3.不但找直接子元素,且在所有后代中查找

4.不要求完全匹配所有class,只要其中一个class

###### 如果没找到 : 返回 空 集合

##### 问题:

每次只能按一个条件查找

如果查找条件复杂时，代码会很繁琐

##### 解决:

用选择器查找

#### 按选择器查找: 2种

##### 何时:

只要查找条件复杂时，都用按选择器查找

##### 1. 之查找一个元素

###### var elem = parent.querySelector("selector")

##### 2.查找多个元素

###### var elems = parent.querySelectorAll("selector")

##### 强调

###### 1.可在任意父元素上调用

###### 2.返回非动态集合 :

**非动态集合 :**

实际存储所有数据,即使反复查找,

也**不会导致反复查找DOM树**

**优点 :** 反复访问,也不会导致反复查找DOM树

**缺点 :** 首次查找,效率低

3.选择器受制于当前浏览器的兼容性要求

### 2.修改

#### 内容

##### 1.elem.innerHTML :

获取或设置元素开始标签到结束标签之间的HTML代码片段

##### 2.elem.textContent

获取或设置元素开始标签到结束标签之间的纯文本

**vs .innerHTML**: 1. 去掉所有内嵌的标签

2. 将转义字符转为正文

##### 3.elem.value

获取或设置表单元素的值

#### 属性

##### 1.标准属性: 2种

**title , style , class , id**

值是String类型

###### 1. 核心 DOM: 万能 , 繁琐

获取:

var value = elem.getAttribute("属性名")

设置:

elem.setAttribute("属性名","值")

移除:

elem.removeAttribute("属性名")

判断包含:

elem.hasAttribute("属性名")

###### 2. HTML DOM ※:

将所有标准属性,提前定义为对象的普通属性,

可用.直接访问.默认值为""

获取:

elem.属性名

设置:

elem.属性名="值"

移除:

elem.属性名=""

判断包含(判断有没有):

elem.属性名!=""

##### 特例 : class属性

###### 问题:

html中的class属性和js对象中的内部属性class冲突了

###### 解决:

elem.className->html中的class

elem.getAttribute("class")没问题

##### 2.三大状态

**disabled selected checked**

值是**bool**类型

**不能用 核心DOM改** , 只能用简化版HTML DOM

**补 :** **选择器 : :disable :select :checked**

##### 3.自定义扩展属性 :

###### data-\*

在程序中查找元素 : id class data-\*

id : 唯一 ; class: 专门定义样式,容易被改变

data-\* : 可选择多个元素,且只有程序使用

###### 何时 :

今后 , 只要为元素添加行为时,

都要用data-\*自定义扩展属性查找

###### 如何: 2步:

1. 定义属性时: <ANY **data-属性名="值"**>

2. 查找: 用**属性选择器: [data-属性名=值]**

**获取/修改: elem.dataset.属性名**

#### 样式 : 2种

##### 内联样式 :

直接写在开始标签的style属性中 , 仅归当前元素独有的样式

###### 优点 :

独有 , 优先级最高!

###### 缺点 :

不可重用!

###### 如何 :

获取/设置: elem.style.css属性名

强调:css属性名都要去横线变驼峰

ex : list-style-type ---> listStyleType

background-color -----> backgroundColor

###### 问题:

只能获得内联样式的属性，无法获得完整样式!

###### 所以，

elem.style通常不用做读取css属性值

仅用作修改css属性值: 独有，优先级最高

###### 如何读取:

获得计算后的样式: ComputedStyle

##### ComputedStyle

###### 什么是:

最终应用到元素上的完整样式

将相对单位的数值计算为绝对单位

###### 何时:

今后只要获得一个元素的任何样式，

保险起见，都要获得计算后的样式!

###### 如何: 2步:

1. 获得计算后的完整style对象

var style=getComputedStyle(elem)

2. 从style中获取想要的css属性值

style.css属性

强调:

计算后的样式是只读，不可更改！

##### 内部/外部样式: 了解

###### 1. 获得样式表对象:

var sheet=document.**styleSheets[i]**;

###### 2. 获得样式表中的cssRule对象

cssRule: 样式表中，每对儿{}，就是一个cssRule对象

var cssRule=sheet.**cssRules[i]**

###### 3. 获得cssRule中的style下的css属性值

var value=cssRule.style.css属性

##### 问题 :

每个style一次只能设置一个css属性,代码及其繁琐

##### 解决 : 2种

###### 1.style.cssText=""

用style字符串 , 覆盖现有style属性

**问题: 字符串拼接很复杂**

###### 2.用.className 批量修改一套样式 --- 最简单,最高效

### 3.添加和删除

#### 添加 : 3步

##### 1.创建空元素

**var a = document.createElement("元素名")**

##### 2.设置关键属性

a.href="#"

a.innerHTML = ""

......

问题 : 元素在DOM树外孤立存在,不会显示

##### 3.将新元素添加到DOM树

**parent.appendChild(a) //追加到父元素的末尾**

**parent.insertBefore(oldElem ,a) //追加到父元素下指定元素之前**

**parent.replaceChild(oldElem ,a) //替换父元素下的指定元素**

(待考证, oldElem 在前还是在后)

##### 优化 :

尽量减少操作DOM树的次数

##### 为什么 :

每操作一次DOM树,都会触发一次layout

反复操作DOM树 , 就会反复触发layout,降低效率

网页加载过程 :

html -> DOM Tree

↓

Render Tree -> layout -> paint

↑

css -> COM

layout : 计算每个元素的绝对布局位置

##### 如何 : 2种

###### 1. 同时添加父元素和子元素 :

先在内存中,将子元素添加到父元素,

再将父元素一次性添加到DOM树

###### 2. 同时添加多个平级子元素 :

**文档片段 : DocumentFragment**

**什么是 :** 内存中,临时存储多个元素的,虚拟的父元素

**何时 :** 只要同时添加多个子元素,

都要将子元素同时加入文档片段中,

再将文档片段整体加入DOM树

如何 : 3步

**1.创建文档片段**

var frag = document.createDocumentFragment();

**2.将子元素追加到文档片段中 :**

frag.appendChild(child)

**3.将文档片段整体追加到DOM树中**

parent.appendChild(frag)

**强调 :** frag将子元素添加到DOM树后,

**自动释放**,不会成为页面上实际的元素

#### 删除 :

parent.removeChild(child)

child.parentNode.removeChild(child)

## HTML DOM常用对象

### Image :

代表一个img元素

#### 创建 :

var img = new Image();

### Select

#### 属性

##### .value :

获得当前选中项的 value值

###### 强调 :

如果选项中没有value 属性

则用innerHTML代替

##### .options

获得当前select 下所有option对象的集合

.options.length 获取或设置option的个数

清除所有option : .options.length=0

.length : 等效于 .options.length

清除所有 option : .length=0

##### .selectedIndex : 当前选中项的下标!

#### 方法

.add(option) 代替了 appendChild(option|frag)

**.add不支持文档片段!**

**.remove(i) 移除第i个option**

#### 事件

.onchange 当选项发生改变时触发

### Option

代表select中的一个option元素

#### 创建 :

var opt = new Option(text, value)

### Table

代表一个table元素

#### 管着行分组

#### 添加 :

.createTHead() 返回 thead对象

.createTBody() 返回 tbody对象

.createTFoot() 返回tfoot对象

#### 删除 :

.deleteTHead()

.deleteTFoot()

#### 获取 :

.tHead

.tFood

.tBodies[i]

#### 行分组管着行 :

**添加 :** .insertRow(i) 返回新添加的tr对象

其中省略i , 表示末尾追加

i为0 , 表示开头添加

**删除 :**  .deleteRow(i)

强调 : i ,是要删除的行,在行分组内的相对下标

**获取 :** .rows

#### 行管着格

**添加 :** .insertCell() //返回新添加的td对象

**删除 :**  .deleteCell(i)

**获取 :** .cells

#### 删除行 : 2种

##### 1.行分组.deleteRow(i) i是在行分组内的下标

问题 : 无法快速获得行在行分组内的下标

##### 2.table.deleteRow(i) i是行在整个表中的下标

优 : 每行都有.rowIndex 属性,表示该行在整个表中的下标,用table删除行,可自动快速获得行的下标

总结 : table.deleteRow(tr.rowIndex)

### Form : 代表一个form元素

#### 获取 :

var form = document.forms[i/id]

#### 属性:

##### .elements:

获得表单中所有表单元素(input select button textarea)

##### .elements.length

获得表单中表单元素的个数

##### .length:

等效于 .elements.length

比如获得倒数第二个按钮:

form.elements[form.length-2]

#### 方法:

##### .submit() 手动提交表单

### Element 代表表单中一个表单元素

#### 获取:

form.elements[i/id/name]

更简单: 如果元素有name属性: form.name

#### 方法:

##### .focus() 让当前元素获得焦点

##### .blur()

## BOM: Browser Object Model

专门操作浏览器窗口的API——没有标准

### window: 2个角色:

#### 1. 代替ES标准中的global，充当全局作用域对象

#### 2. 封装所有DOM，BOM的API

**history: 历史记录**

**location: 地址栏**

**navigator: 配置信息**

**document: DOM中的document根节点**

**screen: 客户端显示设备的信息**

**screen.width**

**<768 手机 xs**

**<992 Pad sm**

**<1024 PC md**

**lg**

**event: 事件处理和事件对象**

### 1.打开和关闭窗口

#### window.open("rl","target")

#### name属性 :

每个窗口都有一个那么属性

在内存中唯一标识一个窗口

浏览器要求, 相同name属性的窗口只能打开一个

后打开的新窗口会覆盖先打开的同名窗口

#### 其实 :

open的第二个参数和a元素的target属性值,都是name属性

#### 三种情况 :

##### 1.如果使用自定义的name值 , 则在新窗口打开,只能打开一个

##### 2.如果使用 \_self , 则新窗口替换当前窗口 :

\_self的意思是,使用当前窗口自己的name值,打开新窗口

##### 3.如果使用\_blank , 则在新窗口打开 , 可随意打开多个

\_blank的意思是,不指定name值,浏览器会随机分配name值,几乎不会重复

#### window.close()

### 2.history 和 location

#### history :

保存当前窗口打开后,成功访问过的url的历史记录栈

##### 如何 : history.go(n)

.go(1) 前进一次

.go(2) 前进两次

.go(0) 刷新

.go(-1) 后退一次

.go(-2) 后退两次

#### location

##### 属性:

.href: 获得完整url

.protocol: 协议

.host: 主机名+端口号

.hostname: 主机名

.port: 端口号

.pathname: 相对路径

.hash: 锚点地址

.search: 查询字符串

打开新链接, 可以后退:

location.assign("url") <==> location.href="url"

<==> open("url","\_self")

其实可以更简单: location="url"

**打开新链接，禁止后退:**

##### location.replace(url)

用新的url代替history中当前url记录

**---禁止后退**

##### 刷新:

###### 普通刷新:

优先从本地缓冲获取文件，

如果缓存没有或过期，才从服务器下载新文件

history.go(0)

F5

location.reload(/\*false\*/) 不强迫

###### 强制刷新:

跳过本地缓存，始终从服务器获取新文件

**location.reload(true) 强迫**

### 3. \*\*\*定时器: 2种:

#### 1. 周期性定时器:

什么是: 让程序每隔一段时间间隔，自动反复执行一项任务

何时: 让程序每隔一段时间间隔，自动反复执行一项任务时

##### 如何: 3件事:

###### 1. 任务函数: 定时器反复调用的函数

###### 2. 启动定时器: var timer=setInterval(task,ms)

创建一个新定时器对象，每隔ms毫秒，反复自动调用task函数

其中: 返回定时器序号timer，专用于停止定时器

###### 3. 停止定时器: clearInterval(timer)

**问题:** 不能自动清除timer中残留的序号

**强烈建议:** **手动timer=null**

2中方式:

**1. 用户手动操作**

**2. 自动停止:**

在任务函数中设定临界值

如果到达临界值，则自动调用clearInterval

#### 2. 一次性定时器:

##### 什么是:

先等待一段时间，再自动执行一次任务，执行后自动停止

##### 何时:

延迟执行一项任务时

##### 如何: 3件事:

###### 1. 任务函数:

###### 2. 启动定时器: var timer=setTimeout(task,ms)

先等待ms毫秒，再执行一次task，执行后自动停止

###### 3. 停止定时器: clearTimeout(timer)

**建议: timer=null**

#### 定时器的原理:

setXXX(task,ms)

在主程序之外，创建定时器对象，保存回调函数task

定时器对象根据ms，决定何时自动调用task函数

定时器调用task函数时，无权立刻直接调用task，必须先将task加入回调队列中等待。

等待主程序全部执行完，回调队列中的函数，才能执行

### 4. \*\*\*event

#### 什么是事件:

浏览器自动触发的，或用户手动触发的页面状态改变

#### 什么是事件处理函数:

当事件发生时，自动调用的函数

#### 绑定事件处理函数:

将一个函数，赋值给元素的事件属性

当事件发生时，自动调用函数

#### 如何绑定: 3种:

##### 1. 在HTML中:

**<ANY on事件名="js语句"**

**问题:** 不符合内容与行为分离的元素，不便于维护和重用

##### 2. 在js中, 用赋值方式绑定:

**elem.on事件名=function(){**

**this->当前触发事件的元素elem**

**}**

**问题:** 一个事件属性，只能绑定一个处理函数

##### 3. 在js中, 用addEventListener

**elem.addEventListener("事件名",fun)**

**优:** 一个事件属性，可同时添加多个事件监听

**解除绑定:** **elem.removeEventListener("事件名",fun)**

**强调:** 如果处理函数可能解绑，就必须用有名称的函数来绑定。解除时，才能用函数名解除绑定。

因为，解除绑定时，要求，必须指定原函数对象

#### 事件模型: 3个阶段:

##### 1. 捕获:

由外向内，

依次记录各级父元素上绑定的相同事件的处理函数

##### 2. 目标触发:

优先触发目标元素上绑定的处理函数

**目标元素:** 最初，实际触发事件的元素

##### 3. 冒泡:

由内向外，按照捕获阶段记录的反向，

依次触发各级父元素上绑定的处理函数

#### 事件对象:

##### 什么是:

事件发生时，自动创建的，

封装事件信息，并提供修改事件行为的API的对象

##### 何时:

只要希望获得事件中的信息，

或希望修改事件的默认行为时

##### 如何:

###### 获取:

**默认，事件对象，总是作为处理函数的第一个参数自动传入！**

**function 处理函数(e){**

**e->事件对象**

**}**

###### API:

**取消冒泡/阻止蔓延: e.stopPropagation()**

**利用冒泡:**

**优化:** 尽量减少事件监听的个数:

**为什么:** 浏览器触发事件靠遍历，事件监听越多，

遍历时间越长，导致响应速度变慢

**如何:** 利用冒泡:

**如何利用:** 如果多个平级子元素，要绑定相同的事件时，

其实，只要在父元素绑一次，所有子元素，就可共用！

**为什么:** 冒泡

##### 2个难题:

###### 1. 获得目标元素:

**错误:** this, this已经指父元素，不再指子元素

**正确:** **e.target** 始终记录着最初点击的目标元素

###### 2. 目标元素不一定是想要的:

在处理函数中，

先判断当前目标元素的特征是否符合要求，

再决定是否继续执行。

###### 阻止默认行为: e.preventDefault();

**何时:** 当事件具有的默认行为，不想要时

典型:

1. a元素作为按钮时，会自动在地址栏添加#xxx锚点地址

2. 如果验证未通过，则阻止表单自动提交

form.onsubmit事件:

在无论以任何方式正式提交表单之前，自动触发。

3. H5中做拖拽之前，必须阻止浏览器默认的拖拽行为

#### 鼠标坐标:

事件发生时，鼠标的坐标位置

**如何: 3组:**

##### 1. 相对于屏幕左上角: e.screenX, e.screenY

##### 2. 相对于文档显示区左上角: e.clientX, e.clientY

##### 3. 相对于所在元素左上角: e.offsetX, e.offsetY

#### 页面滚动:

用户手动滚动页面:

**window.onscroll=function(){**

**document.body.scrollTop//获得页面滚动过的高度**

**}**

用程序控制页面滚动:

**window.scrollTo(left,top)**

**window.scrollBy(left增量,top增量)**

### 5. navigator:

#### 什么是:

封装浏览器配置信息的对象

#### 何时:

只要获得浏览器名称，版本号，插件等配置信息时

#### 如何:

**.cookieEnabled: 判断是否启用了****cookie**

##### 什么是cookie:

客户端，持久存储用户私密信息的小文件

##### 为什么:

程序中的数据，都是内存中临时存储的

程序关闭，内存释放，数据丢失

##### 何时:

只要希望在客户端持久保存的数据，

就用cookie

设置:

设置->高级->隐私...->内容设置->cookie

##### 检查浏览器是否禁用cookie:

**if(navigator.cookieEnabled)**

**document.write(**

**"cookie已启用，请妥善保存个人信息！<br>");**

**else**

**document.write(**

**"cookie已禁用，无法记住密码！<br>");**

#### .plugins: 封装插件信息的集合

##### 什么是:

为浏览器添加新功能的小软件

##### 如何判断是否已经安装指定插件:

#### .userAgent: 保存浏览器名称，版本号信息的字符串

##### 何时:

只要希望获得浏览器的名称和版本号时

##### 如何:

**function checkPlugins(piName){**

**if(navigator.plugins[piName])**

**document.write(**

**"已安装"+piName+"插件<br>");**

**else**

**document.write(**

**'未安装'+piName+'插件,<a href="#">点此下载</a><br>');**

**}**

**checkPlugins("Shockwave Flash");**

**checkPlugins("QQ MUSIC");**

**var browser="unknow",ver=0,**

**ua=navigator.userAgent;**

**document.write(ua+"<br>");**

**if(ua.indexOf("MSIE")!=-1)**

**browser="MSIE";**

**else if(ua.indexOf("Firefox")!=-1)**

**browser="Firefox";**

**else if(ua.indexOf("OPR")!=-1)**

**browser="Opera";**

**else if(ua.indexOf("Chrome")!=-1)**

**browser="Chrome";**

**else if(ua.indexOf("Safari")!=-1)**

**browser="Safari";**

**else if(ua.indexOf("Trident")!=-1){**

**browser="IE"; ver=11;**

**}**

**if(browser!="unknown"||browser!="IE"){**

**var i=ua.indexOf(**

**browser=="Opera"?"OPR":browser);**

**var len=browser=="Opera"?3:browser.length;**

**ver=parseFloat(ua.substr(i+len+1,3));**

**}**

**document.write(**

**"浏览器名称:"+browser+"<br>版本号:"+ver);**

# jQuery

## 什么是jQuery

### 什么是 :

第三方开发,执行DOM操作的,极简化的函数库

### 为什么 : 2大原因

#### 1.对DOM操作的终极简化 :

##### 简化了四大操作 :

###### 1.增删改查

###### 2.事件绑定

###### 3.动画效果

###### 4.Ajax请求

#### 2.屏蔽了绝大多数兼容性问题 :

凡是jQuery让用的,都没有兼容性问题

### 何时 :

今后,只要执行DOM操作,都用jQuery

现在企业中的大型项目,以及前端框架都是用jQuery实现的

## 如何使用

### 1.下载:

#### 版本:

##### 1.x : 兼容旧浏览器

##### 2.x/3.x : 不再兼容就浏览器,代码少,体积小,功能丰富

#### 未压缩 :

有注释,代码格式,有意义的变量名

##### 何时 :

学习时和开发阶段,可读性好

#### 压缩:

删除注释,删除格式,极简化变量名

##### 何时 :

部署到生产环境中,体积小,下载快!

### 2.加载:

先引入jquery.js,再编写自定义代码

#### 如何 : 2种:

##### 1.引入本地的jquery.js;<script src="js/jquery.js"><script>

##### 2.引入CDN网络中的jquery.js

### 3.script到底做了什么

#### 1.在浏览器中添加了一种新的类型:jQuery

##### 构造函数 : 创建jQuery对象

###### 何时 :

今后只要使用jQuery简化版API,

都要先创建jQuery对象

###### 如何 : 不用 new! 2种

1.通过查找创建 : var $jq=jQuery("选择器")

2.可直接封装一个DOM元素:

var $elem=jQuery(elem)

其实,都可简写为$(...)

###### 什么是 : jQuery对象:

封装DOM对象,

并提供操作DOM对象的简化版API的**类数组对象**.

###### 为什么封装DOM对象:

因为只有jQuery类型的子对象,

才能使用jQuery提供的简化版API

DOM对象默认不是jQuery类型的子对象,

所以无法直接使用jQuery的简化版API

###### 强调 :

尽量少的使用$查找和创建jQuery对象

1.**节约内存** : 每调用一次$(..),都会创建新的jQuery对象

2.**效率** : 每$(...)都要查找

##### 原型对象 :

**原型对象 : jQuery.fn = jQuery.prototype**

是所有jQuery对象的公共父对象

保存所有jQuery类型提供的简化版API

### 4.jQuery : API特点:

#### 1.一个函数两用(重载):

如果不提供参数,表示获取

提供参数,表示修改

#### 2.自带遍历效果 :

对jQuery对象集合调用API,

**等效于**,

对集合中每个DOM元素分别调用API

#### 3.每个API几乎都返回正在操作的jQuery对象

**问题 :** 对一个元素,反复执行多个操作

**解决 :** jQuery中可用链式结构

**$(...).操作1().操作2()....**

**为什么:** 每个API几乎都**返回**正在操作的jQuery对象

**优: 1. 减少查找的次数**

**2. 减少变量的个数**

### 学习jQuery还是在学DOM操作 : 只是API简单而已

查找出发事件的对象

绑定事件

查找要修改的元素

修改元素(内容,属性,样式)

添加/删除/替换/克隆

## 查询

### 1.用选择器查找:

jQuery支持所有CSS选择器

只扩展了一小部分而已

**使用 : 优先**使用CSS选择器:

**为什么 :** css选择器由内容排版引擎执行,效率高

jQuery扩展的选择器由程序模拟实现,效率低

#### 基本本选择器 : 同 CSS

**#id .class element \* selector1,selector2,....**

#### 层次选择器 : 同CSS

**parent child parent>chlid child1+child2 child1~** **siblings**

#### 子元素过滤: 同CSS

**:first-child :last-child :nth-child(i) :only-child**

##### 特点:

###### 1. 在每个父元素下，分别编号排列

###### 2. 序号从1开始

#### 基本过滤:(位置过滤)

**:first/last :even/odd :eq/lt/gt(i)**

##### 特点:

1. 先取出所有符合条件的元素组成集合，

再在一个大的集合中统一编号排列

2. 编号从0开始

#### :not(selector) 同CSS

#### 属性过滤: 同CSS

**[属性名] [属性名=值]**

**[属性名^=值] [属性名$=值] [属性名\*=值] [属性名!=值]**

#### 内容过滤:

##### 按文本内容:

:contains(text) 选择内容中包含指定text内容的父元素

##### 按子元素特征:

:has(selector)

选择子元素中包含具有selector特征的子元素的父元素

##### 按是否为空:

**:empty** 选择空元素

**:parent** 选择非空元素

##### 可见性过滤:

**:hidden :visible**

**:hidden** 只能选择: **display:none**的和**type="hidden"**的

不能选择: visibility: hidden的和opacity:0的

#### 表单元素过滤:

**:input** 选择所有表单元素,包括: **input select textarea button**

其实，每种type都对应一种选择器:

**:text :password :submit :reset :button**

**:file :checkbox :radio :hidden :image**

#### 状态过滤: 同CSS

**:enabled :disabled :checked :selected**

### 2.按节点间关系查找:

#### 何时:

已经获得一个元素，找周围元素时

#### 包括: 2大类

##### 1. 父子:

###### $(...).parent()

=> .parentNode

###### $(...).children(["selector"])

=>.children 只找直接子元素

###### $(...).children().first()

=> $(...).children(":first-child")

=> .firstElementChild

###### $(...).children().last()

=> $(...).children(":last-child")

=> .lastElementChild

###### $(...).find("selector")

找**所有后代**元素中符合selector条件的子元素

##### 2. 兄弟:

###### 相邻的一个兄弟:

**$(...).next/prev()**

###### 之前的/之后的所有兄弟:

**$(...). prevAll/nextAll ()**

###### 除我之外所有兄弟:

**$(...).siblings()**

## 修改: 都是一个函数两用

### 内容: 3种:

#### $(...).html(["html代码片段"]) => .innerHTML

#### $(...).text(["文本"]) => .textContent

#### $(...).val(["表单元素值"]) => .value

#### 清空: $(...).empty() => .innerHTML="";

### 属性:

#### 标准属性:

**$(...).attr("属性名"[,"属性值"])**

#### 状态:

**$(...).prop("状态名"[,bool])**

#### 自定义扩展属性:

##### $(...).data("属性名"[,"属性值"]) => .dataset

##### 强调:

自定义扩展属性名不用加data-前缀

### 样式:

#### 获取/修改具体css样式:

##### 仅修改一个css属性值:

**$(...).css("css属性名"[,"属性值"])**

**强调:** 数值不用加px

##### 同时修改多个css属性值:

**$(...).css({**

**css属性名: 属性值,**

**... : ...**

**})**

**强调:** 用.css修改属性时，自动执行.style.属性名,修改内联样式。

用.css 修改属性值时，自动执行getComputedStyle，

获取计算后的完整样式。

#### 问题: 用css样式直接修改，代码繁琐，不便于维护

#### 解决: 用class批量修改元素的样式

##### 如何:

###### $(xxx).hasClass("class")

**判断**元素是否包含该class

###### $(xxx).addClass("class")

仅为元素**添加**class，而不是完整替换

###### $(xxx).removeClass("class")

仅为元素**移除**指定class,不影响元素上其它class

###### $(xxx).toggleClass("class")

在有没有class之间做切换

=> if(.hasClass("class")) .removeClass("class")

else .addClass("class")

## 添加/删除/替换/克隆

### 添加: 2步:

#### 1. 用HTML代码片段创建DOM子树或单个元素

**var $ new =$("html代码片段")**

#### 2. 追加到DOM树:

##### 开头/末尾插入:

###### $(parent).append/prepend($new);

将**右边**的**新元素**，添加到**左边**的**父元素**中，**不**返回新元素

###### var $new=$ new.appendTo/prependTo($parent);

将**左边**的**新元素**，添加到**右边**的**父元素**中，**可返回新元素**， 就可继续对新元素做后续操作。

##### 中间插入:

**$(child).before/after($new)**

### 删除: $(...).remove();

### 替换:

#### $(...).replaceWith($new)

用**右边**的**新元素**，**代替左边**找到的**旧元素**

#### var $new=$(new).replaceAll($old)

用**左边**的**新元素**，**代替右边**找到的**旧元素**

#### 其实:

所有的$new的位置，都可换为一段代码片段

### 克隆:

#### $(...).clone();

浅克隆: 仅复制样式和属性，不复制行为

#### $(...).clone(true);

深克隆: 即复制样式和属性，**又复制行为**

## 事件绑定:

鄙视: jQuery共几种事件绑定方式:

### 1. .bind/unbind

**仅为简化addEventListener/removeEventListener**

#### unbind: 3种重载:

##### 1. unbind("事件名",原处理函数对象)

仅移除当前元素的当前事件上的一个指定处理函数

等效于removeEventListener

##### 2. unbind("事件名")

移除当前元素的当前事件上绑定的所有处理函数

##### 3. unbind() 移除当前元素上的所有事件

### 2. .one 仅触发一次事件，触发后，自动解绑

### 3. .live/die 已废弃

### 4. .delegate/undelegate 其实就是利用冒泡！

**$(parent).delegate("selector","事件名",fn)**

将事件绑定在parent上，利用冒泡

仅限于符合selector条件的子元素触发事件

fn中的this自动指向目标元素

鄙视: **bind vs delegate**

#### 1. 绑定的元素不同:

bind绑在每个子元素上

delegate 绑在父元素上

#### 2. 添加的监听个数不同:

bind 每个子元素都添加一个监听——多

delegate 仅在父元素添加一个监听——少

#### 3. 动态添加子元素时:

.bind 动态添加的新元素，无法自动获得事件处理函数

delegate 动态添加的新的子元素，

自动可使用父元素上绑定的处理函数

### 5. .on:

既可以代替bind: .on/off("事件名",fn)

又可以代替delegate: .on/off("事件名","selector",fn)

### 6. 常用事件: 都提供了简写:

**.事件名(fn)**

### 页面加载后触发: 2个时机

#### 1. 仅DOM内容加载完成，就提前触发: (html, js)

##### 事件:

DOMContentLoaded

##### 何时:

今后，凡是和样式，图片无关的初始化操作(事件绑定)，

都必须在DOMContentLoaded中提前执行

##### 问题: 兼容性

##### jQuery: $(document).ready(fn)

其实可以简写: $().ready(fn)

其实可以**更简写**: **$(fn)**

#### 2. 必须等待所有页面内容加载完成，才触发: (html,css,js,图片)

##### 事件:

onload

##### 何时:

如果依赖于css或图片才能执行的程序，

只能放在onload中，稍后执行

### 鼠标事件:

#### mouseover mouseout :

进出子元素，**同样冒泡触**发父元素的进出事件处理函数

#### mouseenter mouseleave :

进出子元素，**不会冒泡**触发父元素的进出事件处理函数

#### 其实，还可更简化:

如果同时绑定enter和leave时，可用一个hover代替

##### $(...).hover(fn1,fn2)

fn1绑定给mouseenter，fn2绑定给mouseleave

#### 其实还可以更简单:

如果fn1和fn2可统一为一个处理函数，则只需出入一个即可。一个函数即绑定给mouseenter，又绑给mouseleave

### 模拟触发:

用程序模拟触发元素上绑定的事件处理函数

#### 如何:

**$(...).trigger("事件名")**

#### 其实可以更简单:

**$(...).事件名()**

## 其他:

### \*\*\*var i=$(...).index() 查找当前元素是其父元素下第几个

### 而且关系: selector1selector2...

### \*\*\*$(...).is(selector) 判断当前元素是否符合selector的条件

## 动画:

### 简单动画: 3种固定效果:

#### 1. 显示隐藏: $(...).show/hide/toggle();

##### 不带参数:

默认使用**display:block/none瞬间**显示/隐藏

今后，如果只需要瞬间显示隐藏时，

**就可用show/hide代替display:block/none**

##### 带参数:

**$(...).show/hide(ms);** 才能显示过渡效果

#### 2. 上滑下滑: $(...).slideUp/slideDown/slideToggle();

#### 3. 淡入淡出: $(...).fadeIn/fadeOut/fadeToggle();

#### 问题:

##### 1. 效果是写死的，无法订制

##### 2. 用程序的定时器实现的，效率低

#### 解决:

今后，凡是有规律的动画，都用**transition**

### 万能动画:

#### $(...).animate()

$(...).animate({

css属性:目标值,

... : ... ,

},ms)

##### 强调:

只能对单个数值的属性，应用动画效果

#### 排队和并发:

##### 1. 排队:

对同一个元素，反复调用动画函数，多个效果是排队依次执行。

##### 2. 并发:

同一个animate内的多个css属性，是同时变化的

#### 动画结束后执行:

每个动画函数的最后一个参数，都是一个回调函数，

在动画结束后，自动执行善后操作。

#### 停止动画: $(...).stop()

##### 问题:

默认仅停止队列中正在播放的当前动画

之后的动画依然正常执行

##### 解决: $(...).stop(true)

停止动画，并清空队列

### 查找/判断元素是否正在播放动画: 选择器: :animated

### 延迟执行:

#### $(...).delay(ms).动画函数

先延迟等待ms，再播放动画

## 类数组对象操作:

### $(...).size() => .length

### var dom=$(...).get(i) => [i]

### $(...).each((i,elem)=>{ ... }) => .forEach((val,i,arr)=>{ ... })

### var i=$(...).index() => .indexOf(elem,fromi)

## 添加自定义jQuery函数

### 何时:

只要jQuery函数不够用或现有jQuery函数不好用

### 如何: 2步:

#### 1. 验证: 是否先引入了jQuery

#### 2. 向jQuery.fn中添加自定义函数:

#### 强调: this-> 将来调用该函数的jQuery对象

## 插件: plugins

### 什么是:

拥有独立样式，功能的小程序

### 为什么:

项目中，很多功能是大量重复的！

### 何时:

只要发现反复使用的功能，都要封装为一个独立的插件，再反复使用插件。

### 如何: 3种:

#### 1. jQuery官方: jQueryUI

#### 2. 自定义插件

#### 3. 第三方流行插件

## jQueryUI:

jQueryUI是jQuery官方提供的基于jQuery函数库开发的效果和插件的集合。

### 如何:

#### js:

**jquery-ui.js**

#### css:

#### html: 3步

##### 1. 引入插件的js和css:

<link href="jquery-ui.css" rel="stylesheet">

<script src="jquery-1.11.3.js">

<script src="jquery-ui.js">

##### 2. 按插件规定，编写HTML内容和结构:

##### 3. 找到插件的父元素，调用插件函数

##### 可选:

自行补齐插件需要的class样式

### 1. 效果:

#### 1. 重写了addClass/removeClass，可定义过渡时间和回调函数

#### 2. 重写了animate，支持颜色动画

#### 3. 重写了传统动画函数，添加了额外的动画特效

### 2. Widgets: 拥有独立样式和功能的小程序

#### 如何:

**引入: css,jquery,jqueryui**

按部件规定的结构定义内容

##### 1. accordion:

html:

<div id="#xxx">

<ANY>标题</ANY>

<ANY>内容</ANY>

<ANY>标题</ANY>

<ANY>内容</ANY>

...

