**WEB前端相关笔记**

# 目 录

[第一部分 Mysql数据库 20](#_Toc838)

[第一章 数据库服务器概述 20](#_Toc19620)

[1. 如何保存数据 20](#_Toc22794)

[2. 数据库概述 20](#_Toc5953)

[3. 关系型数据库服务器 20](#_Toc7058)

[第二章 MySQL数据库 20](#_Toc22061)

[1. 数据库发展 20](#_Toc614)

[2. 关系型数据库的结构 20](#_Toc4494)

[3. 常见的数据库 20](#_Toc28873)

[4. MySQL软件介绍 20](#_Toc5231)

[4.1. xampp软件 20](#_Toc28459)

[4.2. 安装使用MySQL数据库系统 21](#_Toc12513)

[第三章 mysql常用管理命令 21](#_Toc19754)

[1. SQL语句两种执行方式 21](#_Toc3708)

[1.1. 交互模式： 21](#_Toc6322)

[1.2. 脚本模式： 21](#_Toc7553)

[2. MySQL常用管理命令 21](#_Toc15092)

[第四章 MySQL中的SQL语句——重点 22](#_Toc23655)

[1. SQL语法规范 22](#_Toc21077)

[2. 标准SQL语句的分类 22](#_Toc19114)

[3. 日常开发中常用的SQL语句 22](#_Toc22652)

[4. 数据库中的中文乱码 25](#_Toc31492)

[4.1. 计算机如何存储英文字符 25](#_Toc24102)

[4.2. 计算机中如何存储中文存储 25](#_Toc2171)

[5. 列类型 26](#_Toc12923)

[5.1. 数值类型——数据可以不加引号 26](#_Toc3312)

[5.2. 日期时间类型——必须加引号 27](#_Toc14818)

[5.3. 字符串类型——必须加引号 27](#_Toc27920)

[6. 列约束 29](#_Toc6316)

[6.1. 主键约束——PRIMARY KEY 29](#_Toc11473)

[6.2. 唯一约束——UNIQUE 29](#_Toc14368)

[6.3. 扩展知识：NULL 29](#_Toc22763)

[6.4. 非空约束——NOT NULL 31](#_Toc8343)

[6.5. 默认值约束——DEFAULT 31](#_Toc13535)

[6.6. 检查约束——CHECK 31](#_Toc14798)

[6.7. 外键约束 32](#_Toc6808)

[6.8. 扩展知识：MySQL中的自增列 32](#_Toc19111)

[7. 简单查询 34](#_Toc8100)

[7.1. 查询特定的列 34](#_Toc7134)

[7.2. 查询所有的列 34](#_Toc30096)

[7.3. 给列取别名 34](#_Toc32131)

[7.4. 只显示不同的记录(合并相同的记录) 34](#_Toc26111)

[7.5. 在查询时执行计算 34](#_Toc11581)

[7.6. 查询结果集的排序 35](#_Toc4191)

[7.7. 条件查询 36](#_Toc19355)

[7.8. 模糊条件查询 38](#_Toc32367)

[7.9. 分页查询 38](#_Toc16645)

[8. 复杂查询 39](#_Toc3586)

[8.1. 聚合查询/分组查询 39](#_Toc14081)

[8.2. 子查询 41](#_Toc24974)

[8.3. 多表查询 42](#_Toc12531)

[9. 综合的语法 43](#_Toc22163)

[第二部分 JavaScript 44](#_Toc26491)

[第一章 学一门新语言的基本步骤 44](#_Toc14300)

[第二章 JavaScript概述 44](#_Toc25627)

[1. JS的执行环境 44](#_Toc21288)

[2. 应用领域 44](#_Toc1750)

[3. 发展历史 44](#_Toc5310)

[4. 特点 44](#_Toc11738)

[5. 搭建js的开发环境 45](#_Toc8440)

[6. js的运行环境 45](#_Toc14205)

[6.1. 浏览器 45](#_Toc25287)

[6.2. nodejs下（服务器） 45](#_Toc6413)

[7. JS书写规范 45](#_Toc32703)

[第三章 变量 45](#_Toc22858)

[1. 声明变量使用关键字var-->variable 45](#_Toc25909)

[2. 变量的命名规则 45](#_Toc29568)

[3. 变量的声明 46](#_Toc32612)

[第四章 常量 constant（常数、常量，缩写const） 47](#_Toc4251)

[第五章 数据类型 47](#_Toc29375)

[1. 数值型 47](#_Toc11690)

[1.1. 整数 47](#_Toc12505)

[1.2. 普通小数和指数小数 47](#_Toc17321)

[2. 字符串型 47](#_Toc19854)

[3. 布尔型 48](#_Toc25239)

[4. 未定义型 48](#_Toc24132)

[5. null 48](#_Toc533)

[第六章 数据类型转换 48](#_Toc15497)

[1. 隐式转换 48](#_Toc7205)

[2. 强制转换 49](#_Toc3676)

[2.1. 将任意的类型转为整数 49](#_Toc2756)

[2.2. 将任意的类型转为浮点型 49](#_Toc18901)

[2.3. 将任意的类型转为数值型 49](#_Toc10099)

[2.4. 将数值转为字符串 50](#_Toc22293)

[2.5. 布尔型转化为字符串 50](#_Toc25581)

[第七章 运算符和表达式 50](#_Toc19100)

[1. 算数运算符 50](#_Toc3479)

[2. 比较运算符 51](#_Toc17279)

[2.1. 比较运算符返回布尔型的值 51](#_Toc14704)

[2.2. 字符串型被转成了数值型 51](#_Toc5134)

[2.3. 字符串比较 51](#_Toc1167)

[2.4. NaN的比较 52](#_Toc20715)

[3. 逻辑运算符 52](#_Toc30058)

[4. 逻辑短路 52](#_Toc422)

[5. 位运算符 54](#_Toc27770)

[6. 赋值运算符 54](#_Toc18257)

[7. 三目运算符 55](#_Toc18182)

[第八章 两个浏览器端函数 55](#_Toc12463)

[第九章 流程控制 56](#_Toc11529)

[1. 顺序执行 56](#_Toc28910)

[1.1. if语句 56](#_Toc2966)

[1.2. if-else语句 57](#_Toc5382)

[1.3. if-else语句的嵌套 58](#_Toc1840)

[1.4. switch-case语句 60](#_Toc30001)

[1.4.1. 对比if-else嵌套和switch-case语句 62](#_Toc19938)

[2. 循环执行 62](#_Toc19233)

[2.1. while循环 62](#_Toc12065)

[2.2. do-while循环 65](#_Toc23256)

[2.2.1. 对比while和do-while的区别 67](#_Toc20415)

[2.3. for循环 67](#_Toc19280)

[2.3.1. 对比break和continue的区别 70](#_Toc17018)

[2.4. 循环嵌套 71](#_Toc30039)

[第十章 函数 73](#_Toc20964)

[1. 函数的概念 73](#_Toc25155)

[2. 自定义函数 73](#_Toc28349)

[2.1. 创建普通函数 73](#_Toc16761)

[2.2. 创建带有参数的函数 74](#_Toc3578)

[2.3. 带有返回值的函数 75](#_Toc4813)

[2.3.1. 对比break，continue，return用法 78](#_Toc23554)

[3. 作用域 78](#_Toc30025)

[3.1. 变量的作用域 78](#_Toc5874)

[3.2. 变量声明提升 79](#_Toc17639)

[3.3. 变量的生存期 80](#_Toc12264)

[3.4. 函数的作用域 80](#_Toc18227)

[3.5. 函数声明的提升 82](#_Toc5019)

[4. 递归 82](#_Toc26104)

[5. 匿名函数 84](#_Toc15569)

[5.1. 创建函数——函数表达式 84](#_Toc12391)

[5.1.1. 对比函数表达式和函数声明的区别 84](#_Toc1908)

[5.2. 匿名函数的自调用 85](#_Toc24148)

[5.3. 回调函数 86](#_Toc12623)

[6. 全局函数 87](#_Toc11902)

[7. 对象 88](#_Toc7615)

[7.1. JS中的对象 88](#_Toc2)

[7.2. 创建自定义对象 88](#_Toc23513)

[7.3. 对象字面量创建对象 88](#_Toc18528)

[7.4. 使用new Object创建对象 89](#_Toc10966)

[7.5. 遍历（循环）属性 90](#_Toc21186)

[7.6. 检查对象中是否含有某个属性 91](#_Toc22211)

[7.7. 对象中的方法 92](#_Toc23141)

[8. 数组 94](#_Toc22466)

[8.1. 1.创建数组 94](#_Toc24910)

[8.1.1. 数组字面量 94](#_Toc12397)

[8.1.2. 内置构造函数——new Array 94](#_Toc28090)

[8.2. 数组元素的访问 95](#_Toc11274)

[8.3. 获取数组的长度 95](#_Toc27617)

[8.4. 遍历数组 96](#_Toc25530)

[8.4.1. 遍历索引数组 97](#_Toc7943)

[8.5. 数组的分类 100](#_Toc27081)

[8.6. 数组中的方法（API） 101](#_Toc9206)

[8.7. 二维数组 105](#_Toc13507)

[9. 字符串操作 105](#_Toc22606)

[9.1. 包装对象 105](#_Toc28675)

[9.2. 转义字符——\ 106](#_Toc21842)

[9.3. 字符串中的常用方法 107](#_Toc17869)

[10. 匹配模式 111](#_Toc25111)

[11. Math对象 112](#_Toc5945)

[12. Date对象 115](#_Toc8260)

[12.1. 实例化Date对象 116](#_Toc30841)

[12.2. 本地化时间格式 117](#_Toc4042)

[12.3. 获取时间信息 117](#_Toc15419)

[12.4. 设置时间信息 118](#_Toc10540)

[13. Number对象 120](#_Toc28475)

[14. Boolean对象 120](#_Toc2908)

[第十一章 错误处理 121](#_Toc27966)

[第十二章 异常处理 122](#_Toc24890)

[第十三章 ES6新特征 122](#_Toc17165)

[1. 块级作用域 122](#_Toc1069)

[2. 箭头函数 123](#_Toc27950)

[3. 函数中的参数 123](#_Toc19576)

[4. 模板字符串 124](#_Toc16741)

[第三部分 NodeJavaScript 126](#_Toc17713)

[第一章 nodeJS概述 126](#_Toc326)

[1. NODEJS运行 126](#_Toc26891)

[1.1. 交互模式 node 126](#_Toc18020)

[1.2. 脚本模式 126](#_Toc30287)

[2. Node.js的特点 126](#_Toc15581)

[2.1. 优势 126](#_Toc7094)

[2.2. 劣势 126](#_Toc1872)

[第二章 Node.js全局对象 127](#_Toc3334)

[1. 全局对象 127](#_Toc16259)

[1.1. console对象 127](#_Toc17823)

[1.2. 计算程序的运行时间 128](#_Toc24900)

[1.3. process对象（了解） 129](#_Toc12374)

[1.4. Buffer对象 129](#_Toc6476)

[2. 全局函数 130](#_Toc26454)

[2.1. 一次性定时器 130](#_Toc38)

[2.2. 周期性定时器 130](#_Toc5086)

[2.3. process.nextTick(回调函数) 131](#_Toc18796)

[2.4. 开启 131](#_Toc29971)

[2.5. 同步和异步 131](#_Toc29146)

[第三章 模块 132](#_Toc3650)

[1. 模块的分类 132](#_Toc2820)

[2. 模块封装 132](#_Toc16447)

[3. 相关函数 132](#_Toc31157)

[4. 包和npm 135](#_Toc28367)

[5. 工具模块 136](#_Toc25660)

[5.1. querysting模块 136](#_Toc6710)

[5.2. URL模块 137](#_Toc14389)

[5.3. Buffer对象 139](#_Toc22114)

[5.4. 文件系统模块 139](#_Toc11154)

[5.4.1. 对比同步方法和异步方法的区别 140](#_Toc5196)

[5.5. FS模块 141](#_Toc31471)

[6. HTTP 144](#_Toc6021)

[6.1. HTTP协议 144](#_Toc30578)

[6.1.1. 通用头部General 144](#_Toc270)

[6.1.2. 请求的头信息Request headers 145](#_Toc18632)

[6.1.3. 响应的头信息Reponse headers 145](#_Toc25052)

[6.2. HTTP模块 145](#_Toc11270)

[6.2.1. 作为客户端浏览器 145](#_Toc6858)

[6.2.2. 创建web服务器 146](#_Toc32360)

[6.3. Express 149](#_Toc15552)

[6.3.1. Express概述 149](#_Toc6991)

[6.3.2. 使用Express创建服务器 149](#_Toc29521)

[6.3.3. 路由 149](#_Toc3820)

[6.3.3.1. 请求的对象req 150](#_Toc5417)

[6.3.3.2. 响应的对象res 152](#_Toc27509)

[6.3.4. 获取浏览器端请求的数据 153](#_Toc4347)

[6.3.4.1. POST请求方法的数据 153](#_Toc15579)

[6.3.4.2. GET请求方法的数据 153](#_Toc25297)

[6.3.4.3. 对比POST和GET方法 153](#_Toc14171)