</div>

js: $(...).accordion();

##### 2. autocomplete:

html: <input/>

js: $(...).autocomplete({

source: 本地数组|"xxx.php"

})

php:

$kw=$\_REQUEST["term"]

...

echo 数组

##### 3. datepicker:

html: <input />

js: $(...).datepicker({

changeYear: true,

changeMonth: true,

dateFormat: "yy年mm月dd日",

monthNamesShort: [ "一月", "二月", ..., "十二月" ],

dayNamesMin: [ "日", "一", ..., "六" ],

... ...

})

vs HTML5 的Date

<input type="date"

问题: 在浏览器中写死的样式，不能定制

兼容性问题

##### 4. dialog:

html:

<div title="对话框标题">

<form>...</form>

</div>

js: $(...).dialog({

autoOpen:false,

modal:true,

buttons:[

{ text:"登录",click (){

$("#login").submit();

//$("#dialog-login").dialog("close");

}}

]

})

打开对话框: $("#dialog-login").dialog("open");

##### 5. tabs:

html:

<div>

<ul>

<li><a href="#对应div的id">文本

...

</ul>

<div id="a中对应的id">标签也内容

...

</div>

##### 6. toolTip: 工具提示,代替默认title效果

$(...).tooltip({

tooltipClass:"class名"

})

### 问题:

封装的太死, 且很多效果用程序实现，所以极其不便于定制。

### 解决: bootstrap:

bootstrap中定义了现成的class和行为，但是没有和具体元素绑定

由开发人员自行指定元素使用的class和行为

如果开发人员觉得boot中的class不是想要的，随时可定义同名class替换

### 2. 自定义插件:

#### 前提:

必须先用普通的HTML，css，js已经实现了想要的功能和效果。

#### 如何:

##### 1. 将css，提取到专门的css文件中保存

在css中，尽量不要使用id或元素选择器，尽量用class解决一切问题

##### 2. 按插件规定，定义HTML内容结构

##### 3. 为HTML内容添加样式和行为: 2种:

###### 1. jQuery UI: 自动侵入式

开发人员不用手动添加class和行为，一切都由插件函数自动添加到HTML元素上

问题: 侵入靠程序实现，不便于维护

###### 2. bootstrap:

样式: 开发人员手动添加class

行为: 开发人员手动添加data-\*扩展属性，自动赋予

优点: class和data-\*都由开发人员手动定义，灵活，便于维护

##### 如何使用:

###### 1. 引入css

###### 2. 按插件规定编写HTML内容和结构

###### 3. 引入jquery和插件的js

###### 可选: 调用插件函数

## 流行的第三方插件:

**jQuery Form:**

**jQuery validation:**

**jQueryFileUpload**

**jWysiwyg?**

## jQuery的ajax封装:

### $(parent).load("url",[data],[fn])

### 何时:

多个页面加载统一的页头和页脚

异步请求一段HTML代码片段，自动填充到parent中

### 强调:

#### 1. 不是追加,而是替换！ ($(parent).html(xhr.responseText))

#### 2. 有data参数，自动post，否则，默认都是get

### $.get("url",[data],[fn])

#### 何时:

专门用于向服务器端发送get请求并处理响应结果

回调函数fn: 用于接收响应，处理响应结果

#### 3个参数:

**data:** 接收服务器端返回的数据

如果服务器端定义header为json，则get会自动调用JSON.parse，将json字符串转为客户端对象

如果服务器端未定义header为json，则get只会拿到json字符串，还需要手动转换。

**总结:** php中必须显式指定header

"success": 总是success

**xhr:** 原生ajax的xhr对象

### $.getJSON("url",[data],[fn]) 同$.get()

无论服务器是否指定消息类型为json,都强行调用JSON.parse转化服务器响应结果。

### $.getScript("url",[data],[fn])

获取服务器端一段js代码，到客户端执行

### $.post("url",[data],[fn])

$.ajax({

url:"url",

type:"GET",//"POST"

data:"k1=v1&k2=v2&...."或{k1:v1,k2:v2,...},

success(data){

}

})

### $.ajax({})

**$.ajax({**

**type:"GET",//POST**

**url:"xxx.php",**

**data: "k1=v1&k2=v2&..."或{k1:v1,k2:v2,...}或$(form).serialize()**

**dataType: "服务器响应消息的类型",**

**//text/script/json/xml/jsonp**

**beforeSend:fn,**

**success:fn,**

**error:fn,**

**complete:fn,**

**})**

xhr.onreadystatechange=function(){

if(xhr.readyState==4){

if(xhr.status==200){

success()

}else{

error()

}

complete();

}

}

xhr.open();

beforeSend();

xhr.send();

### 总结:

如果服务端php规定了header，则get可代替getJSON和getScript

## \*\*\*跨域请求

### 什么是:

一个域名下的网页，想访问另一个域名下的资源

### 比如:



### 总结:

协议，主机名，端口号，二级域名不同都算跨域

### 浏览器认为:

多数由程序发起的跨域请求，是非法的。禁止跨域

### 但:

不由程序发起的跨域请求，是允许的:

### 包括:

img, link, script, iframe

### 不允许跨域:

xhr

### jQuery简写跨域:

#### 1. $.getJSON("xxx.php?callback=?",函数对象)

#### 2. $.ajax({

$.ajax({

type:"GET",

url:"xxx.php",

success:doResponse,

dataType:"jsonp"

})

服务器: $fn=$\_REQUEST["callback"];//必须是callback

...

echo "$fn('$json')"

# HTML5新特性(十个新特性-凌乱)

## (1)新的语义标签

## (2)增强表单(表单2.0)

### (1)新的input type <input type=? >

H4:text;password;submit;button;radio;checkbox;

file;hidden;image;

H5:**email; url; number; tel; range; color; date; month; week**

### (2)新的表单元素

H4:input;button;select/option;textarea

H5:**datalist; progress; meter; output**

### (3)新的表单属性

H4:name;value;class;readonly;style;disabled;id;checked;selected

**H5:**

#### (1) placeholder:占位字符

<input type="text" placeholder="请输入用户名" />

(2)autofocus: 自动获取输入焦点

<input autofocus>

#### (3)multiple:允许输入框中出现多个输入值(用逗号分隔)

如:邮箱输入域

<input type="email" multiple/>

#### (4)form 用于把输入域放置到form外部

<form id="f5"></form>

<input form="f5" />

===输入验证相关的新属性====

#### (5)required:必填项,内容不能为空

<input required>

#### (6)maxlength:指定字符串最大长度

<input maxlength="9">

#### (7)minlength: 指定字符串最小长度

<input minlength="6">

#### (8)max:指定数字的最大值

<input type="number" max="100" min="19" >

#### (9)min:指定数字的最小值

<input type="number" max="100" min="19" >

#### (10)pattern:指定输入必须符合正则表达式

<input type="text" pattern="1[3578]\d{9}"

### (4)新的表单元素--datalist

<datalist id="list3"> datalist 本身不可见

<option>XX</option>

<option>YY</option>

</datalist>

<input type="text" list="list3" />

datalist为 input提供**输入建议列表**，用户可以从中选取，也可以手工输入.

### (5)新的表单元素--progress

显示一个进度条有两种形式  
 <progress></progress>左右晃动的进度条  
 <progress value="0.7"></progress>具有指定进度值的进度条

### (6)新的表单元素--meter

meter :刻度尺 用于标识一个值所处的范围，不可接受(红色).可接受(黄色),非常优秀(绿色)

<meter min="可取最小值" max="可取最大值" low="合理下限值" high="合理上限值" optimum="最佳值" value="当前值" ></meter>

### (7)新的表单元素--output

output:输出，语义标签，没有任何外观样式，样式等同于span

商品单价: 3.5

购买数量:<input type="number" value="1">

小计:<output>3.5</output>

8: 新的表单新属性

H4:name;value;class;readonly;style;disabled;id;checked;selected

H5:

(1) placeholder:占位字符

<input type="text" placeholder="请输入用户名" />

(2)autofocus: 自动获取输入焦点

<input autofocus>

(3)multiple:允许输入框中出现多个输入值(用逗号分隔)

如:邮箱输入域

<input type="email" multiple/>

(4)form 用于把输入域放置到form外部

<form id="f5"></form>

<input form="f5" />

===输入验证相关的新属性====

(5)required:必填项,内容不能为空

<input required>

(6)maxlength:指定字符串最大长度

<input maxlength="9">

(7)minlength: 指定字符串最小长度

<input minlength="6">

(8)max:指定数字的最大值

<input type="number" max="100" min="19" >

(9)min:指定数字的最小值

(10)pattern:指定输入必须符合正则表达式

<input type="text" pattern="1[3578]\d{9}"

## (3)音频和视频(好玩)

### 1.HTML5的新特性之二-视频/音频

Flash绘图(AS/Flex) ==> Canvas/SVG

Flash动画 ==> 定时器+Canvas

Flash视频/音频 ==> VIDEO/AUDIO

Flash客户端存储 ==> WebStorage

### 2.HTM5新特性--视频和音频

h5提供一个新标签用于播放视频

<video src="res/birds.mp4"></video>

<video>

<source src=" res/birds.mp4"></source>

<source src=" res/birds.avi"></source>

<source src=" res/birds.ogg"></source>

</video>

它本身是一个300\*150的inline-block元素

#### 成员属性

autoplay:false 是否自动播放

controls:false 是否显示播放控件

loop:false 是否循环播放

muted:false 是否静音

poster:"" 在播放第一帧之前显示的海报

preload: 视频的预加载策略

auto 预加载视频的元数据以及缓冲一定时长

metadata 预加载视频的元数据(时长，第一帧内容)

none 不预加载任何数据

#### ===js对象属性===

currentTime: 当前播放时间

duration: 总时长

paused:true 当前视频是否处理暂停状态

volume:1 音量

playbackRate:1 播放速度

#### ====js对象方法===

play() 播放视频

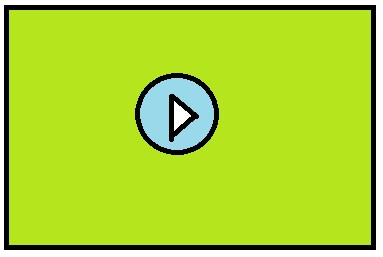
pause() 暂停播放

#### ====js事件===

onplay: 当前视频开始播放时触发事件

onpause: 当前视频暂停触发事件

练习1:不使用video 自带的constrols自定义播放/ 暂停按钮



鼠标移出视频区域，隐藏按钮,鼠标移入，显示按钮

练习2：不论何种原因，只要视频暂停，就隐藏广告

### 3.HTM5新特性--音频

H5提供了一个新标签用于播音频

<audio src="res/m.mp3">您的浏览器不支持音频播放</audio>

<audio>

<source src="res/m.mp3"></source>

<source src="res/m.wav"></source>

您的浏览器不支持音频播放

</audio>

它默认300\*30的inline-block元素;若没有controls属性，则display:none

#### 成员属性

autoplay:false 是否自动播放

controls:false 是否显示播放控件

loop:false 是否循环播放

muted:false 是否静音

preload: 视频的预加载策略

auto 预加载视频的元数据以及缓冲一定时长

metadata 预加载视频的元数据(时长)

none 不预加载任何数据

#### ===js对象属性===

currentTime: 当前播放时间

duration: 总时长

paused:true 当前视频是否处理暂停状态

volume:1 音量

playbackRate:1 播放速度

#### ====js对象方法===

play() 播放音频

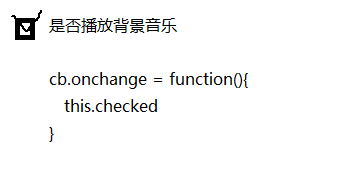
pause() 暂停播放

#### ====js事件===

onplay: 当前音频开始播放时触发事件

onpause: 当前音频暂停触发事件

练习:使用复选框控制网页背景音乐



## (4)Canvas绘图----重点&难点

网页中可用绘图技术 ：走势图，统计图，网页游戏，地图应用都要使用绘图技术，有三种绘图技术可以选择.

(1)SVG 绘图:2D矢量图技术,2000出现，后纳入H5标准

(2)Canvas绘图:2D位图技术,H5提出绘图技术

(3)WebGL绘图:3D绘图技术，尚未纳入H5标准

|  |
| --- |
| Canvas绘图难点二个  1:坐标系  2:单词比较多 |

### 1.H5新特性-Canvas绘图

canvas:画布，是H5提供2D绘图技术

<canvas width="500" height="400">

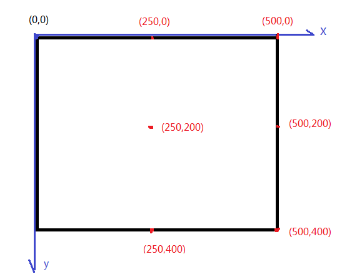
</canvas>

canvas标签在浏览器默认300\*150的inline-block.

画布宽度和高度只能使用html5属性来赋值，不能用css样式赋值.

每个画布上有且只有一个"画笔"对象.

var ctx = canvas.getContext("2d");



#### (1)使用canvas绘制矩形

矩形的定位点在自己左上角

ctx.lineWidth 描边宽度

ctx.fillStyle 填充样式(颜色)

ctx.strokeStyle 描边样式(颜色)

ctx.fillRect(x,y,w,h) 填充一个矩形

ctx.strokeRect(x,y,w,h) 描边一个矩形

ctx.clearRect(x,y,w,h) 清除一个矩形范围内所有绘图

练习:

在画布的左上角填充一个100\*80矩形

在画布的右上角填充一个100\*80矩形

在画布的左下角填充一个100\*80矩形

在画布的右下角填充一个100\*80矩形

在画布的居中填充一个100\*80矩形

练习:

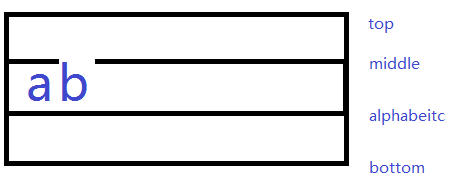
在画布上描边一个可以左右移动的 100\*80矩形

提示:使用定时器,先清除画布上己有内容,再重新绘制一个(x不停增加)

在画布上描边一个可以上下移动的 100\*80矩形(y 不停增加)

在画布上描边一个可以对角线90度方向移动的 100\*80矩形(x y 不停增加)

### 2.H5新特性-Canvas绘制文本



一段文字的定位在其文本的***基线***的起点

ctx.textBaseline = "alphabetic"; 文本的基线

ctx.font = "12px sans-serif" 文本大小和字体

ctx.fillText(str,x,y) 填充文本

ctx.strokeText(str,x,y) 描边一段文本

ctx.measureText(str) 测量文本返回一个对象{width}

练习:

在画布左上角描边一段黑色的文本: Tarena

在画布左下角描边一段黑色的文本: Tarena

在画布右上角描边一段黑色的文本: Tarena

在画布右下角描边一段黑色的文本: Tarena

在画布居中角描边一段黑色的文本: Tarena

在画布左上角描边一段黑色的文本: Tarena 左右移动

### 3.Canvas 绘图

Canvas绘图中使用渐变对象

线性渐变:linearGradient

径向渐变:radialGradient

var g = ctx.createLinearGradient(x1,y1,x2,y2);

g.addColorStop(offset,color)

...

g.addColorStop(offset,color)

ctx.strokeStyle = g ;

ctx.fillStyle = g;

练习:学子商城全年销售统计图

#### 1:动态请求

salesdata.php 向客户输出json 数据

[

{"月份":"一月","value":280},

{"月份":"二月","value":270},

{"月份":"三月","value":300},

....

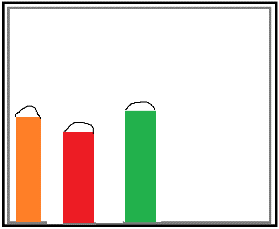
]

#### 2:静态请求

/02\_salesdata.html 向客户输出空白画布，加载完成之后异步

请求动态数据并且创建销售统计图

/jq.js



### 4.使用Canvas进行绘图---路径

路径:由多个坐标点组成任意形状，路径本身不可见，可用于

"描边","填充"

ctx.beginPath() 开始新路径

ctx.closePath() 闭合当前路径

ctx.moveTo(x,y); 移动到指定点

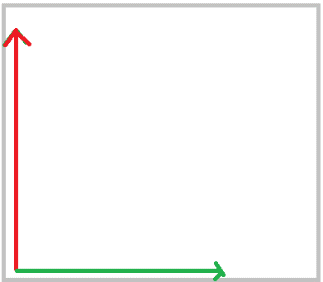
ctx.lineTo(x,y); 从当前点到指定点画直线

ctx.arc(cx,cy,r,start,end); 绘制圆拱形路径(弧度)

ctx.stroke(); 对当前路径描边

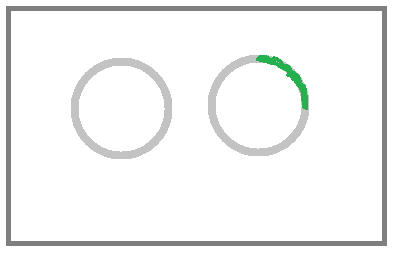
ctx.fill(); 对当前路径填充

练习:使用路径描边绘制一个坐标轴

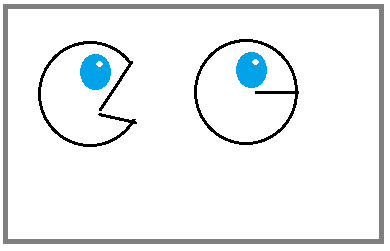


练习:使用圆拱形式绘制可以前进的进度条

提示:开始角度-90使用定时器，不停的修改结束角



练习:创建二个函数 openMouth() closeMouth();



使用定时器，每隔1s中交替调用以上二个方法

### 5.使用Canvas进行绘图---图像

canvas属于客户端技术，图片在服务器中，所以浏览器必须先下载绘制的图片，且等待图片异步加载完成.

var p3 = new Image(); //创建图片对象

p3.src = "img/p3.png"; //浏览器会异步请求加载图片(时间)

console.log(p3.width); //0

p3.onload = function(){ //事件:图片下载完成(onload)

console.log(p3.width); //200

ctx.drawImage(p3,x,y); //原始大小

ctx.drawImage(p3,x,y,w,h); //拉伸绘图

}

练习:画布左上角，右上角，左下角，右下角，居中原始大小飞机  
 练习:左右移动小飞机  
 练习:跟随鼠标移动小飞机

### 6.使用Canvas进行绘图---变形操作

canvas绘图也有变化技术，可以针对于某个图形/图像绘制过程进行变形: rotate(弧度) translate();平移原点

ctx.rotate(弧度); 旋转绘制画笔,轴点画布原点(0,0)

ctx.translate(x,y); 将整个画布原点平移到指定点

ctx.save(); 保存画笔当前所有变形状态值

ctx.restore(); 恢复画布状态到最近一次保存

***ctx.translate(100,100);*** 平移

***ctx.rotate(10);*** 旋转

***ctx.drawImage(..)*** 绘制

***ctx.rotate(-10);*** 逆向旋转

***ctx.translate(-100,-100);*** 逆向平移

提示:每个小飞机"平移+旋转+绘制+逆向旋转+逆向平移"

练习:在画布左上角画一个绕自己为中飞的飞机1;  
练习:在画布左上角画一个绕自己为中飞的飞机1;在同一个画布右

上角画一个旋转速度是飞机1，2倍速的飞机3

## (5)SVG绘图--------重点&难点

### SVG 可缩放的矢量图

|  |  |  |
| --- | --- | --- |
|  | canvas绘图 | SVG绘图 |
| 类型 | 2D位图 | 2D矢量图 |
| 如何绘图 | 使用JS代码图 | 使用标签绘图 |
| 事件绑定 | 每个图形不是元素，无法直接绑定事件 | 每个图形都是元素，可以直接绑定事件 |
| 应用场合 | 统计,游戏 | 地图 |

SVG技术诞生于2000年,早期作为XML扩展应用出现,H5标准常用标签采纳为标准，但有些被废弃.

### a:SVG 技术在HTML5之前使用方法

(1)在一个XML文档中声明要绘制图形

(2)再编写HTML文档，使用img/object 应用xml文档即可

### b: SVG 技术在HTML5之后使用方法

直接创建HTML5文档，在其中书写SVG标签即可

<svg></svg> 本身是300\*150的inline-block

### 5.1使用SVG标签绘制矩形

<rect>

|  |
| --- |
| **SVG-着重注意**  (1)所有图形默认只有填充色(黑色)没有描边色  (2)SVG图形的样式可以用元素声明方式，也可以使用CSS形式来声明，但用CSS声明时，只能使用SVG专用样式，不能用CSS样式，如边框只能用stroke,而不用border  (3)图形可以使用JS属性赋值,  但能用核心DOM操作可以不能用HTML DOM  r.x=100,r.width = 100;;//无效  r.setAttibute("width",1000)//有效  (4)动态添加SVG图形有两种方式  1)HTML字符串拼接  var html = "<rect></rect>";  svg.innerHTML = html;  2)使用  var xx = document.createElementNS("http://www.w3.org/2000/svg",  "标签名"); |

练习:绘制两个柱子，初始高度都是0,使用定时器，不断修改它们的高度，一个高度变为300停止，另一个变为150停止.

提示:用JS修改SVG图形的属性只能用核心DOM方法.

练习:创建php salesdata.php 四个季度销售额 json  
 创建html salesdata.html 获取json创建统计图

### 5.2使用SVG标签绘制圆形

<circle r="" cx="" cy="" fill=""

fill-opacity="" storke="" stroke-opacity="">

练习:在SVG画布随机创建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;点击某个圆形之后，它慢慢变大/淡直至消失从DOM上删除

### 5.3使用SVG标签绘制椭圆形

<ecllipse cx="" cy="" rx="" ry="">

rx:水平半径

ry:垂直半径

### 5.4用SVG标签绘制直线

<line x1="" y1="" x2="" y2="" stroke="" stroke-width=""

stroke-linecap="butt/square/round">



### 5.5 使用SVG标签绘制折线

一条折线上可以有任意多个连接点

<polyline points="50,50 100,300 ...." stroke="#f00" fill="transparent">

### 5.6使用SVG标签绘制多边形

<polygon points="50,50 100,300 ..">

### 5.7使用SVG标签绘制文本

<text font-size="" fill="" stroke="' x="" y="'>文本内容</text>

### 5.8使用SVG标签绘制图像

<image xlink:href="p3.png" with="" height="" x="" y=""></image>

### 5.9: SVG绘图

#### a:渐变对象

定义特效对象:渐变对象属于-一种特效对象

<defs>

<linearGradient id="g2" x1="" y1="" x2="" y2="">

<stop offset="0" stop-color="red" stop-opacity=""/>

<stop offset="1" stop-color="green" stop-opacity=""/>

</lineaGradient>

</defs>

<ANY fill="url(#g2)"></ANY>

#### b:滤镜

<defs>

<filter id="f2">

<feGaussianBlur stdDeviation="3" />

</filter>

</defs>

<ANY filter="url(#f2)"></ANY>

## (6)地理定位

**Geolocation:**地理定位,使用JS获得当前浏览器所在地理坐标(经度;维度;海拨;速度)数据，用于实现应用:如 高德导航.饿了么

### 手机浏览器如何定位信息

#### (1)首选手机中的GPS 芯片与卫星通信，定位精度在米

#### (2)次选手机通信基站进行定位获取，定位精度在公里

### pc浏览器如何定位信息

#### (3)通过IP 地址反向解析，定位精度取决IP地址库的大小

### HTML5中提供了一个新的对象,用于获取当前浏览器定位

window.navigator.geolocation{

getCurrentPosition:fn, //获得当前定位信息

watchPosition:fn //监视定位数据改变

clearWatch:fn //清除监视

};

### 扩展知识:在网页中如何:百度地图

#### (1)注册百度开发发帐号

http://lbsyun.baidu.com/

#### (2)创建一个网站,为网站申请一个访问密钥 AccessKey

#### (3)在自己网页中嵌入百度地图提供API

## (7)拖放API

### Drag & Drop:拖动和释放

HTML5为拖放行为提供7个事件，分为两组

### 拖动的源对象(会动)可以触的事件(3)

#### dragstart 拖动开始

#### drag 拖动中

#### dragend 拖动结束

### 拖动的目标对象(不动)可以触发事件(4)

#### dragenter: 拖动进入

#### dragover: 拖动着悬停在上方

#### dragleave: 拖动着离开

#### drop: 在上方释放

### 注意:必须阻止dragover默认行为,自动触发dragleave

## (8)Web Worker

### 程序:

Program 指可被CPU执行代码，存储在外存中.

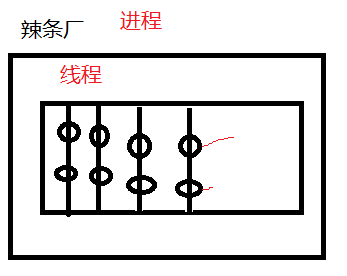
(java;c++;c#;php;js;python)

### 进程:

Process/Task 指程序被OS调入内存，随时供CPU执行

### 线程:

Thread 线程是进程内执行代码基本单位



### 进程和线程

#### (1)进程是操作系统分配内存的基本单位

#### (2)线程是CPU执行代码的基本单位

#### (3)线程必须处理某个进程内部

#### (4)一个进程内必须至少有一个线程，也可以多个

#### (5)一个操作系统中可能同时存在几千个线程,它们"并发执行"

--宏观上看同时执行, 微观看依次循环执行.

### Chrome浏览器的线程模型

一个chrome进程内，**至少有6个线程**，可以"并发"向web服务器发起http请求，以获得所需的资源

--资源请求线程

还有一个线程负责将所有内容绘制到浏览器页面中--UI主线程--

不允许多个线程同时绘图,防止内容布局错乱.

观察如下一段代码执行特点

<button id="bt1">11</button>

<script src="1.js"></script>

<button id="bt1">22</button>

**解决方案:**创建一个并发执行的新线程，让它来执行耗时的JS任务;

### Worker线程有缺陷

#### 浏览器不允许Worker线程操作任务的DOM/BOM对象.

#### Worker线程可以给UI线程发数据消息

##### UI主线程

var w5 = new Worker("10.js");

w5.onmessage = function(e){e.data}

##### Worker线程

postMessage(stringMsg)

##### UI主线程也可以给Worker线程发数据消息

##### UI主线程

var w5 = new Worker("10.js");

w5.postMessage(strinMeg);

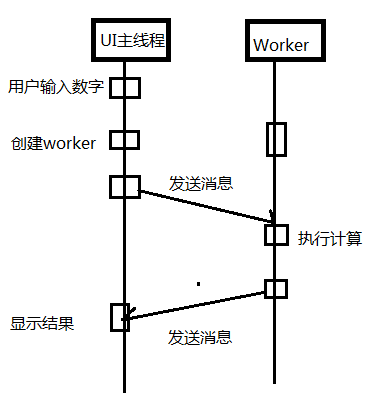
##### Worker线程

onmessage = function(e){e.data}

练习:在HTML中有一个INPUT,按钮“开始计算累加和”,点击按钮，

创建一个Worker线程来计算出用户输入的数子累加和，

在下方一个div显示计算结果 14:30-14:35



## (9)Web Stoage

在浏览器中存储当前用户**专有**的数据:访问历史、内容定制、样式定制.

### 客户端存储数据可以使用的技术

#### (1)cookie 浏览器兼容性好，不能超过4KB，操作复杂

#### (2)Flash存储 依赖于Flash播放器

#### (3)H5 WebStorage 不能超过8MB，操作简单

#### (4)IndexedDB 可存储大量的数据，还不是标准技术

|  |
| --- |
| Session:会话:浏览器从打开某个网站的一个页面开始，中间可能打开很多页面，直到关闭浏览器，整个过程称为"一次会话" |

WebSrorage技术中，浏览器为用户提供二个对象

### (1)window.sessionStorage:类数组对象，会话级数据存储

在浏览器进程所分得内存中存储一次web会话可用的数据，

可供此次会话所有的页面共同使用; 浏览器一旦关闭数据消失了，作用:如 登录用户名，用户编号

sessionStorage[key]=val //保存一个数据

sessionStorage.setItem(key,val) //保存一个数据

var val = sessionStorage[key] //读取一个数据

var val = sessionStorage.getItem(key) //读取一个数据

sessionStorage.removeItem(key) //清除一个数据

sessionStorage.clear() //清除所有数据

sessionStorage.length //数据个数

sessionStorage.key(i) //获取第i个key

练习:15:55-16:00

创建 index.html 右上角超链接"请登录"

创建 login.html 输入用户名，密码，提交

3秒钟后转换首页 index.html

setTimeout(function(){location.href="index.html"},3000);

sessionStorage["uname"]="tom"

返回 index.html "欢迎回来:xxx 退出登录"

创建 logout.html 提示"您己退出登录，3秒钟后转回首页"

sessionStorage.removeItem("uname");

### (2)window.localStorage

在浏览器所能管理的外存(硬盘)中存储用户数据，可供次会话及后续的会话页面共同使用。即使浏览器关闭-永久存在.

当前客户所有会话共享数据; 网页访问次数,***用户样式***

**localStorage [key]=val //保存一个数据**

**localStorage.setItem(key,val) //保存一个数据**

**var val = localStorage [key] //读取一个数据**

**var val = localStorage.getItem(key) //读取一个数据**

**localStorage.removeItem(key) //清除一个数据**

**localStorage.clear() //清除所有数据**

**localStorage.length //数据个数**

**localStorage.key(i) //获取第i个key**

localStorage中若数据发生改变，会触事件onstorage事件

可以监听此事件，实现监视localStorage数据改变目的.

不能监视sessionStorage数据修改.

练习:

创建index.html 有一个下拉菜单,"请选择你喜欢的主题"

"蔚蓝天空" "芭比公主" "暗黑主题" 各对应一个class名

.blue{background:#ddf;color:#33a}

.pink{ background:#fdf;color:#a3a }

.dark{ background:#333;color:#eee }

用户选中某个选项(select.onchange) 为当前页面(body)使用指定样式.

创建usercenter.html 打开此页面应用index.html 中一样的主题式.

返回 index.html 也应该自动应用之前选主题样式

## (10)Web Scoket

# Bootstrap

## 什么是响应式网页

1G: 只能通话

2G: 可以通话，短信

2.5G: GPRS 可以通话、短信、上网--WML

3G: IOS/Android(Linux) 可以通话，短信，视频，上网--HTML

京东: www.jd.com m.jd.com

**Responsive Web Page:** 响应式网页/自适应的网页,2010年提出,

一个网页，会自动根据用户***浏览器设备***不同，自动改变布局,可以被PC/PAD/PHONE正常浏览器.

## 响应式网页必备

### (1)流式布局 float:left; display:inline-block

### (2)可以改变尺的图片

### (3)可以改变大小的文字

## 响应式网页的不足

#### 代码更加复杂,不适合于大型页面

## 如何测试响应式的网页

### (1)使用真实的物理设备浏览器

不足:测试任务量太大

好处:测试结果真实可靠

### (2)使用第三方测试软件

### (3)使用Chrome内置的设备模拟器

好处:测试简单,功能丰富

不足:测试结果有待进一步验证

## 如何编写响应式网页--- 重点&难点

如何做网页适用(phone/pad)网页?