[6.3.5. 路由传递参数 155](#_Toc23788)

[6.3.5.1. 配置路由 155](#_Toc24412)

[6.3.5.2. 请求URL 155](#_Toc10390)

[6.3.6. 路由器 156](#_Toc7543)

[6.3.6.1. 使用步骤 156](#_Toc18070)

[6.3.7. 引入第三方模块 158](#_Toc1958)

[6.3.8. express中间件 158](#_Toc21536)

[6.3.8.1. 中间件概述 158](#_Toc16511)

[6.3.8.2. 中间件分类 160](#_Toc6839)

[6.3.8.3. 第三方的中间件 161](#_Toc16581)

[6.3.8.4. 错误级的中间件 161](#_Toc169)

[6.3.8.5. npm使用 162](#_Toc31300)

[6.3.8.6. req对象 163](#_Toc13129)

[7. MySQL模块 163](#_Toc25001)

[7.1. 普通连接 163](#_Toc10606)

[7.2. 使用连接池 164](#_Toc31711)

[第四章 连接错误 167](#_Toc24578)

[第五章 项目一 167](#_Toc22314)

[第四部分 HTTP 169](#_Toc9516)

[第一章 导论 169](#_Toc1573)

[1. 课程安排 169](#_Toc4587)

[2. 第二阶段知识特点 169](#_Toc29563)

[3. 学习要求 169](#_Toc19798)

[第二章 web基础知识 170](#_Toc23698)

[1. web和internet 170](#_Toc20259)

[1.1. web简介 170](#_Toc29326)

[1.2. Internet简介 170](#_Toc17189)

[1.3. web和internet 170](#_Toc23625)

[2. Internet上的应用程序 170](#_Toc32213)

[第三章 web运行原理 171](#_Toc26936)

[1. 服务器 171](#_Toc12847)

[1.1. 功能 171](#_Toc1111)

[1.2. 服务器的产品 171](#_Toc23399)

[1.3. 写服务器的技术: 172](#_Toc12176)

[2. 浏览器 172](#_Toc8507)

[2.1. 功能 172](#_Toc21304)

[2.2. 浏览器的技术 172](#_Toc29684)

[2.3. 浏览器的产品 172](#_Toc19096)

[第四章 HTML快速入门 172](#_Toc17748)

[1. 什么是HTML 172](#_Toc7580)

[2. HTML的特点 173](#_Toc11773)

[3. HTML基础语法 173](#_Toc6431)

[3.1. 标记 173](#_Toc2844)

[3.2. 标记的分类 173](#_Toc17003)

[3.3. 元素 174](#_Toc15963)

[3.4. 标签的嵌套 174](#_Toc18405)

[3.4.1. 什么是嵌套 174](#_Toc19043)

[3.4.2. 语法 174](#_Toc24582)

[3.5. 属性和值 175](#_Toc6264)

[3.6. 公有属性 175](#_Toc20737)

[3.7. 注释 175](#_Toc22520)

[第五章 HTML的文档结构 176](#_Toc28368)

[1. 文档类型的声明 176](#_Toc11967)

[2. 网页的结构 176](#_Toc2688)

[3. head元素 176](#_Toc27071)

[4. body元素 177](#_Toc25837)

[5. 文本标记 177](#_Toc5969)

[5.1. 标题元素 177](#_Toc2127)

[5.2. 段落标记 p 178](#_Toc260)

[5.2.1. 特点 178](#_Toc18249)

[5.2.1.1. 独占一行 178](#_Toc26874)

[5.2.1.2. 文本上下有空白间距 178](#_Toc21851)

[5.2.1.3. 换行 178](#_Toc20239)

[5.2.1.4. 水平线 178](#_Toc19420)

[5.2.1.5. 预格式化 178](#_Toc25699)

[5.2.1.6. 特殊字符(实体) 179](#_Toc18422)

[5.2.1.7. 文本样式 179](#_Toc18425)

[5.2.2. 分区元素(逻辑分区) 179](#_Toc28973)

[5.2.2.1. 块分区 179](#_Toc15753)

[5.2.2.2. 行分区 179](#_Toc30077)

[5.2.3. 级元素和行内元素(行级元素) 180](#_Toc22462)

[5.2.3.1. 块级元素 180](#_Toc12053)

[5.2.3.2. 行内元素 180](#_Toc6509)

[5.2.3.3. 行内块元素 180](#_Toc2359)

[5.2.3.4. table 180](#_Toc28050)

[6. 图像和链接 180](#_Toc9414)

[6.1. 图像的使用 180](#_Toc641)

[7. URL 181](#_Toc20857)

[7.1. 目录结构 181](#_Toc26635)

[7.2. URL 181](#_Toc25236)

[7.2.1. 目录结构 181](#_Toc7720)

[7.2.2. URL 181](#_Toc30738)

[7.2.3. 路径的表现形式(url) 181](#_Toc22994)

[7.2.3.1. 绝对路径 181](#_Toc26062)

[7.2.3.2. 相对路径 181](#_Toc28439)

[7.3. 属性 182](#_Toc18611)

[7.4. 链接 182](#_Toc9191)

[7.4.1. a标签的其他表现形式 182](#_Toc16606)

[7.4.1.1. 调用js代码 182](#_Toc27186)

[7.4.1.2. 下载资源——不重要 182](#_Toc2491)

[7.4.1.3. 连接到电子邮件的客户端——不重要 183](#_Toc17410)

[7.4.1.4. 返回页面顶部——重要 183](#_Toc18483)

[7.4.2. a标签的锚点 183](#_Toc27009)

[7.4.2.1. 定义锚点 183](#_Toc26404)

[7.4.2.2. 跳转到锚点 183](#_Toc18436)

[7.4.2.3. 跳转到另外一个页面的某个锚点 183](#_Toc23736)

[8. 表格 184](#_Toc27281)

[8.1. 语法 184](#_Toc18264)

[8.2. 属性 184](#_Toc8062)

[8.2.1. table的属性 184](#_Toc1867)

[8.2.2. tr的属性 185](#_Toc3095)

[8.2.3. td的属性 185](#_Toc25158)

[8.3. 不规则的表格使用 185](#_Toc9843)

[8.4. 可选标签 186](#_Toc26737)

[8.5. 表格复杂的应用 186](#_Toc3799)

[8.5.1. 行分组(分区) 186](#_Toc20148)

[8.5.2. 表格嵌套 186](#_Toc19807)

[9. 列表(重点) 186](#_Toc20552)

[9.1. 列表的作用 186](#_Toc24004)

[9.2. 列表组成 186](#_Toc31191)

[9.3. 列表属性 187](#_Toc631)

[9.3.1. 有序列表的属性 187](#_Toc4797)

[9.3.2. 无序列表的属性 187](#_Toc24937)

[9.4. 列表嵌套 188](#_Toc15649)

[9.4.1. 列表嵌套列表 188](#_Toc16435)

[9.4.2. 列表嵌套其他元素 189](#_Toc27646)

[9.5. 定义列表 189](#_Toc8839)

[10. 结构标记<div></div> 语义化标签 189](#_Toc1626)

[10.1. 结构标记的作用 189](#_Toc25866)

[10.2. 常用的语义标签 190](#_Toc2355)

[11. 表单(重点&难点) 191](#_Toc0)

[11.1. 作用 191](#_Toc17906)

[11.2. 表单的组成 191](#_Toc14460)

[11.2.1. 前端部分 191](#_Toc2409)

[11.2.2. 后端服务 192](#_Toc31731)

[11.2.3. 表单 192](#_Toc6038)

[11.2.4. enctype编码方式 192](#_Toc16726)

[11.3. 表单控件(在表单中可以输入数据的标签) 193](#_Toc2766)

[11.3.1. 分类 193](#_Toc25641)

[11.3.2. input控件 193](#_Toc22785)

[11.3.3. 属性 193](#_Toc20786)

[11.3.3.1. type 193](#_Toc6180)

[11.3.3.2. name 193](#_Toc22960)

[11.3.3.3. value 193](#_Toc8981)

[11.3.3.4. disabled 193](#_Toc26941)

[11.3.4. input详解 194](#_Toc2672)

[11.3.4.1. 文本框和密码框 194](#_Toc31685)

[11.3.4.2. 按钮 194](#_Toc22826)

[11.3.4.2.1. 提交按钮 194](#_Toc23613)

[11.3.4.2.2. 重置按钮 195](#_Toc13042)

[11.3.4.2.3. 普通按钮 195](#_Toc14840)

[11.3.4.3. 单选框 195](#_Toc17062)

[11.3.4.4. 复选框 195](#_Toc20946)

[11.3.4.5. 隐藏域 195](#_Toc24922)

[11.3.4.6. 文件的选择框 195](#_Toc4490)

[11.3.5. 非input标签 196](#_Toc25293)

[11.3.5.1. 文本域 196](#_Toc3835)

[11.3.5.2. select和option 196](#_Toc6713)

[11.4. 其他控件 198](#_Toc14874)

[11.4.1. 关联文本域表单控件 198](#_Toc731)

[11.4.2. 为控件分组的标签 198](#_Toc10205)

[11.4.3. 浮动框架 198](#_Toc4534)

[11.5. 新表单元素 198](#_Toc1408)

[11.5.1. 电子邮件 199](#_Toc8784)

[11.5.2. 搜索框 199](#_Toc32448)

[11.5.3. URL类型的文本框 199](#_Toc16002)

[11.5.4. 电话号码文本框 199](#_Toc30458)

[11.5.5. 数字类型文本框 199](#_Toc16780)

[11.5.6. 范围选择 199](#_Toc13403)

[11.5.7. 颜色拾取器 200](#_Toc6414)

[11.5.8. 日期类型 200](#_Toc21894)

[11.5.9. 月份类型 200](#_Toc15003)

[11.5.10. 周类型 200](#_Toc27985)

[12. HTTP协议 204](#_Toc23291)

[12.1. URL 204](#_Toc15758)

[12.2. HTTP 205](#_Toc7250)

[12.3. HTTP详解 206](#_Toc21202)

[12.3.1. 请求消息 206](#_Toc30154)

[12.3.1.1. 请求的起始行 206](#_Toc15259)

[12.3.1.2. 请求头 206](#_Toc25960)

[12.3.1.3. 请求主体 FORM DATA 207](#_Toc8181)

[12.3.2. 响应消息 207](#_Toc2915)

[12.3.2.1. 响应起始行 207](#_Toc28429)

[12.3.2.2. 响应头 207](#_Toc11426)

[12.3.2.3. 响应主体 208](#_Toc27349)

[12.3.3. 响应状态码 208](#_Toc31060)

[12.3.4. 缓存 208](#_Toc10432)

[12.3.4.1. 缓存完整流程 209](#_Toc10035)

[12.3.5. 与缓存相关的消息头 209](#_Toc28006)

[12.3.5.1. Cache-Control http/1.1的用法 209](#_Toc24342)

[12.3.5.2. Expires http/1.0用法 209](#_Toc25183)

[12.3.6. 在网页中添加缓存 210](#_Toc8168)

[12.3.7. HTTP的性能优化 210](#_Toc24528)

[12.3.7.1. HTTP连接过程 210](#_Toc2684)

[12.3.7.2. HTTP连接性能的优化 210](#_Toc3966)

[12.3.7.3. 安全的http协议(了解) 210](#_Toc25813)

[13. DOM操作 210](#_Toc7528)

[13.1. 完整的JavaScript组成 210](#_Toc22748)

[13.2. 使用js获取页面上的某个元素 210](#_Toc2606)

[13.3. 获取/设置表单控件的数据 212](#_Toc20889)

[13.4. 获取/设置表单控件的数据 212](#_Toc5598)

[13.4.1. 关于方法名的要求 212](#_Toc13167)

[13.4.2. 简化 213](#_Toc11727)

[13.5. HTML的事件 213](#_Toc20196)

[13.5.1. 什么是事件 213](#_Toc24247)

[13.5.2. 要学习的事件 213](#_Toc11002)

[第六章 AJAX 215](#_Toc5158)

[1. 名词解释 215](#_Toc27446)

[1.1. 同步(Synchronous) 215](#_Toc30974)

[1.2. 异步(Asynchronous) 215](#_Toc30382)

[1.3. 什么是AJAX 215](#_Toc10416)

[2. 异步请求的步骤 216](#_Toc28905)

[2.1. 创建异步对象 216](#_Toc7991)

[2.2. 异步对象的属性 217](#_Toc23117)

[2.2.1. readyState属性 217](#_Toc1589)

[2.2.2. status属性 217](#_Toc7537)

[2.2.3. onreadystatechange——事件 217](#_Toc12181)

[2.3. 创建请求,打开链接 218](#_Toc14223)

[2.4. 发送请求 218](#_Toc21151)

[2.5. ajax使用步骤总结 219](#_Toc24035)

[2.5.1. 创建异步对象 XMLHttpRequest 219](#_Toc1645)

[2.5.2. 绑定监听事件 219](#_Toc20164)

[2.5.3. 创建请求，打开链接 219](#_Toc3341)

[2.5.4. 发送请求 219](#_Toc13492)

[2.5.5. 带参数的get请求 220](#_Toc23785)