### (1)声明viewport元素

<meta name="viewport"

content="width=device-width, initial-scale=1">

|  |
| --- |
| viewport元标签  早期的3G手机为浏览大尺寸的页面，只能强行把页面缩小，  导致图片，文字，超链接都很小.  IOS提出了"视口"的概念,视口用于盛放网页内容，尺寸大小可以随意指定;    width=device-width 表示"视口"宽度是设备屏幕的宽度  initial-scale=1 表示初始化缩放比例 |

### (2)使用流式布局

实现方式: float:left display:inline-block

### (3)所有容器使用相对尺寸，不用绝对尺寸

.container{

/\*width:500px;\*/

width:100%;

}

### (4)所有文字使相对尺寸，不用绝对尺寸

body{

/\*font-size:12px;\*/

font-size:1.2em;

}

### (5)所有图片使用相对尺寸，不用绝对尺寸

img{

/\*width:130px;\*/

width:50%;指定父容器中宽度占比

height:auto;

}

### (6) 最重要的原则 使用CSS3 Media Query技术

## CSS 媒体查询技术

Media:指浏览网页的设备,如 screen(pc/pad/phone) print

Media Query:查询出当前浏览网页设备的类型,以及特性

(尺寸，手持方向等)随着设备不同，而选择执行某些CSS代码

而忽略另外一些。

媒体查询的两种使用方式

### (1)根据媒体查询结果执行不同的css文件

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

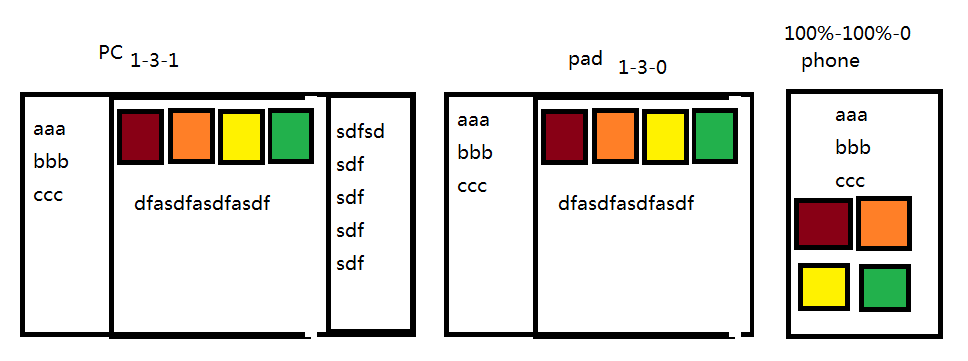
### (2)根据媒体查询执行CSS片段中不同部分选择器(使用比较多)

@media screen and (min-width:768px) and (max-width:991px){

选择器{样式设定}

}

练习:使用CSS3 Media Query编写响应式网页



提示:四个图片是小难点

|  |
| --- |
| bootstrap:起步,引导程序 |

## Twitter Bootstrap 框架

官网: https://getbootstrap.com

中文网: http://www.bootcss.com/

Bootstrap是一个HTML/CSS/JS框架,用于开发响应式布局、移动设备优先web项目

内容分为五部分

### (1)起步

### (2)全局CSS样式

### (3)组件

### (4)JS插件

### (5)定制

提示:Bootstrap知识点不难，但难在:1000+单词忆记

## Bootstrap第一部分:起步

### 1:下载 css/fonts/js

### 2:基本模板

|  |
| --- |
| <html lang="zh-CN">  Language:指定当前网页所有自然语言:zh简体中文CN 中国  作用:为浏览器翻译功能确定基础语言,为读取屏软件确定基础发音 |
| <meta name="viewport" content="width=device-width, initial  视口 |
| <meta http-equiv="X-UA-Compatible" content="IE=6">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  指定IE浏览器兼容性指定--仅IE浏览器支持此头部  IE 6:  IE 7:  ...  IE 11: |
| html5shiv.min.js  第三方编写JS文件，让老IE兼容HTML5新的标签 header/footer/... |
| respond.min.js  第三方编写JS文件,让老IE支持响应式网页必备内容--  CSS3 Media Query |

bootstrap.css 提供HTML标签默认样式重置(CSS Reset)

## Bootstrap第二部分: 全局CSS样式

### 1:Bootstrap 全局css样式---按钮

.btn 按钮的基础样式

.btn-default 白底黑字的按钮

.btn-danger/warning/success/info/primary 五种颜色按钮

.btn-lg/sm/xs 按钮大小

.btn-block 块级按钮

### 2:Bootstrap 全局css样式---图片

.img-rounded 圆角图片

.img-circle 图形图片

.img-thumbnail 缩略图片

.img-responsive 响应式图片

### 3:Bootstrap 全局css样式---文本

.text-danger/.text-success/.text-warning/.text-info/.text-primary

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

.text-left/center/right/justify

.text-uppercase/.lowercase/capitalize

### 4: Bootstrap 全局css样式---列表

.list-unstyled 去除提示符号

.list-inline 行内列表

|  |
| --- |
| Bootstrap中屏幕的分类  **(1)大型PC屏幕(Large-lg) w>=1200px**  **(2)中等PC屏幕(Medium-md) 1200px>w>992px**  **(3)小型PAD屏幕(Small-sm) 992px>w>768px**  **(4)超小PHONE屏幕(ExtraSmall-xs) 768px>w** |

### 5: Bootstrap 全局css样式---表格

.table

.table-bordered 带边框的表格

.table-striped 隔行变色的表格

.table-hover 带悬停变色的表格

.table-responsive 响应表格 !!(必须用在TABLE的父元素div)

|  |
| --- |
| Bootlint是Bootrap官方所支持HTML检测工具，在使用Bootrap页面上，它能自动检查某些常见的HTML/CSS错误.  建议将Bootlint加入开发工具中，这样帮助你在项目开发中避免简单错误。 |

### 6:栅格布局系统--重点&难点--(Grid System)

网页布局可用的方法

#### (1)TABLE布局

好处:简单易控制

不足:语义错误,页面渲染效率低

#### (2)DIV+CSS布局

好处:语义正确,页面渲染效快

不足:不容易控制

#### (3)Bootstrap的栅格布局系统

好处:简易控制，语义正确，页面渲染速度快，支持响应

不足:没有

#### 使用方法

##### (1)最外层必须使用容器

div.container或div.container-fluid

##### (2)容器里可以放置任何内容，若想使用栅格系统，必须用div.row

.container > .row

##### (3)一个.row中不能放其他内容，只能放置 .col,列中放置任何内容

.container > .row > .col

##### (4)Bootstrap中的行默认均分为12等份，每个列必须指定在行中占有比

##### (5)栅格系统针对不同的屏幕提供不同的列

.col-lg-1/2/3...12

.col-md-1/2/3..12

.col-sm

.col-xs

##### (6)可以使用"列偏移"实现指定列及其后的列向右偏移的效果,

如果不足 12列,居中

.col-lg-offset-1/2/3/../12

.col-md-offset-1/2/3/../12

.col-sm-offset-1/2/3/../12

.col-xs-offset-1/2/3/../12

##### (7)需要注意不同屏幕下列的适用性问题

.col-sx-\* 适合于xs/sm/md/lg屏幕

.col-sm-\* 适合于 sm/md/lg 屏幕

.col-md-\* 适合于 md/lg 屏幕

.col-lg-\* 适合于lg 屏幕

列宽对于当前屏幕以及更大屏幕都有效，列偏移也一样

##### (8)一个列可以指定在不同屏幕下宽占比

div.col-xs-12 col-sm-9 col-md-6

div.col-xs-12 col-md-6

##### (9)一个列可以指定在特定屏幕下隐藏

hidden-xs 仅在xs 屏幕下隐藏

hidden-sm 仅在sm 屏幕下隐藏

hidden-md 仅在md 屏幕下隐藏

hidden-lg 仅在lg 屏幕下隐藏

##### (10)栅格系统可以嵌套，规则:在行中再嵌入行，行中再有列

.container >

.row >

.col-\*-\* >

.row >

.col-\*-\*

### 7:表单---难点

#### (1)默认表单



<form>

<div class="form-group">

<label class="control-label"></label>

<input type="text" class="form-control">

<span class="help-block"></span>

</div>

</form>

#### (2)行内表单



<form class="form-inline">

<div class="form-group">

<label class="sr-only">用户名</label>

<input type="text" class="form-control">

<span class="help-block"></span>

</div>

</form>

#### (3)水平表单

水平表单 = 表单+栅格系统(变种)

|  |  |  |
| --- | --- | --- |
|  | 默认栅格系统 | 水平表单中栅格 |
| 最外层元素 | div.container | form.form-horizontal |
| 行 | div.row | div.form-group |
| 列 | div.col-\*-\* | div.col-\*-\* |

<form class=" form-horizontal ">

<div class=" form-group">

<div class="col-\*-\*">

<label class="sr-only"></label>

</div>

<div class="col-\*-\*">

<input type="text" class="form-control" />

</div>

</div>

</form>

## bootstrap第三部分：组件

### 3.1: 组件 - 下拉菜单

下拉菜单必需三级结构

<div class="dropdown">

<a data-toggle="dropdown">触发元素</a>

<div/ul class="dropdown-menu">隐藏元素</div/ul>

</div>

### 3.2: 组件 - 导航

提示:bootstrap中的导航(nav)不同导航条(navbar)

#### 导航组件分为三种

##### (1)标签页式导航--页签组件

<ul class="nav nav-tab">

<li><a data-toggle="tab" href="#">十元套餐</a></li>

<li class="active"><a data-toggle="tab" href="#">二十元套餐</a></li>

<li><a data-toggle="tab" href="#">三十元套餐</a></li>

</ul>

##### (2)胶囊式导航

<ul class="nav nav-pills">

<li><a data-toggle="tab" href="#">十元套餐</a></li>

<li class="active"><a data-toggle="tab" href="#">二十元套餐</a></li>

<li><a data-toggle="tab" href="#">三十元套餐</a></li>

</ul>

练习:(字体图标)

### 3.3: 组件 - 图标字体

#### Web项目中常用图标字体

(1)FontAwesome - 675个(免费)的图标

(2)Glyphicons - 800 个收费的图标

#### 服务器端字体的使用方法

#### (1)web服务器项目目录必须有字体文件

#### (2)css中声明一个新字体, 并指定字体文件下载位置

@font-face {

font-family: 'Glyphicons Halflings';

src: url('../fonts/glyphicons-halflings-regular.eot');

}

#### (3)声明一个基础的class,使用到该字体

.glyphicon {

font-family: 'Glyphicons Halflings';

}

#### (4)在HTML页面中输图标对应的unicode码

.glyphicon-minus:before {

content: "\2212";

}

对于页面开始者，图标字体最后的使用方法

<span class=" glyphicon glyphicon-minus"></span>

练习:

主页/用户/配置/汉堡包/刷新/打分/我喜欢(心形)/

发邮件/拍照/定位/购物车/放大镜/删除(垃圾箱)/

加号/减号/对勾/前进/后退/前一张/后一张/

播放/暂停/停止/快进/快退

### 3.4: 组件 - 按钮组

.btn-group > .btn\* n

.btn-group-vertical > .btn \* n

### 3.5: 组件 - 警告框

<div class="alert alert-四种颜色">

<span class="close">X</span>

<p>....</p>

</div>

### 3.6: 组件 - 进度条

.progress > .progress-bar progress-bar-四种颜色

练习:

面包屑 .breadcrumb

分页条 .pagination

分页器 .pager

徽章 .badge

标签 .label

巨幕 .jumbotron

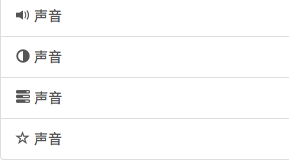
页头 .page-header

水井 .well

### 3.7: 组件 - 列表组

方式1：ul.list-group > li.list-group-item

方式2：div.list-group > a.list-group-item



练习:使用列表组+图标字体 实现上述效果

### 3.8: 组件 - 缩略图

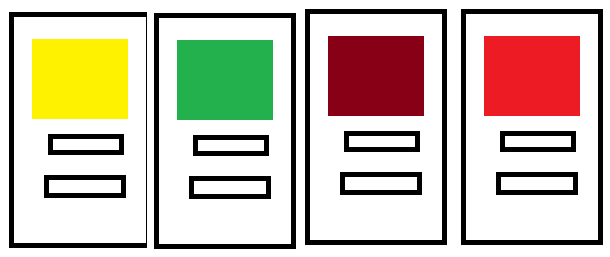
<div class="thumbnail">

<img src="">

<div class="caption"></div>

</div>

注意：缩略图本身是一个block组件,需要配合栅格系统控制宽度



### 3.9: 组件 - 媒体对象

以“左中/中右/左中右”横向三部分来展示一个商品/文章对象

<div class="media">

<div class="media-left">img</div>

<div class="media-body">p/div</div>

<div class="medi-right">img</div>

</div>

### 3.10: 组件 - 面板

以"中/上中/中下/上中下"纵向三部分来展示数据

<div class="panel panel-颜色">

<div class="panel-heading"></div>

<div class="panel-body"></div>

<div class="panel-footer"></div>

</div>

面板一个重要的用途:实现手风琴组件

|  |
| --- |
| Bootstrap基于Jquery提供了十几个插件函数(bootstrap.js)  调用方法有两种  (1)js编码方法  $(".dropdown>a").dropdown();  (2)data扩展属性方法:  <a data-toggle="dropdown"></a> |

## Bootstrap第四部分: jquery插件

### 4.1: jquery插件 -- 折叠(collapse)

<a href="#my-target" data-toggle="collapse">展开/收起</a>

<button data-toggle="collapse" data-target="#my-target"></button>

<div id="my-target" class="collapse in"></div>

折叠插件实现 手风琴=多个面板+多个折叠插件

### 4.2: jquery插件 -- 响应式导航条(复杂)

.navbar-toggle {三横线}

媒体查询:屏幕大于768px,就隐藏

.navbar-collapse 的媒体查询:屏幕的宽度大于768px,

定位.navbar-header内,否则定位 .navbar-header下方

#### 导航条颜色:

浅色底深色字: .navbar-default

深色底淡色字: .navbar-inverse

#### 导航定位位:

相对定位:默认

固定定位: .navbar-fixed-\*

#### 导航条位置

顶部: .navbar-fixed-top

底部: .navbar-fixed-bottom

### 4.3: jquery插件 -- 模态框

Model:模态,一个父窗口打开了一个子窗口，若子窗口不关闭父窗口无法获得输入焦点，该子窗口称为模态对话框

window.alert()/confirm()/prompt();

<div class="modal fade"> 动画效果

<div class="modal-dialog"> 尺寸定位

<div class="modal-content"> 背景/边框/阴影

<div class="modal-header">

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

</div>

<div class="modal-body">主体</div>

<div class="modal-footer">尾部</div>

</div>

</div>

</div>

### 4.4: 插件 -- 轮播广告/提示框

提示:轮播广无需记忆，使用 BootLint提示功能，可以全写出来

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

<div class="carousel-inner">

<div class="item active">

img+.carousel-caption

</div>

...

</div>

</div>

### 4.5: 第四部分：插件 工具提示

把元素title 属性值在一个黑色的提示框中显示

<a data-toggle="tooltip" href="#" title="返回达内首页">返回</a>

注意: 该插件必须调用一下 js

$("[data-toogle='tooltip']").tooltip();

### 4.6: 第四部分：插件 弹出框

<button type="button" title="弹出框的标题"

data-content="弹出框中的详细内容"

data-toggle="popover" data-trigger="hover">关闭</button>

注意:该插件必须调用一下js

$("data-toggle='popover'").popover();

## bootstrap第五部分：重构(定制)

### 5.1:样式语言的分类

#### (1)静态样式语言:

CSS 可以直接被浏览器解析，但作为一门语言，CSS并不完整

,缺少编程语言必需的概念：数据类型;变量;运算; 函数;

导致样式***可维护性***差.

#### (2)动态样式语言:

第三种:Sass;Less;Stylus;在CSS的基础上添加动态编程语言必须的特性,如数据类型,变量,运算,函数，提高了样式可维护性.

需要注意:浏览器只能识别CSS，所有的动态样式语言的代码，

必须转换CSS，才能被浏览器识别,该过程称为"编译"

### 5.2: Less样式语言

#### 官网: http://lesscss.org

#### 中文网: http://lesscss.cn

Less是一门CSS预处理语言--它扩展CSS语言，增加变量,函数

等特性，使CSS更易于维护和扩展.

### 5.3:Less使用方法

#### (1)在客户使用Less

##### 1.1 编写 x.less

##### 1.2 在HTM引入x.less，同时再引入一个Less编译程序

less.min.js #注意顺序

##### 1.3 在客户请求下，下载x.less和less.min.js 客户端运行js

把less文件编译为 css

会减慢客户端式的呈现速度(学习时可以使用)

#### (2)在服务器端使用Less--必需掌握

##### 2.1 编写 x.less

##### 2.2 安装*less编译环境*，执行x.less 编译为 x.css

##### 2.3: 再编写 html,引入x.css

##### 2.4: 客户端请求HTML文件，下载css文件即可(生产环境)

### 5.4: Less语法学习

##### (1) Less支持所有CSS语法;

##### (2) Less 支持多行/单引注释,但只有多行注释会被编译CSS

文件中--推荐单行注释

##### (3)Less变量(Variable)概念

定义: @变量名:值;

示例:

@baseColor:"#abc"

.box{

color:@baseColor;

}

变量可以取值为任意合法样式值;

变量名:英文/匈牙利/

##### (3)Less支持变量和常量的算术运算

+ - \* / %

##### (4)Less支持在一个选择器中"混入"另一个选择器定义的样式

选择器1{}

选择器2{ ... 选择器1...}

##### (5)Less在样式混入可以指定参数

选择器1(@参1,@参2){}

选择器2{...选择器(值1,值2)}

##### (6)Less提供几十样式操作函数,

ceil();

floor();

percentage(num) 把小数转百分比形式

darken(color,percentage) 把指定颜色变暗

lighten(color,percentage) 把指定颜色变亮

image-width(url) 返回指定图片宽度

image-height(url) 返回指定图片高度

##### (7)Less中可以使用@import实现文件包含--

### 5.5:通过修改Bootstrap的less源文件实现重构(服务器定制)

练习:操作定制默认按钮(背景颜色/边框/文字)

#### (1)检查node正常使用

node -v

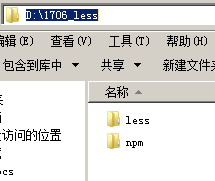
#### (2)创建目录(没有中文没有空格)

D:\1706\_less

#### (3)在此目录创建less\ 复制所有bootstrap less文件

#### (4)复制编译工具

npm--->D:\1706\_less\



#### (5)修改less文件 variables.less 按钮默认颜色

#### (6)将bootstrap.less 编译成 bootstrap.css

##### 6.1:进入命令行

##### 6.2:切换目录

cd D:\1706\_less\npm

D:

##### 6.3:编译 lessc 源文件 目标文件

lessc D:\1706\_less\less\bootstrap.less D:\1706\_less\bootstrap.css

### 5.6:通过修改Bootstrap的Less源文件定制

#### (1) 瘦身:删除不需要的样式

只需要注释掉bootstrap.less不要的@import

#### (2) 粗粒度定制

只需要修改varables.less中定义的变量值即可

#### (3) 细粒度定制:精准控制某个组件的样式细节

修改对应的.less文件,如dropdowns.less

# 框架

## 什么是框架

框架指的是一套非常优秀可被反复使用的代码,帮助开发者解决一些复杂的问题

## 框架的好处

### 1)提高了代码的复用率

### 2)降低了模块之间的耦合度

### 3)提高了开发速度

### 4)提高代码质量

# Vue.js

## WebStorm提供了一个模板的功能，常见一个模板：

①编写模板

②拷贝模板当中的所有的内容

③点击webstorm左上角的file--》new--》edit file templates

④点击弹窗中的左上角的加号，指定模板的名字（vue）和结尾的名字（html）

⑤保存就可以

## Vue的概述

Vue的开发模式 和 之前接触的jQuery、原生JSDOM操作是不同的，之前要想修改试图，首先找元素；在使用Vue时，专心把精力放在修改数据。

DOM驱动 ---》 数据驱动

### What?

Vue的开发者时尤雨溪,是受到了angular的启发,是一个实现ui层的渐进式的js框架

版本 : V2.4.0

注意 :

1. 并不是所有的ui类的库或者框架 , 都提供了丰富的样式类
2. 渐进式 : step by step

### Where?

饿了么 , 滴滴 , 阿里巴巴

Gitlab…..

VueJS可以用在很多地方,从简单的表单验证,到复杂的大型企业应用程序,Vue都可以胜任;

核心库是实现ui层的

### Why?

#### 1)非常好的中文文档

#### 2)学习曲线比较缓和(容易上手)

#### 3)速度快

#### 4)体积小

#### 5)基于组件( web component )化的开发方式

#### 6)代码的可读性,可维护性比较好

### How?

#### 搭建环境有两种方式 :

##### 方式1 : CLI (command line interface 命令行窗口)

#全局安装 vue – cli

Npm install – global vue –cli

#创建一个基于 webpack 模板的新项目

Vue init webpack my-project

#安装依赖

Cd my-project

Npm install

Npm run dev

#### 方式2: 直接引入对应的js文件

## Vue的基础使用

### 第一个例子

通过new的方式来得到了Vue的实例,在new通过一个对象来对Vue的实例配置(el,data);

#### 接触了双花括号(mustache/interpolation)的语法:

可以直接读取变量的值,显示在当前调用的元素的innerHTML中

练习时间：

新建一个文件，通过vue的方式 ，在html中执行

一个三目运算、一个逻辑运算；将运算的结果显示不同的h1标签中。

### 双花括号

#### 语法 :

**<any> { { 表达式 } } </any>**

#### 作用 :

执行表达式,将表达式的结果 ,

输出到当前调用的元素innerHTML中

### Vue的常用指令

循环,选择判断,显示隐藏….

#### 循环指令

##### 语法1:

###### <any v-for=” tmp in 集合” ></any>

**v-for :** 根据**in**关键词后的集合去循环创建多个标签

###### 举例:

MyList:[100,200,300]

<a v-for=”tmp in MyList” >{ { tmp } }</a>

##### 语法2:

###### <any v-for=” (value , key) in 集合” ></any>

练习：

去新建一个文件，通过Vue将一个对象数组 显示在table(thead和tbody)中。

对象数组有5个对象，

每个对象指定3个属性：name/sex/age.

基本步骤

1. 定义数据

[{},{}]

1. 在视图中创建table
2. 将数据通过v-for和双花括号将值显示出来

#### 选择指令

##### <any v-if="表达式"></any>

v-if指令 ： 根据表达式执行的结果的真假 来选择是否要挂载到DOM

v-else-if

v-else

练习：

通过Vue，渲染一个h1,ul.

在刚开始时，h1：正在加载数据 ,ul显示一个空的列表在vue实例创建完毕1s之后，h1:从DOM中移除掉，同时给数组初始化5条数据 显示在ul中。

data需要用到的数据：

isDataReady:false

myList:[]

①搞定数据

②显示在视图中

③指定定时器

#### Vue事件绑定

##### <any v-on:eventName="处理函数名字"></any>

通过v-on去给指定的事件绑定一个处理函数

举个例子：

<button v-on:click="handleClick">clickme</button>

当触发click事件，就会调用handleClick方法。

注意事项：

方法在定义时，要放在Vue实例的methods属性

new Vue({

methods:{

handleClick:function(){

//做一些操作

}

}

})

事件绑定支持支持一种简写形式：

<button v-on:click="handleClick">clickme</button>

//通过@后边跟上事件的名字，和v-on:是一样的效果

##### <button @click="handleClick">clickme</button>

Vue.js 为 v-on 提供了 事件修饰符。通过由点(.)表示的指令后缀来调用修饰符。

v-on:submit.prevent="handleSubmit"

#### Vue其它常用指令

##### v-bind

将变量中的值 通过 v-bind 绑定到元素指定的属性

img v-bind:src="myImage"

a v-bind:href="myLink"

v-bind:style='{backgroundColor:myBGColor}'

v-bind:class='{myRed:isRed}'

v-bind:disabled="!isValid"

###### v-bind也支持简写

**a v-bind:href="myLink"**

**===》**

**a :href="myLink"**

##### v-show

v-show="表达式" 根据表达式执行的结果的真假 来切换display显示还是隐藏

##### v-text

v-text='变量' 根据变量更新对应的文本内容

##### v-html

v-html="变量" 根据变量更新对应的innerHTML

...

#### Vue中的v-model

##### 双向数据绑定：

###### 方向1：data --> view

之前所接触的循环、判断、显示隐藏等指令，主要的功能是将数据 绑定到视图；

**方式：①双花括号 ②常见的指令**

###### 方向2：view --》 data

将**表单控件**中 用户操作的 结果 绑定到 数据

**方式：v-model**

v-model指令本身就是一个双向数据绑定的指令：既可以将数据绑定到视图，又可以将视图中的结果绑定到数据。

#### 修饰符：

##### .lazy

- 取代 input 监听 change 事件（在输入完之后，失去焦点 再去更新(替换成blur?)）

##### .number

- 输入字符串转为数字

##### .trim

- 输入首尾空格过滤

## 三.自定义指令的创建和使用

Vue 自带的指令很多:

v-for/v-if/v-else/v-else-if/v-model/v-bind/v-on/v-show/v-text/v-html…

但是这些指令都是比较偏向工于具化,有些时候在实现具体的业务逻辑的时候,发现不够用,如何来自定义指令,

### 1)自定义指令

**new Vue**({

**directives**:{

**change**:{  
 bind:function(){}

update:function(){}

unbind:function(){}  
 }  
 }  
})

在自定义指令时,在指令对应的配置对象中有3个处理函数指令对应的操作

#### Bind

指令在绑定到元素要执行的操作

#### Update

如果在掉调用指令时候,传了参数,当参数变化时候,就会执行update所指定的操作

#### Unbind

解除绑定时要执行的操作

### 2)自定义指令

**change**:{  
 bind:function(){}

update:function(){}

unbind:function(){}  
 }

v-change

### 注意事项：

建议在给指令的命名采用小驼峰式的命名方式，比如changeBackgroundColor,在使用的时候，采用烤串式写法 v-change-background-color

练习：实现一个自定义指令v-change-background-color,

changeBackgroundColor

在调用指令时，可以去传递参数，根据参数的值 修改调用该指令的元素的背景色。

## 四. 自定义过滤器的创建和使用

过滤器：生活中有很多例子，净水器 空气净化器 。

过滤器的作用：实现数据的筛选、过滤、格式化。

vue1.\*版本是有内置的过滤器，但是在vue2.\*所有的版本都已经没有自带的过滤器了。

### 1、过滤器创建

过滤器的本质 是一个有参数 有返回值的方法

new Vue({

filters:{

myCurrency:function(myInput){

return 处理后的数据

}

}

})

### 2、过滤器使用

语法：

<any>{{表达式 | 过滤器}}</any>

举个例子：

<h1>{{price | myCurrency}}</h1>

### 3、过滤器高级用法

在使用过滤器的时候，还可以指定参数，来告诉过滤器按照参数进行数据的过滤。

①如何给过滤器传参？

<h1>{{price | myCurrency('￥',true)}}</h1>

②如何在过滤器中接收到？

new Vue({

filters:{

//myInput是通过管道传来的数据

//arg1在调用过滤器时在圆括号中第一个参数

//arg2在调用过滤器时在圆括号中第二个参数

myCurrency:function(myInput,arg1,arg2){

return 处理后的数据

} }})

练习：

实现一个过滤器myTextTransform,功能是实现将一个字符串转换为大写或者小写，取决于传递给过滤器的参数，如果是true转换为大写 否则小写。

myInput.toUpperCase()

myInput.toLowerCase()

## 五、组件

w3c为什么有组件概念？

现在的前端领域，对于代码的质量和数量有着非常高的要求，特别容易出问题。

通过组件的方式来完成代码的管理和编写。

### 组件：是一个可以被反复使用的 带有指定功能的 视图

组件化编程 一切都是组件component，把可以被反复使用的任何的视图 都可以封装成组件。

类似玩积木。

### 组件化的好处（封装）：

①提高了代码的复用率

②提高开发速度

③降低测试难度

④降低整个应用的耦合度

。。。

### 1、组件的创建

Vue.component('my-component',{

template:'<h1>it is a component</h1>'

})

### 2、组件的使用

组件的使用就像是使用普通的html标签一样

<my-component> </my-component>

**建议：**组件的命名和使用 建议来使用烤串式命名规则。

（如果组件的名字写成驼峰式 myComponent,使用依然还得是烤串式）

### 注意：

①如果一个组件要渲染多个元素，将多个元素放到一个顶层标签；否则会报错。

### 3、复合组件

组件可以像html一样来使用，复合组件并不是一个新的概念，就是一个组件，只不过该组件中又调用了其它的组件。

一个完整的Vue的项目，可以比作是一颗组件树，组件树的根 就是 根组件（my-article）....

组件树可以帮助我们降低业务的复杂度，避免出现一些错误，提高开发速度。

### 4、组件的生命周期

#### 分为4个阶段：

**Create / mount / update / destroy**

每一个阶段都对应着有自己的处理函数

##### create:

beforeCreate created

初始化

##### mount:

beforeMount mounted

和挂载相关的处理

##### update:

beforeUpdate updated

根据要更新的数据 做逻辑判断

##### destroy:

beforeDestroy destroyed

清理工作

### 在Vue的实例或者在组件，都有哪些属性？

el

data

methods

directives

filters

watch

练习：

生命周期。

创建一个组件，该组件渲染一个h1的标签。

要求：在组件挂载完毕之后，启动定时器，修改h1标签的透明度的值 0~1

v-bind:style='{backgroundColor:myBGColor}'

每日一练：

创建一个组件，my-game:

猜数字大小。

组件：一个input和一个p构成

当组件准备挂载的时候，初始化一个随机数，在input取输入的时候，如果输入的数字小了，在p显示：输入的太小了；如果输入的数字大了，在p显示：输入的太大了；否则就提示输入正确。

## 六.组件间通信（父组件--》子组件）

### 步骤：

#### ①父组件在调用子组件 传值

<child-component myValue="123"> </child-component>

#### ②在子组件中 获取父组件传来的值

Vue.component('child-component',{

props:['myValue'],

template:''

})

练习：登录窗口

创建4个组件，分别是my-label my-input my-button my-login(复合组件)

## 七、组件间通信（子组件传值给父组件）

通过事件的方式来完成数据的传输。

### ①在父组件中 定义一个方法，用来接收子组件所通过事件传来的值

methods:{

recvMsg:function(msg){

//参数msg就是子组件通过事件出来的数据

}

}

### ②绑定事件处理函数

事件一般情况 都是自定义事件

<child-component @myEvent="recvMsg">

</child-component>

### ③在子组件触发事件

this.$emit('myEvent',myPhone)

//触发一个叫做myEvent的事件，同时把第二个参数数据传递给事件对应的处理函数

总结：

在 Vue 中，父子组件的关系可以总结为 props down, events up。父组件通过 props 向下传递数据给子组件，子组件通过 events 给父组件发送消息。

练习：

在子组件中 放上一个input，点击按钮 把用户输入的内容发给父组件

练习：

有两个组件，分别是main-component,header-component.

main-component是由header-component和一个列表（有5条数据 [100,200,300,400,500]），

header-component是由一个h1的标签：'这是页头',有一个数据isUserLogin:true

在渲染main-component时候，读取header-component在挂载完毕之后通过事件传递来的数据（isUserLogin），根据该数据的真假来决定列表是否显示.