[第七章 项目二 220](#_Toc29370)

[1. 准备工作 220](#_Toc548)

[1.1. 建立服务器 220](#_Toc11459)

[1.2. 建立路由器 221](#_Toc1)

[1.3. 建立挂载路由文件 221](#_Toc13281)

[1.4. 建立通用js工具库 222](#_Toc23205)

[2. 建立登录界面 222](#_Toc32154)

[2.1. 建立前端文件 222](#_Toc29607)

[2.2. 建立后台文件(在myPro.js中定义接口login) 223](#_Toc18093)

[3. 建立用户列表界面 224](#_Toc30789)

[3.1. 设立基本功能 224](#_Toc17649)

[3.1.1. 建立前端页面 224](#_Toc22082)

[3.1.2. 建立后台文件(在mypro.js中定义接口list) 225](#_Toc10556)

[3.2. 设立删除功能 225](#_Toc5305)

[3.2.1. 修改前端页面 225](#_Toc31092)

[3.2.2. 在后台文件(在mypro.js中定义接口deleteUser) 226](#_Toc6238)

[3.3. 设立修改功能 227](#_Toc19513)

[4. 建立修改页面 227](#_Toc22288)

[4.1. 建立基本功能 227](#_Toc31431)

[4.1.1. 建立前端页面 227](#_Toc23714)

[4.1.2. 建立后台文件(在mypro.js中定义接口selUser) 229](#_Toc17152)

[4.2. 建立修改功能 229](#_Toc21317)

[4.2.1. 修改前端页面 229](#_Toc25286)

[4.2.2. 建立后台文件(在mypro.js中定义接口updateUser) 230](#_Toc5122)

[5. 注册功能 231](#_Toc16131)

[5.1. 设立前端页面 231](#_Toc10261)

[5.2. 用户名验证 232](#_Toc31761)

[5.2.1. 修改前端页面 232](#_Toc10806)

[5.2.2. 建立后台文件(在mypro.js中定义接口checkUname) 233](#_Toc23148)

[5.3. 用户注册 233](#_Toc12046)

[5.3.1. 修改前端页面 233](#_Toc456)

[5.3.2. 建立后台文件(在mypro.js中定义接口register) 234](#_Toc22122)

[6. 邮箱验证功能 235](#_Toc30212)

[6.1. 修改前端页面 235](#_Toc29417)

[6.2. 建立后台文件(在mypro.js中定义接口doubleEmail) 235](#_Toc3219)

[7. 两次密码输入一致验证功能 236](#_Toc25655)

[7.1. 修改前端页面 236](#_Toc11859)

[第五部分 CSS3 237](#_Toc24601)

[第一章 CSS3概述 237](#_Toc20710)

[1. 名词解释 237](#_Toc2258)

[2. 作用 237](#_Toc11273)

[3. HTML与CSS的关系 237](#_Toc25508)

[4. HTML与CSS属性的使用原则 237](#_Toc4739)

[第二章 CSS3 基础 237](#_Toc6275)

[1. CSS3概述 237](#_Toc21404)

[1.1. 名词解释 237](#_Toc27698)

[1.2. 作用 237](#_Toc28992)

[1.3. HTML与CSS的关系 238](#_Toc12377)

[1.4. HTML与CSS属性的使用原则 238](#_Toc6195)

[2. CSS3的语法概述 238](#_Toc355)

[2.1. CSS的使用方式 238](#_Toc29450)

[2.1.1. 行内样式\内联样式 238](#_Toc10269)

[2.1.1.1. 常见样式属性 238](#_Toc30147)

[2.1.1.2. 内联样式的特点 239](#_Toc25958)

[2.1.2. 内部样式 239](#_Toc28458)

[2.1.3. 外部样式 240](#_Toc24708)

[2.2. CSS语法规范 241](#_Toc22135)

[2.3. CSS的特性 241](#_Toc29896)

[2.3.1. 继承性 241](#_Toc16071)

[2.3.2. 层叠性 241](#_Toc16808)

[3. 优先级的问题 242](#_Toc14473)

[4. 调整优先级 242](#_Toc22188)

[5. CSS的基础选择器(重点) 242](#_Toc32185)

[5.1. 选择器的作用 242](#_Toc8224)

[5.2. 通用选择器 通配符 242](#_Toc24181)

[5.3. 元素选择器 243](#_Toc11852)

[5.4. 类选择器 243](#_Toc27632)

[5.5. 特殊用法 244](#_Toc632)

[5.5.1. 多类选择器 244](#_Toc2993)

[5.5.2. 分类选择器 244](#_Toc6147)

[5.6. id选择器 245](#_Toc4037)

[5.7. 群组选择器 245](#_Toc21576)

[5.8. 后代选择器 246](#_Toc15765)

[5.9. 子代选择器 246](#_Toc28319)

[5.10. 伪类选择器 247](#_Toc8290)

[5.10.1. 链接伪类 247](#_Toc22882)

[5.10.2. 动态伪类 247](#_Toc11449)

[5.11. 选择器权值 248](#_Toc15391)

[5.11.1. 权值的计算（重点） 248](#_Toc9303)

[6. 尺寸和边框 249](#_Toc20579)

[6.1. 尺寸属性 249](#_Toc26195)

[6.1.1. 作用 249](#_Toc15803)

[6.1.2. 属性 249](#_Toc30095)

[6.1.3. 页面允许设置尺寸的元素 249](#_Toc14045)

[6.2. 尺寸单位(附加知识点) 250](#_Toc4032)

[6.3. 合法的颜色(附加知识点) 250](#_Toc26916)

[6.4. 溢出的处理 250](#_Toc16771)

[6.5. 边框属性 251](#_Toc6325)

[6.5.1. 边框的单边设置 251](#_Toc2144)

[6.5.2. 单属性设置 251](#_Toc22284)

[6.5.3. 边框倒角 252](#_Toc22957)

[6.5.4. 边框阴影 253](#_Toc28361)

[6.5.5. 轮廓 253](#_Toc23975)

[7. 框模型 254](#_Toc14057)

[7.1. 什么是框模型 254](#_Toc26688)

[7.1.1. 框模型的默认计算方式 254](#_Toc20694)

[7.2. 外边距 254](#_Toc17777)

[7.2.1. 简写方式 255](#_Toc12278)

[7.2.2. 自带外边距的元素 255](#_Toc23830)

[7.2.3. 外边距的特殊效果(重要) 255](#_Toc32410)

[7.3. 内边距 256](#_Toc4758)

[7.3.1. 语法 256](#_Toc21371)

[7.3.2. 简写方式 257](#_Toc9962)

[7.3.3. box-sizing 257](#_Toc1077)

[8. 背景 257](#_Toc22463)

[8.1. 背景图片 257](#_Toc1103)

[8.2. 背景颜色 257](#_Toc3271)

[8.3. 背景图片的平铺 257](#_Toc11384)

[8.4. 背景图的定位 258](#_Toc30619)

[8.5. 背景图尺寸 258](#_Toc15296)

[8.6. 背景图固定 258](#_Toc16018)

[8.7. 背景属性的简写方式 259](#_Toc18566)

[9. 渐变 259](#_Toc13024)

[9.1. 什么是渐变 259](#_Toc23778)

[9.2. 渐变的主要因素 259](#_Toc24215)

[9.3. 渐变色的分类 259](#_Toc27002)

[9.4. 线性渐变 259](#_Toc11068)

[9.5. 径向渐变 260](#_Toc18279)

[9.6. 重复渐变 260](#_Toc29827)

[9.6.1. 重复线性渐变 260](#_Toc13958)

[9.6.2. 重复径向渐变 261](#_Toc27113)

[9.7. 浏览器的兼容性 261](#_Toc9004)

[10. 文字格式化(重点) 261](#_Toc32271)

[10.1. 字体属性 261](#_Toc5776)

[10.2. 字体大小 261](#_Toc23046)

[10.3. 字体加粗 261](#_Toc3942)

[10.4. 字体样式 262](#_Toc15344)

[10.5. 小型大写字母 262](#_Toc11397)

[10.6. 字体属性简写模式 262](#_Toc9114)

[11. 文本格式化 262](#_Toc8592)

[11.1. 文本颜色 262](#_Toc1279)

[11.2. 文本的水平方向排列 263](#_Toc28144)

[11.3. 文本的垂直方向排列 263](#_Toc14294)

[11.4. 线条修饰 263](#_Toc20648)

[11.5. 首行缩进 263](#_Toc21655)

[11.6. 文本阴影 263](#_Toc1995)

[12. 表格的样式属性 264](#_Toc29338)

[12.1. 表格的常用样式属性 264](#_Toc8346)

[12.2. 合并边框 264](#_Toc6206)

[12.3. 边框边距 264](#_Toc1428)

[12.4. 标题的位置 265](#_Toc30011)

[12.5. table的显示规则 265](#_Toc8701)

[12.6. 自动布局与固定布局对比 265](#_Toc136)

[13. 定位 265](#_Toc28324)

[13.1. 什么是定位 265](#_Toc7812)

[13.2. 分类 266](#_Toc6653)

[13.3. 普通流定位 266](#_Toc22841)

[13.4. 浮动定位 266](#_Toc19901)

[13.4.1. 什么是浮动&浮动的特点 266](#_Toc17402)

[13.4.2. 语法 266](#_Toc26264)

[13.4.3. 浮动引发的特殊情况 266](#_Toc11434)

[13.4.4. 清除浮动 267](#_Toc22242)

[13.4.5. 子元素浮动（高度坍塌） 267](#_Toc4196)

[14. 显示方式 268](#_Toc29454)

[14.1. 什么是显示方式 268](#_Toc13934)

[14.2. 语法 268](#_Toc8043)

[14.2.1. 行内元素、块级元素、行内块元素对比 268](#_Toc11147)

[14.3. 显示效果 269](#_Toc20626)

[14.4. 透明度 270](#_Toc26932)

[14.5. 垂直对齐方式 270](#_Toc7256)

[14.6. 光标 270](#_Toc3903)

[15. 列表 271](#_Toc18246)

[15.1. 列表项标识 271](#_Toc3055)

[15.2. 列表项使用图像 271](#_Toc23270)

[15.3. 列表项的位置 271](#_Toc11674)

[15.4. 简写方式 271](#_Toc29542)

[16. 定位 272](#_Toc12853)

[16.1. 定位属性 272](#_Toc10080)

[16.2. 偏移属性 272](#_Toc19593)

[16.3. 定位方式 272](#_Toc17091)

[16.3.1. 相对定位relative 272](#_Toc5093)

[16.3.2. 绝对定位absolute 272](#_Toc5688)

[16.3.2.1. 绝对定位的参照物 273](#_Toc14983)

[16.3.3. 固定定位fixed 273](#_Toc26228)

[16.3.4. 堆叠顺序 273](#_Toc2061)

[第三章 CSS3 Core核心 274](#_Toc9046)

[1. 复杂选择器 274](#_Toc16556)

[1.1. 兄弟选择器 274](#_Toc3272)

[1.1.1. 相邻兄弟选择器 274](#_Toc9995)

[1.1.2. 通用兄弟选择器 274](#_Toc23472)

[1.2. 属性选择器 274](#_Toc30183)

[1.3. 伪类选择器 275](#_Toc30591)

[1.3.1. 目标伪类 275](#_Toc22514)

[1.3.2. 结构伪类 276](#_Toc17691)

[1.4. 伪类选择器——修饰标签内容 279](#_Toc22408)

[1.4.1. 匹配某个元素的首字符 279](#_Toc16953)

[1.4.2. 匹配某个元素的首行 279](#_Toc19759)

[1.4.3. 匹配用户选取的内容 279](#_Toc30816)

[1.4.4. 内容生成content 280](#_Toc19201)

[1.4.4.1. 匹配到某元素的内容区域之前 280](#_Toc13420)

[1.4.4.2. 匹配到某元素内容区域之后 280](#_Toc8595)

[1.4.4.3. 内容属性 280](#_Toc27885)

[1.4.4.4. 解决的问题 280](#_Toc32)

[2. 弹性布局(重点) 281](#_Toc7604)

[2.1. 弹性布局是什么 281](#_Toc23200)

[2.2. 弹性布局的相关概念 282](#_Toc20817)

[2.2.1. 弹性布局的容器 282](#_Toc22926)

[2.2.2. 弹性布局的项目 282](#_Toc24024)

[2.2.3. 主轴 282](#_Toc5159)

[2.2.4. 交叉轴 282](#_Toc6434)

[2.3. 语法 282](#_Toc32299)

[2.4. 容器的属性 282](#_Toc28690)

[2.4.1. 指定容器的主轴及其排列的方向 282](#_Toc138)

[2.4.2. 换行 283](#_Toc264)

[2.4.3. 属性简写 283](#_Toc31886)

[2.4.4. 定义项目在主轴对齐方式 283](#_Toc6889)

[2.4.5. 项目在交叉轴上的对齐方式 283](#_Toc13636)

[2.4.6. 项目在多根轴上的对齐方式 284](#_Toc19810)

[2.5. 项目属性 284](#_Toc14635)

[2.5.1. 定义项目的排序 284](#_Toc23636)

[2.5.2. 定义项目的放大比例 284](#_Toc30332)

[2.5.3. 定义项目的缩小比例 285](#_Toc14218)

[2.5.4. 定义某一个项目在交叉轴上的对齐方式 285](#_Toc13954)

[3. CSS hack（见拓展视频） 285](#_Toc24866)

[3.1. 相关概述 285](#_Toc24793)

[3.2. 浏览器运行模式分类 285](#_Toc16474)

[3.3. 触发相关模式 286](#_Toc11670)

[3.3.1. 触发混杂模式 286](#_Toc2969)

[3.3.2. 触发准标准模式 286](#_Toc26009)

[3.3.3. 触发标准模式 286](#_Toc2721)

[3.4. 什么是CSS Hack 286](#_Toc19806)

[3.5. Hack的原理 287](#_Toc19677)

[3.5.1. CSS类内部Hack 287](#_Toc10924)

[3.5.2. 选择器Hack 287](#_Toc8710)

[3.5.3. HTML头部引用Hack 287](#_Toc22677)

[4. 转换（重点） 287](#_Toc12871)

[4.1. 转化属性 287](#_Toc29570)

[4.1.1. 转换属性 287](#_Toc12679)

[4.1.2. 转换原点 287](#_Toc27720)

[4.2. 2D转换 288](#_Toc12231)

[4.2.1. 位移 288](#_Toc21359)

[4.2.2. 缩放 288](#_Toc440)

[4.2.3. 旋转 289](#_Toc16397)

[4.2.4. 倾斜 289](#_Toc10930)

[4.3. 3D转换 289](#_Toc13893)

[4.3.1. 透视视距 289](#_Toc15283)

[4.3.2. 3D旋转 290](#_Toc21219)

[4.3.3. 3D缩放 290](#_Toc30485)

[5. 过渡（重点） 290](#_Toc27821)

[5.1. 什么是过度 290](#_Toc18190)

[5.2. 语法 291](#_Toc8542)

[5.2.1. 指定过渡属性 291](#_Toc17736)

[5.2.2. 指定过渡的时长 291](#_Toc13163)

[5.2.3. 指定过渡时间曲线函数 291](#_Toc17418)

[5.2.4. 指定过渡的延迟时间 292](#_Toc1796)

[5.2.5. 过渡属性的编写位置 292](#_Toc3526)

[5.2.6. 过渡的简写 292](#_Toc16356)

[6. 动画（重点） 292](#_Toc30221)

[6.1. 什么是动画 292](#_Toc20214)

[6.2. 动画是通过“关键帧”，来控制动画的每一步 292](#_Toc30173)

[6.3. 关键帧的使用步骤 293](#_Toc12883)

[6.3.1. 声明动画 293](#_Toc14291)

[6.3.2. 调用动画 293](#_Toc13023)

[6.4. 动画的兼容性 294](#_Toc10885)

[7. CSS优化 295](#_Toc9952)

[7.1. 目的 295](#_Toc28377)

[7.2. CSS优化原则 295](#_Toc27395)

[7.3. 代码优化 295](#_Toc6326)

[第四章 Bootstrap 295](#_Toc3513)

[1. Bootstrap概述 295](#_Toc17997)

[1.1. 响应式布局 295](#_Toc24553)

[1.2. 响应式布局的要求（重要） 295](#_Toc27256)

[1.2.1. 响应式布局存在问题 296](#_Toc8275)

[1.3. 如何测试响应式网页 296](#_Toc11037)

[1.4. 如何编写响应式网页（重点） 296](#_Toc16089)

[1.4.1. meta 296](#_Toc31687)

[1.4.2. 数值 297](#_Toc24396)

[1.4.3. 布局 297](#_Toc7621)

[1.4.4. 使用条件 297](#_Toc16241)

[1.5. 使用media query技术,做媒体查询的方式 297](#_Toc14998)

[2. Bootstrap框架 298](#_Toc601)

[2.1. 如何使用boot 298](#_Toc12911)

[2.2. 全局css样式 298](#_Toc13117)

[2.2.1. BTN 298](#_Toc18451)

[2.2.1.1. 控制按钮颜色 298](#_Toc9344)

[2.2.1.2. 按钮边框 299](#_Toc23646)

[2.2.1.3. 3种尺寸 299](#_Toc13192)

[2.2.1.4. 其他颜色 299](#_Toc16173)

[2.2.1.5. 图片相关的css 300](#_Toc135)

[2.2.1.6. 文字相关的css 300](#_Toc2188)

[2.2.1.7. 列表相关的class 300](#_Toc24945)

[2.2.1.8. table的class 300](#_Toc17491)

[2.3. 辅助类名---大部分标签都可以使用的class 301](#_Toc20908)

[2.3.1. 基本类名 301](#_Toc19346)

[2.3.1.1. 边框 301](#_Toc27713)

[2.3.1.2. 浮动 301](#_Toc17023)

[2.3.1.3. 显示 301](#_Toc11414)

[2.3.1.4. 背景颜色 301](#_Toc9929)

[2.3.2. 内外边距 301](#_Toc20987)

[2.3.3. 各个方向 301](#_Toc25740)

[2.3.4. 高度 302](#_Toc10415)

[2.4. 栅格样式布局(重点) 302](#_Toc3637)

[2.4.1. web页面布局方法有三种 302](#_Toc16708)

[2.4.2. 使用栅格布局 302](#_Toc20709)

[2.4.3. 针对不同屏幕需要使用不同的比例 302](#_Toc27788)

[2.4.4. 注意不同屏幕下，列的适应性问题 303](#_Toc11047)

[2.4.5. 列偏移 303](#_Toc10004)

[2.4.6. 响应式 303](#_Toc1727)

[2.4.7. 栅格嵌套 303](#_Toc15701)

[2.5. 弹性布局 304](#_Toc26882)

[2.5.1. 使用dflex/d-inline-flex 类创建一个弹性容器 304](#_Toc18211)

[2.5.2. 响应式flex类 304](#_Toc23284)

[3. 公共CSS 305](#_Toc28547)

[3.1. 表单 305](#_Toc2057)

[3.1.1. form的样式 305](#_Toc16458)

[3.1.2. input的样式 305](#_Toc15815)

[3.1.3. 总结row和form-inline类似 306](#_Toc6973)

[4. 组件 307](#_Toc21994)

[4.1. 下拉菜单 307](#_Toc15781)

[4.1.1. 外层有容器dropdown 307](#_Toc19553)

[4.1.2. 容器中包含两部分 307](#_Toc26330)

[4.1.2.1. 下拉菜单的3级结构 307](#_Toc14868)

[4.2. 按钮组 307](#_Toc13413)

[4.3. 信息提示框 308](#_Toc5546)

[4.4. 导航 308](#_Toc15523)

[4.4.1. 水平导航 308](#_Toc29968)

[4.4.2. 选项卡导航 308](#_Toc23213)

[4.4.3. 胶囊导航 309](#_Toc30808)

[4.5. 导航栏 309](#_Toc580)

[4.6. 折叠 309](#_Toc4200)

[4.7. 折叠导航栏 310](#_Toc9269)

[4.8. 卡片 310](#_Toc10322)

[4.9. 媒体对象（重点） 310](#_Toc11995)

[4.10. 轮播 311](#_Toc11860)

[4.11. 模态框 311](#_Toc21408)

[5. 其他组件 312](#_Toc8643)

[5.1. 徽章 312](#_Toc16822)

[5.2. 巨幕 312](#_Toc13686)

[5.3. 分页 312](#_Toc18268)

[5.4. 面包屑 312](#_Toc21101)

[5.5. 进度条 312](#_Toc5726)

[第五章 SCSS动态样式语言 313](#_Toc25799)

[1. css的缺点 313](#_Toc3529)

[1.1. 常见的动态样式语言 313](#_Toc27969)

[2. scss是什么 313](#_Toc14805)

[3. scss的作用 313](#_Toc25554)

[4. 使用SCSS 313](#_Toc1181)

[4.1. 在服务器端使用 313](#_Toc7212)

[4.1.1. 测试是否安装成功 313](#_Toc9501)

[4.1.2. 执行转换命令 314](#_Toc8992)

[4.2. SCSS的基础语法 314](#_Toc1927)

[4.2.1. 变量 314](#_Toc1075)

[4.2.2. 嵌套规则 315](#_Toc249)

[4.2.3. 群组选择器嵌套 315](#_Toc8435)

[4.2.4. 属性嵌套 315](#_Toc29202)

[4.2.5. 导入SCSS文件 315](#_Toc20418)

[4.2.6. 混合器 316](#_Toc22264)

[4.2.6.1. 低版本浏览器，需要做兼容 316](#_Toc26167)

[4.2.6.2. 声明混合器(封装) 316](#_Toc30925)

[4.2.6.3. 使用混合器(调用) 316](#_Toc13121)

[4.2.6.4. 带参数的混合器 317](#_Toc21422)

[4.2.7. 继承 317](#_Toc20405)

[4.3. 运算 317](#_Toc7593)

[4.3.1. 数字运算 317](#_Toc8169)

[4.3.2. 颜色 318](#_Toc32325)

[4.4. 函数 318](#_Toc30602)

[4.4.1. 从css调用的函数 318](#_Toc17314)

[4.4.1.1. 颜色函数 318](#_Toc6358)

[4.4.1.2. 数字函数 319](#_Toc2357)

[4.4.1.3. 字符串函数 319](#_Toc21207)

[4.4.2. 自定义函数 319](#_Toc9633)

[4.5. 控制指令 319](#_Toc24157)

[第六章 项目阶段 320](#_Toc8271)

[1. 完成头部标签 320](#_Toc22459)

[2. 完成底部标签 321](#_Toc13473)

[3. 一楼的结构 321](#_Toc25503)

[4. 商品列表页 322](#_Toc22528)

[4.1. 思路 322](#_Toc16254)

[4.2. 步骤 322](#_Toc8465)

[5. 商品详情 323](#_Toc16990)

[第六部分 324](#_Toc8660)

# Mysql数据库

## 数据库服务器概述

### 如何保存数据

内存，速度快，空间有限；

自定义文件，速度慢；

Access/excel；

数据库存储。

### 数据库概述

Database:按照特定的数据结构存储数据的软件。

### 关系型数据库服务器

SERVER->Database->Table->Row->Column

## MySQL数据库

Oracle分支https://www.mysql.com/

Martin分支https://mariadb.org/

### 数据库发展

网状数据库->层次型数据库->关系型数据库->非关系型数据库

### 关系型数据库的结构

RDBMS中的数据在逻辑结构上通常呈现下述层级关系：

DATABASE->TABLE->ROW->COLUMN

### 常见的数据库

SQlite、MySQL、PostgreSQL、SQLServer、Oracle、DB2

LAMP组合：Linux+Apache+MySQL+PHP

（适合中小型站点，需要快速部署web应用）

### MySQL软件介绍

#### xampp软件

xampp 服务器套装；包含有多个开源服务器：Apache，MySQL，PHP

mysqld.exe 服务器启动文件

d->deamon 精灵，守护者，服务器软件

mysql.exe 连接数据库服务器

data 保存数据库服务器中的数据

#### 安装使用MySQL数据库系统

1. **服务器端：**下载并安装mysql服务器软件
2. **服务器端：**启动mysql服务器

保证端口 3306 被mysql占用

1. **客户端：**使用mysql.exe连接数据库
2. **语法**

mysql.exe [-h服务器地址] [-p服务器监听端口] [-u登录用户名] [-p登录密码] [数据库名]

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

简写形式： mysql –uroot

## mysql常用管理命令

### SQL语句两种执行方式

#### 交互模式：

在客户端输入一行，点击回车，服务器执行一行。适用于临时性的查看数据。

mysql -uroot 回车

#### 脚本模式：

客户端把要执行的命令写在一个文本文件中，一次性提交服务器执行。适用于批量的增删改查。

mysql -uroot < c:/xampp/htdocs…../2.sql 回车

### MySQL常用管理命令

前提：必须要连接成功mysql数据库

show databases; 查看有哪些数据库

quit; 退出mysql

use数据库名称; 使用数据库(进入数据库)

show tables; 显示当前数据库中所有的数据表

desc数据表名称; 描述表中都有哪些列(表头);

## MySQL中的SQL语句——重点

SQL：Structrued Query Language,结构化的查询语言，用于操作关系型数据服务器中的数据——增删改查。

SQL语言最早是ANSI提出的，后提交给ISO最终成为了数据库行业标准的语言。分为SQL-87、SQL-92、SQL-99,被绝大多数关系型数据库所支持。

### SQL语法规范

1. **每条SQL语句以英文的分号结尾，一条语句可以跨越多行(use 除外)，见到分号认为语句结束。**
2. **假如第N条语句语法错误，则此语句之后的所有语句都不会再执行。**
3. **SQL语句不区分大小写，习惯上关键字用大写，非关键字用小写。**
4. **SQL语句分为单行注册(# )和多行注释(/\* … \*/ )，注释的语句不被服务器执行。**

### 标准SQL语句的分类

1. **DDL：Data Define Language定义数据语言**

CREATE（新建）/ DROP（丢弃）/ ALTER(修改)/ TRUNCATE（删除）

当你不再需要该表时，用drop；

当你仍要保留该表，但要删除所有记录时，用truncate；

当你要删除部分记录时（always with a WHERE clause),用delete.