<child-component @myEvent="recvMsg">

</child-component>

this.$emit('myEvent',myPhone)

## 八、组件间通信（$parent $refs）

父组件要想获取子组件的数据：

### ①在调用子组件的时候，指定ref属性

<child-component ref="mySon"></child-component>

### ②根据指定的引用的名字 找到子组件的实例对象

this.$refs.mySon

子组件要想获取父组件的数据：

#### ①直接读取

this.$parent

组件间通信综合练习：

（props down,events up）

有2个组件：chat-room,user-component

user-component是由label input button构成

chat-room是由两个user-component和一个列表构成

①在chat-room调用user-component指定label的名字

②在user-component，

点击按钮时，将当前用户输入的信息发送给chat-room组件，chat-room接收到数据显示在列表中

## 九、兄弟组件间通信（event）

借助于一个公共的Vue的实例对象，不同的组件可以通过该对象完成事件的绑定和触发

var bus = new Vue();

bus.$emit()

bus.$on()

熊大想要发消息给熊二，

### 接收方（熊二）：事件绑定

**bus.$on('customEvent',function(msg){**

**//msg就是通过事件 传递来的数据**

**})**

### 发送方（熊大）：触发事件

**bus.$emit('customEvent',123);**

练习：

在熊二中 加上一个button，

点击按钮时告诉熊大:'快跑！'

接收方：事件绑定

发送方：触发事件

总结：

父组件--》子组件

①通过属性

步骤1：

<son myName="michael" myPhone='123'></son>

<son :myName="userList[0]"></son>

步骤2：

Vue.component('son',{

props:['myName','myPhone']

})

②通过$parent

直接在子组件中通过this.$parent得到调用子组件的父组件

子组件--》父组件

①events up

步骤1：在父组件中 调用子组件的时候 绑定一个自定义事件 和 对应的处理函数

methods:{

recvMsg:function(msg){

//msg就是传递来的数据

}

}，

template:'

<son @customEvent="recvMsg"></son>

'

步骤2：在子组件中 把要发送的数据通过触发自定义事件传递给父组件

this.$emit('customEvent',123)

②$refs

reference 引用

步骤1：在调用子组件的时候 可以指定ref属性

<son ref='zhangsan'></son>

步骤2：通过$refs得到指定引用名称对应的组件实例

this.$refs.zhangsan

兄弟组件通信

步骤1：创建一个Vue的实例 作为事件绑定触发的公共的对象

var bus = new Vue();

步骤2：在接收方的组件 绑定 自定义的事件

bus.$on('customEvent',function(msg){

//msg是通过事件传递来的数据 (传递来的123)

});

步骤3：在发送方的组件 触发 自定义的事件

bus.$emit('customEvent',123);

每日一练：

创建2个组件，main-component,son-component

视图：

main-component 显示一个按钮

son-component 显示一个p标签

功能：

main-component 定义一个变量count,初始化为0，将count传递给son-component,son-component接收到数据显示在p标签中

main-component 在点击按钮时，实现对count的自增操作，要求son-component能够实时显示count对应的数据

son-component在接收到count之后，在count大于10的时候，将main-component的按钮禁用掉

（参考：<button v-bind:disabled="!isValid">clickMe</button>）

## SPA的概述

SPA（single page application）单页面应用程序，在一个完成的应用或者站点中，只有一个完整的html页面，这个页面有一个容器，可以把需要加载的代码片段插入到该容器中。

SPA的工作原理：

http://127.0.0.1/index.html#/start

### ①根据地址栏中url解析完整的页面：index.html

加载index.html

### ②根据地址栏中url解析#后的路由地址： start

根据路由地址，去在当前应用的配置中 找该路由地址的配置对象去查找该路由地址 所对应的模板的页面地址

发起异步请求加载该页面地址

### 把请求来的数据加载到指定的容器中

## 十、通过VueRouter来实现一个SPA的基本步骤

### ①引入对应的vue-router.js

### ②指定一个盛放代码片段的容器

<router-view></router-view>

### ③创建业务所需要的各个组件

### ④配置路由词典

每一个路由地址的配置对象（要加载哪个页面...）

const myRoutes = [

{path:'/myLogin',component:TestLogin},

{path:'/myRegister',component:TestRegister}

]

const myRouter = new VueRouter({

routes:myRoutes

})

new Vue({

router:myRouter

})

### ⑤测试

在地址栏中 输入对应的不同的路由地址 确认是否能够加载对应的组件

练习：

需要大家创建一个SPA，这个SPA有3个组件，分别对应的是collect/detail/order

功能需求：

在地址栏中路由地址是:

/myColllect --> 收藏页组件

/myDetail --> 详情页组件

/myOrder --> 订单页组件

## 十一、通过VueRouter来实现组件之间的跳转

提供了3种方式实现跳转：

### ①直接修改地址栏中的路由地址

### ②通过router-link实现跳转

<router-link to="/myRegister">注册</router-link>

### ③通过js的编程的方式

jumpToLogin: function () {

this.$router.push('/myLogin');

}

## 十二、通过VueRouter来实现组件之间的跳转：参数的传递

login ---用户名---》main

### ①明确发送方和接收方

### ②配置接收方的路由地址

{path:'/myTest',component:TestComponent}

-->

{path:'/myTest/:id',component:TestComponent}

### ③接收方获取传递来的数据

this.$route.params.id

### ④跳转的时候，发送参数

this.$router.push('/myTest/20')

<router-link :to="'/myTest'+id">跳转</router-link>

### 练习：

视图：有两个组件

product-list ： 构造一个具有5条数据的数组，显示在列表中，每个列表项都有一个超链接

product-detail:p 显示字符串：这是id为\*\*的详情信息

功能：点击product-list的列表项，跳转到product-detail，并将数组中该元素的下标发送给product-detail，product-detail接收到数据显示在p标签。

#### ①明确发送 接收

发送：list

接收：detail

#### ②配置接收方的路由地址

{path:'/detail',} --》 {path:'/detail/:id',}

#### ③接收方取值

this.$route.params.id

#### ④发送参数

<router-link :to="'/detail/'+index"></router-link>

this.$router.push('/detail/20')

## 十三、路由设置高级用法

### alias 别名

{path:'/list',component:MyList,alias:'/lists'}

### redirect 重定向

{path:'/productList',redirect:'/list'}

### path:'\*' 异常处理

{path:'\*',component:'NotFound'}

## 十四、路由嵌套

举个例子：

163邮箱为例，

login/mail(根据需求 加载inbox或者outbox)

login mail

--inbox

--outbox

路由嵌套的SPA实现的步骤：

A（/a）组件需要嵌套B组件(/b)和C组件(/c)

### ①准备嵌套其它组价的父组件 指定一个容器

在A组件指定一个容器

<router-view></router-ivew>

### ②在A组件的路由配置对象中指定children属性

{

path:'/a',

component:A,

children:[

{path:'/b',component:B},

{path:'/c',component:C},

]

}

**补充：**

//数字如果超出记录的次数，是不行的。

this.$router.go(num);

如果num是正数，向前进

如果num是负数，向后退

## 十五.网络请求

需要vue-resource.js插件

this.$http

.get('url')

.then(function(response){

// response.data 就是服务器端所返回的数据

})

## 十六、综合组件之间的通信

实现一个ToDoList。

①完成所有的组件的创建和使用

②add

点击add按钮时候，将用户输入的内容（todoinput），显示在（todolist）

### 核心代码：兄弟组件间通信

#### 步骤1：var bus = new Vue()

#### 步骤2：在准备接受数据的组件

**bus.$on('addEvent',function(){**

**})**

#### 步骤3：触发事件

**bus.$emit('addEvent',123)**

将todolist中数组的元素 渲染在todoitem的每一个span标签。（父子组件通信）

③delete

在todoitem中点击delete按钮时，将该todoitem删除，由于todoitem的数量 取决于 todolist中数组

子组件 和 父组件通信、

## 十七、基于Vue-Cli所创建的模板项目使用方式

### 好处：

#### ①当代码修改，直接预览效果

hot reload

#### ②框架已经搭建好了，快速进入到开发

#### ③组织代码的方式

将组件通过.vue的文件 单独组织 放在src目录components

### 1、webstorm设置

为了提高webStrom在Vue项目中响应速度，设置：

选中项目，点击右键，mark directory as,excluded

进入到CLI：

windows+R

cmd

d:

cd vue\_cli

npm run dev

### 2、介绍单文件组件概念

文件扩展名为 .vue 的 single-file components(单文件组件) ；

有3个标记：template(是html) script（组件的设置） style（样式的设置）

### 3、vue项目启动流程

在执行npm run dev的时候，会去在当前文件夹下的项目中找package.json文件,启动开发服务器，默认端口是8080；

找到src的main.js文件，在该文件中new Vue的实例，要加载的模板内容App，App是src目录下的App.vue结尾的文件，在App.vue所对应的模板当中，有一个router-view

在src目录下有一个router文件夹，该文件夹有个index.js文件，该文件是配置路由词典，指定了路由地址是空，加载Hello组件

### 4、使用Vue的组件

#### 步骤1：创建.vue结尾的文件

MyHeader.vue

#### 步骤2：创建组件

在main.js文件中，

##### ①引入要使用的组件

import MyHeader from '@/components/MyHeader'

##### ②创建组件

Vue.component('my-header',MyHeader)

#### 步骤3：使用组件

MyList.vue

<template>

<my-header> </my-header>

</template>

### 5、Vue项目 如何 部署在真正的服务器端

在工程所在的目录中，执行npm run build，

会生成一个dist的文件夹

就是我们可以直接部署（deploy）在服务器端的文件。

练习：

创建两个组件，my-login,my-main,要求，当路由地址是空或者'/login'的时候，访问my-login组件；

当路由地址是'/main'，访问my-main组件。

步骤1：

在components创建中创建MyLogin.vue,MyMain.vue

步骤2：

配置路由词典

参数传递：

①明确发送 接收

login main

②配置接收方路由地址

{path:'/main'}

-->

{path:'/main/:uName'}

③在接收方接收数据

this.$route.params.uName

④发送参数

this.$router.push('/main/michale')

## 资料

①安装nodejs

https://nodejs.org

下载nodejs，安装的时候，点击同意，一路next就可以。

会自带一个npm（npm是个包管理器，有啥用？是个仓库，需要用到啥 直接npm install packageName就可以了）

②安装cnpm

npm install -g cnpm --registry=https://registry.npm.taobao.org

③安装vue-cli

windows电脑，同时按下widnows+r，输入cmd，点击回车，进入到命令行模式；

在命令行中输入以下指令，最后点击回车

cnpm install --global vue-cli

④根据vue-cli安装后提供的vue指令创建模板项目

使用老师提供的压缩包(tpls.zip)

⑤安装项目需要用到的依赖

将上述文件夹解压缩之后，进入到该目录中执行指令：

cnpm install

⑥执行指令，启动开发服务器

npm run dev

# Angular

## 一、AngularJS概述

### 官网：

https://angular.cn/ 中文

https://angular.io/ 英文

哪些公司在使用AngularJS？

Google、网页版微信、锤子手机。。。

### what?

#### 使用的angular是哪个版本？ ng4.0 与老版本的区别

①ng2.0之后所有的版本 基本上 没有大的变化

②ng1.\* 与 ng2.0之后所有版本的区别 就像是雷锋和雷峰塔的关系

③命名：将angular1.\* 称之为angularJS,将angular2.\*以及以后所有的版本统称为angular

④angular引入了component的概念，舍弃了老版本的$scope/controller..

⑤angular速度更快

⑥angular的代码由于是基于组件化的开发，代码的可读性非常好，调试方便，测试简单

#### angular是一个Google推出的js的框架，

### when?

主要实现SPA（single page application 单页面应用程序），用在数据操作比较频繁的场景；是实现大而全的平台，能够实现移动端和桌面端的开发

### why?

angularJS在一些传统企业中依然在大量使用，angular主要是用在新兴的互联网公司。

#### ①迎合开发的潮流，引入了组件的概念

（提高了代码的复用率，降低了测试难度，提高了开发速度）

#### ②代码具有更强的可读性和可维护性

#### ③angular引入了更优秀的提高性能的工具，让开发和部署更简单，让应用程序速度更快

#### ④angular采用的语言是 typescript

ts是由微软和google合作联合推出的一种基于es6的超集的语言；支持了新特性，比如：js是弱类型检查语言，而ts是强类型检查语言，让代码更健壮！

### how?

如何来搭建开发环境？

提供两种方式，基于cli：

#### 方式1：官方推荐

Angular CLI是一个命令行界面工具，它可以创建项目、添加文件以及执行一大堆开发任务，比如测试、打包和发布。

①安装nodejs

如果node -v ; npm -v能够显示对应的版本信息，已经安装，

否则nodejs.org官网下载最新版本，一路next去安装

②通过npm安装 angular cli

npm install -g @angular/cli

③通过安装的cli去创建一个项目

ng new my-app

④启动开发服务器

cd my-app

ng serve --open

#### 方式2：

借助于angular所提供的quickstart

(17,819 个文件，2,434 个文件夹)

①下载快速开始用到的文件

https://codeload.github.com/angular/quickstart/zip/master

②解压缩

安装该项目依赖的所有的资源文件

npm install

（将nodeModules.zip文件解压缩，将拷贝到quickstart-master文件夹中）

③启动开发服务器：

npm start

## 二、AngularJS开发基础知识

### 1、执行npm start时，Angular是如何启动？

在启动开发服务器的时候，会去寻找src目录中main.ts文件；

在main.ts文件中，指定要启动一个模块AppModule；

在src目录中有一个app目录，有一个app.module.ts文件；

在app.module.ts文件中，指定bootStrap要启动哪一个组件？--》 AppComponent（一般情况下，都是整个应用程序的根组件）

在app.component.ts文件中，指定了template用来渲染该组件的视图（Hello Angular）

### 2、组件的创建和使用

#### 创建：

在app目录中，创建一个文件夹，在该文件夹中创建一个\*\*\*.component.ts(比如：demo01.component.ts)

在该文件中， 通过 @Component装饰当前的类为一个组件类，然后通过export导出一个类

#### 使用：

##### ①先到模块中声明

import {Demo01Component} from './demo01/demo01.component'

在 @ngModule装饰器 的declarations数组中，添加上Demo01Component

##### ②使用组件

就像使用html标签一样去使用（指定的选择器的名字）

练习：实现一个组件，demo02文件夹中，命名为demo02.component.ts,在该组件中，显示一个有5个元素的列表。

## 三.软件领域中的设计原则和设计模式

### 1、反面案例

不够灵活、不够健壮、反复的重复。。。

### 2、设计原则

设计原则：前辈帮总结好的编程技巧、经验

#### ①YAGNI：

you aren't gonna need it

不写不需要的代码，不要把我们认为会出现的需求写到项目中来

#### ②KISS：

keep it simple and stupid

代码越简单越好，傻瓜式，遵循命名规则，加上合理的注释

#### ③DRY：

don't repeat yourself

不要重复自己，善于封装和重用

#### ④高内聚低耦合：

耦合 藕断丝连

低耦合指的就是不同的模块之间关系 要尽可能的少；将降低耦合度的过程，叫做解耦

内聚：一个模块内部之间组件或者方法等之间的关系

耦合度过高会带来什么问题？

比如淘宝，有配送模块，有用户模块，有商品模块，有订单模块，如果配送模块和商品模块有着非常紧密的关系，万一配送模块出了问题，在配送模块进行升级和维护时候，一不小心改到了商品模块，就有可能导致其他问题的出现

#### ⑤SRP

single responsibility principle

单一责任原则

指的是可以把每一个功能都封装在一个方法或者一个文件中，要尽量做到每一个文件、每一个方法都只做一件事

#### ⑥OCP

open close principle

开闭原则

开放：扩展

关闭：修改

#### ⑦最少知识法则

封装的功能单位要小，尽量只与自己相关（和高内聚低耦合、SRP有点类似）

昨天学习了angular的概述、组件创建和使用

## 四、angular所支持的常见指令

### 知识点回顾：

vue

v-for v-if v-else v-else-if

v-bind(简写:src) :class="{myRed:true}"

v-on( 简写 @click)

v-show

v-model 双向数据绑定的指令

在angular中，指令有很多类型，其中一种称之为结构型指令，它在使用的时候，将影响到html的结构。

### 1、循环指令

<any \*ngFor="let tmp of list"></any>

<any

\*ngFor="let tmp of list;let myIndex=index"></any>

作用：将遍历集合（数组、对象）中的每一个元素，

根据元素的个数，来创建多个标签（调用该指令的标签）

### 2、选择指令

<any \*ngIf="expression"></any>

根据表达式执行的结果的真假 来决定是否要将该元素挂载到DOM树

### 3、多重选择指令

\*ngSwitchCase 是多重判断的一个分支

\*ngSwitchDefault 如果不是switchCase当中任何一个，是多重判断的最后一个处理

answer = 'a';

<div [ngSwitch]="answer">

<p \*ngSwitchCase="'a'"> A </p>

<p \*ngSwitchCase="'b'"> B </p>

<p \*ngSwitchDefault> 请输入正确的答案 </p>

</div>

练习：

创建一个组件，demo04.component.ts

在这个组件中，有一个变量uName，希望在视图中根据uName来实现一个列表项（如果uName是lucy，显示'Hello lucy';如果uName是mike，显示‘ 你好，mike’；否则显示‘hello’）;

有一个学生数组（对象数组有5条数据），每一个对象都包含stuName、stuScore、stuAddr；将这5条数据显示在table中，而且只显示成绩在80分以上的数据

### 注意事项：

不允许把结构型指令(\*开头的指令) 用在同一个元素上边.

解决方案：是借助于**ng-container**来实现，ng-container在使用的过程中 并不会引入一个新的html层级

### 4、事件绑定的方式（圆括号）

<any (eventName)="处理函数"></any>

比如：

<button (click)="handleClick">clickMe</button>

### 5、属性绑定的方式 （中括号）

<any [attr]="变量"></any>

比如：

<img [src]="imgUrl"/>

<a [href]="linkUrl"></a>

动态绑定样式类或者样式

ngClass

ngStyle

<!--借助于ngStyle来实现样式的动态修改-->

<p [ngStyle]="{color: myColor}">hello,i'm a paragraph</p>

<!--借助于ngClass来实现样式类的动态绑定-->

<h2 [ngClass]="{myBlue:false}">myColor will change</h2>

扩展：如果要给一个组件指定某一个css文件，怎么指定？？

@Component({

//指定要加载的模板的页面地址

templateUrl:''

styleUrls:['./test.css','./hello.css']

})

### 6、插值表达式

<any>{{表达式}}</any>

双花括号、插值表达式、mustache、interpolation

---》

指的都是双花括号

将表达式执行的结果 输出到当前元素的innerHTML中

练习：

创建一个组件demo06.component.ts

该组件要求它的模板内容 存储在一个html文件中；

模板内容：有一个button，有一个p标签

功能：点击button，启动一个定时器，定时器处理函数中修改一个变量，这个变量会动态修改p标签的透明度（0~1，继续从0~1）

(click)

[ngStyle]

### 7、双向数据绑定的指令

**ngModel指令 属于FormsModule（表单模块）**；

在app.module.ts默认只依赖browserModule，如果要使用其他模块，直接import引入。

#### ①在app.module.ts文件中 引入

import {FormsModule} from '@angular/forms';

#### ②在app.module.ts文件中 通过imports数组来指定当前模块所依赖的其它模块

@NgModule({

imports:[FormsModule]

})

#### ③使用双向数据绑定指令

<input type='text' [(ngModel)]="myMsg"/>

注意：

如何给表单元素监听用户的操作结果？

绑定ngModelChange事件

练习：创建一个组件demo08.component.ts

组件的模板中：

功能1：双向数据绑定

有一个复选框，有一个按钮，当复选框选中，按钮可以被点击，否则禁用

功能2：ngModelChange

有一个输入框，在输入框有一个无序列表，要求无序列表显示用户输入过的所有的信息

补充：

typescript 常见基本数据类型：

boolean/number/string/number[]/string[]/any..

定义一个整型的数据count

count:number = 0;

编写具有typescript语法的代码，浏览器默认能识别吗？

tsc :typescript compiler 将typescript转换成浏览器能识别的js的语法

## 总结：

①设计原则

7大设计原则

②angular提供的常见指令

\*ngFor

\*ngIf

\*ngSwitchCase

\*ngSwitchDefault

() 事件绑定

[] 属性绑定

{{}} 将双花括号中表达式的结果输出到当前元素的innerHTML

[ngClass] 动态样式类的绑定

[ngStyle] 动态样式的绑定

[(ngModel)]="kw" 前提：在app.module.ts中指定依赖于FormsModule

(ngModelChange)=""

## 知识点回顾：

Vue路由模块：建立url和代码片段之间的映射关系

### ①SPA工作原理

什么SPA？

单一页面应用程序，只有一个完整的html页面，其余都是代码片段

SPA的工作原理？

http://localhost/index.html/#/start

①解析完整的index.html,加载（一般情况，该页面都会有一个用来盛放代码片段的容器）

②解析路由地址 start

③会从路由词典中 找到start所对应的配置信息

④加载对应的代码片段 显示出来

### ②vue去实现SPA基本步骤

①引入vue-router.js

②指定容器 <router-view></router-view>

③创建业务所需要用到的各个组件

④配置路由词典

new Vue({

router:new VueRouter({

routes:[

{path:'/start',component:StartComponent},

{path:'/login',component:LoginComponent}

]

})

})

⑤测试

在地址栏中 取输入对应的路由地址 ，检测是否能够加载所设置的组件

### ③跳转的方式

①修改地址栏

②js

this.$router.push('/login')

③超链接

<router-link to="/login">去登录</router-link>

### ④传参的方式

①明确发送 接收方

②配置接收方的路由地址

{path:'/start',component:StartComponent}

--》

{path:'/start/:id',component:StartComponent},

③接收参数

this.$routes.params.id

④发送参数

this.$router.push('/start/10')

<router-link to="/start/20">去登录</router-link>

### ⑤路由嵌套

什么是路由嵌套？

在一个模板中 嵌套另一个模板

实现的方式：

①指定一个容器

<router-view></router-view>

②配置路由地址

const myRoutes = [

{path:'/login',component:LoginComponent},

{path:'/mail',component:MailComponent,

children:[

{path:'/inbox',component:InboxComponent}

]

}

]

## 五、angular中路由模块的使用

路由模块：建立起url和组件之间的关系

补充：在angular中，框架会自动解决模块之间依赖关系

### 1、实现SPA的基本步骤

#### ①指定模块 依赖于 路由模块

在app目录下，创建一个app.router.ts,

通过插件中的快捷方式a-module-routing ，创建一个自定义模块，已经指定依赖于Routes,RouterModule

①引入组件

②在对象数组中配置

③forChild --> forRoot

④修改了导出的自定义模块类名称 AppRoutingModule

找到app.module.ts,引入AppRoutingModule并在imports数组中指定依赖于AppRoutingModule

#### ②需要有一个容器用来盛放加载的组件

<router-outlet></router-outlet>

#### ③创建所需要用到的组件

#### ④配置路由

const routes:Routes = [

{path:'',component:Demo09Component}

]

#### ⑤测试

练习：

实现一个SPA，创建一个文件夹 demo10\_spa,有两个组件

demo10\_login.component.ts->Demo10LoginComponent

demo10\_main.component.ts --》Demo10MainComponent

要求：

当路由地址为空或者login显示 Demo10LoginCompoent

当路由地址为main,显示Demo10MainComponent

当地址栏中路由地址 不是配置过的任何一个，就通过\*\*来设置要进行异常处理时 要加载的组件，

注意：一般放在数组的最后一个元素的位置

### 2、跳转的方式

#### 方式1：直接修改地址栏（开发人员自己测试）

#### 方式2：通过js的方式

##### ①import {Router} from '@angular/router'

##### ②在构造函数中 完成变量定义

constructor(private router:Router){

}

##### ③调用方法来实现跳转

jump(){

this.router.navigateByUrl('/login');

}

#### 方式3：通过超链接

##### <a routerLink="/login">跳转到login</a>

#### 补充：

在angular如何来通过路由模块，实现返回上一页的功能？

##### ①引入Location

import {Location} from '@angular/common'

##### ②定义Location类型的变量

constructor(private myLocation:Location){

}

##### ③实现返回上一页

this.myLocation.back();

#### 通过Router中的navigate方法来跳转？

##### ①引入Router

import {Router} from '@angular/router'

##### ②定义Router类型的变量

constructor(private router:Router){

}

##### ③跳转

this.router.navigate(['/login'])

练习：实现一个SPA

创建一个文件夹demo12\_jump，在该文件夹中创建3个组件，

notFound.component.ts

(有一个h1:404，有一个button：返回上一页)

-->NotFoundComponent

demo12List.component.ts

(列表页面：渲染一个有5个元素的列表，要求点击列表项 通过js的方式跳转到详情页面；超链接，点击也可以跳转到详情页面)

-->ListComponent --》myList

demo12Detail.component.ts（

详情页面：显示h1标签；超链接，故意给个错的path）

-->DetailComponent -->myDetail

### 3、传参

#### ①理清楚发送 接收

Demo13StartComponent Demo13DesComponent

#### ②配置接收方

找到接收方 所对应的路由地址

{path:'myDes',component:Demo13DesComponent}

-->

{path:'myDes/:uname',component:Demo13DesComponent}

#### ③接收参数

import {ActivatedRoute} from '@angular/router'

constructor(private aRouter:ActivateRoute){

}

this.aRouter.params.subscribe((data)=>{

this.rcvMsg = data.uname;

})

#### ④发送参数

this.myRouter.navigateByUrl('/myDes/123')

<a [routerLink]="'/myDes/'+myInput">跳转</a>

#### 补充：

##### 异步请求的方式？

###### ①ajax

###### ②事件绑定

###### ③promise

###### ④rxjs 是由微软所推出的基于消息的订阅机制

observable/subscribe

##### 传参的另外一种方式：

###### ①明确发送方和接收方

###### ②到接收方准备接收参数

ActivatedRoute

###### ③ 发送参数

this.router.navigate(['/myWork',{id:2,uname:'zhangsan'}])

### 4、路由嵌套

{

path:'mail',

component:MailComponent,

children:[

{path:'',component:InboxComponent},

{path:'inbox',component:InboxComponent},

{path:'outbox',component:OutboxComponent}

]

},

和Vue中不一样的地方：

@Component({

selector: 'mail',

template:`

<a routerLink="/mail/inbox">收件箱</a>

<a routerLink="/mail/outbox">发件箱</a>

<a routerLink="/mail/trash">垃圾箱</a>

<router-outlet></router-outlet>

`

})

综合练习：

创建一个文件夹demo15\_homework，创建3个组件

check.component.ts

CheckComponent /storeCheck

有一个按钮，点击发送10给detail

detail.component.ts

DetailComponent /storeDetail

detail接收数据显示在该页面，放上一个超链接，点击跳转到send

send.component.ts

SendComponent /storeSend

button 点击跳转到check

## 回顾Vue组件间通信的方式

### 方式1：props down

Vue.component('parent',{

data:function(){

return {

sonName:'zhangsan'

}

},

template:`

<son :myName="sonName"></son>

`

})

Vue.component('son',{

props:['myName'],

template:`

<h1>{{myName}}</h1>

`

})

### 方式2：events up

Vue.component('parent',{

data:function(){

return {

sonName:'zhangsan'

}

},

methods:{

recvMsg:function(data){

}

},

template:`

<son @customEvent="recvMsg"></son>

`

})

Vue.component('son',{

methods:{

sendToFather:function(){

this.$emit('customEvent','hello father');

}

},

template:`

<button @click="sendToFather">send</button>

`

})

### 方式3：$refs $parent

Vue.component('parent',{

template:`

<son ref="mySon"></son>

`,

mounted:function(){

//得到子组件的实例 this.$refs.mySon

}

})

Vue.component('son',{

methods:{

getFather:function(){

//得到父组件的实例 this.$parent

}

},

template:`

<button @click="getFather">send</button>

`

})

### 方式4：兄弟组件通信

var bus = new Vue();

Vue.component('xiong-da',{

mounted:function(){

bus.$on('msgToXiongDa',function(data){})

}

})

Vue.component('xiong-er',{

methods:{

sendToXiongDa:function(){

bus.$emit('msgToXiongDa','...')

}

}

})

## 六、组件间通信

### 1、props down

#### 在父组件中：

<son myName="zhangsan"></son>

#### 在子组件中：

import {Input} from '@angular/core'

export class Demo16SonComponent{

@Input() myName="";

}

### 2、events up

event:

#### ①绑定事件

##### 在父组件中：

###### 步骤1：定义一个事件处理函数

recvMsg(data:any){

}

###### 步骤2：绑定事件处理函数

<son (customEvent)="recvMsg($event)"></son>

#### ②触发事件

##### 在子组件中：

###### 步骤1：

import {Output,EventEmitter} from '@anular/core'

###### 步骤2：

在该组件类中

@Output() customEvent = new EventEmitter();

###### 步骤3：触发事件

this.customEvent.emit('hello father');

练习：

创建一个文件夹 demo17\_tongxin

demo17-main.component.ts

Demo17MainComponent 路由地址：demo17Main

demo17-header.component.ts

Demo17HeaderComponent

功能要求：

在Demo17HeaderComponent组件中，定义一个判断用户是否登录的方法，返回true、false

在Demo17MainComponent组件中 是由Demo17HeaderComponent和一个列表构成的，如果用户已经登录，加载列表（模拟3条数据），否则显示一个<h1>提示请登录

核心步骤：将子组件中 是否已登录 传递给父组件

触发事件可以在ngOnInit中触发

### 3、父组件与子组件通过本地变量互动

<p>{{mySon.name}}</p>

<son #mySon></son>

## 七、angular管道

### 1、内置的管道

**date/json/uppercase/lowercase/**

**number/currency/percent/slice**

number:[整数部分保留最小位数].[小数部分保留最小位数]

比如：

{{price | number:'3.4'}}

currency:currencyCode:symbolDisplay:小数点

比如：

{{price | currency:'USD':true:'3.4'}}

percent：小数点

比如：

{{value | percent}}

{{value | percent:'3.4'}}

### 2、自定义管道的实现

**在angular这个框架中 有8个核心概念。**

**模块、组件、模板、元数据、指令、服务、依赖注入、数据绑定。**

#### 步骤1：创建一个文件 demo18.pipe.ts

#### 步骤2：在demo18.pipe.ts文件：

import {Pipe,PipeTransform} from '@angular/core'

@Pipe({

name:'myCurrency'

})

export class Demo18Pipe implements PipeTransform{

transform(value: any, ...args: any[]): any {

//必须有返回值

//value管道操作符前面传来的数据

}

}

#### 步骤3：在模块中 声明管道

import {Demo18Pipe} from '...'

@NgModel({

delcarations : [Demo18Pipe]

})

#### 步骤4：调用管道

{{ expression | myCurrency}}

## 八、angular服务

**组件：一个具有特定功能的，可被反复使用的视图**

**管道：本质是一个方法，用来实现对于数据的筛选 过滤 格式化**

**服务：把具有公共都需要用到的方法 封装成一个服务**

### 术语：

依赖注入:将我们需要用到的东西 注入 到当前的对象中去用，是通过提供商的设置来实现的

#### 提供商：

providers，将依赖注入到当前的对象中

providers:[LogService,ShowService,HttpService]

#### 单例模式：

在angular中 服务就是单例模式的类；单例模式指的在一个应用程序中，这个对象只保留单个实例。

### 7大设计原则：

**SRP/YANGNI/OCP/kISS/高内聚低耦合/DRY/LKP**

### 23种设计模式：

#### 总体来说设计模式分为三大类：

#### 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。

常见：单例模式、观察者模式、工厂模式、抽象工厂模式。。。

### 1、服务的创建

#### 步骤1：创建一个log.service.ts的文件

#### 步骤2：引入Injectable

import { Injectable } from '@angular/core';

#### 步骤3：在服务类中 定义需要用到的方法和数据

@Injectable()

export class LogService {

isDev:boolean = false;

constructor() { }

print(msg:string){

if(this.isDev)

{

console.log(msg);

}

}

}

### 2、服务的使用

在组件中调用服务的方式：

#### ①在组件中 引入 对应的服务类

import {LogService} from './log.service'

#### ②在组件的元数据添加一个Providers（用来指定需要提供哪些服务）

@Component({

providers:[LogService]

})

#### ③得到服务的实例

constructor(private myService:LogService){

}

#### ④调用服务的方法

this.myService.\*\*\*\*();

练习：创建一个自定义的服务

HeartBeatService,实现两个方法，一个方法开始心跳：启动了周期性定时器，每隔0.5s，在控制台输出一句话“正在心跳”

一个结束心跳：结束周期性定时器

创建一个文件夹demo20，创建一个组件 demo20.component.ts，导出的类 Demo20Component,路由地址 demo20

在这个组件中有两个按钮，希望能够分别调用服务中的开始和结束心跳的方法。

## 九、封装一个网络服务

### DI： Dependency Injection 依赖注入

ngModel FormModule

router RouterModule

http HttpModule

创建一个网络服务并去调用完整的过程？

### ①创建一个网络服务

//a-service-http

### ②在服务中封装实现网络请求的方法

在服务类中定义一个方法

sendRequest(myUrl:string){

//发起网络请求

return this.http.get(myUrl)

.map((response: Response) => response.json());

}

### ③在组件中准备调用

import {MyHttpService} from './myhttp.service'

在组件的元数据中指定提供商

@Component({

providers:[MyHttpService]

})

在构造函数中得到该服务的实例

constructor(private httpService:MyHttpService){}

### ④使用服务

this.httpService

.sendReuqest("./data.json")

.subscribe((data)=>{

})

### 总结异步请求方式：

ajax、callback（回调函数）、promise、RxJS

## 十、路由守卫

### 路由守卫：

保护指定的路由，当需要切换到该路由地址的时候，会根据指定的值的true、false来决定是否可以跳转。

### 服务知识的补充：

服务可以注入到一个组件，也可以注入到一个模块中。

如果服务通过不同组件的providers，注入到不同的组件，那么服务是被创建了很多次，每一个都是一个单独的实例，都没有关系

如果服务注入到了一个组件中，那么这个组件 由有其它的子组件构成，那么在子组件中如果引入服务类（不会通过providers去指定），那么所实例化的服务类对象 是同一个实例对象。

如果服务是注入到了一个模块中，那么该模块任何一个组件所import和实例化得到的服务都是同一个实例

### 路由守卫：

### 步骤1：创建一个服务can-enter.service.ts

//a-guard-can-activate 快速创建一个路由守卫的服务

### 步骤2：

在这个服务canActivate方法中，处理是否能够访问,通过返回值是true（允许访问）/false(禁止访问)

### 步骤3：

在app.module.ts对应的模块中 指定一个提供商属性

import {CanEnterGuard} from ''

@NgModule({

providers:[CanEnterGuard]

})

### 步骤4：

在app.router.ts对应的路由配置对象中，指定的属性

{

path:'admin',

component:AdminComponent,

canActivate:[CanEnterGuard]

}

## 十一、自定义指令

在vue中要想实现一个自定义指令

new Vue({

directives:{

change:{

bind:function(el){},

update:function(el){}

unbind:function(){}

}

}

})

在angular中实现一个自定义指令的步骤：

### ①创建一个指令类

//a-directive

import {Directive} from '@angular/core';

@Directive({

selector:[test]

})

export class TestDirective{}

### ②到模块中声明

import {TestDirective} from ''

@NgModule({

declarations:[TestDirective]

})

### ③使用

<h1 test></h1>

练习：考察自定义指令

创建一个自定义指令 changBgColor

<h3 changeBgColor="#ff0000">

hello custom directive

</h3>

指令接收到数据，修改背景色传递来的数据。

## 我们是这样写 Angular 应用的：

用 Angular 扩展语法编写 HTML 模板（angular指令、管道、插值表达式等），

用组件类管理这些模板（通过组件来指定模板显示），

用服务添加应用逻辑（将应用中 需要经常用到的逻辑封装在服务中，方便服用），

用模块打包发布组件与服务（组件、服务需要在一个模块中去创建去声明）。

## 学子商城的ng版本：

采用angular框架，

### 1、通过angular中提供的指令和插值表达式来实现组件模板

### 2、通过router来管理路由（SPA）：建立其地址栏中路由地址和组件之间的映射关系

const routes:Routes = [

{path:'',component:IndexComponent},

{path:'index',component:IndexComponent},

{path:'list',component:ListComponent},

{path:'\*\*',component:NotFoundComponent}

]

### 3、实现具体的业务逻辑

#### 3.1 把经常用到的方法或者数据 封装在服务中

创建一个MyHttp.service.ts,导出的类：MyHttpService

准备在IndexComponent调用服务：

##### ①import

##### ②在哪个地方，通过提供商指定需要依赖注入的服务

providers:[MyHttpService]

在组件中：该组件以及它的子组件可以使用该服务的实例

在模块中:一旦通过提供商注入到模块中，那么模块中所有的组件再去该服务时，无序再去指定providers

##### ③实例化

constructor(private myHttp:MyHttpService){

}

##### ④调用服务的对象

#### 3.2 实现具体的业务逻辑之前，明确具体的需求

一切以需求为导向

#### 3.3 遇到常见错误的处理方式

##### ①与服务器端通信的错误

请求的API接口要正确

请求API接口时，如果有参数，保证参数的个数和格式要符合要求

请求来的数据结果 要做一个验证，比如格式、是否为空

network（headers、response）

##### ②can't read property 'name' of undefined

是因为操作的对象或者数组，没有有效的值

##### ③DI error

dependency Injection

依赖注入出错

服务在使用之前有没有通过提供商指定(providers)、所需要用的模块有没有依赖(imports)

##### ④Unhandled Promise rejection: Failed to load \*\*\*\*

在载入某一个模板或者模板页面时候，出错了，检查

template、templateUrl有没有正确的进行赋值

##### ⑤奇怪的错误

单词拼写错误、中英文符号错误、大小写错误

##### ⑥没有指定事件的处理函数

Unhandled Promise rejection: Template parse errors:

Empty expressions are not allowed ("

### angular框架的核心知识点：

组件、模块、模板、

服务、依赖注入、

元数据、数据绑定、指令、管道

list商品列表组件

application 应用程序

IDE：(Intergarted Develop Enviroment 集成开发环境）

前端开发：webStorm、visual studio code...