1. **DML:Data Manipulate Language操作数据语言**

INSERT（插入）/ DELETE（删除）/ UPDATE（修改）

1. **DQL:Data Query Language查询数据语言**

SELECT（选择）

1. **DCL:Data Control Lanuage控制数据语言**

GRANT(授权，管理数据库)/ REVOKE(收权)

### 日常开发中常用的SQL语句

1. **丢弃指定的数据库，如果存在的话**

DROP DATABASE IF EXISTS jd;

1. **创建新的数据库**

CREATE DATABASE jd;

1. **进入创建的数据库**

USE jd;

1. **创建数据表**

CREATE TABLE emp(

id INT,

name VARCHAR(16),

addr VARCHAR(16),

phone VARCHAR(11)

);

1. **插入数据**

INSERT INTO emp

VALUES(‘1’,’tom’,’bj’,’13111111111’);

1. **查询数据**

SELECT \* FROM emp;

1. **添加数据**

INSERT INTO user VALUES('1','dingding'…),('2','dangdang')…;

1. **删除数据**

DELETE FROM user WHERE uid='1';

1. **更改数据**

UPDATE user SET upwd='654321',sex='m'WHERE uid='2';

1. **查询数据**

SELECT\*FROM user;

练习：创建一个脚本文件 xz.sql；丢弃数据库xz，如果存在；创建数据库xz，进入该数据库，创建数据表user，包含列uid，uname，upwd，email，phone，sex，userName(真实姓名)，registerTime(注册时间 2018-08-20)； 插入5条记录，查询最终的数据

#设置客户端连接服务器端使用的编码为utf8

SET NAMES UTF8;

#丢弃数据库xz，如果存在

DROP DATABASE IF EXISTS xz;

#创建数据库xz，使用utf8编码

CREATE DATABASE xz CHARSET=UTF8;

#进入该数据库

USE xz;

#创建数据表 user

CREATE TABLE user(

uid INT,

uname VARCHAR(20),

upwd VARCHAR(16),

email VARCHAR(32),

phone VARCHAR(11),

sex VARCHAR(1),

userName VARCHAR(5),

registerTime VARCHAR(10)

);

#插入数据

INSERT INTO user VALUES('1','dingding','123456','dingding@qq.com','13112345678','m','张三','2018-9-4');

INSERT INTO user VALUES

('2','dangdang','123456','dangdang@jd.com','18812345678','w','KATE','2018-8-31'),

('3','baidu','123456','baidu@163.com','19912345678','m','JERRY','2018-7-1');

#更改数据

UPDATE user SET upwd='654321',email='baidu@sina.com' WHERE uid='3';

#删除数据

DELETE FROM user WHERE uid>'1';

#查询数据

SELECT \* FROM user;

### 数据库中的中文乱码

#### 计算机如何存储英文字符

**ASCII：**总共有128个，对所有的英文字符以及符号进行的编码。AB 6566

http://ascii.911cha.com/

**Latin-1：**总共有256个，兼容ASCII码，同时对欧洲的符号进行了编码。

MySQL默认使用Latin-1编码，而Latin-1不支持中文字符，直接插入中文会出现乱码

#### 计算机中如何存储中文存储

**GB 2312**对六千多个中文字符进行编码，主要是简体中文，兼容ASCII码。

**GBK**对两万多个中文字符进行编码，简繁体，兼容ASCII。

**Unicode**对世界上主要国家的语言进行了编码，兼容ASCII码；有UTF-8，UTF-16，UTF-32三种存储方案。

|  |  |
| --- | --- |
| **解决方案：**   1. **sql脚本文件存储使用UTF-8** 2. **客户端连接服务器使用UTF-8** 3. **服务器存储数据使用UTF-8** |  |

练习：创建一个脚本文件01\_dangdang.sql，先删除再创建数据库dangdang,使用utf8编码；进入数据库，创建数据表book，包含bid，title，price，pubDate(出版时间)，isJia(是/否)是否加价购；插入4条数据；更改第2条数据，删除第3条数据。

#设置客户端连接服务器使用utf8

SET NAMES UTF8;

#丢弃数据dangdang，如果存在

DROP DATABASE IF EXISTS dangdang;

#创建数据库dangdang，使用utf8编码存储数据

CREATE DATABASE dangdang CHARSET=UTF8;

#进入数据库

USE dangdang;

#创建数据表 book

CREATE TABLE user(

bid INT,

title VARCHAR(40),

price DECIMAL(5,2), #99999.99

pubDate VARCHAR(10),

isJia BOOL(1)

);

#插入数据

INTSERT INTO book VALUES

('1','三国演义','105','2018-6-6','是'),

('2','水浒传','88','2001-1-1','否'),

('3','西游记','22','1998-3-3','是')，

('4','红楼梦','33','2010-1-2','是');

#更改数据

UPDATE book SET title='孟子1',pubDate='2001-2-2' WHERE bid='2';

#删除数据

DELETE FROM book WHERE bid='3';

### 列类型

创建数据表的时候，指定的列所保存的数据类型

create table book(bid列类型);

#### 数值类型——数据可以不加引号

TINYINT 微整型，占用1个字节，存储范围-128~127

SMALLINT 小整型，占用2个字节，存储范围-32768~32767

INT 整型，占用4个字节，存储范围-2147483648~2147483647

BIGINT 大整型，占用8个字节，存储范围很大

FLOAT 单精度浮点型，占4个字节，取出几位作为指数使用,实际存储的范围要INT大的多。

DOUBLE 双精度浮点型，占8个字节，实际存储范围要比BIGINT大的多1.79E308

DECIMAL(M,D) 定点小数，不会产生误差,M总共有效位数(不包含小数点)，D表示小数点后的有效位数。

BOOL 布尔型只有两个值TRUE/1和FALSE/0，最终的存储方式使用数字1和0

#### 日期时间类型——必须加引号

DATE 日期类型例如：2018-09-04

TIME 时间类型例如：14:50:30

DATETIME 日期时间类型例如：2018-09-04 14:50:30

#### 字符串类型——必须加引号

VARCHAR(M) 变长字符串，查询速度相对比较慢，不存在空间浪费。M最多存储65535

CHAR(M) 定长字符串，例如手机号，身份证号…，查询速度比较快，可能存在空间浪费。M最多存储255

TEXT(M) 大型变长字符串，M最多存储2G

|  |  |  |
| --- | --- | --- |
|  | VARCHAR(5) | CHAR(5) |
| a | a\0 | a\0\0\0\0 |
| ab | ab\0 | ab\0\0\0 |
| abc | abc\0 | abc\0\0 |
| 一二 | 一二\0 | 一二\0\0\0 |

存储时间，存储的是距离1970年1月1日的毫秒数。

1000毫秒=1秒

练习：创建一个脚本文件02\_xz.sql，设置编码为utf8，丢弃数据库，再创建数据库xz，进入该数据库，创建保存笔记本信息的表laptop，包含列lid，title，price，stockCount(库存量)，shelfTime(上架时间)，isIndex(是否显示在首页)；插入4条记录

#设置客户端连接服务器端使用的编码为utf8

SET NAMES UTF8;

#丢弃数据库xz，如果存在

DROP DATABASE IF EXISTS xz;

#创建数据库xz，使用utf8编码

CREATE DATABASE xz CHARSET=UTF8;

#进入该数据库

USE xz;

#创建数据表 laptop

CREATE TABLE laptop(

lid INT,

title VARCHAR(20),

price DECIMAL(8,2),

stockCount SMALLINT,

sheltTime DATE,

isIndex BOOL

);

#插入数据

INSERT INTO laptop VALUES

(1,'a',4999,100,'2018-1-5',1),

(2,'b',5999,400,'2017-6-6',0),

(3,'c',5299,500,'2018-5-15',0),

(4,'d',6199,600,'2016-7-7',1);

练习：使用恰当的列类型

创建脚本文件03\_tedu.sql，设置存储的编码为utf8；丢弃数据库tedu，如果存在；创建数据库tedu，进入该数据库；

创建保存部门信息的表dept，包含did，dname，empCount员工数量；

10 研发部 3

20 市场部 2

30 运营部 2

创建保存员工信息的表emp，包含eid，ename，sex，phone，birthday，salary工资，deptId部门的编号

1 tom 1 18711111111 1989-1-1 8000 20

#设置连接服务器使用的编码UTF-8

SET NAMES UTF8

#丢弃数据库tedu，如果存在

DROP DATABASE IF EXISTS tedu;

#创建数据库tedu，设置UTF-8

CREATE DATEBASE tedu CHARSET=UTF8;

#进入数据库

USE tedu;

#创建数据表dept

CREATE TABLE dept(

did SMALLINT,

dname VARCHAR(10),

empCount SMALLINT

);

#插入数据

INSET INTO dept VALUES

(10,'研发部',3),

(20,'市场部',2),

(30,'运营部',2);

#创建员工表

CREATE TABLE emp(

eid INT,

dname VARCHAR(6),

sex BOOL,

phone CHAR(11),

birthday DATE,

salary DECIMAL(7,2),#99999.99

deptId SMALLINT

);

#插入数据

INSET INTO emp VALUES

(1,'Tom',1,'18711111111','1989-1-1',80000,20),

(2,'Jay',1,'13112345678','1988-2-2',70000,10),

(3,'Gay',0,'15122222222','1991-3-3',60000,30);

### 列约束

MySQL可以对插入表中的数据进行特定的验证，只有满足条件的数据才允许插入成功！

例如：一个人的年龄范围是在0~120之间，商品的价格只能是正数…

#### 主键约束——PRIMARY KEY

声明了主键约束的列，不允许插入相同的数据。不应插入NULL；一个表中只能使用一个主键，使用主键约束会加快查找的速度。

#### 唯一约束——UNIQUE

声明了唯一约束的列上不能插入重复的值，允许插入一个或者多个NULL；一个表中可以使用多个唯一约束。

#### 扩展知识：NULL

NULL：空的，空缺的。

通常用于某个列，还不能确定具体要保存的数据。

例如：年终奖，部门经理暂时还不能确定。

**注意：NULL和任何值都不相等，甚至自身。**

练习：创建脚本文件xuezi.sql，设置存储的编码；删除数据库xuezi，如果存在；创建数据库xuezi，进入该数据库。

创建笔记本型号表 laptop\_family ( fid，fname，laptopCount每个型号的笔记本数量 )

10 戴尔燃700 3

20 小米Air 2

30 联想小新系列 2

创建笔记本信息表laptop(lid，title，price，shelfTime，stockCount库存数量，isOnsale 是否在售，fid隶属的系列)

插入7条记录；修改第3条记录，删除第5条记录。

#设置客户端连接使用的编码

SET NAMES UTF8;

#丢弃数据库xuezi，如果存在

DROP DATABASE IF EXISTS xuezi;

#创建数据库，存储的编码

CREATE DATABASE xuezi CHARSET=UTF8;

#进入数据库xuezi

USE xuezi;

#创建数据表laptop

CREATE TABLE laptop\_family(

fid SMALLINT PRIMARY KEY,

fname VARCHAR(16)UNIQUE,

laptopCount SMALLINT NOT NULL

);

#插入数据

INSERT INTO laptop\_family VALUES

(10,'戴尔燃700',3),

(20,'小米AIR',2),

(30,'联想小新系列',2),

(40,NULL,NULL);

#创建笔记本信息表

CREATE TABLE laptop(

lid INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(128)UNIQUE,

price DECIMAL(8,2)DEFAULT 3000,

shelfTime DATE,

ctockCount SMALLINT DEFAULT 9999,

isOnsale BOOL,

fid SMALLINT,

FOREIGN KEY(fid)REFERENCES laptop\_family(fid)

);

#插入数据

INSERT INTO laptop VALUES

(NULL,'联想1',DEFAULT,'2018-5-1',DEFAULT,TRUE,30),

(NULL,'小米1',5100,'2018-7-1',1000,FALSE,20),

(NULL,'戴尔1',6199,'2018-6-1',5001,TRUE,10),

(NULL,'联想2',4699,'2018-5-1',8000,TRUE,30),

(NULL,'小米2',5100,'2018-7-1',1000,FALSE,20),

(6,'戴尔2',6199,'2018-6-1',500,1,10),

(NULL,'戴尔3',6199,'2018-6-1',500,1,10);

INSERT INTO laptop(lid,title) VALUES(NULL,'戴尔4000');

#删除的笔记本是小米，小米系列也应该减1

DELETE FROM laptop WHERE lid=5;

#在原来的基础上减1

UPDATE laptop\_family SET laptopCount=laptopCount-1 WHERE fid=20;

#删除一款戴尔笔记本，戴尔系列也应该减1

DELETE FROM laptop WHERE lid=7;

#在原来的基础上减1

UPDATE laptop\_family SET laptopCount=laptopCount-1 WHERE fid=10;

#### 非空约束——NOT NULL

禁止插入NULL，如果插入的值是NULL，不会报错；字符串保存的值变成空的，数值型保存的值变成了0。

#### 默认值约束——DEFAULT

可以使用DEFAULT关键字为列声明默认值。

有两种方法可以使用默认值

INSERT INTO user VALUES(1,’tom’,DEFAULT);

INSERT INTO user(uid,uname) VALUES(1,’tom’);

#### 检查约束——CHECK

检查约束对要插入的数据进行检验

CREATE TABLE student(

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

);

MySQL**不支持**检查约束，会**降低**数据的**插入速度**。

#### 外键约束

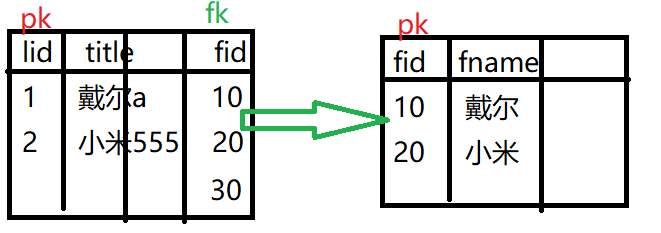
声明的外键约束的列，取值必须在另一个表的主键列上出现过。

CREATE TABLE laptop(

….