Android: AndroidStudio 、Eclipse

iOS： Xcode

# WEBAPP

## 一、移动端开发的常见的3种模式(混合编程非常受欢迎)

### 1、WebApp 移动端的网页应用程序

采用前端技术：html/css/js，来实现提供了一种类似于原生应用程序体验的网页

(运行在browser)

很多基于h5的应用程序、移动端的站点（淘宝京东。。）

### 2、NativeApp 原生应用程序

采用java、oc、swift、kotlin等原生开发的语言，调用由google、apple、microsfot所提供的原生组件和服务进行编程，而生成的可以在手机的操作系统(os:Operating System)中直接运行的应用程序称之为NativeApp

Android/iOS/windowsPhone/BlackBerry

在应用商店（appStore/应用宝/华为应用商店。。）所下载的app，多数都是原生应用程序。

### 3、HybridApp 混合编程

采用原生组件 结合着 前端技能进行混合开发，所生成的应用程序 称之为 HybridApp.

常见的实现方案：phonegap/ionic/DCloud/369Cloud..

## 二、准备工作

混合编程中要使用原生组件，配置下环境。

iOS？--》iOS模拟器必须要求在macOS

Android

jetBrains/kotlin/java

准备Android的开发环境：

### ①安装jdk

java development kit （java开发工具箱）

### ②安装SDK

Software development kit （软件开发包）

在Android环境搭建的时候，并不会直接安装SDK，会去安装的是由Google所提供的供新手使用的ADT（Android Developer tools,自带安装好的sdk和eclipse）

http://www.androiddevtools.cn/ 下载Android开发用到的所有的资源

eclipse的使用：

如何显示工具条？ window->show toolbar

设置字体大小？

window==> preferences -->appereance-->colors and fonts -->basic -->text font 双击进行设置

创建一个模拟器？

找到工具条中,AndroidVirtualDeviceManager,点击new，在窗口进行设置：

①虚拟设备名字：testPhone

②选中device：4'(480\*800)

③backCamera : emulated

④指定sd：512

## 三、第一个Android原生应用程序

### 1、Android工程的创建

①点击eclipse中file --> new -->AndroidApplicationProject

②给Android工程指定ApplicationName（会显示在程序列表中） PackageName（区分其它的应用程序

com.公司或者项目名字简写.应用名字）

③设置icon图标（显示在程序列表中）

### 2、Android工程的使用

#### ①找到项目，点击右键

#### ②run as android application

作为android工程去运行

了解：（将java进行编译，最终生成一个apk（Android package）结尾的文件，通过一个叫做adb(Android debug bridge)的服务传输给模拟器或者真实手机，自动安装并调用该应用程序）

练习：

创建一个android项目，工程名字demo02，包的名字叫做com.tedu.demo02

运行在模拟器中显示： Welcome to tarena

## ionic使用常见错误：

### 1、忘记声明

Uncaught (in promise): Error: No component factory found for Demo01BasicPage. Did you add it to @NgModule.entryComponents?

app.module.ts中引入页面类，在entryComponents和declarations声明

### 2、是一个警告，提示引入了一个类，但是没有使用

tslint: ...:/xampp/htdocs/Framework\_codes/webApp/template/template/src/app/app.component.ts, line: 9

Unused import: 'Demo02AlertPage

## 复习：

### ng中router的5个知识点：

#### 1、通过ng来实现SPA的基本步骤

①创建一个自定义的模块app.router.ts，导出了一个模块类，AppRoutingModule

指定依赖于RouterModule

②在该文件中指定一个对象数组

const routes:Routes = [

{path:'',component:IndexComponent},

...

]

③在app.module.ts指定依赖于AppRoutingModule

④测试对应的路由地址能否正确加载设置过的组件

#### 2、跳转方式

##### 方式1:js

import {Router} from '@angular/router'

constructor(private myRouter:Router){

}

this.myRouter.navigate/navigateByUrl

##### 方式2：routerLink

<a routerLink=“/index”></a>

<button routerLink=“/index”></button>

##### 方式3：返回上一页

import {Location} from '@angular/common'

constructor(private myLocation:Location){

}

this.myLocation.back();

#### 3、传参基本步骤

两种方式：

##### 方式1 借助于url

###### ①明确发送方和接受方

###### ②配置接收方的路由地址

{path:'detail/:id',component:DetailComponent}

###### ③接收参数

import {ActivatedRoute} from '@angular/router'

constructor(private myAR:ActivatedRoute){}

this.myAR.params.subscribe((data:any)=>{

//data就是接收到的参数：对象

//data.id

})

###### ④发送参数

<a routerLink="/detail/5">

this.navigateByUrl('/detail/5')

##### 方式2 无需配置接收方路由

###### ①明确发送方和接受方

###### ②接收参数

###### ③发送参数

<a [routerLink]="['/detail',{id:myId}]">

this.navigate(['/detail',{id:5}])

#### 4、路由嵌套

mail:

inbox/outbox

##### ①配置准备嵌套其它组件的组件的容器

<router-outlet></router-outlet>

##### ②给要嵌套的组件 配置路由

{

path:'mail',

component:MailComponent,

children:[

{path:'',component:InboxComponent},

{path:'inbox',component:InboxComponent},

{path:'Outbox',component:OutboxComponent},

]

}

注意事项：

需要导航到发件箱

routerLink="/mail/outbox"

#### 5、路由守卫

守卫某个路由地址能否访问，比如：userCenter

##### ①将路由守卫 定义在一个服务中

CanEnterService

在该服务中要实现一个接口canActivate，如果该接口的返回值为true意味着允许访问，否则就不能访问（可以在此去跳转到某个页面进行处理）

##### ②将服务 在模块中 通过providers依赖注入到整个模块，

在该模块中，任何一个组件都可以使用该路由守卫

##### ③

{

path:'mail',

component:MailComponent,

canActivate:[CanEnterService]

}

#### 管道:

date/percent/number/currency/slice/uppercase/lowercase/json....

##### 自定义管道

transform

##### 自定义指令：

import {ElementRef} from '@angular/core'

constructor(private elRef:ElementRef){

}

this.elRef.nativeElement.style.\*\*\* = "";

自定义服务：服务是常用的方法的封装，主要是为了重用很多的业务逻辑，在使用服务的过程中，配置providers指定依赖注入该服务

移动端的3种常见开发模式、搭建Android的开发环境、完成Android工程的创建和使用

## 一、第一个HybridApp

创建Android工程之后，如何添加权限？

AndroidManifest.xml--》permissions --> add -->选中最后一个uses permission -->右侧下拉菜单，选中需要用到的权限。

Android工程的目录结构：

src: 存储的是java文件

assets:资源文件，比如要混编用到的html/css/js

res/layout: 布局文件 activity\_main.xml

AndroidManifest.xml文件中，包含版本号、权限的设置、图标的设置..

基本步骤：

①编写前端代码，完成指定的功能，保证测试是没问题

②创建一个Android项目

③将前端代码拷贝到Android工程的assets目录

④调用原生组件(WebView)来加载前端代码

3行java代码

步骤1：创建WebView类型的变量

WebView wv = new WebView(getApplicationContext());

步骤2：让wv载入前端代码

wv.loadUrl("file:///android\_asset/demo01.html");

步骤3：将载入了前端页面的wv 设置为当前的内容视图

setContentView(wv);

练习：创建一个HybridApp

要求在应用中实现：p button,点击button，隐藏p

如果混合编程应用程序 要想执行js:

wv.getSettings().setJavaScriptEnabled(true);

## 二、Ionic概述

为什么讲Ionic？因为所实现的混合编程app，已经能够在移动端的os去运行，但是整体页面效果和真正的原生的体验有点距离，可以使用Ionic这个移动端的框架。

常见的移动端的框架 :Ionic/jQueryMobile/SenchaTouch。。。

what？ 用来构建移动端应用程序的开源框架

when? 用在任何一个需要实现移动端效果的应用程序

why?

①成本低，可以使用前端技术来轻松编写出跨平台运行的应用程序

②Ionic框架基于Angular、Cordova来实现的

③支持es6、ts

④强大的命令行工具的支持

how？搭建开发环境

①通过npm去安装ionic

npm install -g cordova ionic

②通过ionic指令去创建官网所提供的模板项目

ionic start myApp blank/tabs/sidemenu

③运行项目

cd myApp

ionic serve

## 准备工作：

1、Ionic\_For\_Stu.zip解压缩，里头有个压缩template.zip文件，

2、将这个压缩包放到c/xampp/htdocs/Framewrok\_Codes/WebApp/这个位置，在当前文件夹中解压缩该压缩包

3、打开visualStudioCode，通过点击右上角的文件，选择打开文件夹，找到对应的目录

如果电脑执行ionic出错，怎么解决？

①将cmd.rar压缩包 解压缩到c/xampp/htdocs/Framewrok\_Codes/WebApp/

②拷贝ionic.cmd所在的文件夹的路径

C:\xampp\htdocs\Framework\_codes\webApp\cmd\cmd

③配置环境变量

电脑桌面，选中计算机，点击右键，查看属性，系统高级设置 --》 环境变量 --》系统Path，-->编辑，在值的最后变加上分号，在分号将ionic.cmd所在的文件夹路径信息粘贴在分号后边

做好准备工作之后，通过visual studio code,打开了放在c盘目录的template文件夹，通过ctrl+`,启动了终端，在终端中输入的指令:ionic serve ,可以启动开发服务器，端口号默认8100

## 三、启动开发服务器时，为什么显示是当前页面？

步骤1：找到index.html

步骤2：引入了main.js文件，对应的是app目录main.ts

发现启动了模块AppModule

步骤3：在AppModule中，指定要启动来自于ionic-angular中的IonicApp，而且由于在imports指定了

将MyApp作为模块的根组件

步骤4：在app.component.ts中已经通过rootPage来指定要显示的组件是HelloIonicPage，那么该页面就是当前所显示的页面

## 四、ionic的组件的使用

### ionic serve -l

注意：Ionic并不会直接创建很多个组件，通过创建page指定项目中各个页面，比如说index/detail/list....

### 1、完成page页面的创建

在vs(Visual studio code)打开新的终端，输入指令：

#### ionic g page demo01-basic

执行之后，在pages目录创建一个文件夹demo01-basic,在这个文件夹中有4个文件，

demo01-basic.ts 一个页面（可以定义数据、方法） demo01-basic.html指定页面要显示的模板内容

### 2、page页面怎么使用

①将page页面导出的类，比如Demo01BasicPage;

在app.module.ts进行声明:declarations/entryComponetns

②找到app.component.ts,引入Demo01BasicPage，设置rootPage=Demo01BasicPage

## 五、AlertController

### 步骤1：引入

import {AlertController} from 'ionic-angular'

### 步骤2：实例化

constructor(public alertCtrl:AlertController){}

### 步骤3：创建窗口去显示

let myAlert = this.alertCtrl.create({

title:'',

subTitle:'',

message:'',

buttons:[]

})

myAlert.present();

（17:10 - 17:30）

练习：创建一个page demo03-myalert

页面效果：输入框，按钮，在点击按钮时候，将用户输入的数据通过弹窗显示。

alert/prompt/confirm

## angular如何解决与服务器端通信时，由于服务器采用session所导致的问题？

两个步骤：

### ①在前端请求时，加上属性{withCredentials:true}

比如：

sendRequest(url:string){

//a-http-get

return this.http.get(url,{withCredentials:true})

.map((response: Response) => response.json());

}

### ②在后台的php文件中，加上两句话：

header('Access-Control-Allow-Credentials:true');

header('Access-Control-Allow-Origin:http://localhost:3000');

**ionic自带的图标：**

**https://ionicframework.com/docs/ionicons/**

## 六、button

### ionic自带的颜色：

primary 默认，蓝色

secondary 绿色

danger 红色

light 浅灰色

dark 黑色

### ionic中怎么自定义颜色属性？

theme-->variables.scss --> 添加颜色设置

按钮如果要符合移动端的展示效果:

<button ion-button></button>

指定颜色: color="primary/secondary/danger/light/dark"

round/outline/clear/small/large

带有图标的按钮:

<button ion-button icon-left/right/only>

<ion-icon name=""></ion-icon>

</button>

## 七、卡片 card

<ion-card>

<ion-card-header></ion-card-header>

<ion-card-content></ion-card-content>

</ion-card>

## 八、list

基本用法:默认情况每个列表项之间是有横线隔开的，

可以指定no-lines去取消横线

<ion-list>

<ion-item>test01</ion-item>

</ion-list>

分组：

<ion-item-group>

<ion-item-divider>A</ion-item-divder>

<ion-item>apple</ion-item>

<ion-item>and</ion-item>

<ion-item-divider>B</ion-item-divder>

<ion-item>apple</ion-item>

<ion-item>and</ion-item>

</ion-item-group>

在列表中通过指定item-start/item-end来调整元素显示在列表项的左边或者右边

### icon-list:一个有图标构成的列表

<ion-list>

<ion-item>

<ion-icon name="heart" item-start></ion-icon>

test

</ion-item>

</ion-list>

### avatar-list:一个有头像构成的列表

<ion-list>

<ion-item>

<ion-avatar item-start>

<img src=""/>

</ion-avatar>

h2

p

</ion-item>

</ion-list>

### thumbnail-list:一个有缩略图构成的列表

<ion-list>

<ion-item>

<ion-thumbnail item-start>

<img src=""/>

</ion-thumbnail>

h2

p

</ion-item>

</ion-list>

### sliding-list:支持侧滑的列表

<ion-list>

<ion-item-sliding>

<ion-item>test<ion-item>

<ion-item-options>

<button ion-button>btn1</button>

</ion-item-options>

</ion-item-sliding>

</ion-list>

## 九、ActionSheet

### 步骤1：import {ActionSheetController} from 'ionic-angular'

### 步骤2：实例化

constructor(public myAS:ActionSheetController){}

### 步骤3：创建

let myMenu = this.myAS.create({

title:"",

buttons:[

{

text:'',

handler:()=>{},

role:'cancel/destructive'

}

]

}

)

### 步骤4：显示

myMenu.present();

练习：

创建一个page demo09-list

视图：有一个列表，点击列表项，从下往上弹出一个菜单，点击菜单中的选项，提示警告窗口

要求：

列表是一个由5个元素构成的缩略图列表，

弹出的菜单：有2个选项分别是删除、取消，

点击删除，从列表中删除该元素，删除完之后提示警告窗：删除成功

list/actionSheet/alert/ng

## 十、fab 悬浮式操作按钮（主要是用在android设备）

<ion-fab top/bottom/left/right/center>

<button ion-fab>share</button>

<ion-fab-list side="left/right/top/bottom">

<button ion-fab>

<ion-icon></ion-icon>

</button>

</ion-fab-list>

</ion-fab>

## 十一、loading

主要是在进行数据操作、网络请求比较耗时的场景下面，通过LoadingController来实现一个正在加载的窗口

使用步骤：

import {LoadingController} from 'ionic-angular';

constructor(public loadCtrl:LoadingController){}

let myLoading = this.loadCtrl.create({

content:'',

duration:2000

})

myLoading.present();

myLoading.dismiss();

实现一个服务，sendRequest是用来处理网络请求；

调用该方法，就提示正在加载，服务器端返回数据，取消该加载中的窗口

## 十二、toast

ToastController

实例化

create({

message:'',

duration:1000,

position

})

present()

## 常见错误：

### 1、提示net::ERR\_CONNECTION\_REFUSED

?ionicplatform=ios&ionicstatusbarpadding=true:8 GET http://localhost:8100/\_\_ion-dev-server/ion-dev.js?v=1.3.7 net::ERR\_CONNECTION\_REFUSED

解决办法：查看ionic serve 所启动的开发服务器是否还在运行

tabs/grid/badge/icon/navigation/forms...

启动开发服务器：

ionc serve -l

## 十三、grid

### 基本布局：

<ion-grid>

<ion-row>

<ion-col></ion-col>

</ion-row>

</ion-grid>

col-\* 指定当前这一列的宽度占据百分比

offset-\* 距离左边列偏移量

align-items-start/center/end 调整一行中所有的列纵向对齐方式

align-self-start/center/end 调整某一列针对这一行的纵向对齐方式

justify-content-start/center/end 一行中所有的列是靠左边、中间、右边进行布局

push-3 将当前列 向右移动3/12

pull-9 将当前列 向左移动9/12

## 十四、navigation 导航

ng、vue --》路由模块

### 跳转：

#### 方式1：通过js的方式

import {TestPage} from '...'

this.navCtrl.push(TestPage)

#### 方式2：通过navPush属性

##### ①import {TestPage} from '...'

##### ②在目的地的页面类中 定义一个变量

destinationPage

##### ③在构造函数或者ionViewDidLoad处理函数对变量进行赋值

this.destinationPage = TestPage

##### ④ <button [navPush]="destinationPage"></button>

### 传参：

#### 方式1：通过js

import {TestPage} from '...'

this.navCtrl.push(TestPage,{id:3})

#### 方式2：通过属性 navParams

##### ①import {TestPage} from '...'

##### ②在目的地的页面类中 定义一个变量

destinationPage

##### ③在构造函数或者ionViewDidLoad处理函数对变量进行赋值

this.destinationPage = TestPage

##### ④ <button [navPush]="destinationPage" [navParams]="{id:myInput}"></button>

##### 接受参数：

this.navParams.get('id');

##### 返回：

this.navCtrl.pop();

练习时间：

创建两个页面 demo14-list.page.ts demo14-detail.page.ts

需求：

[123,456,789]

在list页面，实现一个具有3个元素的列表，点击列表项，将列表项中的内容 发送给detail页面；

detail页面接收到数据，显示在视图中:

LoadingController

/\*先去显示一个加载中的窗口（在前2s，显示：正在连接服务器，到了第4s，显示：连接成功，正在获取数据；到了第6s：关掉）\*/

## 3、forms 表单

复选框、单选框、下拉菜单、输入框、滑动开关、滑动组件。。

<ion-checkbox/>

<ion-radio> --> 在ion-list指定radio-group

<ion-select>

<ion-option></ion-option>

</ion-select>

<ion-label fixed/floating/stacked></ion-label>

<ion-toggle></ion-toggle>

<ion-range pin="true" step="5" snaps="true" min="0" max="100">

<ion-label range-left>0</ion-label>

<ion-label range-right>100</ion-label>

</ion-range>

## 十五、下拉刷新、上拉加载更多

### 基本步骤:

#### ①在模板内容中 去调用指定的元素

#### ②绑定事件处理函数

#### ③在处理函数中 指定刷新或者加载更多操作，通知ionic结束正在刷新的动作

### 下拉刷新：

#### ①指定模板

<ion-refresher>

<ion-refresher-content

pullingText=""

pullingIcon=""

refreshingText=""

refreshingSpinner=""

></ion-refresher-content>

</ion-refresher>

#### ②绑定事件

<ion-refresher (ionRefresh)="doRefresh($event)">

</ion-refresher>

#### ③

function doRefresh(myRefresher){

//对数据进行操作（比如从服务器端获取数据，操作本地数据等），当数据操作完之后，调用complete的方法

setTimeout(()=>{

this.myList.unshift(100);;

//结束刷新动作

myRefresher.complete();

},3000)

}

### 上拉加载更多

#### ①指定模板

<ion-infinite-scroll>

<ion-infinite-scroll-content

pullingText=""

pullingIcon=""

refreshingText=""

refreshingSpinner=""

></ion-infinite-scroll-content>

</ion-infinite-scroll>

#### ②绑定事件

<ion-infinite-scroll (ionInfinite)="loadMore($event)">

</ion-infinite-scroll>

#### ③

function loadMore(myInfiniteScroll){

//对数据进行操作（比如从服务器端获取数据，操作本地数据等），当数据操作完之后，调用complete的方法

setTimeout(()=>{

this.myList.push(100);;

//结束刷新动作

myInfiniteScroll.complete();

},3000)

}

## 十六、lifecycle 生命周期

vue ：create/mount/update/destroy

ng : onInit/onDestroy

ionic:

**ionViewDidLoad**

**ionViewWillEnter**

**ionViewDidEnter**

**ionViewWillLeave**

**ionViewDidLeave**

**ionViewWillUnload**

**ionViewCanEnter**

**ionViewCanLeave**

## ionic的组件：modal/slides/tabs

## 十七、modal 模态窗

alertController/loadingCtroller/actionSheetController/modalController(视图的自定义)

ionic的模态窗口 是从屏幕底部 划入当前视图的临时ui层，通过可以用来做登录注册、消息提示、选项操作等窗口

使用方式：

### ①引入

import {ModalController} from 'ionic-angular'

### ②实例化

constructor(prviate modalCtr:ModalController){}

### ③创建模态窗中 要显示的page或者component

LoginComponent

### ④创建模态窗

import LoginComponent from '...';

let myModal = this.modalCtr.create(LoginComponent);

### ⑤显示

myModal.present()

### 注意：

①modalController的create方法第一个参数不是一个对象，而是要显示在模态窗中的组件或者页面的类

②关掉模态窗 ViewController得到当前的模态窗口，dismiss方法

## 十八、slides

pager 显示slide一共有几个 通过进度，比如小圆点显示

paginationType:bullets\fraction\progress

autoplay=1000 自动播放

loop 指定循环播放

effect去指定slide之间切换的时候 动画效果:slide/fade/cube/flip

speed 动画效果持续的时间

direction horizontal/vertical

<ion-slides>

<ion-slide>

</ion-slide>

<ion-slide>

</ion-slide>

</ion-slides>

## 十九、tabs 标签页/选项卡

### ①创建每一个tab所点击时 要显示的page

### ②创建一个page，指定ion-tabs组件

<ion-tabs></ion-tabs>

### ③在ion-tabs中指定ion-tab绑定root属性

<ion-tabs>

<ion-tab [root]=""></ion-tab>

</ion-tabs>

### 属性： tabTitle/tabIcon/tabsHideOnSubPages

举个例子：

<ion-tabs>

<ion-tab tabsHideOnSubPages="true" tabIcon="home" tabTitle="首页" [root]="listPage"></ion-tab>

<ion-tab tabIcon="cart" tabTitle="购物车" [root]="toastPage"></ion-tab>

<ion-tab tabIcon="settings" tabTitle="设置" [root]="loadingPage"></ion-tab>

</ion-tabs>

综合练习：

创建一个移动端的应用程序，支持通过tabs进行导航，有两个tab。

有4个页面，my-app作为跟页面：product-list product-detail cart-list

product-list:

显示一个缩略图列表（5条数据）：每一个列表项显示图片、标题、描述信息

要求：第一个tab中 显示该页面；然后点击列表项，跳转到product-detail,将下标传递给详情

product-detail

显示一个h1的标签，显示一个‘添加到购物车’的按钮

要求：h1的标签要求显示传递来的参数，按钮点击，通过toast在页面底部显示一个‘添加成功’的通知

（localStorage保存序号）

cart-list

在购物车列表页面中，显示一个列表（只显示序号就可以）,记录的就是在detail添加到购物车中的数据

（localStorage 读取序号）

考察知识点：

①tabs的基本用法

②跳转和传参

③本地存储 localStorage

# React

学习安排：

①概述

②搭建开发环境

③React的核心思想和

核心概念

④React的组件的

基础知识

## 一、概述

React是是由facebook推出的,一个是通过组件化，用来实现ui层的开源的js库

### why?

①大量的数据操作，带来了大量的DOM操作,浏览器遇到了性能瓶颈问题--》虚拟DOM

②大量的数据操作背后，复杂的业务逻辑，代码的维护和开发成本提高

--》state和组件化

### React特点：

①组件化

②高效（通过虚拟DOM，最大限度的减少真实DOM的操作）

③JSX：不是一种新的语言，是对于js的扩展，允许在js中编写html

④灵活 可以非常方便的

搭配其他库来使用

### React开发环境的搭建方式：

### 方式1：

直接引入对应的js文件

### 方式2：

通过命令行的方式

//安装create-react-app

npm install -g create-react-app

//创建一个my-app的项目

create-react-app my-app

//切换到对应的目录

cd my-app/

//启动开发服务器

npm start

### 使用React时要考虑的

问题：

①React是一个相对年轻的框架，配套使用的库

并不是特别多

②并不是所有的项目 都适合React来实现

（数据操作非常频繁，页面更新非常多）

③React不支持ie8以下的任何浏览器（官方以后

也不支持）

核心思想：组件（是一个可被反复使用的带有特定功能的视图）

Everything is Component.

## 二、React开发

### 1、第一个例子

#### ①引入3个js的文件

react.js 核心文件

react-dom.js 和DOm相关的文件

browser.min.js 浏览器

#### ②指定了一个容器，用来渲染元素（容器是用来盛放根组件的）

#### ③ReactDOM.render(要渲染的元素，准备渲染的容器);

ReactDOM.render(

<h1>Hello ReactJS</h1>,

document.getElementById('example')

)

在js中编写html标签，将这种语法称之为JSX.

#### ④babel

<script type="text/babel">

</script>

babel是一个js的编译器，能够将es6的语法、或者jsx的语法转换为浏览器能够识别的js语法，从而来实现页面的展示

### 2、JSX （java script xml）

注意：JSX不是一种新的语言，只不过是允许在js中编写html

React是可以完全不使用JSX的，但是用JSX可以更简单的开发，所以还是提倡使用JSX

JSX语法 借鉴了XML标签打开与关闭的优点，提高了组件的可读性；JSX中可以直接使用HTML标签，赋予React强大的表现能力，方便进行代码的开发，最早是受php中一个插件库的影响。

基本用法：

遇到HTML标签(<),就会使用HTML来解析

遇到了代码块({),就是使用js来解析

注意事项：如果我们要渲染多个元素，要将返回的多个元素放到一个顶层标签中。

jsx练习：(11:10 - 11:20)

新建一个文件，在该文件中 指定一个容器，通过jsx语法，渲染h1(逻辑运算)、span(三目运算)

### 3、组件

#### 在Vue中 创建组件：

Vue.component()

#### 在angular中，创建组件类:

import {Component} from '@angular/core'

@Component({

selector:'',

tempalte:''

})

export class Demo01Component{

}

#### ①在React中 创建组件

//组件类的名称 必须是全驼峰（即使是第一个单词，首字母也要大写）

var ComponentName = React.createClass({

render:function(){

return ‘要渲染的元素’

}

})

#### ②在react中，使用组件

是按照html标签的方式 来调用组件

<ComponentName></ComponentName>

<ComponentName/>

React有一个特点，根据标签的首字母大小写，来决定是按照组件、html标签的方式 来解析内容

#### 组件使用的过程3个注意事项：

##### ①首字母必须大写

##### ②不允许在render方法返回时，顶层标签或者第一个元素，不允许直接换行

##### ③不允许直接返回多个元素，要放在一个顶层标签中

#### 组件的工作原理：

React允许将需要反复使用的视图通过代码块的方式 封装在一个组件中，然后像插入html标签一样，在网页中插入这个组件，将组件类进行实例化，将render方法的返回元素 渲染在指定的容器中

### 4、React中，是以组件为基本单位

复合组件：并不是一个新的概念，只不过是由其他的组件构成的，本质依然是一个组件

组件树：在实现一个页面的时候，可以将页面中有共性的部分，封装成组件，把这些组件放在一起，构造成一个最终渲染的根组件，那么这个时候，就有一颗组件树。

树的根就是最终渲染的复合组件，树的树叶、树枝就是创建的其它的组件

练习：要求创建一个登录窗口，

离线安装vscode插件的方式：

C:\Users\web-01\.vscode\extensions

将离线好的插件 解压缩到该目录中，然后重启vscode 就可以了

### 5、props

(React中的核心概念：JSX/组件/props/refs/state)

复习Vue、angular组件之间通信的方式

#### 5.1 react 父组件与子组件通信的方式

实现具体步骤：

##### ①在调用子组件时候，通过属性传值

<son tips="something"/>

##### ②在子组件内部，通过props属性接收值

this.props.tips

#### 5.2 子组件想与父组件通信的方式 props。

父组件

子组件

有数据要传递给父组件

##### ①在父组件中 定义一个用来 接收数据的方法

rcvMsg:function(data){

}

##### ②在父组件中调用子组件时，可以将父组件中的方法传递给子组件

<son callback={this.rcvMsg}>

##### ③子组件在需要给父组件传值时，调用该方法this.props.callback(this.uname)

#### 5.3 children属性

this.props.children得到在调用组件时，标签内部 所写的元素，比如:

<MyList>

<li>test01</li>

</MyList>

this.props.children 是null 是对象 是数组

React.Children.map(

this.props.children,

function(child){

}

)

### 6、refs

reference 引用/参考的意思

#### ①直接ref

<input type="text" ref="myInput"/>

#### ②通过ref去找到元素

this.refs.myInput

综合练习：

实现3个组件

MyInput 只有一个输入框

MyButton 只有一个按钮

MyMain是由上述两个组件构成的

功能：在点击MyButton 获取 MyInput用户所输入的值

思路：将MyInput的值保存在MyMain中，在MyButton点击时去MyMain中取值

MyInput OnChange

步骤1：将MyInput的值保存在MyMain中

（子组件 与 父组件通信）

①在父组件中 定义一个用来 接收数据的方法

rcvMsg:function(data){

}

②在父组件中调用子组件时，可以将父组件中的方法传递给子组件

<son callback={this.rcvMsg}>

③子组件在需要给父组件传值时，调用该方法this.props.callback(this.uname)

步骤2：在MyButton点击时去MyMain中取值

## 三、state

### 1、state 概述

组件是一个可被反复使用的视图，呈现给用户，在和用户交互的时候，就避免不了数据的变化，在react中 将数据比作是状态。

React将每一个组件 都比作是一个状态机（由各种各样的状态构成的组件）

### 2、state的3个基本操作

#### 初始化:getInitialState:function(){

return {myStatus:'吃饭'}

}

#### 读: this.state.myStatus

#### 写: this.setState({myStatus:'睡觉'},function(){

//状态设置异步操作 成功设置完成之后会调用的方法

})

#### 重要：

一旦将状态中数据 绑定到了视图，当状态中数据发生变化，视图会自动的更新

（从数据绑定到视图）

组件要显示的内容 只取决于render方法返回的内容；一旦将状态中的数据绑定到 视图，当数据发生变化的时候，react框架观察数据发生了变化，自动的采用最优算法最高效率 来完成DOM的更新(重新去执行render)

#### 注意：

不要在render方法中，做任何状态的设置。

练习：（14:17 - 14:27）

MyInput、MyButton、MyMain

按照状态的方式来处理。

## 四、lifecycle 生命周期

初始化、数据变化、消亡

React的组件 也有自己的生命周期，也有从创建到消亡的过程

分为3个阶段：

mount/update/unmount

对应的生命周期的某个阶段对应的处理函数（钩子函数）

componentWillMount

componentDidMount

componentWillUpdate

componentDidUpdate

componentWillUnmount

### 注意事项：

①如果要使用ref去寻找元素，必须等到componentDidMount

②在componentWill/DidUpdate的处理函数中是由参数的，分别对应的是准备更新/更新之前的props和state

③在一个组件中，建议：

将生命周期的处理函数 一般都是放在组件的上面，

其次自定的方法，

最下边是render方法

练习：(15:40 - 15:55)

创建一个组件并显示；组件渲染一个h1的标签，内容随便写；

要求：当组件挂载完毕之后，启动一个周期性定时器，在定时器中修改一个数据，然后实现h1的标签透明度的变化

(0~1 0~1)

提示：style={{opacity:this.state.OpacityValue}}

步骤1：完成组件的创建和显示

步骤2：初始化状态

步骤3：指定生命周期处理函数，启动定时器

## 五、表单受控元素

React中并不是所有的表单都得按照受控元素的方式去处理，这是React建议和解决方案。

什么是表单受控元素？

是一些表单元素，如果指定value/checked/selected,那么

就是表单受控元素（现象：针对表单操作的结果是不对的)

### 解决思路：

#### ①初始化状态

#### ②将状态的值 绑定属性value/checked/selected

#### ③在对应的onChange处理函数中 完成对应的状态值的修改

### 注意事项：

#### 如果一个组件包含多个状态，需要初始化：

getInitialState:function(){

return {

num1:'',

num2:''

}

}

#### 需要读取状态：

this.state.num1;

this.state.num2;

#### 设置状态：

需要修改哪个状态，就在setState方法中设置该指定的状态就可以了，其余的不需要修改的无需理会

this.setState({num1:''});

综合练习：

①完成4个组件的创建和使用

ToDoBox

ToDoInput

ToDoList

ToDoItem

②实现add

todoinput要想把数据直接给todolist，属于兄弟组件通信，借助于共同的父组件todobox

当点击add按钮时候，将用户的输入的值 传递给todobox

步骤1：在todobox中定义一个方法（参数）

步骤2：把方法通过属性传递给todoinput

步骤3：按钮单击，调用该方法

③实现delete

点击todoitem中的按钮，调用todobox中的删除数组中指定元素的方法

考察的知识点：

①组件的创建的使用

②props

③state

三个基本操作以及功能（绑定）

1. jsx
2. ref

## 三连棋游戏：

## 1、给Square组件中的按钮 指定 单击处理函数

修改Board组件中 状态的值

（子与父通信）

基本步骤：

①在父组件中 定义一个带有参数的方法

修改数组、修改xIsNext

②将带有参数的方法 传递给子组件

③在子组件中 调用该方法

要求：在状态设置成功的处理函数中 打印当前的数组

## 2、判断输赢

在组件更新完成之后 去判断-输赢

componentDidUpdate

将下棋结果通过弹窗显示(\*\*\*赢了、平局)

# ReactNative

## 一、ReactNative的概述

移动端开发的3种常见模式：

WebApp/NativeApp/HybridApp

### React:

(由React提出的核心概念：Component、JSX、Props、State、Refs)

ReactJS：是一个在web前端中实现层ui的js库

ReactNative：15年 由fb推出的ReactNative是一个用前端技能（React中的核心概念）来编写原生应用程序;在实现基于ReactNative的app时，主要是调用由ReactNative所封装的各种各样的组件（在被渲染时，是会变成真正的原生组件）

### 强调：

ReactNative不是用来实现webApp、HybridApp，是用来实现nativeApp,即使没学过原生的Android\iOS开发，依然可以使用React中的核心概念来完成app

### why?

free\open source

成本低（学习成本、时间成本）

nativeApp(性能要好)

learn once,write anywhere

### where?

低成本的开发真正的原生应用程序

### how？

搭建ReactNative的开发环境

#### ①方式1 cli

//安装

npm install -g create-react-native-app

//创建一个叫做my-app的项目

create-react-native-app my-app

//切换到当前目录下的my-app

cd my-app/

//执行start脚本，启动开发服务器

npm start

#### ②方式2 cli (建议)

//安装 对应的命令行工具 react-native-cli

npm install -g react-native-cli

//安装yarn

npm install -g yarn

//创建项目(会去自动的安装依赖的包)

react-native init my-app

//切换到my-app

cd my-app

//启动开发服务器

react-native start

#### 大致步骤：

①下载ADT

（Android Developer Tools 内置了eclipse/sdk）

②配置环境变量

ANDROID\_HOME(将c盘adt-bundle-window\*\*,中的sdk的路径赋值给ANDROID\_HOME)

(计算机-》右键-》属性-》查看高级设置->环境变量-》创建或者编辑或者删除)

③安装Java运行环境和工具包

一般情况下，安装之后系统的环境已经自动的设置过了

④处理和ReactNative

启动开发服务器：

①进到e盘，找到software，找到RNTPL，点进去，找到myapp

②进入到myapp文件夹，点击右键，选择git bash here

去启动了一个终端（有点类似windows+R->cmd）

③在终端中，react-native start

如果能够成功的启动8081端口，说明环境是ok的，否则环境是有问题的

#### 环境：

①pc端

react-native start 启动开发服务器

在编辑器中 编写项目中的代码

②mobile移动端

安装一个myapp的安卓系统下的应用程序

打开该应用程序，设置要连接的服务器端的ip地址和端口号（一旦连接成功，可以实现当pc端修改reactnative

项目代码，移动端可以直接去预览）

如何将一个apk结尾的文件 安装到模拟器中？

①服务器

将app-debug.apk放到htdocs

②指令

adb(android debug bridge) 实现模拟器或者真机 和pc的连接

adb install app-debug.apk

步骤1：

找到adb所在的文件夹

c:\adt-bundle-windows\*\*\*\*\adt-budnle\*\*\*\sdk\platform-tools

步骤2：

windows+R --> cmd ,将上一步找到的adb.exe直接拖拽到该命令行工具

步骤3：在命令行中 输入 空格 install 空格

步骤4：找到c xampp htdocs目录下的apk文件夹中的app-debug.apk，直接拖拽到install空格的后面，点回车即开始安装

（已经安装过，重复安装是会报错的；没有启动模拟器，安装也会失败）

## 二、ReactNative的知识点

学习ReactNative所封装的常用组件，包括View/Text/TextInput/Modal/...

需要将模拟器中的myapp软件 连接当前pc端的通过reactnative start所启动的开发服务器

①cmd->ipconfig

记录当前pc端的ip地址

172.163.0.155

②打开模拟器中的myapp，点击menu按键，选中Dev Settings,设置Debug Server Host&Port

如果模拟器连接debug server失败：

①debug server 是否在运行

②确认下在模拟器中通过浏览器 访问debug server比如(172.163.0.155:8081)

③在myapp这个软件中，点击menu按键，查看debug js remotely选项，如果当前处于stop remote js debugging,点即该选项把远程调试 关掉，再去重新reload

④重启模拟器

### 1、常用reactNative组件类

View 指定一个容器

Text 指定一段文本内容

StyleSheet 指定样式：

①import {StyleSheet} from 'react-native'

②根据需要去创建要用到的样式类

const myStyles = StyleSheet.create({

myText:{

color:'red'

},

myView:{

backgroundColor:'#eeeeee'

}

})

③使用样式类

<Text style={myStyles.myText}></Text>

注意事项：

①在ReactNative中指定宽高等数据时候，无需加上单位，reactNative会帮助我们自动处理

②指定样式时候，要采用驼峰式的写法

backgroundColor

fontSize

任务：复习css core day1 flexbox

## 准备环境

①启动开发服务器

找到e\software\RNTPL\myapp ，点击右键，git bash here,

在命令行窗口中：react-native start

②打开eclipse，找到avd(android virtual device manager安卓虚拟设备管理器),启动模拟器

## 三、Button

使用步骤：

### ①引入

import {Button} from 'react-native';

### ②使用

<Button title="clickMe" onPress={this.handlePress}></Button>

## 四、State

### ①在构造函数中去初始化状态

constructor(){

super();

//初始化状态

this.state={count:0}

}

### ②读

<Text>{this.state.count}</Text>

### ③设置

this.setState({count:1})

## 五、Image

### ①引入对应的组件类

import {Image} from 'react-native';

### ②根据要加载图片的路径

#### 本地图片:

<Image source={require('../img/1.jpg')}/>

#### 网路图片

<Image source={{uri:'\*\*\*\*.com/\*\*\*.png'}}/>

## 六、TextInput

使用方式：

### ①引入

import {TextInput} from 'react-native'

### ②使用

handleChange=(data)=>{

console.log(data);

}

<TextInput

onChangeText={this.handleChange}

placeholder="请输入用户名"/>

练习：

新建一个文件夹demo07，在该文件夹中创建一个文件demo07\_lianxi.js

在该文件中，创建一个基于ReactNative的一个组件，渲染一个TextInput,要求默认显示'zhangsan'

按照表单受控元素的方式来处理

常见错误：

unable to connect the remote debugger

①查看debugger service是否在运行

1. 检查模拟器 通过浏览器 测试 是否能够连接开发服务器，如果不行,重启模拟器

## 七、FlatList

### ①引入

import {FlatList} from 'react-native'

### ②调用

this.list = [];

showListItem=()=>{

//必须要返回组件

return <Text/>

}

<FlatList

data={this.list}

renderItem={this.showListItem}

/>

## 八、Fetch接口的使用

fetch('url')

.then((result)=>{return result.json()})

.then((result)=>{

//result就是服务器端返回的json格式的数据

})

## 九、TouchableOpacity

给容器中的元素 添加 触控或者点击时 透明度变化的效果

### ①引入

import {TouchableOpacity} from 'react-native'

### ②直接使用（作为容器去使用）

<TouchableOpacity>

<Text></Text>

<Image source={{uri:''}}/>

</TouchableOpacity>

练习：

创建一个demo11的文件夹，在该文件夹中创建一个demo11\_lianxi.js的文件

FlatList --> 列表项 Text (在点击Text时有透明度的变化)

当组件挂载完毕之后，向服务器发起请求，将请求到的数据渲染在列表中，当点击列表时候，弹窗显示当前的的列表项的序号

数据存在状态中

# PhoneGap

## 一、PhoneGap概述

PhoneGap是一个移动端的框架，插件、云端打包等服务让phoneGap功能更强大，可以使用PhoneGap所提供的各种各样的插件，来编写前端代码， 从而实现对于智能设备底层硬件的调用。

phoneGap和cordova关系：

phoneGap被Adobe收购之后，将核心的跨平台代码 贡献给了Adobe，依然保留商品所有权；Adobe将phoneGap的核心代码开源形成了一个新的项目：Cordova；

有点类似chrome和webkit的关系

React:

learn once,write anywhere

phoneGap的开发理念：

write once,run everywhere

网站推荐：

phonegap.com

cordova.apache.org

phonegap-plugins.com/

## 二、搭建PhoneGap的开发环境

步骤1：安装PC端软件

步骤2：安装Mobile端的软件

步骤3：在PC端，打开软件，创建新项目或者打开一个已经存在的项目，同时会自动的启动开发服务器

步骤4：打开移动端软件，输入ip地址和端口号，连接开发服务器

连接成功之后，当pc端修改项目代码时，将自动的同步到移动端进行预览

①C:\xampp\htdocs\Framework\_codes\phonegap

将apk结尾的包 放在该文件件中

②移动端 打开浏览器，去访问pc端的ip地址

172.163.0.155

,找到对应的apk文件，先去下载

③点击安装

## 三、使用phoneGap的插件

phoneGap的启动流程：

在index.html引入的cordova.js是一个非常重要的文件，是用来区分该项目是一个普通的项目还是一个基于cordova的项目

在index.js刚开始，有4个方法，第二个方法是用来实现事件的绑定，默认情况下绑定了deviceready对应的处理函数，在处理函数中去控制一些元素的显示或者隐藏

### 1、phoneGap支持事件

deviceready/volumedownbutton/volumeupbutton/backbutton/pause/resume/online/offline/batteryStatus/batteryCritical/batterylow

### 2、phoneGap所提供的核心插件

#### 2.1 device 获取设备的软硬件信息

device.platform 获取平台信息（android/ios/...）

device.uuid 获取设备的编号（每一台设备都是唯一的）

device.cordova 获取当前cordova版本

#### 2.2 accelerometer 加速度传感器

##### 方法：获取当前

navigator.accelerometer.getCurrentAcceleration(

funcSucess,funcError

);

##### 方法：开始监听

var watchId = navigator.accelerometer.watchAcceleration(

funcSucess,funcError,{frequency:1000})

##### 方法：结束监听

navigator.accelerometer.clearWatch(watchId);

#### 2.3 camera

##### 用途1：拍照

##### 用途2：从相册中获取指定的照片

navigator.camera.getPicture(

function(data){

alert(data);

},

function(){

alert('choose failed');

},

{

sourceType:0

}

)

#### 2.4 file

##### 写文件

步骤1：获取文件系统 fileSystem

步骤2：通过文件系统获取到操作文件的一个

入口对象 fileEntry

步骤3：创建写对象 fileWriter

步骤4：完成文件的写操作

##### 读文件

步骤1：获取文件系统 fileSystem

步骤2：通过文件系统获取到操作文件的一个

入口对象 fileEntry

步骤3：创建读对象 fileReader

步骤4：完成文件的读操作

file

#### 2.5 notification

听觉、视觉、触觉

navigator.notification.alert();

navigator.notification.confirm();

navigator.notification.beep(3);

navigator.notification.vibrate(3000);

# 微信

## 微信公众号

1.<script src="http://res.wx.qq.com/open/js/jweixin-1.0.0.js"></script>

2.微信sdk帮助文档

http://mp.weixin.qq.com/wiki/7/aaa137b55fb2e0456bf8dd9148dd613f.html

3.根据基本配置里面的

appid

AppSecret

4.到开发者工具-在线接口调试工具

https://mp.weixin.qq.com/debug

https://api.weixin.qq.com/cgi-bin/token?grant\_type=client\_credential&appid=APPID&secret=secret

生成access\_token

date

5.根据这两个值到

https://api.weixin.qq.com/cgi-bin/ticket/getticket?access\_token=你的access\_token&type=jsapi

生成json数据 ticket也就是jsapi\_ticket

6.到

http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=jsapisign

进行其他变量的获取

signature

noncestr

timestamp

7.进行配置

8.进行调用

## 微信公众号的维护和开发

1.微信公众号的维护和开发

订阅号(企业) 菜单可以自定义 可以调用特殊的接口

服务号(企业) 菜单可以自定义 可以调用特殊的接口

2.安全域名

weixin.js

wx.config({

})

wx.ready(function(){ })

3.commonweb5.php

自定义公众号调用

##写好页面

##拷贝commonweb5.php

##引入微信 weixin.js

##写

<script>

var a=document.createElement("script")

a.src="commonweb5.php?page="+window.location.href

document.getElementsByTagName("head")[0].appendChild(a)

</script>

## 写调用接口

## appid appsec 修改成自己公司的值

4.

http://xwkj.org.cn/weixin/pro1/glhy.html

http://xwkj.org.cn/weixin/pro2/glhy.html

http://xwkj.org.cn/weixin/pro3/glhy.html

1991 1996

## 微信小程序 2017-01-09

### 1.微信--我--设置--关于微信 6.5.4

发现--小程序--

京东购物

饿了么

摩拜单车

腾讯自选股票

滴滴出行

美团外卖

北京新资计算器

小年糕

### 2.是什么?

运行于微信中的程序。软件~~~~支付宝~~~qq

### 3.为何?

#### 3.1

@@@ 为了区别公众号

公众号 只能局限于微信中，而且只能做微信中的导航

@@@ 提供更多的自定义开发的权限

@@@ 挤占其它app开发

#### 3.2特点

@@@ 小 1M

@@@ 节省人力 时间

android ios sql 服务器 美工

100W

@@@ 兼容性好

@@@ 不需要下载

@@@ 频率低 安全性低 交互性低

### 4.如何做?

微信开发者工具

qq.com

mp.weixin.qq.com

小程序---开发---工具--微信开发者工具

https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html

ftp day02

172.163.0.155/wxw

### 5.微信扫描二维码

新建本地小程序

+

无id

目录 空文件

文件组成

找到能够修改 hello 张三

### 6.文件组成

js 功能

json 配置

wxml 布局 html huper text markup language

weixin markup language 标记名称

view image input form text map checkbox

wxss 样式 css

weixin stylesheet

### 7.目录结构

xinzi/

app.js 小程序加载的业务 名字不能改

app.json 小程序全局配置 名字不能改

app.wxss 全局样式 名字不能改 删掉

/utils/工具库文件夹

/pages/xxx页面/

xxx.js 页面js

xxx.json 页面配置

xxx.wxml 页面布局

xxx.wxss 页面css

/xxx页面/

xxx.js

xxx.json

xxx.wxml

xxx.wxss

#### 7.1 mp.weixin.qq.com

小程序 开发 框架

app.json 全局配置文件 json

"pages":[] 路径 pages/login/login 注册页面

$$$不注册的页面无法跳转过去

$$$写到第一个的是首页

"window"

"tabBar" 选项卡栏目

pages/images/图片

"networkTimeout" 超时时间

#### 7.2 app.js 全局的东西

App({

onLaunch:fn,//初始化的东西 本地存储 得到手机状态

onShow:fn,

onHide:fn,

onError:fn,

abc:fn,

dd:34

})

var s=getApp();

s.abc

查看网络状态

console 调试

source 资源

network 网络

#### 7.3 app.wxss 全局样式表 link href

每个页面自动引用app.wxss

每个页面再次引用同名 xxx.wxss

common.css

rpx 所有手机屏幕 750rpx 100%

屏幕宽750px

1rpx===?px

360px

#### 7.4 xxx.wxss

#### 7.5 xxx.wxml

写一个页面

打开 文件 拖拽

ctrl+r

右键 像素

ctrl+alt+i 750

拉参考线 移动工具

alt+滚轮

f8

5工具右键 三个 切片工具

ctrl+alt+shift+s

jpeg 品质60

存储

名字 默认设置 仅限图片 选中的切片

#### 7.6 reg.json 页面配置

@@@@ 可以没有

@@@@ {}

@@@@ 配置的属性 就是app.json window的大括号之内的内容

#### 7.7reg.js 页面逻辑 10+1

@@@@

Page({}) 都必须有此方法

@@@@

10个事件 onlaunch 小程序启动的那一刻

onload 每个页面启动

onLoad--->onShow

data {} 页面变量对象

onLoad 当加载的时候 页面初始化的处理

onReady 初次渲染

onShow 当显示

onHide 隐藏

onUnload 当关闭页面

onPullDownRefresh 当下拉刷新

onReachBottom 触底

onShareAppMessage 当分享此页面

onPageScroll 当滚动

## 自定义方法和属性

### 2.和前端的区别

#### 2.1事件区别

web前端 div onclick="kkk()" function kkk(){}

小程序 view bindtap="kkk" kkk:function(){}

#### 2.2修改内容

没有DOM BOM jq zepto依赖于dom

document window

{{kkk}}

web前端 div id function kkk(){ t.innerHTML="ddd"}

小程序 view {{xxx}} kkk:function(){ this.setData({xxx:"ddd"})}

#### 2.3修改颜色

web前端 div id function kkk(){ t.style.color="#f00"}

小程序 view style="color:{{d}}" kkk:function(){ this.setData({d:"#f00"})}

#### 2.4获取原来的值

emmet #abc .abc div\*5

#### 2.5 传参

web前端 div onclick="kk(0)" function kk(n){}

小程序 view bindtap="kk" id="a1" data-xy="0"

kk:function(e){

e.currentTarget.id

}

kk:function(e){

e.currentTarget.dataset.xy

}

#### 2.6 定时器

web前端 div onclick="kk(0)" function kk(n){}

setInterval setTimeout

window.setInterval()

小程序 setInterval setTimeout

ecmascript 核心类

app.js 公共的方法

xxx.js 库

var k=require("xxx.js")

k.ccc()

#### 2.7焦点 选项卡

#### 2.8修改类

web前端 div class="x" t.className="active"

小程序 view class="{{y}}" that.setData({y:"active"})

#### 2.9 DOM添加标记 删除标记

web前端 div t.appendChild()

removeChild()

createElement()

insertBefore()

小程序 view wx:if

js 选择 if(){}

if(){}else{}

if(){}else if(){} else if(){} else{}

for while do while fo in

宿主

js 浏览器 ecmascript DOM BOM

js Node.js ecmascript global http

js asp ecmascript Rquest

js 小程序 ecmascript wx.

2.10 发送ajax请求

前端 js new jq $.ajax

小程序 wx.request({})

https://zhang.com/ddd/xx.php

wx.request({

url: 'test.php', //仅为示例，并非真实的接口地址

data: {

page: 0 ,

y: ''

},

header: {

'content-type': 'application/json' // 默认值

},

success: function(res) {

console.log(res.data);

that.setData({

kk:res.data

})

},

fail:function(res){},

complete:function(res){}

})

vw vh

显示器 100vw 100vh

作业： 饿了么小程序

### 3.上线

####注册一个账号 mp.weixin.qq.com

####登录进去 appid

设置--开发设置 wxa503da876bc540c8

#### 用编辑软件 打开

项目栏目 预览 上传

#### 临时仓库

提交审核

# NodeJs

Node.js是一个基于V8引擎的JS解释器，适用于I/O密集型应用；第一次使得使用一种开发语言统一前后台开发。

掌握一门新语言的基本步骤：

(1)了解背景

(2)搭建环境，写出HelloWorld

(3)变量和常量

(4)数据类型

**原始/基础/值类型：**string、number、boolean、null、undefined

**引用对象类型：**ES原生对象类型、Node.js核心对象、用户自定义对象

(5)运算符

(6)逻辑结构

(7)通用小程序

(8)函数和对象

(9)第三方组件、库、框架

(10)实用项目

## 1.语义化版本控制 —— SemVer

Semantic Version：语义化的版本，制定一套统一的规则，让应用的版本号更加的体现出语义。

semver.org：

**主版本号.次版本号.修订版本号**

不兼容 兼容 更正

主版本号：0.x.x 尚处于初始开发阶段，尚不推荐商业应用

alpha：内测

beta：公测

milestone：里程碑版，更多功能有待继续开发

rc：Release Candicate，发行版的候选版

ga：General Available，公共可用版

## 2.Node.js核心模块 —— Globals

global：Node.js中的全局对象是global，不是window；

|  |
| --- |
| 交互模式下，所有的全局变量都自动成为global的成员！——也存在污染问题。  脚本模式下，每个JS文件都会被封装为一个对象，看似全局变量其实只是当前模块对象的内部局部变量，不会污染global对象！ |

**global.console** = new Console( process.stdout, process.stderr ); 控制台对象，默认的标准输出和错误输出对象都是“屏幕”。console对象的常用成员：

console.log( data )：向stdout输出指定数据

console.dir( obj[, options] )：向stdout输出指定对象的检查数据

console.info( data )：等同于log()

console.warn( data )：等同于error()

console.error( data )：向stderr输出指定数据

console.trace( data )：向stderr输出当前的函数调用堆栈

console.assert( bool表达式, 错误消息)：断言，用于编写单元测试用例；若bool表达式为true，则继续执行后面的代码；否则，表示测试用例失败了，会输出错误消息

console.time()/timeEnd()：用于粗略计算一段代码的执行时间，单位为ms

|  |
| --- |
| 输出数据的严重级别：log < info < warn < error |
| Node.js中数据的严重级别： log == info < warn == error |

|  |
| --- |
| Node.js中计算代码的耗时两种方法：  **粗略计算（单位为ms）：**  console.time() .... console.timeEnd()  **精确计算（单位为ns）：**  process.hrtime() |

练习：使用递归调用方式计算1~10的累加和，最后一次累加时输出函数调用堆栈。

**global.process：**进程，该对象指代Node.js应用在操作系统中对应的进程。通过该对象可以获取当前操作系统的相关信息。

process.hrtime([time])：High Resolution Time，高解析度的计时器。若没有参数直接返回当前高精度的系统时间；若指定了参数，则返回当前时间与指定时间的差值。

process.pid：返回当前进程的id号

process.kill( pid )：杀死指定id号的进程，给指定的进程发送“终止”信号

process.exit( )：终止当前进程的执行

process.stdin：标准输入对象，默认指键盘

process.stdout：标准输出对象，默认为显示器

process.stderr：标准错误输出对象，默认为显示器

-----------------Node.js四种定时器---------------

process.nextTick(fn)：等价于setTimeout(fn, 0)，把定时器任务追加到事件循环队列的当前循环的队尾

global.setInterval(fn, delay) / clearInterval()：设置周期性定时器

global.setTimeout(fn, delay) / clearTimeout()：设置一次性/超时定时器

global.setImmediate(fn) / clearImmediate()：等价于setTimeout(fn, 0)，把定时器任务追加到事件循环队列的下一次循环的队首

-----------------------------------------------------

练习：使用process.hrtime()计算一个耗时1s的定时器实际最终等待时长是多久？

练习：创建一个10\_1.js，输出自己的进程id号，等待10s后自动退出；创建一个10\_2.js，一运行就设法杀死10\_1.js对应的进程。

练习：创建一个10\_1.js，输出自己的进程id号，等待10s后自动退出；创建一个10\_2.js，一运行就设法杀死10\_1.js对应的进程(向10\_1进程发送SIGTERM信号)。 10\_1终止前输出自己的实际已经等待的时长(高精度值)——监听信号事件。

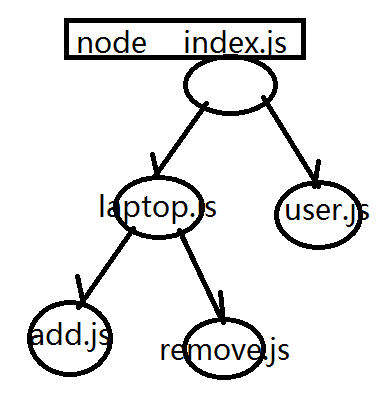
## 3.Node.js中的模块

Node.js中每一个JS文件都被看作一个“模块”；

应用启动时运行的那个模块称为“主模块”——mainModule；

主模块可以调用其它模块提供的内容——require()；

每个模块中声明的变量都是私有的，但可以导出某些数据供其它模块使用——exports。



课后任务：

(1)Node.js中如何在cmd命令行中输出不同颜色的文字。

(2)process.uncaughtException事件在项目中的作用

(3)Node.js如何实现一行一行的读取用户的键盘输入

## 4.Express框架

Express是一个基于Node.js http模块的高层封装，用于创建Web服务器(静态+动态)。

|  |
| --- |
| DOM => jQuery |
| Node.js/http => Express |

Express的三个核心概念：

(1)路由：定义特定的URL如何来处理

(2)中间件

(3)模板引擎

### 2.Express路由

#### 路由的三要素： RESTful

(1)请求方法：GET / POST / PUT / DELETE

(2)请求地址：/index、/user/10、/news/edu/bj、/user/tom/123

(3)响应函数：(req, res)=>{}

#### 如何获取请求数据？

(1)查找字符串形式的请求数据

/user/login?uname=tom&upwd=456

req.query

(2)在请求URL中使用路由参数

/user/login/tom/123

req.params

#### 如何给客户端以各种类型的响应？

res.send( 字符串 )

res.sendFile( 文件路径 )

res.json( obj )

res.download( 文件路径 )

res.redirect( url )

|  |
| --- |
| “中间件”行业中定义：指一般不能直接供最终用户使用，需要进行二次开发的中间产品。 |

### 3.Express中间件

Middleware： Express中的中间件就是函数，可使用req和res（路由处理函数本质就是一个中间件）。中间件可以实现：

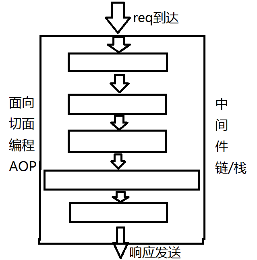
(1) 执行任何代码，如权限检查、日志记录。

(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。

提示：Express中的中间件体现的是“面向切面的编程思想”。



Express中间件的分类——难点&面试题

#### (1)应用级中间件

app.use( 中间件 )

app.use( url, 中间件 )

#### (2)路由级中间件

app.get/post/put/delete(url, 中间件)

#### (3)错误处理中间件

#### (4)内置中间件

Express3.x中内置了十几个常用的中间件，Express4.x中几乎全部剔除，只保留了一个express.static()中间件

#### (5)第三方中间件

npmjs.org：body-parser、cookie-parser....

|  |  |
| --- | --- |
| 思考题：应用级中间件和路由级中间件的区别？  1.应用级中间件可以不指定特定的URL（绑定到所有的请求URL上）；路由级中间件必须绑定到一个明确的URL  2.应用级中间件默认对所有的请求方法有效；而一般情况下路由级中间件只对特定的请求方法有效。  3.应用级中间件挂载地址只是一个起始说明，实际可以拦截的地址包括后续添加任何子路径的地址；路由级中间件的挂载地址是精确匹配的 | |
| 应用级中间件的绑定地址 /user，可拦截的请求URL | 路由级中间件绑定地址 /user，可拦截的请求URL |
| /user、/user/login、  /user/login/news、....等等形如： /user/\*/\*/\*.... | /user |

### 4.Express模板引擎

模板技术：在Node.js中实现静态页面和动态数据的服务器端拼合 —— 这种技术没有实现“前后端分离”，不推荐使用。

示例：独立使用EJS模板引擎

示例：把Express框架中整合使用EJS模板引擎

课后任务：

(1)把学子商城前台主要功能用Node.js+Express+MySQL改写。

(2)回忆PHP Session技术原理；在Express中使用第三方的SessionParser。

(3)自学koa框架。

Git/Github部署=>SAE云服务器=>简历写Node项目

yeoman

scaffolding：脚手架，

### 5.使用Express-CLI脚手架工具搭建一个Express项目的基本结构

在任意目录下执行：

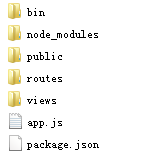
npm i express-generator -g

切换到需要创建项目的目录下执行命令

express.cmd mypro

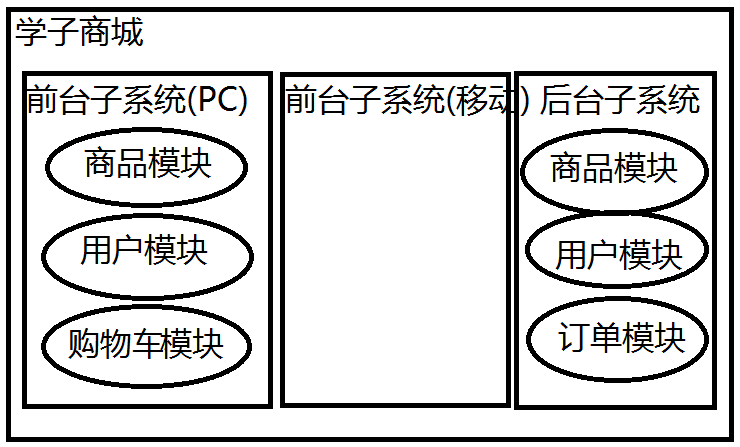
进入项目目录(mypro)

npm i



执行package.json指定的启动脚本，运行Web服务器应用

npm start



### 6.Express中的Router

Router：路由器，就是一个路由的集合对象；在项目中可以根据模块的功能划分，把不同模块中的路由划分到不同的路由器中。

**routerUser.js**

let router = express.Router();

router.get('/del', fn);

router.post('/add', fn);

module.exports = router;

**app.js**

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

### 7.Epxress中的中间件分类

(1)应用级中间件 app.use(url, fn)

(2)路由级中间件 app.get/post/put/delete(url, fn)

(3)错误处理中间件

(4)内置中间件 app.use(express.static(...))