FOREIGN KEY(列名称) REFERENCES laptop\_family(列名称)

);



#### 扩展知识：MySQL中的自增列

AUTO\_INCREMENT，只能用于列类型为整型的主键列上。

作用：在原来最大的值基础上加1

允许手动赋值，如果设置的是NULL，就会自增。

练习：创建03\_tedu.sql， 设置存储的编码为utf8，丢弃数据库tedu，如果存在，创建数据库，进入该数据库；

创建部门表dept，包含列(did，dname)

10 研发部 development

20 测试部

30 运营部

40 市场部

创建员工表emp，包含(eid，ename，sex，phone，birthday，salary，deptId)；

插入15条记录，有一个员工部门为NULL，其中市场部没有员工

#设置客户端连接使用的编码UTF8

SET NAMES UTF8;

#丢弃数据库tedu，如果存在

DROP DATABASE IF EXISTS tedu;

#创建数据库，存储的编码UTF8

CREATE DATABASE tedu CHARSET=UTF8;

#进入数据库tedu

USE tedu;

#创建数据表部门表dept

CREATE TABLE dept(

did SMALLINT PRIMARY KEY AUTO\_INCREMENT,

dname VARCHAR(16) UNIQUE

);

#插入数据

INSERT INTO dept VALUES

(10,'研发部'),

(20,'测试部'),

(30,'运营部'),

(40,'市场部');

#创建员工表emp

CREATE TABLE emp(

eid INT PRIMARY KEY AUTO\_INCREMENT,

ename VARCHAR(12),

sex BOOL,

phone CHAR(11),

birthday DATE,

salary DECIMAL(7,2),

deptId SMALLINT,

FOREIGN KEY(deptId) REFERENCES dept(did)

);

#插入数据

INSERT INTO emp VALUES

(NULL,'A',TRUE,'13112345678','1991-1-1',5000,10),

(NULL,'B',FALSE,'13212345678','1992-1-1',6000,20),

(NULL,'C',TRUE,'13132345678','1993-1-1',7000,30),

(NULL,'D',FALSE,'13412345678','1994-1-1',50000,10),

(NULL,'E',TRUE,'13152345678','1991-5-1',7000,20),

(NULL,'F',FALSE,'13612345678','1996-1-1',5000,30),

(NULL,'G',TRUE,'13172345678','1991-7-1',6000,10),

(NULL,'H',FALSE,'13812345678','1991-8-1',5000,20),

(NULL,'I',TRUE,'13192345678','1999-1-1',8000,30),

(NULL,'G',TRUE,'13102345678','1991-9-1',9000,30),

(NULL,'H',FALSE,'13132345678','1991-1-1',5000,20),

(NULL,'I',1,'13114345678','1991-1-8',9000,20),

(NULL,'J',TRUE,'13115345678','1995-5-1',5300,10),

(NULL,'K',0,'13117345678','1996-6-1',5200,20),

(NULL,'L',TRUE,'13116345678','1993-2-4',5000,NULL);

### 简单查询

#### 查询特定的列

例：查询所有员工的编号和姓名。

SELECT eid,ename FROM emp;

练习：查询员工的姓名，性别，工资，生日。

SELECT ename,sex,salary,birthday FROM emp;

#### 查询所有的列

SELECT \* FROM emp;

#### 给列取别名

例：查询员工的姓名，工资，列名称使用汉字。

SELECT ename AS 姓名,salary 工资 FROM emp;

SELECT ename 姓名,salary 工资 FROM emp;

**AS可省略**，但要留空格,注意逗号需为英文符号

练习：查询所有员工的编号，姓名，性别，生日，全部使用中文别名。

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

#### 只显示不同的记录(合并相同的记录)

例：查询哪些部门下有员工(员工都在哪些部门)?

SELECT DISTINCT deptId FROM emp;

练习：查询公司都有哪些性别的员工？

SELECT DISTINCT sex FROM emp;

#### 在查询时执行计算

例：计算3\*4/5+2

SELECT 3\*4/5+2 AS 结果;

例：查询出所有员工的姓名和年薪是多少

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

练习：假设每个员工工资增加500，年终奖是5000，查询所有员工的姓名和年薪是多少

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

#### 查询结果集的排序

默认按照**升序排列**(asc)

排序可以按照数值，日期，字符串排列。

例：查询出所有部门的信息，结果按照部门编号升序排序

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

例：查询出所有部门的信息，结果按照部门编号降序排序

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

练习：查询所有的员工，结果按照工资由低到高排序；

SELECT \* FROM emp ORDER BY salary ASC;

练习：查询所有的员工，结果按照年龄由高到低排序；

SELECT \* FROM emp ORDER BY birthday ASC;

练习：查询所有的员工，结果按照姓名升序排序

SELECT \* FROM emp ORDER BY ename ASC;

练习：查询所有的员工，结果按照工资升序排序，如果工资相同，按照姓名升序排序。

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

练习：查询所有的员工，结果按照生日升序排序，如果生日相同，按照工资降序排序。

SELECT \* FROM emp ORDER BY birthday ASC,salary DESC;

#### 条件查询

例：查询出编号为5的员工所有信息

SELECT \* FROM emp WHERE eid=5;

练习：查询出姓名为KING的员工的编号、工资、生日

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

练习：查询出20号部门下所有员工的信息

SELECT \* FROM emp WHERE deptId=20;

练习：查询出所有的男员工信息

SELECT \* FROM emp WHERE sex=1;

例：查询出工资大于等于5000的员工所有信息

SELECT \* FROM emp WHERE salary>=5000;

|  |
| --- |
| SQL语句中比较运算符:  > >= = != < <= |

练习：查询出1991-1-1之后出生的员工所有信息

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

练习：查询出不在10号部门的所有员工信息

SELECT \* FROM emp WHERE deptId!=10;

练习：查询出暂时没有部门的员工信息(部门编号为NULL)

SELECT \* FROM emp WHERE deptId IS NULL;

练习：查询出有明确部门的员工信息

SELECT \* FROM emp WHERE deptId IS NOT NULL;

练习：查询出工资大于6000的女员工所有信息

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

练习：查询出工资在5000~5999之间的员工信息

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

SELECT \* FROM emp WHERE salary BETWEEN 5000 AND 5999;

练习：查询出在1991年出生的员工信息

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

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

练习：查询出工资小于4000和大于8000的员工信息

S T \* FROM emp WHERE salary NOT BETWEEN 4000 AND 8000;

练习：查询出在1990年之前和1993之后出生的员工信息

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

SELECT \* FROM emp WHERE birthday NOT BETWEEN '1990-1-1' AND '1993-12-31';

|  |
| --- |
| BETWEEN ... AND ... 在两者之间  NOT BETWEEN...AND...不在两者之间 |

练习：查询出在10和30号部门的员工信息

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

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

练习：查询出在10、30、50、80号部门的员工信息

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

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

练习：查询出不在10和30号部门的员工信息

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

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

|  |
| --- |
| IN(10,30) 满足其中一个条件  NOT IN(10,30) 不满足任意一个条件 |

练习：删除工资在5000~10000之间的员工信息

DELETE FROM emp WHERE salary BETWEEN 5000 AND 10000;

#### 模糊条件查询

例：查询出姓名中包含字母E的员工所有信息

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

练习：查询出姓名以E结尾的员工所有信息

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

练习：查询出姓名中倒数第二个字符是E的员工所有信息

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

|  |
| --- |
| % 任意多个字符 >=0  \_ 任意一个字符 =1  **注意：%和\_必须要结合着LIKE关键字使用** |

#### 分页查询

分页显示：查询结果太多的记录行，一次显示不完，可以分页显示。不同的数据库有不同的查询方式。MySQL的分页查询最简单。

语法：

SELECT\*FROM emp LIMIT start,count;

start:是一个数字，表示从结果集中的哪一行开始读取

count:是一个数字，表示读取的最多行数

start=(页码-1)\*count

练习：假设每一页对多显示4条记录

第1页 SELECT\*FROM emp LIMIT 0,4;

第2页 SELECT\*FROM emp LIMIT 4,4;

第3页 SELECT\*FROM emp LIMIT 8,4;

第4页 SELECT\*FROM emp LIMIT 12,4;

第5页 SELECT\*FROM emp LIMIT 16,4;

练习：假设每一页对多显示5条记录

第1页 SELECT\*FROM emp LIMIT 0,5;

第2页 SELECT\*FROM emp LIMIT 5,5;

第3页 SELECT\*FROM emp LIMIT 10,5;

练习：查询工资最低的5个员工。

SELECT\*FROM emp ORDER BY salary ASC LIMIT 0,5;

练习：查询男员工中工资最高的5个人

SELECT\*FROM emp WHERE sex=1 ORDER BY salary DESC LIMIT 0,5;

### 复杂查询

#### 聚合查询/分组查询

聚合函数：COUNT(...)/SUM(...)/AVG(...)/MAX(...)/MIN(...)

函数：功能体，接收若干个数据，返回特定的计算结果。

例：查询出所有员工的数量

SELECT COUNT(eid) FROM emp;

SELECT COUNT(ename) FROM emp;

SELECT COUNT(\*) FROM emp;

SELECT COUNT(deptId) FROM emp; #无法识别NULL

练习：查询男员工的数量

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

例：查询出所有的员工的工资总和

SELECT SUM(salary) FROM emp;

练习：产出平均工资是多少

SELECT SUM(sarlary)/SUM(\*) FROM emp;

SELECT AVG(salary) FROM emp;

例：查询出工资最高的员工

SELECT MAX(salary) FROM emp;

SELECT MIN(salary) FROM emp;

练习：查询年龄最大的员工

SELECT MIN(birthday) FROM emp;

例：查询每个部门的员工数量是多少

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

分组查询：只能查询分组的条件和聚合函数。

练习：查询出男女员工的平均工资

SELECT sex,ACG(salary) FROM emp GROUP BY sex;

练习：查询出男女员工的平均工资、最多、最少

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

|  |
| --- |
| year()获取日期中的年份；  month()获取日期中的月份；  day()获取日期中的日子； |

例：查询1991年出生的员工

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

例：查询3月份出生的员工

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

例：查询5号份出生的员工

SELECT\*FROM emp WHERE day(birthday)=5;

#### 子查询

本质上就是一个SQL语句的查询结果是另一SQL语句的查询条件

例：查询出研发部所有员工的信息

步骤1：查询出研发部的部门编号——10

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

步骤2：查询出员工表中部门编号为10的员工

SELECT \* FROM emp WHERE deptId=10;

综合：

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

练习：查出工资比tom高的所有员工的信息

步骤1：查询出tom的工资——6000

SELECT salary FROM emp WHERE ename='tom';

步骤2：查询出工资大于6000的员工

SELECT \* FROM emp WHERE salary>6000;

综合：

SELECT\*FROM emp WHERE salary>( SELECT salary FROM emp WHERE ename='tom');

练习：查询出与tom同一年出生的员工信息

法一、

步骤1：查询出tom的生日——1990-5-5

SELECT birthday FROM emp WHERE ename='tom';

步骤2：查询出1990年出生的员工信息

SELECT\*FROM emp WHERE year(birthday)=year('1990-5-5');

综合：

SELECT\*FROM emp WHERE year(birthday)=year((SELECT birthday FROM emp WHERE ename='tom'));

法二、

步骤1：查询出tom的生日——1990

SELECT year(birthday) FROM emp WHERE ename='tom';

步骤2：查询出1990年出生的员工信息

SELECT\*FROM emp WHERE year(birthday)=1990;

综合：

SELECT\*FROM emp WHERE year(birthday)=(SELECT year(birthday) FROM emp WHERE ename='tom');

#### 多表查询

例：查询出所有员工的姓名及其部门名称

SELECT ename,dname FROM emp,dept;

错误：笛卡尔积现象。

多表查询如何避免“笛卡尔积”，为查询添加条件。

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

|  |
| --- |
| 上述的语法是SQL-92中的多表查询语法，无法查询出没有部门的员工，也不能查询出没有员工的部门。  SQL-99中提出了新的多表查询方法。 |