(5)第三方中间件 app.use(bodyParser())

错误处理中间件专用于执行“错误处理”功能，向客户端输出4xx或5xx响应消息。

//声明一个错误处理中间件

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

|  |
| --- |
| next()函数的参数有如下形式：  (1) next( )：继续交由栈中下一个中间件执行(应用中间件/路由中间件)  (2) next('route')：在一个路由中间件中，若某个处理函数声明了next('route')则当前路由中的后续处理函数将会被跳过，继续执行后续的路由/应用中间件  (3) next( err )：参数err是一个Error类型/子类型的对象，直接跳转到后续最近的一个错误处理中间件 —— 错误处理中间件可以对next(err)或同步错误都有效 |

## 5.MVC设计模式

### 后端MVC：Express/MySQL ——

Model(数据，业务处理过程和数据库访问）

View(视图，即接口，接收请求，给出JSON响应数据)

### 前端MVC：NG/Vue ——

Model(数据，通过AJAX从服务器异步请求得到)、 View(视图，即HTML/DOM)

Repositoy：仓库，其中有所有的项目文件和修改历史记录

## 6.VCS系统的使用

Version Control System，版本(软件开发版本)控制系统。用于项目代码的存储、共享、合并、历史追溯、历史对比/回退等等项目代码的管理问题。VCS软件有两大类：

(1)集中式VCS: 由一个中央服务器来管理仓库

VCS服务器：负责永久存储所有项目文件，及修改记录

VCS客户端：负责下载/修改/提交项目文件

常见的集中式VCS: CVS、Subversion(SVN)

(2)分布式VCS : 每个客户端都保存一份完整的仓库——每个客户端都是服务器

常见的分布式VCS: Git

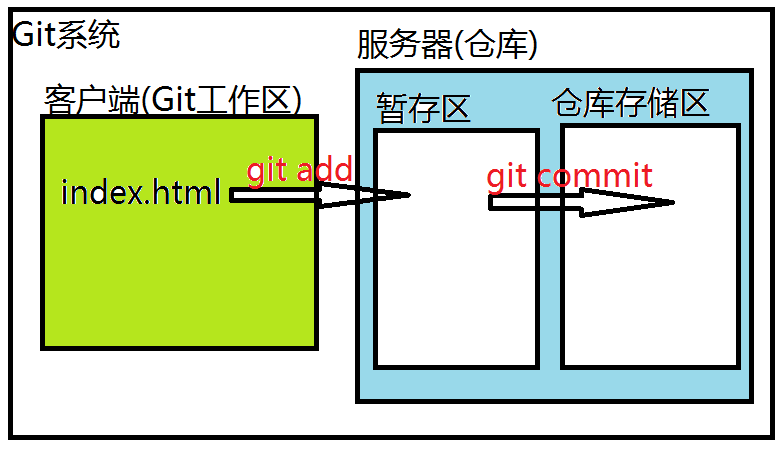
## 7.使用Git系统

官网：https://git-scm.com/

官方文档：https://git-scm.com/doc

自学文档：https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000

|  |
| --- |
| Git：是一个CVS软件，由Linus编写，用于分布式版本控制；每个安装了Git的计算机就是一个Git仓库  GitHub：是一个网站，全球最大的开源Git仓库 |



### (1)下载并安装Git软件

### (2)配置一下当前计算机的Git账户

git config --global user.name "我的用户名"

git config --global user.email "我的邮箱"

### (3)创建一个不包含中文和空白字符的目录，进入该目录，将其初始化为一个Git 仓库

git init

### (4)在仓库中创建项目文件，添加到Git暂存区中，再提交给仓库分支

git add \*/文件名

git commit -g "一段说明文字"

### (5) 在工作区中删除文件，并从仓库中也删除

git rm 文件名

git commit -g "一段说明文字"

### (6)修改一个工作区中的文件后，再提交给仓库

git add \*/文件名

git commit -g "一段说明文字"

### (7)回退到版本历史上的某个版本

git reset --hard 版本号

----------------------------------------------

### (8)查看工作区文件状态

git status

### (9)查看当前所处提交历史记录

git log

### (10)查看所有的提交历史记录

git reflog

### (11)复制另一个仓库(如github上的仓库)

git clone ...

### (12)把当前仓库内容推送给远程仓库

git push ...

### (13)从其他仓库重新拉取最新内容

git pull ...

自学手册：新浪云服务器上使用Node.js服务器：

http://www.sinacloud.com/doc/sae/docker/nodejs-getting-started.html

复习：

VCS系统的分类：

(1)集中式VCS：Subversion

(2)分布式VCS：Git

|  |
| --- |
| git --help 显示所有可用的命令  git 命令 --help 显示指定命令的使用指南 |

## 8.Git的使用

### (1)下载并安装一个Git软件（命令行/GUI版本）

### (2)配置个人信息

git config --global user.name "LiWenhua"

git config --global user.email "lwh@tedu.cn"

这两个命令其实是在修改c:/Users/用户名/.gitconfig

### (3)创建一个项目所在的目录，初始化为git仓库

git init

### (4)创建/修改/删除文件，添加暂存区，...，最终提交到分支库

git add **.**  把所有有改变文件/目录添加到暂存区

git commit -m "注释内容(英文)"

### (5)从仓库中找回某个文件之前版本中的内容，版本库并不变

git checkout -- 文件名 找回仓库中最新版本的指定文件内容，覆盖当前文件中的内容

git checkout 版本号 -- 文件名 找回仓库中指定版本中的指定文件内容，覆盖当前文件中的内容

### (6)回退到历史上的指定版本，把工作区强制恢复到当时的状态

git reset --hard 版本号

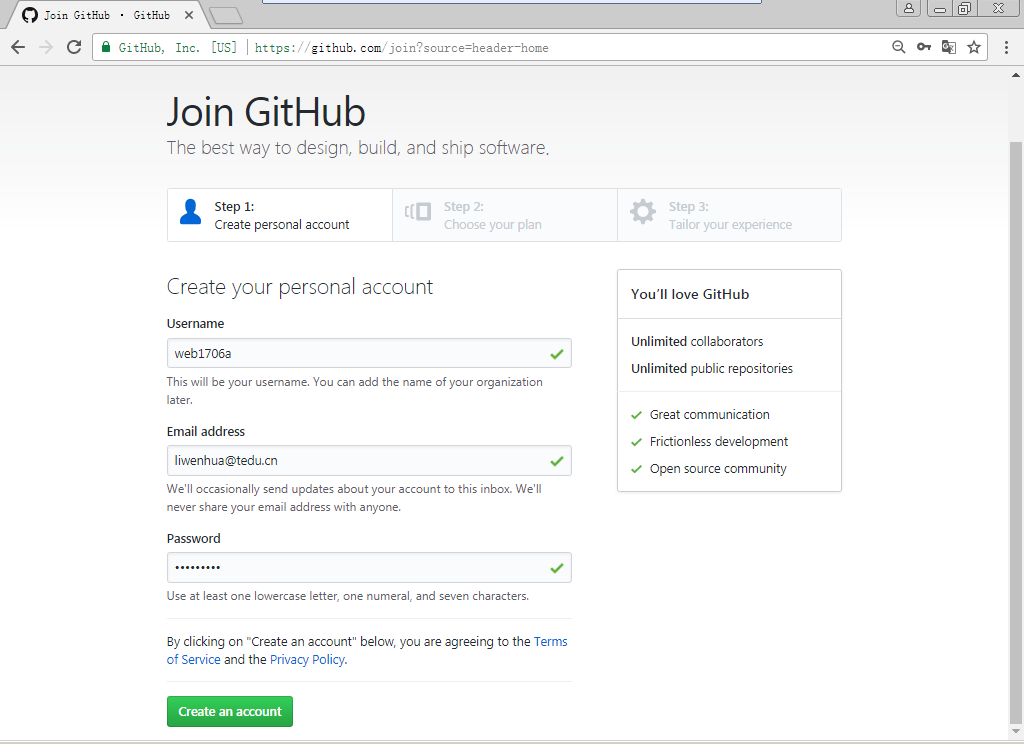
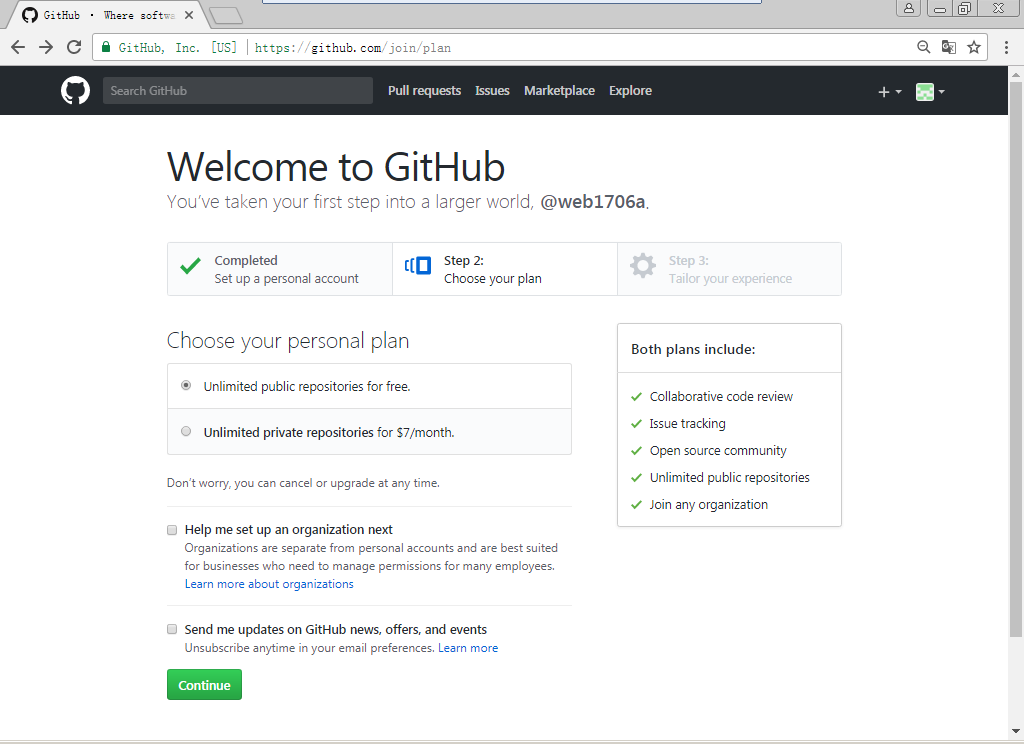
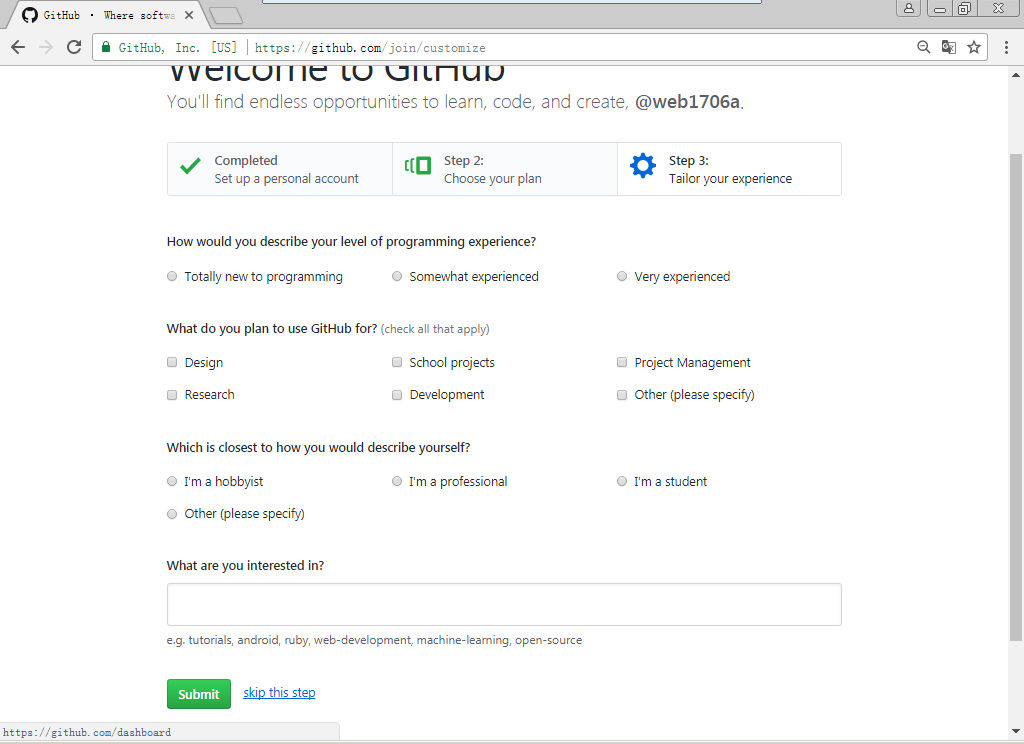
|  |
| --- |
| git status 查看当前工作区状态  git log 查看仓库的版本日志（当前+之前）  git reflog 查看仓库的所有版本日志（之后+当前+之前）  git remote 查看当前仓库所关联的远程仓库 |

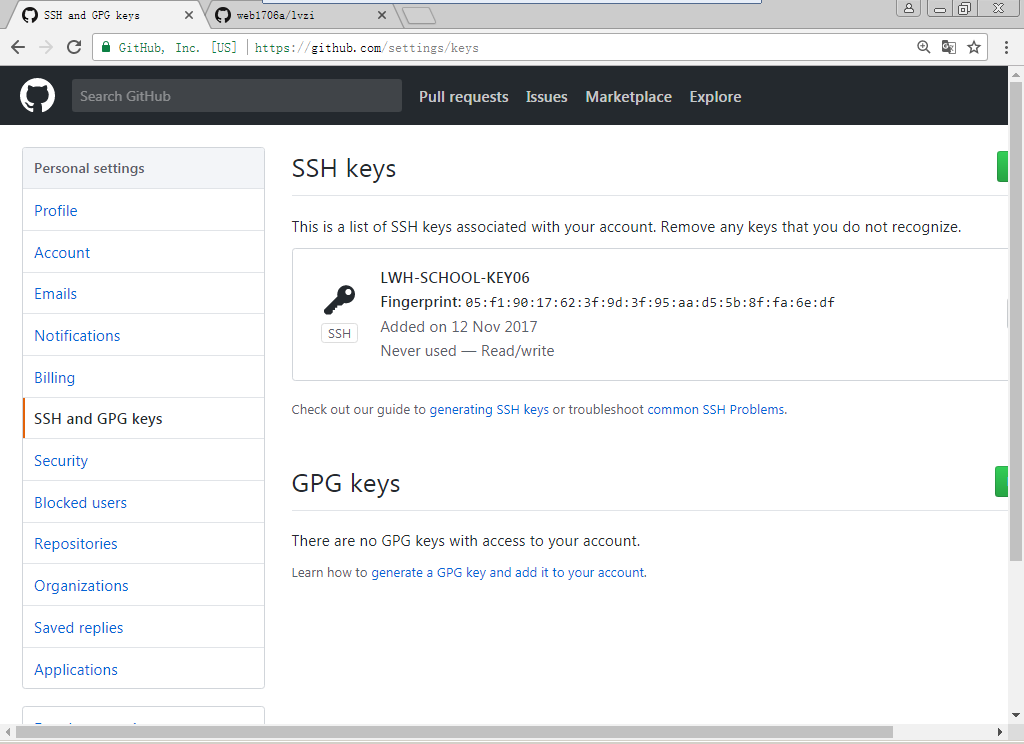
公司多人协作项目最好建一个7\*24工作的Git版本库，两种方法：

(1)公司搭建一台Linux Git仓库

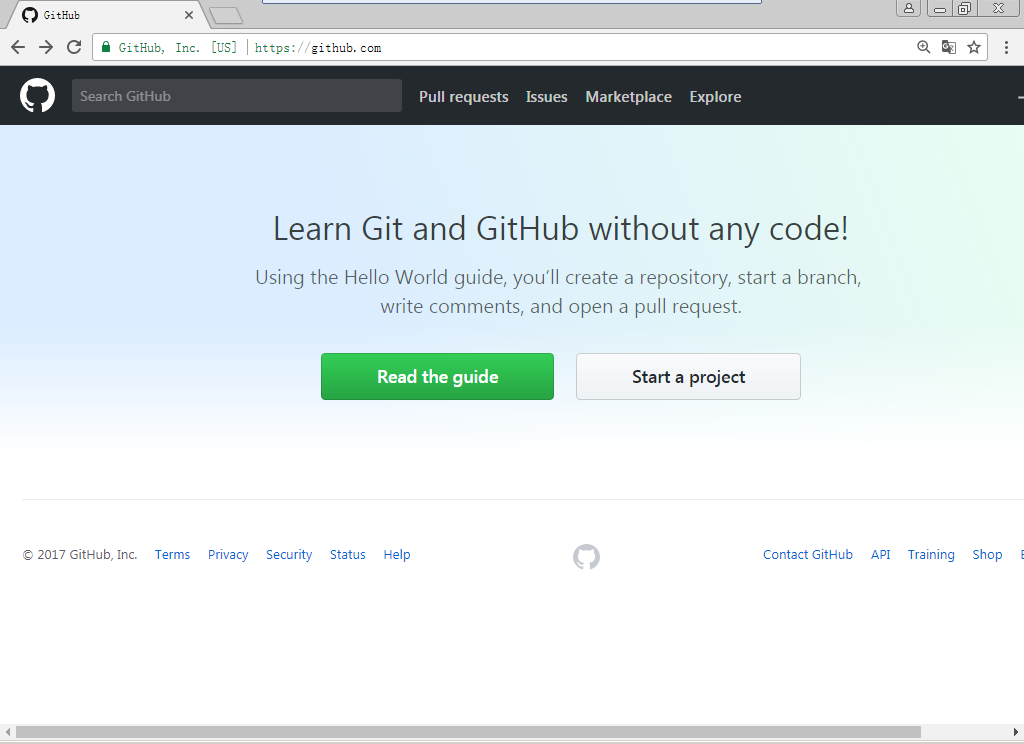
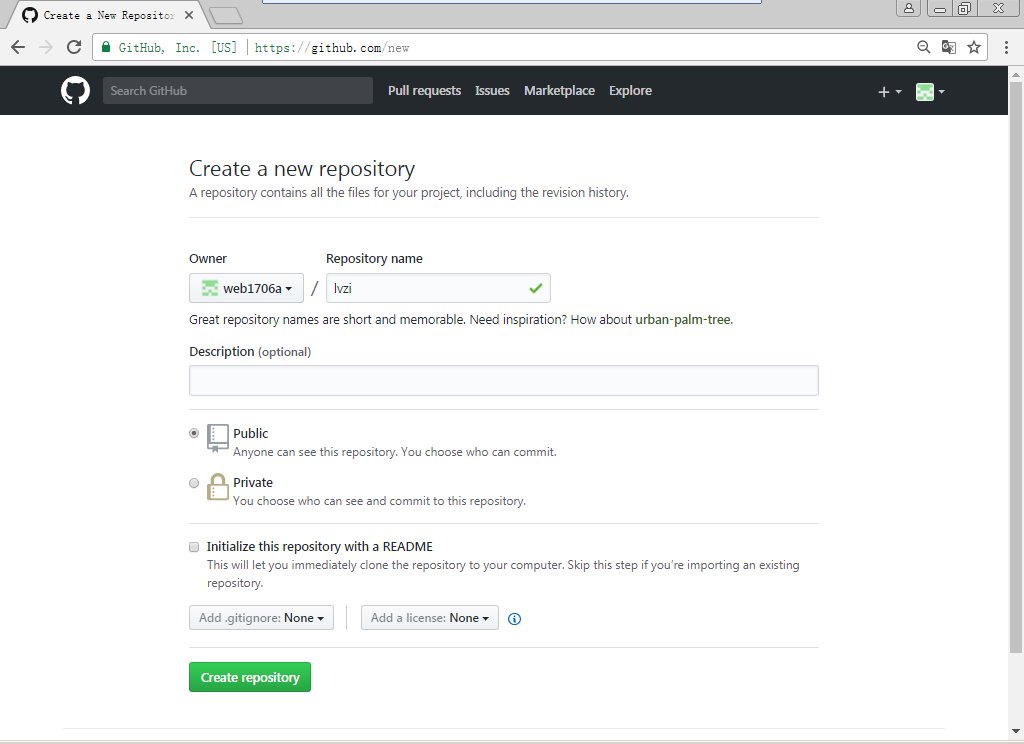
(2)使用第三方网站Git仓库，如github.com 集市/集中器

注册Github.com账号：

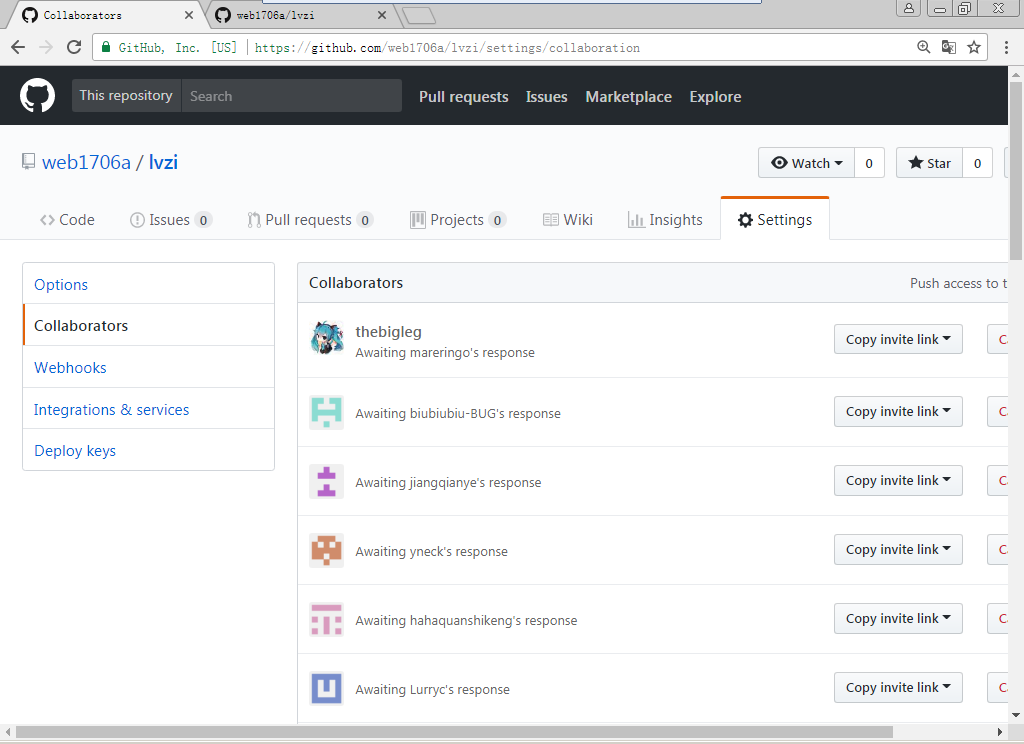
  

项目发起者：创建一个新项目：

项目发起者：添加项目合作者：



**项目发起者：在自己的仓库中创建项目的初始内容，推送到Github.com对应的仓库上**

git init

拷贝项目必需的初始文件

git add .

git commit -m "initial commit"

#将当前仓库关联到一个远程仓库

git remote add lvzi git@github.com:web1706a/lvzi.git

#将当前仓库中指定分支的内容“推送”给远程仓库

git push -u lvzi master #-u表将两个master分支合并

**项目合作者：将远程仓库克隆一份到本地**

git clone git@github.com:web1706a/lvzi.git

#进入克隆下来的仓库目录

cd lvzi

#修改/添加/删除项目文件

git add . #添加到本地暂存区

git commit -m "说明文字" #提交给本地的仓库

#把本地仓库内容提交/上传/推送到远程仓库（克隆下来的仓库名默认都叫origin）

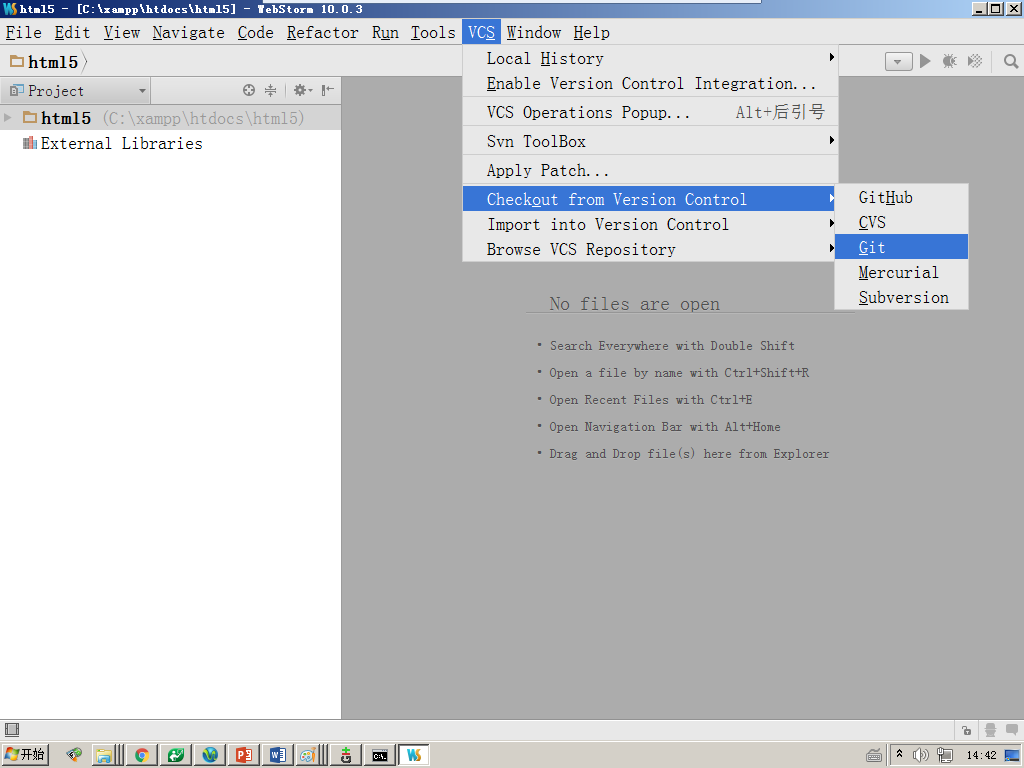
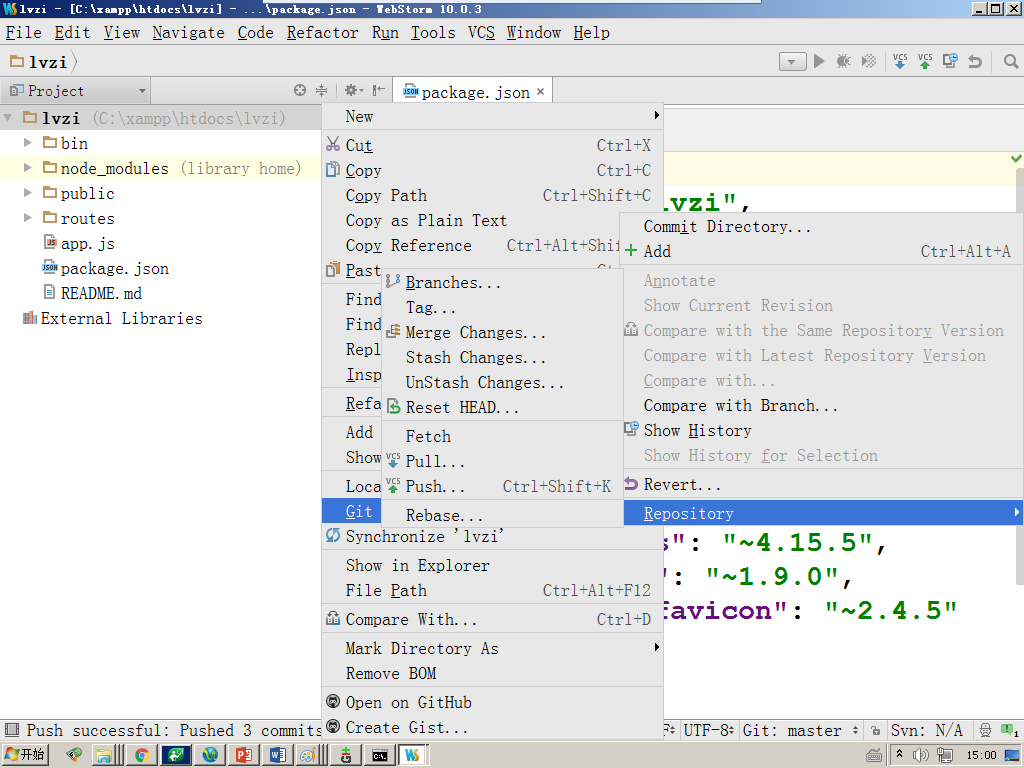
git push origin master #把本地仓库的主分支推送到远程仓库

#也可以随时从远程服务器上下载/拉取内容到本地仓库

git pull origin

|  |
| --- |
| **使用Github.com每日的工作流程：**  早上：拉取github.com上的所有最新版本  git pull origin  白天：修改/添加/删除项目文件，提交给本地仓库  git add .  git commit -m ""  下班前：将本地仓库内容推送到远程仓库  git push origin master |

项目合作者：也可以直接使用WebStorm克隆远程仓库、本地修改/提交、推送到远程仓库

|  |
| --- |
| 新浪云让Apache占80端口，监听客户端请求；  所有Node.js默认占用5050端口；  当有客户端请求到达，80端口的Apache会将根据域名会将**端口映射**为5050的应用。 |

## 9.将项目上传到新浪云Node.js服务器

### (1)创建/克隆一个本地的Git仓库，其中包含项目文件，注意：

pacakge.json：

{

"scripts":{

"start": "node ../bin/www"

}

}

### (2)登录新浪云，创建自己的云应用，获得二级域名，自动分配Git仓库的地址

### (3)将新浪云的远程仓库与本地仓库做一次绑定

git remote add sae https://git.sinacloud.com/web1706a

推送本地仓库的内容到新浪云远程仓库

git push sae master

推送完成后会自动运行整个Node.js应用程序

### (4)使用客户端访问自己的域名

|  |
| --- |
| 注意：Node.js应用部署到新浪云服务器失败原因：  (1)Node服务器监听端口必须是5050  (2)package.json文件必须有 "scripts":{ "start": "...."} 指令 |

## 10.HTTP协议和WS协议

### HTTP协议：

属于典型的“请求-响应模型”的协议，一个请求会获得一个响应，没有请求就没有响应，且是一个“无状态”协议。不便于创建“实时性要求高的应用”——实时走势、聊天室、联机游戏、协同办公等 —— 解决办法：“长轮询”= 定时器 + 异步请求

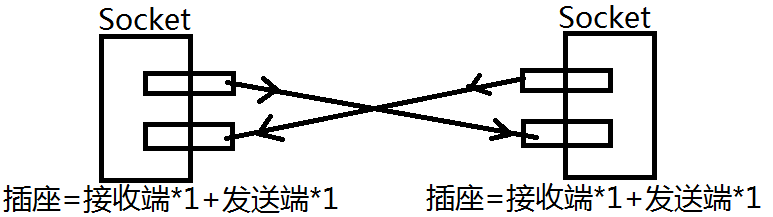
### WS协议：

属于“广播模型”的协议，一个可以一对多双向通信协议，一方可以随时不停的向对方发送多个消息(Message)。

WS协议的支持情况：

客户端：各大浏览器基本都支持，除了(IE10以下的浏览器)

服务器：各种主流服务器端编程语言基本都支持



## 11.Socket.IO

是一个第三方提供的可以实现“双向实时通信”的工具库，包含两部分：

(1)Node.js服务器端应用(net+crypto)

(2)JS客户端应用(封装H5 WS API/其它浏览器兼容方案)

官网：https://socket.io/

快速入门：https://socket.io/get-started/chat/

### 使用Socket.IO创建类聊天室实时应用的步骤：

#### (1)使用HTTP或Express模块创建一个Web服务器

#### (2)下载socket.io模块，添加到项目模块依赖列表

npm i socket.io --save

#### (3)修改服务器端代码，让socket.io拦截客户端请求

let io = require('socket.io')(server);

io.on('connection', (socket)=>{

//一个实时通信客户端连接到服务器

}

#### (4)客户端请求的页面中引入script：/socket.io/socket.io.js，连接到Socket.IO服务器：

<script src="/socket.io/socket.io.js"><script>

<script>

var socket = io();

</script>

#### (5)服务器和客户端可以使用socket对象进行双向通信：

发送消息： socket.send('msg');

接收消息： socket.on('message', (msg)=>{})

|  |
| --- |
| 面试题（百度）：  HTTP协议中的“三次握手”和“四次挥手”各是什么意思？  WS协议中的“握手”只需要一个请求消息和响应消息。 |

## Buffer

### Iteration

- length

- keys()

- values()

- entires()