1. **内连接 INNER JOIN...ON...**——和SQL-92结果一致

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

JOIN 后跟被连接的表；ON 后接条件

1. **左外连接 LEFT OUTER JOIN...ON...**

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

查询结果是左侧所有记录都显示；OUTER可以省略。

1. **右外链接 RIGHT OUTER JOIN...ON...**

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

查询结果是右侧所有记录都显示；OUTER可以省略。

1. **全连接 FULL JOIN**

显示左侧和右侧所有的记录——MySQL不支持

MySQL的全连接：UNION; UNION ALL

|  |
| --- |
| 两个查询结果：  集中合并记录 UNION  不合并相同记录 UNION ALL |

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

**UNION**

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

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

**UNION ALL**

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

### 综合的语法

SELECT > FROM > WHERE > ORER BY > LIMIT

注意：LIMIT后的两个值不能加引号。

# JavaScript

## 学一门新语言的基本步骤

1. **了解背景知识：历史，现状，特点，应用场景。**
2. **搭建开发环境，编码 hello world**
3. **变量和常量**
4. **数据类型**
5. **运算符**
6. **逻辑结构**
7. **通用小程序**
8. **函数和对象**
9. **第三方库，框**

## JavaScript概述

### JS的执行环境

浏览器内置的JS解释器——客户端

nodeJS中的JS解释器——服务器

### 应用领域

网页中的特效、移动APP、web服务器

### 发展历史

1995年，JavaScript出现在网景浏览器中。

但JavaScript与Java没有任何关系

2009年，JavaScript遵循commonJS规范，开始出现在服务器端。

（JS可在浏览器端和客户端同时运行）

### 特点

1. **弱类型语言**
2. **解释型语言，无需编译，编译一行，执行一行**
3. **基于对象**
4. **跨平台**

### 搭建js的开发环境

nodejs 下载地址：https://nodejs.org/en/

选择左侧（8.11.4 LTS）

### js的运行环境

#### 浏览器

创建2.html和2.js文件

在2.js中输入 console.log('hello world');

把2.js引入到2.html中；

<script scr="2.js"></script>

#### nodejs下（服务器）

查看nodejs版本号，输入node -v 回车

运行 node C:/xampp/..../2.js 回车

### JS书写规范

1. **区分大小写**
2. **每行结束的分号可加可不加；提倡都加。**
3. **单行注释（//....）和多行注释（/\*...\*/）**

（达成后，语句会变成绿色）

1. **字符需加引号**

## 变量

变量是用于存储数据的容器。

x=1 y=2

### **声明变量使用关键字var-->variable**

var x=1;

var y=2;

### 变量的命名规则

可以以字母、下划线、美元符号（$）开头；后续除了以上三个，也可以使用数字。

多个单词之间通常使用驼峰命名法和下滑线命名法。

userName（驼峰命名法）；

user\_name（下划线命名法）；

不能使用关键字作为变量名称。

标准关键字：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| break | case | catch | continue | default | delete |
| do | else | false | finally | for | function |
| if | in | instanceof | new | null | return |
| switch | this | throw | true | try | typeof |
| var | void | while | with | undefined | ... |

预保留关键字:

class、int、float等

### 变量的声明

声明后（var）就在计算机里创建一个空间，变量名称（empName）是为了查找。

不加var，同变量名，后面赋的值会覆盖之前赋的值

加var，同变量名，会建立一个相同的空间，覆盖原空间；（结果相同，但会影响效率）

变量的声明之后，可以重现为变量赋值；可以赋任何类型的值。

这个特点也是弱类型语言的特点。

|  |
| --- |
| \*弱类型语言可以多次赋值，并可以赋不同类型的值；而强类型语言则不能多次改变赋值类型。 |

可以一次性声明多个变量

例：var price=5000,isOnsale=1;

注意：

1. **使用一个未声明的变量，会报错；**
2. **声明了变量，未赋值，不报错，结果是undefined（无定义、未赋值）。**

例：var shelftime;

console.log(shelftime); //undefined

练习：声明两个变量，一个保存长方形的宽度和高度，计算面积多少。

var width=5, height=2,area;

area=width\*height;

console.log(width); //5

console.log(height); //2

console.log(area); //10

## 常量 constant（常数、常量，缩写const）

命名规则

同变量，可以以字母、下划线、美元符号（$）开头；后续除了以上三个，也可以使用数字。

常量一旦声明不能重新赋值。

使用const关键字声明，建议用大写字母。

例：cost PI=3.14;

## 数据类型

分为原始类型和引用类型。

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

### 数值型

#### 整数

整数在内存中占4个字节，小数占8个字节。

8进制（以0开头）、10进制、16进制（以0x开头，不区分大小写）

#### 普通小数和指数小数



|  |
| --- |
| **typeof(数据) 检测数据类型** |

### 字符串型

数据被引号所包含就是字符串类型；大部分不区分单双引号（个别有区别）。

查看一个字符的Unicode编码值

'a'.charCodeAt()

注：括号内不需填写任何数据

练习：查看自己名字的Unicode的编码

console.log('宋'.charCodeAt()); //23435

console.log('潇'.charCodeAt()); //28487

console.log('杉'.charCodeAt()); //26441

### 布尔型

在程序中表示真或假的结果。

true/false（小写）

常用于一些为是否的结果

isLogin=true； isOnsale=false；

两个数字比较的结果也是布尔型 1<2

### 未定义型

声明了变量未赋值，结果就是undefined；

### null

用于释放内存空间，只有一个值null。

例：变量名称=null;（无法看见演示效果，属于底层运算）

## 数据类型转换

### 隐式转换

数字+字符串：数字会转成字符串

1+'a' //'1a'

数字+布尔型：布尔型会转成数字 true->1;false->0

1+true //2

1+false //1

字符串+布尔型：布尔型转换成字符串

'hello'+true //'hellotrue'

总结：

JS中字符串和任何数据类型使用“+”，会执行拼接；

除字符串外使用“+”会执行算术运算；

使用“\*”，“/”，“-”，会算术执行运算；

会尝试将符号的两端转换成数字执行运算，如果无法转换，回NaN（Not a Number，不是一个数字）；

NaN的类型是number。

练习：

var num1=3,num2=true,nume3='hello';

num1+num2+num3=? //'4hello'

num2+num3+num1=? //'truehello3'

num3+num1+num2=? //'hello3true'

### 强制转换

#### 将任意的类型转为整数

parseInt()

例子：parseInt('1') //1

练习：转换这几个值 '1.5' '3a' 'a8'

console.log(parseInt('1.5')); //1

console.log(parseInt('3a')); //3

console.log(parseInt('a8')); //NaN

console.log(parseInt(true)); //NaN

console.log(parseInt(undefined)); //NaN

#### 将任意的类型转为浮点型

parseFloat()

例：

console.log(parseFloat('1.5a')); //1.5

console.log(parseFloat('a1.5')); //NaN

console.log(parseFloat('3.14c15926')); //3.14

#### 将任意的类型转为数值型

Number()

练习：转换 '1' '1.5a' 'a8' '3a' true undefined

console.log(Number('1')); //1

console.log(Number('1.5a')); //NaN

console.log(Number('a8')); //NaN

console.log(Number('3a')); //NaN

console.log(Number(true)); //1

console.log(Number(undefined)); //NaN

如果要转换的值中出现非数字就会返回NaN，特例true->1、false->0。

#### 将数值转为字符串

toString()

例：

var num=12;

num.toString(); //'12'

num.toString(16); //'c'

练习：'达'得到Unicode码，再把Unicode值转成16进制

var c='达';

c=c.charCodeAt();

c=c.toString(16);

console.log(c); //8fbe

综合：

console.log('达'.charCodeAt().toString(16)); //8fbe

#### 布尔型转化为字符串

var d=true;

console.log(d.toString());

d=d.toString();

console.log(typeof(d)); //string

## 运算符和表达式

由运算符连接操作的数据，所组成的形式就称为表达式。

### 算数运算符

+ - \* / % ++ --

% 取余；

++ 自增，在原来的基础上加1；

-- 自减，在原来的基础上减1。

例：console.log(num++); //先打印num的值，然后自增

console.log(++num); //先自增，然后打印num的值

练习：

var a=3;

console.log(a++ + ++a); //8

var b=5;

console.log(--b+b++ + ++b); //14

### 比较运算符

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

==（等于） 仅比较值是否相同

===（全等于） 不仅比较值，还会比较类型是否相同

!==（不全等于） 数值不相同或类型不相同

#### 比较运算符返回布尔型的值

3>'10' //false

#### 字符串型被转成了数值型

'3'>'10' //ture

#### 字符串比较

比较的是Unicode码值，而一次只表示一个字符。

‘3’->51 ‘1’->49

即‘3’的Unicode的码值比‘1’的大，故‘3’>‘10’为真。

'雷锋'>'雷峰塔'为真

而第一个字符相同时，比较第二个字符

锋->38155 峰->23792

例：

3>'10a' //false

3<'10a' //false

3=='10a' //false

'10a'->NaN

#### NaN的比较

NaN和任何数比较（>,>=,<,<=,==）都返回false，而（!=,!==）返回true。

NaN和NaN相比返回为false。

### 逻辑运算符

或 ||； 且 && ；非 ！

|| 关联的两个条件只需满足其一，结果是true，否则false。

&& 关联的两个条件都满足，结果是true，否则false。

！ 反向，逆向；取反。

例：判断一个人是否优秀 90~100

var score=81;

console.log(score>=90 && score<=100); //false

例：年龄大于65，或者小于12都可以打折

var age=40

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

例：取反

console.log(!(2>1)); //false

练习：实现登录，声明变量，变量保存用户名和密码。只有用户名是root，密码是123456，才返回true

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

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

### 逻辑短路

&& 当第1个条件为true的时候，就需要再执行第2个条件。

|| 当第1个条件为false的时候，就需要再执行第2个条件。

（当第一个条件已经可以判断逻辑判断是否成立时，不会再执行第二个条件了。）

例：var num=3;

num>1 && console.log('大于1'); //大于1

第一个条件为真时，第二个条件执行

例：var num2=2;

num2>3 || console.log('小于3'); //小于3

第一个条件为假时，第二个条件执行

练习：下边代码是否报错

var a=3;

a>5 && console.log(num); //不会报错

a<1 || console.log(num); //会报错

因为a>5是false，&&后的代码没有被执行

因为a<1是false，会执行，或，后的代码

练习：判断一个年份是否为闰年。

闰年，能被4整除，并且不能被100整除的是闰年；或者能被400整除的是闰年。

//判断一个年份是否为闰年。

//能被4整除 为真，并且不能被100整除的是闰年

//year%4==0--true; year%100==0--false;

//或能被400整除的是闰年

//year%400==0--true;

var year="2008",x=4,y=100,z=400,a,b,c,d,f;

var a=year%x==0; //能被4整除为真

var b=year%y==0; //能被100整除为假

var c=year%z==0; //能被400整除为真

var d=(a && !(b)); //能被4且不能被100整除为真

var f=(d || c); //能被400整除为真

f && console.log("闰年");

var year="200",x=4,y=100,z=x\*y,a;

var a=(((year%x==0) && !(year%y==0)) || (year%z==0));

a && console.log("闰年"); //a->true

a || console.log("不是闰年"); //a->false

简写：

a=year%x==0 && year %y!=0 ||year%z==0;

练习：用“逻辑短路”判断，一个人是否大于18，若大于18，则打印“成年人。”

var age="20";

age<18 || console.log("成年人"); //成年人

### 位运算符

将数值转成二进制，然后执行运算

2进制：1、10、11、110、111、1000、...