### fs.readFile(path[, option], callback(err, data))

- data: buffer

- `\_\_dirname` global

- `\_\_filename` global

## Timer

- setTimeout() `delay` 延时执行，下一个队列的开头

- setInterval() `delay` 循环执行，下一个队列的开头

- setImmediate() 立即执行，下一个队列的开头

- process.nextTick() 立即执行，本次队列的结尾

## Module

- 核心模块

- 第三方模块

- 自定义模块

- 文件模块

- 目录模块

- `index.js`

- `package.json`

### - 模块的使用

- `exports` 导出

- `require` 引入

## NPM

```

npm install <package-name>

npm install -g cnpm --registry=https://registry.npm.taobao.org

npm init

npm install

```

## querystring

### - `parse()`

### - `stringify()`

## url

### - `parse()`

### - `format()`

## util

### - `isXxx()`

## fs

- 同步操作和异步操作的区别

### - `readFile()`

### - `readFileSync()`

### - `stat()`

### - `readdir()`

### - `writeFile()` 写出文件

### - `appendFile()` 追加写出

### - `mkdir()`

---

## Github

1. File-Settings-git-test [https://git-scm.com/](https://git-scm.com/)

2. File-Settings-github-password-test

3. C:\Users\web-01\.gitconfig

```

[user]

name = your\_github\_username

email = your\_github\_email

```

4. .gitignore

5. VCS-Import into Version Control-Share Project on GitHub

6. commit & push `Ctrl + K | Shift + Ctrl + K`

7. update `Ctrl + T`

## JS和Node.js区别：

|  |  |
| --- | --- |
| JS和Node.js区别： | |
| JS | Node.js |
| 客户端/浏览器技术 | 服务器端技术，竞争对手是PHP/JSP |
| 有严重的兼容性问题 | 没有兼容性问题(ES6/7) |
| 数据类型：  值类型：...  引用类型：  ES原生对象  宿主/浏览器对象(B&D)  自定义对象 | 数据类型：  值类型：...  引用类型：  ES原生对象  Node.js扩展对象(手册)  自定义对象 |

学习Node.js：

(1)架构体系——语法 (2)API中的对象——单词

## Node.js核心模块——http模块

http模块用于创建HTTP协议的服务器端应用(代替Apache+PHP功能)——接收客户端请求，解析并处理请求，返回响应消息

http模块还用于创建HTTP协议的客户端应用(模拟浏览器功能)——向其他的服务器发起请求，等待返回响应消息

Node.js中使用http模块创建服务器端应用：

const http = require('http');

let server = http.createServer(function(req, res){

//读取请求消息 req.method/url/headers

//输出响应消息 res.writeHead()/write()/end()

});

server.listen(80);

午间练习：使用HTTP模块接收不同的请求URL，服务器返回不同的响应消息主体

/index

/user/login

/user/register

原生DOM操作(复杂) => 第三方库jQuery

Node.js http模块(复杂) => 第三方框架Express/Koa

## Node.js http模块的第三方扩展框架——Express

官网：expressjs.com

中文网：expressjs.com.cn

Express本质是一个HTTP请求消息处理函数，可以解析请求消息，处理请求数据，向客户端输出响应消息

使用步骤：

(1)使用NPM工具到npmjs.org下载最新的Express模块

npm i express

(2)使用http模块创建Web服务器，运行express，让其承担请求监听器的任务

var app = express();

http.createServer( app ).listen(80);

(3)为app对象添加特定URL的请求处理

....

|  |
| --- |
| Express三大核心概念：  (1)路由  (2)中间件  (3)模板引擎 |

# Express三个核心概念：

## (1)路由 = 请求方法 + 请求URL + 处理函数

## (2)中间件

## (3)模板引擎

## 1.Express核心概念之一： 路由

Route：路由，针对客户端的某个请求方法所请求的特定URL，如何给出响应消息的过程，称为“路由”

**路由 = 请求方法 + 请求URL + 处理函数**

一个典型的路由形如：

app.get('/user', function(req, res){

//接收请求消息中的数据

req.query //获取查询字符串中的数据{ }

req.params //获取请求参数(手册自学)

req.on('data', function(buf){}) //获取请求主体中的数据

//发送响应消息

res.send(HTML文本);

res.sendFile(任意类型的文件);

res.json(对象/数组数据);

})

课后任务：

创建Node.js/Express服务器端应用，仿照mfresh.codeboy.com实现其中的如下功能点：

GET /index

GET /contact

GET /register

POST /register

GET /login

POST /login

必需的静态资源可以直接从codeboy.com服务器上抓取

## 2.中间件(Middleware)

Express中的中间件就是函数，形如：

function(req, res, next){

}

注意：路由处理函数本质也是一个中间件。

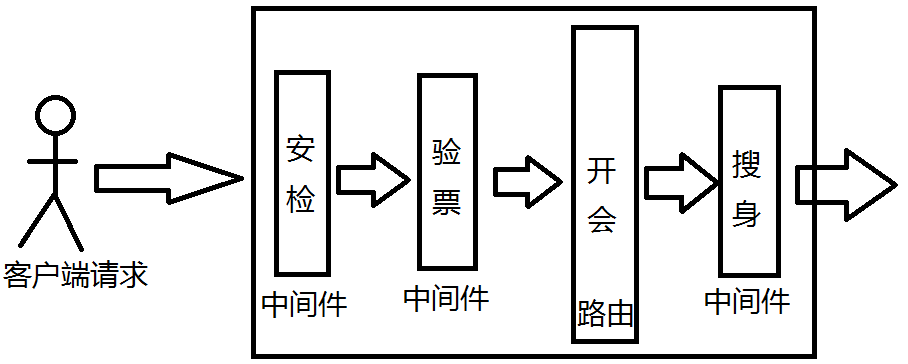
### 中间件的功能：

#### (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。



一个典型的“中间件栈”：

#### //前置中间件

app.use( function(req, res, next){ ...**next();** })

app.get('/index', function(req, res, next){

res.json(...);

**next();**

})

#### //后置中间件

app.use( function(req, res, next){ ...**next();** })

### 项目中常用的中间件

#### (1)向客户端返回指定目录下的所有静态资源文件

app.use(express.static('./public'));

#### (2) body-parser

将客户端POST请求主体中的数据封装到req.body中

npm i body-parser

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

app.use( bodyParser.urlencoded({extended: false}));

#### (3) cookie-parser

将请求头部中的Cookie数据封装到req.cookies中

npm i cookie-parser

const cookieParser = require('cookie-parser');

app.use(cookieParser());

|  |
| --- |
| PHP: $\_REQUEST = $\_GET + $\_POST + $\_COOKIE |
| **H5 sessionStorage技术**：把k=v数据保存在客户端  **服务器端session技术**：把每个客户端会话数据保存在服务器端，客户端仅用Cookie保存一个SessionID号。$\_SESSION['loginName'] = 'tom'; |

#### (4) express-session

服务器的响应消息中发送Set-Cookie提供SessionID，客户端再次请求时可以自动提交该Cookie，出示自己的SessionID；服务器接收该SessionID，查找当前客户端对应的服务器端存储数据——封装在req.session对象中。

npm i express-session

const session = require('express-session');

app.use( session( {

resave: false,

saveUninitialized: true,

secret: 'xxxyyyyzzz'

} ) );

...

req.session

#### (5)cors

使用cors中间件修改响应消息的头部，以允许指定来源的跨域请求

午间复习：

什么是跨域请求？如何实现跨域请求？CORS方案是什么？

## 什么是跨域请求?

http://www.ruanyifeng.com/blog/2016/04/cors.html

当前URL与目标URL的“协议名或主机名或端口”三者中有一项不同，目标URL请求就称为“跨域请求”。

### 浏览器允许跨域的情形：

<img><script><link><iframe>

### 浏览器禁止跨域的情形：

XHR请求

### XHR跨域被禁止的解决方案：

#### (1)JSONP

##### JSONP包含两部分：回调函数和数据。

##### 回调函数

是当响应到来时要放在当前页面被调用的函数。

##### 数据

就是传入回调函数中的json数据，也就是回调函数的参数了。

##### 参考代码

function handleResponse(response){

console.log('The responsed data is: '+response.data);

}

var script = document.createElement('script');

script.src = 'http://www.baidu.com/json/?callback=handleResponse';

document.body.insertBefore(script, document.body.firstChild);

/\*handleResonse({"data": "zhe"})\*/

//原理如下：

//当我们通过script标签请求时

//后台就会根据相应的参数(json,handleResponse)

//来生成相应的json数据(handleResponse({"data": "zhe"}))

//最后这个返回的json数据(代码)就会被放在当前js文件中被执行

//至此跨域通信完成

##### jsonp虽然很简单，但是有如下缺点：

###### 1）安全问题(请求代码中可能存在安全隐患)

###### 2）要确定jsonp请求是否失败并不容易

#### (2)CORS

CORS背后的思想，就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功，还是应该失败。

##### 参考代码

###### IE中对CORS的实现是xdr

var xdr = new XDomainRequest();

xdr.onload = function(){

console.log(xdr.responseText);

}

xdr.open('get', 'http://www.baidu.com');

....

xdr.send(null);

###### 其它浏览器中的实现就在xhr中

var xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {

if(xhr.readyState == 4){

if(xhr.status >= 200 && xhr.status < 304 || xhr.status == 304){

console.log(xhr.responseText);

}

}

}

xhr.open('get', 'http://www.baidu.com');

......

xhr.send(null);

###### 实现跨浏览器的CORS

function createCORS(method, url){

var xhr = new XMLHttpRequest();

if('withCredentials' in xhr){

xhr.open(method, url, true);

}else if(typeof XDomainRequest != 'undefined'){

var xhr = new XDomainRequest();

xhr.open(method, url);

}else{

xhr = null;

}

return xhr;

}

var request = createCORS('get', 'http://www.baidu.com');

if(request){

request.onload = function(){

......

};

request.send();

}

#### document.domain + iframe

(只有在主域相同的时候才能使用该方法)

##### 1) 在www.a.com/a.html中：

document.domain = 'a.com';

var ifr = document.createElement('iframe');

ifr.src = 'http://www.script.a.com/b.html';

ifr.display = none;

document.body.appendChild(ifr);

ifr.onload = function(){

var doc = ifr.contentDocument || ifr.contentWindow.document;

//在这里操作doc，也就是b.html

ifr.onload = null;

};

##### 2) 在www.script.a.com/b.html中：

document.domain = 'a.com';

#### (4) 动态创建script

这个没什么好说的，因为script标签不受同源策略的限制。

##### 参考代码

function loadScript(url, func) {

var head = document.head || document.getElementByTagName('head')[0];

var script = document.createElement('script');

script.src = url;

script.onload = script.onreadystatechange = function(){

if(!this.readyState || this.readyState=='loaded' || this.readyState=='complete'){

func();

script.onload = script.onreadystatechange = null;

}

};

head.insertBefore(script, 0);

}

window.baidu = {

sug: function(data){

console.log(data);

}

}

loadScript('http://suggestion.baidu.com/su?wd=w',function(){console.log('loaded')});

//我们请求的内容在哪里？

//我们可以在chorme调试面板的source中看到script引入的内容

#### (5) location.hash + iframe

原理是利用location.hash来进行传值。

假设域名a.com下的文件cs1.html要和cnblogs.com域名下的cs2.html传递信息。

1) cs1.html首先创建自动创建一个隐藏的iframe，iframe的src指向cnblogs.com域名下的cs2.html页面

2) cs2.html响应请求后再将通过修改cs1.html的hash值来传递数据

3) 同时在cs1.html上加一个定时器，隔一段时间来判断location.hash的值有没有变化，一旦有变化则获取获取hash值

##### 注：

由于两个页面不在同一个域下IE、Chrome不允许修改parent.location.hash的值，所以要借助于a.com域名下的一个代理iframe

##### 代码如下：

###### 先是a.com下的文件cs1.html文件：

function startRequest(){

var ifr = document.createElement('iframe');

ifr.style.display = 'none';

ifr.src = 'http://www.cnblogs.com/lab/cscript/cs2.html#paramdo';

document.body.appendChild(ifr);

}

function checkHash() {

try {

var data = location.hash ? location.hash.substring(1) : '';

if (console.log) {

console.log('Now the data is '+data);

}

} catch(e) {};

}

setInterval(checkHash, 2000);

###### cnblogs.com域名下的cs2.html:

//模拟一个简单的参数处理操作

switch(location.hash){

case '#paramdo':

callBack();

break;

case '#paramset':

//do something……

break;

}

function callBack(){

try {

parent.location.hash = 'somedata';

} catch (e) {

// ie、chrome的安全机制无法修改parent.location.hash，

// 所以要利用一个中间的cnblogs域下的代理iframe

var ifrproxy = document.createElement('iframe');

ifrproxy.style.display = 'none';

ifrproxy.src = 'http://a.com/test/cscript/cs3.html#somedata'; // 注意该文件在"a.com"域下

document.body.appendChild(ifrproxy);

}

}

###### a.com下的域名cs3.html

//因为parent.parent和自身属于同一个域，所以可以改变其location.hash的值

parent.parent.location.hash = self.location.hash.substring(1);

#### (6) window.name + iframe

window.name 的美妙之处：name 值在不同的页面（甚至不同域名）加载后依旧存在，并且可以支持非常长的 name 值（2MB）。

###### 1) 创建a.com/cs1.html

###### 2) 创建a.com/proxy.html，并加入如下代码

<head>

<script>

function proxy(url, func){

var isFirst = true,

ifr = document.createElement('iframe'),

loadFunc = function(){

if(isFirst){

ifr.contentWindow.location = 'http://a.com/cs1.html';

isFirst = false;

}else{

func(ifr.contentWindow.name);

ifr.contentWindow.close();

document.body.removeChild(ifr);

ifr.src = '';

ifr = null;

}

};

ifr.src = url;

ifr.style.display = 'none';

if(ifr.attachEvent) ifr.attachEvent('onload', loadFunc);

else ifr.onload = loadFunc;

document.body.appendChild(iframe);

}

</script>

</head>

<body>

<script>

proxy('http://www.baidu.com/', function(data){

console.log(data);

});

</script>

</body>

###### 3) 在b.com/cs1.html中包含：

<script>

window.name = '要传送的内容';

</script>

#### (7) postMessage（HTML5中的XMLHttpRequest Level 2中的API）

##### 1) a.com/index.html中的代码：

<iframe id="ifr" src="b.com/index.html"></iframe>

<script type="text/javascript">

window.onload = function() {

var ifr = document.getElementById('ifr');

var targetOrigin = 'http://b.com'; // 若写成'http://b.com/c/proxy.html'效果一样

// 若写成'http://c.com'就不会执行postMessage了

ifr.contentWindow.postMessage('I was there!', targetOrigin);

};

</script>

##### 2) b.com/index.html中的代码：

<script type="text/javascript">

window.addEventListener('message', function(event){

// 通过origin属性判断消息来源地址

if (event.origin == 'http://a.com') {

alert(event.data); // 弹出"I was there!"

alert(event.source); // 对a.com、index.html中window对象的引用

// 但由于同源策略，这里event.source不可以访问window对象

}

}, false);

</script>

#### (8) web sockets

web sockets是一种浏览器的API，它的目标是在一个单独的持久连接上提供全双工、双向通信。(同源策略对web sockets不适用)

web sockets原理：在JS创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。

只有在支持web socket协议的服务器上才能正常工作。

##### 参考代码：

var socket = new WebSockt('ws://www.baidu.com');//http->ws; https->wss

socket.send('hello WebSockt');

socket.onmessage = function(event){

var data = event.data;

}

## Express中的“路由器”

Router：路由器，是一个可以包含多个路由声明的对象，可以相关的路由（Route）放置到同一个路由器（Router）中

|  |
| --- |
| //news.js  let router = require('express').Router();  router.get(**'/list'**, function(req, res){}) //news/list  router.get(**'/detail'**, function(req, res){}) //news/detail  module.exports = router; |
| //app.js  let routerNews = require('./news');  app.use('**/news'**, routerNews); |

## Node/Express项目中使用MySQL数据库

必须下载第三方的mysql模块：

npm i mysql

const mysql = require('mysql');

let pool = mysql.createPool({

host: '127.0.0.1',

user: 'root',

password: "",

database: "mfresh",

connectionLimit: 10

});

pool.query("SQL.....", function(err, result, cols){

})

# Gulp

## 1. 打开命令行：

1.1 WebStorm里：Alt + F12

1.2 进入目录：按 Shift，点击鼠标右键-在此处打开命令窗口

## 2. Dos 命令

2.1 cd <目录名> 切换目录 cd .. 回到上一级

2.2 dir 列出当前目录下的所有文件和目录

2.3 快捷方式 文件、目录名的时候，前几个字符，按Tab键

2.4 上、下一个命令：上、下箭头

2.5 盘符切换：d:

## 3. NPM 命令

3.1 package.json 的创建：当前目录下命令：npm init

3.2 cnpm: npm install -g cnpm --registry=https://registry.npm.taobao.org

3.3 npm install <包名> 安装第三方包，简写为 npm i <包名>

3.4 npm install <包名> --save 安装第三方包，把包加入package.json的dependencies里，项目的依赖

3.5 npm install <包名> --save-dev 安装第三方包，把包加入package.json的devDependencies里，开发版的依赖

## 4. Gulp

4.1 安装 Node.js

4.2 创建一个目录 gulp-demo

4.3 目录里执行命令：npm init，创建 package.json

4.4 目录里执行命令：cnpm install gulp -g 和 cnpm install gulp --save-dev 安装 gulp，放入package.json的devDependencies里

4.5 Gulp 的作用

a. 检查 JavaScript 的错误

b. 编译 CSS 预处理器文件 Less、Scss、Sass、Stylus

c. 合并项目的所有的 JavaScript 文件

d. 压缩文件

4.6 安装 Gulp 的依赖,当前目录\*\*\*以管理员身份\*\*\*执行：

cnpm install jshint gulp-jshint gulp-sass gulp-concat gulp-uglify gulp-rename --save-dev

4.7 目录里手动创建 gulp 的配置文件 gulpfile.js

4.8 Gulp 的方法有五个：

a. task：gulp 的任务

b. src：源代码的路径

c. dest：destination 目的地路径

d. run：运行，执行 Gulp 任务

e. watch：监听文件的变化

4.9 执行 Gulp 任务：当前项目目录命令行运行：

gulp [default]

# webpack

1. 创建目录 webpack-demo

2. 创建文件 package.json cmd：npm init

3. 安装 webpack， 加入全局依赖 cmd: cnpm install webpack -g

4. 加入项目开发依赖 cmd: cnpm install webpack --save-dev

5. 创建目录 app，创建文件 app/hello.js

6. 创建文件 app/main.js

7. 创建目录 public，创建文件 public/index.html

8. 运行命令：webpack app/main.js public/bundle.js

9. 浏览器打开 index.html

------------------------

10. 项目根目录下，创建 Webpack 的配置文件：webpack.config.js

11. 运行命令 cmd: webpack

12. 修改 package.json 文件，在 scripts 里加入 "start":"webpack"

13. 运行命令 cmd: npm [run] start，run 仅在 start 时可以省略

------------------------

14. 安装 dev 依赖：cnpm install webpack-dev-server --save-dev

15. 修改 webpack.config.js，加入服务器配置项

16. 修改 package.json，在 scripts 里加入 "server":"webpack-dev-server --open"

17. 运行命令 cmd: npm run server

------------------------

18. 创建文件：app/config.json

19. 修改文件：app/hello.js

------------------------

20. 加入项目开发依赖 cmd: cnpm install babel-core babel-loader babel-preset-es2015 babel-preset-react --save-dev

21. 修改文件：./webpack.config.js

22. 加入项目依赖 cmd: cnpm install react react-dom

23. 修改文件：app/hello.js

24. 修改文件：app/main.js

------------------------

25. 加入项目开发依赖 cmd: cnpm install style-loader css-loader –-save-dev

25.1 css-loader: 使用 import 引入 css

25.2 style-loader: 把 css 加入 JavaScript

26. 修改文件：./webpack.config.js

# Mocha

1. 全局安装 mocha：cmd: cnpm i -g mocha

2. cmd: npm init

3. 安装 mocha 为项目 dev 依赖：cmd: cnpm i --save-dev mocha

4. 安装 chai 为项目 dev 依赖：cmd: cnpm i --save-dev chai

5. 创建文件 t1/add.js 被测试的函数

6. 创建文件 t1/add.test.js 测试用例

7. cmd: mocha add.test.js

# 就业辅导

## 前端三大难点：

兼容性、Web访问优化、Web安全

## 掌握技能

### （1）

熟练掌握PS抠图，切图，将UI设计转化为符合W3C规范的DIV+CSS静态页面，确保浏览器及平台的性能和兼容性；熟练掌握HTML5、CSS3，能够熟练实现页面布局、弹性、响应式布局，实现动画特效；能够解决常见的浏览器的兼容性问题；熟练html5shiv解决H5兼容性、使用normalize.css进行CSSReset。熟练使用Transtion/KeyFrames动画，以及animate.css实现动态特效。

### （2）

熟练掌握JavaScript面向对象编程、原生DOM&BOM操作，屏蔽常见的兼容性问题；掌握ES6、TypeScript；同时熟练使用jQuery常用封装函数进行DOM操作、动画特效、AJAX调用；并能熟练进行jQuery插件编写。

### （3）

熟悉HTTP/1.1内容，了解HTTP/2.0新特性。熟练原生掌握AJAX调用，熟练JSON数据解析，并可以使用多种方法解决XHR跨域请求。

### （4）

熟悉HTML5新特性，视频和音频播放控制——有播放器编写经验；使用Canvas绘图统计图表/数据可视化、2D网页游戏；使用WebWorker并发的数据处理；WebSocket实现双向即时应用。

### （5）

能够CSS Media Query编写响应式布局，熟练Bootstrap样式库。会使用Less、SCSS等动态样式语言。

### （6）

熟悉常见的设计模式，如单例模式、工厂模式、观察者模式等；熟悉 Vue、Angular、React等MV\*框架，熟悉模块化编程，实现SPA应用。

### （7）

熟悉Ionic，能够使用特有的标签实现一定功能的页面；WebApp；PhoneGap实现混合编程。了解微信公众号的开发，微信小程序;

### （8）

使用Gulp进行自动化构建、Webpack项目打包、Mocha/Karma单元测试、Yeoman脚手架工具创建项目；熟练SVN/Git源代码管理以及协同开发

### （9）

有PHP使用经验创建动态Web应用、Node.js创建前后端完全分离的Web应用。

### （10）

熟练使用SQL操作关系型数据库，如MySQL；非关系型数据库MongoDB、Redis；

|  |
| --- |
| 参考招聘启事：  熟悉W3C标准和各主流浏览器在前端开发中的差异，能熟练运用HTML5、CSS3，提供针对不同浏览器的前端页面解决方案；  精通Javascript语言的各特性，有原声js性能优化的实践经验（对js跨域访问有实际处理经验者佳） |

## 项目经验：

### 项目名称：

学子商城(www.codeboy.com)

### 2.所用技术：

AJAX/跨域、jQuery、Bootstrap、Vue.js、Ionic、Angular.js、Node/Express、MySQL、Webpack、Mocha

### 3.项目描述：

学子商城分为“前台(响应式)子系统”、“后台管理端(Vue.js)”、“WebApp移动客户端(Ionic/RN)”，前后端彻底分离，实现了用户模块（包括手机号/微信/微博登录、注册）、商品模块(模糊检索、分类检索、详情查看)、购物车模块相关操作(添加/维护购物车、生成订单)。进行数据可视化开发，进行了相关的Web访问优化。解决了若干框架级冲突。

# 面试题:

## 1.)char(8)和varchar(8)的区别.

**定长字符串,有可能会产生空间浪费,但读取速度快**

**Char(8):**

**‘a’ 实际为 ’a\0\0\0\0\0\0\0’ (‘\0’表示字符串结束)**

**‘ab’ 实际为 ‘ab\0\0\0\0\0\0’**

**‘abc’ 实际为 ‘abc\0\0\0\0\0’**

**…**

**‘abcdefg’ 实际为 ‘abcdefg\0’**

**‘abcdefgh’ 实际为 ‘abcdefgh’**

**‘abcdefghi’ 实际为 ‘abcdefgh’**

**变长字符串,不会产生空间浪费,但读取速度稍慢**

**Varchar(8)**

**‘a’ 实际为 ‘a\0’**

**‘ab’ 实际为 ‘ab\0’**

**‘abc’ 实际为 ‘abc\0’**

**…..**

**‘abcdefg’ 实际为 ‘abcdefg\0’**

**‘abcdefgh’ 实际为 ‘abcdefgh’**

**‘abcdefghi’ 实际为 ‘abcdefgh’**

## 2)

**$n=2;**

**$r=$n++ + ++$n + $n++; //$r=10; $n=5;**

**求$n和 $r**

## 3)如何计算一个数字\*8得到的结果

**$num = $num\*8;**

**$num \*= 8;**

**$num << 3; //速度最快**

**面试题:如何计算一个数字\*16得到的结果**

**$num << 4; //速度最快**

## 4)visibility : hidden 与 display : none 的区别

**display : 会脱离文档流 , 所以不占页面空间**

**visibility : 并未脱离文档流 , 所以还占页面空间**

# 鄙视 :

## 1.)请列举JS中所有内置对象 : 11个

**包含 : String Number Boolean**

**Array Date RegExp Math**

**Error Function Object Global(在浏览器中被window代替)**

## 2) 对包装类型的理解

## 3) 定义函数 , 判断一个数是不是质数/素数

**function isPrime(n){**

**if(n<=3) return true;**

**else if(n%2==0) return false;**

**else{**

**for(var i=3;i<Math.sqrt(n);i+=2){**

**if(n%i==0) return false;**

**}**

**return true;**

**}**

**}**

## 4)错误的类型 :

### 1.SyntaxError : 语法错误

### 2.ReferenceError : 引用错误

要用的变量 / 对象 , 没找到!

### 3.TypeError : 类型错误

错误的使用了类型 :

1.错误的使用了( ) 或 [ ]

2.错误的调用了对象的方法

### 4.RangeError : 范围错误

参数超范围;

### 5.EvalError

### 6.URIError

## 5)以下创建函数正确的是:

**function cmp(a,b){return a-b;} √**

**var cmp = function(a,b){return a-b;}; √**

**var cmp = new Function("a","b","return a-b"); √**

**var cmp = new Function(a,b,"return a-b"); ×**

## 6)判断一个对象是不是数组类型，有几种方法:

### typeof不行！

### 1. 判断原型对象:

**Object.getPrototypeOf(obj)==Array.prototype**

判断obj是数组类型的子对象

**问题: \_\_proto\_\_是内部属性，本不应该被访问到**

**解决: 用Object.getPrototypeOf(obj) 代替\_\_proto\_\_**

### 2. 判断构造函数:

实例

**obj instanceof Array**

判断obj是不是被构造函数Array创造出来的

**instanceof 不仅判断直接父类型，而是所有在原型链上的类型，都返回true！**

### 3. 判断对象的内部class属性

每个对象内部，都有一个隐藏的**class**属性，记录该对象创建时的数据类型

**class**属性不会随继承关系的改变而改变

**问题1: class是内部属性**

**解决:** 只有最顶层的toString()才能输出对象的class属性值

[object class名]

**问题2:** 内置类型的原型对象中几乎都重写了新的toString()

**解决:** 用call强行调用:

**call:** 让一个对象，调用一个本来无法调用到的函数

何时: 只要希望调用一个本无法调用到的函数

如何: 要调用的函数.call(对象)

**Object.prototype.toString.call(obj)=="[object Array]"**

说明obj的内部属性class的值为"Array"

### 4.使用Array中的isArray函数

**Array.isArray(obj)**

## 7) call vs apply vs bind

call ,apply 立刻调用函数

apply: 同时，打散数组参数为单个参数值，再传给fun

bind: 基于原函数，创建一个新函数，永久绑定this为指定对象

## 8)定义函数，遍历指定父节点下所有子元素

#### 如何:

### 方法一: 递归: 2步:

1. 定义函数，仅查找指定父节点下所有直接子元素

2. 对每个子元素，调用和父元素完全相同的操作

递归: arguments.callee

深度优先遍历: 当同时有兄弟元素和子元素时，首先遍历子元素。所有子元素遍历完，才返回遍历兄弟元素。

### 方法二: 循环: 2步:

1. 创建节点迭代器对象:

节点迭代器: 自动根据深度优先顺序，依次访问每个节点的小对象

2. 反复调用迭代器对象的nextNode()方法，获得下一个节点对象

## 9)按 HTML vs 按选择器

### 1.返回值 :

1.按HTML 返回动态集合

2.按选择器 慢

### 2.首次查找效率 :

按 HTMl 块

按选择器 慢

### 3.易用性 :

条件复杂: 按HTML繁琐 ,按选择器简洁

### 总结 :

如果一个条件就能找到想要元素时就按HTML查找

如果查找条件复杂时就用按选择器查找

jQuery中，就是这样选择的。

**jspref //查效率**

## 10)如何快速学习一门新技术:

### 1.上官网 : 看是什么

### 2.找快速入门案例:

### 3.找手册:

### 4.做一个完整的案例

## 11)jQuery共几种事件绑定方式:

### 1. .bind/unbind

仅为简化addEventListener/removeEventListener

unbind: 3种重载:

1. unbind("事件名",原处理函数对象)

仅移除当前元素的当前事件上的一个指定处理函数

等效于removeEventListener

2. unbind("事件名")

移除当前元素的当前事件上绑定的所有处理函数

3. unbind() 移除当前元素上的所有事件

### 2. .one 仅触发一次事件，触发后，自动解绑

### 3. .live/die 已废弃

### 4. .delegate/undelegate 其实就是利用冒泡！

$(parent).delegate("selector","事件名",fn)

将事件绑定在parent上，利用冒泡

仅限于符合selector条件的子元素触发事件

fn中的this自动指向目标元素

### 5. .on:

既可以代替bind: .on/off("事件名",fn)

又可以代替delegate: .on/off("事件名","selector",fn)

### 6. 常用事件: 都提供了简写:

.事件名(fn)

## 12)bind vs delegate

### 1. 绑定的元素不同:

bind绑在每个子元素上

delegate 绑在父元素上

### 2. 添加的监听个数不同:

bind 每个子元素都添加一个监听——多

delegate 仅在父元素添加一个监听——少

### 3. 动态添加子元素时:

.bind 动态添加的新元素，无法自动获得事件处理函数

delegate 动态添加的新的子元素，

自动可使用父元素上绑定的处理函数

## 13)$(document).ready() vs onload

### 1. 加载时机不同: ready()早，onload晚

### 2. ready()可绑定多个处理函数

onload用等号赋值，只能绑定一个处理函数

## 斐波拉切数列

**function f1(n){**

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

**return 1;**

**}else{**

**return f(n-1)+f(n-2);**

**}**

**}//面试题标准答案**

**function f2(n){**

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

**return 1;**

**}else{**

**/\*f1 f2 fn**

**1 + 1 = 2 n=3**

**1 + 2 = 3 n=4**

**2 + 3 = 5 n=5**

**3 + 5 = 8 n=6**

**\*/**

**var f1=1,f2=1,fn;**

**for(var i=3;i<=n;i++){**

**fn=f1+f2;**

**f1=f2;**

**f2=fn;**

**}**

**return fn;**

**}**

**}//效率更高**

## 14)如何让一个回调函数无论ajax请求正确与否都能执行

### 错误:

$().load() $.get/post/... 只能在成功时才执行

### 正确:

只有$.ajax({ ..., complete(){...},...})

无论正确与否，都执行。

### 其实:

load,get,post底层调用都是ajax

## 15)公司里多台服务器或多个子网站间，发起异步ajax请求: OA, HR, MAIL,...或.html, .php, .图片,

### $.get/post肯定不行！

### 如何实现: 4种:

#### 1.

#### 2.

#### 3.

#### 4. JSONP: 专门解决xhr跨域请求问题

##### 什么是: JSON with Padding 填充式json

##### 如何: xhr无论如何不可能跨域请求

只能用允许跨域请求的元素(script)，代替xhr

##### 原生: 3步:

###### 1. 客户端定义一个专门处理响应的有名字的函数

function doResponse(data){

处理返回的data结果

}

强调:处理函数中仅接受并处理服务端json数据

不用xhr对象，更不用判断状态。。。

###### 2. 服务器端php:

处理完数据后

返回一条js语句:

echo 'doResponse($json)'

###### 3. 动态创建一个script元素，设置src指向服务器端php地址。

## 补充小知识

如何存储项目中的'日期/时间'数据

方式1:'DATE/DATETIME',不足: 不同国家的人日期格式不同

方式2:varchar,不足: 不方便比较大小

方式3:bigint ,表示距离计算机元年经过了多少毫秒 1000 =>1970-1-1 0:0:1

计算机元年:格林威治时间:1970-01-01 00:00:00