例：11110（B）转为十进制是多少

11110（B）=1\*2^4+1\*2^3+1\*2^2+1\*2^1+1\*2^0=16+8+4+2+0=30

左移 << 会变大，在最后一位的后边补0

右移 >> 会变小，删除最后几位

例：30>>2

console.log(30>>2); //7 11110->111->7

例：7<<1

console.log(7<<1); //12 111->1110->12

练习： 5>>2；10<<1

console.log(5>>2); //5 101->01->2

console.log(10<<1); //10 1010->10100->20

### 赋值运算符

=、+=、-=、\*=、/=、%=

练习：一件商品售价800，8折

var price=500;

price\*=0.8;

console.log(price); //400

### 三目运算符

需要由3个运算数或者表达式组组成。可嵌套。

条件表达式 ？ 表达式1 ：表达式2

如果条件表达式为 true，执行表达式1；

如果条件表达式为 false，执行表达式2。

例：判断一个人的年龄，如果大于等于18，打印成年人；否则打印未成年人

var age=20;

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

练习：判断一个人的年龄，如果大于80或者小于6岁，打印“免费”，否则打印“全价”

var age=70;

age>=80 || age<6 ? console.log('免费') : console.log('全价');

例：三目运算的嵌套

var score = 75;

var result = score >= 90 ? "aa" : (score>=80 ? "bb" : (score < 60 ? "cc" : "dd") );

console.log(result); //dd

## 两个浏览器端函数

alert() 弹出警示窗口

prompt() 弹出提示窗口，输入的值为字符串类型

（创建05.html文件和05.js文件，在使用xammp软件在05.html文件中输入"<script src="05.js"></script>"。使用xammp软件在05.js文件中输入"alert('hello world'); //弹出警示窗口"，

"//弹出提示窗口

var str=prompt('input your name');

console.log(str);

console.log( typeof(str) );//数据类型"。

在浏览器中打开05.html文件，刷新后，分别可见“警示提示窗口”和“弹出提示窗口”。

在输入栏输入“5”，可在浏览器控制台看见该数据类型“string”。）

## 流程控制

程序 = 算法 + 数据

程序分为：顺序执行、选择执行、循环执行

例如：如果满300元，减30； //选择执行

### 顺序执行

#### if语句

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

练习：通过浏览器输入商品的总价，总价满足500打8折，计算应付金额。

var money=prompt('input total');

if(money>=500){

money\*=0.8;

}

console.log(money);

练习：查看签名，如果签名为空，打印“这家伙很懒”；若有签名，就打印签名。

var txt='';

if (txt==''){ //如果签名为空，若一个等于号则是赋值

console.log('这家伙很懒');

}

（表达式2“若有签名，就打印签名”，暂时还无法达成）

|  |
| --- |
| 以下几种在条件表达式中会转成false  undefined、NaN、null、0、'' |

例：if(1){

console.log('hello');

} //'hello'

例：if(NaN){

console.log('hello');

} //输出为空

#### if-else语句

|  |  |
| --- | --- |
| 语句0；  if（条件表达式）{  语句1...  }else{  语句2...  }  语句3； |  |

if-slse语句和三目运算符的原理一致，使用上也可以相互替换。

例：如果签名为空，打印“这家伙很懒”；若有签名，就打印签名内容。

var txt='问天问地问大海';

if(txt==''){

console.log('这家伙很懒');

}else{

console.log(txt);

}

练习：判断一个人的年龄，满足18，打印“成年人”；否则打印“未成年人”。

var age=20;

if (age>=18){

console.log('成年人');

}else{

console.log('未成年人');

}

练习：收银系统，输入商品的总价，如果满500，打八折；付款的时候，若银行卡中余额不足，会打印余额不足，否则打印支付成功

var total=600;

if (total>=500){

total\*=0.8;

}

//付款——卡内余额是否大于商品的总额

var money=300;

if (money>=total){

console.log('支付成功');

}else{

console.log('余额不足');

}

练习：声明两个变量保存用户输入的用户名和密码，假如登录名为root，密码为123456；如果输入正确，打印“登录成功”，否则打印“用户名或密码错误”

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

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

console.log('登录成功');

}else{

console.log('用户名或密码错误');

}

#### if-else语句的嵌套

语句0；

if(条件表达式1){

语句1；

}else if(条件表达式2){

语句2；

}else...if(条件表达式n){

语句n;

}else {

//以上所有的条件都为false时

}

例：判断一个人的政治面貌

var a='团员';

if (a=='党员'){

console.log ('这个人是党员'

}else if(a=='团员'){

console.log('这个人是团员');

}else{

console.log('这个人是群众');

}

练习：学子商城中保存的订单状态。10—待付款,20—备货中,30—运输中,40—派送中,50—已完成, 其他—不可识别。声明一个变量保存订单的状态码，根据状态码打印出汉字的提示

var state=20;

if (state==10){

console.log('待付款');

}else if (state==20){

console.log('备货中');

}else if (state==30){

console.log('运输中');

}else if (state==40){

console.log('派送中');

}else if (state==50){

console.log('已完成');

}else{

console.log('不可识别');

}

例：90~ 优秀；80~90 良好；70~80 中等；60~70 及格；60以下 不及格

var score=85;

if (score>=90){ //false -> <90

console.log('优秀');

}

else if (score>=80){ //false -> <80

console.log('良好');

}

else if (score>=70){ //false -> <70

console.log('中等');

}

else if (score>=60){ //false -> <60

console.log('及格');

}

else{

console.log('不及格');

}

例：从60分以下开始判断，直到90以上

var score=85;

if (score<60){

console.log('不及格');

}else if (score<70){

console.log('及格');

}else if (score<80){

console.log('中等');

}else if (score<90){

console.log('良好');

}else{

console.log('优秀');

}

练：银行对客户存款进行分类，存款大于1000万是钻石用户，存款大于100万是贵宾用户，存款大于10万是优质用户，否则提示普通用户

var money=8000000;

if (money>=1e7){

console.log('钻石用户');

}else if (money>=1e6){

console.log('贵宾用户');

}else if (money>=1e5){

console.log('优质用户');

}else{

console.log('普通用户');

}

#### switch-case语句

语句0；

switch(表达式){

case 1:

语句1... ;

break;

case 2:

语句2... ;

break;

case n:

语句n... ;

break;

default:

//以上情况都不满足

语句n+1;

}

例：var day=3;//存储的是星期，其中这里的“=”表示的是全等于

switch(day){

case 1:

console.log('吃拉面');

break;

case 2:

console.log('盖浇饭');

break;

case 3:

console.log('红烧肉');

break;

case 4:

console.log('烧茄子');

break;

case 5:

console.log('吃烧烤');

break;

case 6:

console.log('麻小就啤酒');

break;

case 0:

console.log('减肥');

break;

default:

console.log('错误的星期');

}

|  |
| --- |
| break：终止后续代码的执行。  在比较的过程中使用全等于（===）。 |

练习：100：非常优秀；90~100：优秀；80~90：良好；70~80：中等；60~70：及格；60以下：不及格。

var score=86;

score=parseInt(score/10); //6,7,8,9...使用强行转换将小数转换为小数

switch (score){

case 10:console.log('非常优秀');

break;

case 9:console.log('优秀');

break;

case 8:console.log('良好');

break;

case 7:console.log('中等');

break;

case 6:console.log('及格');

break;

default:console.log('不及格');

}

##### 对比if-else嵌套和switch-case语句

相同的：两者都可以用于多项分支语句，都可以判断条件中是否相等的情况。

不同点：if-else可以条件中大于、小于等情况；

switch-case语句结构比较清晰，执行效率更高一些。

### 循环执行

循环：一遍又一遍执行相同或者相似的代码。

循环执行的两个要素：

1. **循环条件：**循环可以继续执行的条件
2. **循环体：**重复执行的相同或者相似的代码

#### while循环

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

例：打印10次'hello'

var i=0;

while(i<10){ //循环条件

//循环体

console.log('hello');

//i的值变化

i++;

}

练：打印0,2,4,6,8

var i=0;

while (i<=8){

console.log(i);

i+=2;

}

练：打印5,6,7,8,9

var i=5;

while (i<10){

console.log(i);

i++;

}

练：打印30~20之间的每一个数字

var i=30;

while (i>=20){

console.log(i);

i--;

}

练：打印1~10之间所有整数的和

var i=1;

var sum=0; //声明变量用于存储数字的和

while (i<=10){

//i:1~10

//i加到sum中——sum在原来的基础上加i

sum+=i,i++;

}

//当所有的值都加入sum后，打印最后的结果

console.log(sum);

练习：默认拨打客服电话，1-查话费 2-查流量 3-缴费充值 4-办理宽带 5-转人工 其他-请重新输入代码

var code=3;

switch(code){

case 1:

console.log('查话费');

break;

case 2:

console.log('查流量');

break;

case 3:

console.log('缴费充值');

break;

case 4:

console.log('办理宽带');

break;

case 5:

console.log('转人工');

break;

default:

console.log('请重新输入代码');

}

|  |
| --- |
| 程序中的“{ }”，当只有一个语句时，该“{ }”可省略。 |

|  |
| --- |
| break可以结束任何形式的循环。 |

练习：声明一个变量存储一个数字；循环弹出提示框（prompt），输入数字；如果输入的数字大于存储的数字，警示框（alert）提示“猜大了”；如果输入的数字大于存储的数字，警示框（alert）提示“猜小了”；否则提示“猜对了”，结束循环（break）。

var num=8;

while (true){

//使用变量保存输入的值

var a=prompt('input a number'); //输入的是字符串，字符串和8比较，字符串转成NaN

if (a>num){ //输入的大于之前存储的

alert('big');

}else if (a<num){ //输入的小于存储的

alert('samll');

}else { //输入的等于存储的

alert('right!!!');

break; //结束循环

}

}

#### do-while循环

do{

循环体

}while(循环条件);

例：循环打印0~9

var i=0;

do{

console.log(i);

i++;

}while (i<10);

练习：打印2,4,6,8

var i=2;

do{

console.log(i);

i+=2;

}

while (i<10);

练习：计算1~100之间的和

var i=1,sum=0;

do{

sum+=i;

i++;

}

while (i<=100);

console.log(sum);

练习：打印100,95,90,...,50

var i=100;

do{

console.log(i);

i-=5

}

while (i>=50);

练习：计算1~10之间的乘积

var i=1,acc=1;

do{

acc\*=i;

i++;

}

while (i<11);

console.log(acc);

练习：打印1~100之间所有能被3整除的数字

var i=1;

do{ //判断能被3整除的数字：用i和3取余

//true->打印

if (i%3==0) {

console.log(i);

}

i++;

}

while (i<=100);

练习：计算1~100之间所有能被7整除的数字的和

var i=1,sum=0;

do{

if (i%7==0) {

sum+=i;

}

i++;

}

while (i<=100);

console.log(sum);

练习：声明变量，保存密码‘123456’，循环（do-while）弹出提示框，如果输入的正确，警示框弹出“login success”，结束循环（break）；

var pwd='123456';

do{ //获取输入的值

var a=prompt('input password');

if (a==pwd){

alert('login success');

break;

}

}

while (true);

##### 对比while和do-while的区别

while 先判断，再执行

do-while 先执行一次，再判断

若条件为false，while一次都不会执行，do-while会执行一次。

#### for循环

for（初始值①；循环条件②；自增④）{

循环体③；

}

for循环中，for（；；）中所有的条件，若在之前声明，均可在式中省略，但不能更改其格式

例：打印0~9

for(var i=0;i<10;i++){

console.log(i);

}

练习：打印0,2,4,6,8

for (var i=0;i<10;i+=2 ){

console.log(i);

}

练习：计算1~100之间的和

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

sum+=i;

}

console.log(sum);

练习：计算1~100之间所有奇数的和

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

if (i%2==1) {

sum+=i;

}

}

console.log(sum);

练习：计算1~20之间所有偶数的乘积

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

if (i%2==0){

acc\*=i;

}

}

console.log(acc);

练习：计算本世纪所有的闰年

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

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

console.log(i);

}

}

练习：打印5个星，成一行

var str='';

for (var i=0;i<5 ;i++ ){ //console.log('\*');

//每次把一个\*放到str中

str+='\*';//str=str+\*

}

console.log(str);

练习：打印九九乘法表第5行

//i从1~5

//1\*5=5 2\*5=10 3\*5=15 4\*5=20 5\*5=25

var str='';

for (i=1;i<6 ;i++ ){

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

}

console.log(str);

练习：打印99乘法表第9行

for (i=1,str='';i<10 ;i++ ){

str+=i+'\*9='+i\*9+' ';

}

console.log(str);

例：求1~10的和(之前声明,之后可省略)

1. var sum=0;

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

sum+=i;

}

console.log(sum);

1. var sum=0;

var i=1;

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

sum+=i;

}

console.log(sum);

1. var sum=0;

var i=1;

for(;i<=10;){

sum+=i;

i++;

}

console.log(sum);

1. for(var i=1,sum=0;i<=10;i++){

sum+=i;

}

console.log(sum);

练习：打印1~10，用break终止。

var i=1;

for (; ;){

console.log(i);

if (i==10){

break;

}

i++;

}

##### 对比break和continue的区别

break：结束循环，后续不会再执行其他的循环

continue：跳出本次循环，继续下一次循环

例：打印0,2,4,6,8

//自增，每次加1

for (var i=0;i<10 ;i++ ){ //遇到奇数，跳过(跳出循环体)

if (i%2==1){

continue; //跳过

}

console.log(i);

}

练习：打印1~100之间，不能被3整除的数字（遇到能被3整除的数字，跳过）；

for (i=1; i<101;i++ ){

if (i%3==0) {

continue;

}

console.log(i);

}

练习：打印本世纪中的闰年一共有几个；

var count=0

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

if (i%4==0 && i%100!=0 || i%400==0){ //满足是闰年，让count+1

count++;

}

}

console.log(count);

练习：打印本世纪中的闰年，只要10个；

var count=0;

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

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

console.log(i);count++;//是闰年，count加1

//已经有10个闰年，结束循环

if (count==10){

break;

}

}

}

练习：本金10000，每年的利率3%，计算5年后连本带利有多少

var money=10000;

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

money\*=1.03;

console.log(money);

}

练习：for (var i=1,j=5;i<=10 ;i++,j+=2){

console.log(i+','+j);

}

练习：求5/3+6/5+7/7+8/9+9/11

var sum=0;

for (var i=5,j=3; i<=9;i++,j+=2 ){

sum+=i/j;

}

console.log(sum);

#### 循环嵌套

while、do-while、for三种循环之间可以相互嵌套。

例：打印1行5列

var str='';

for (var i=1;i<6 ;i++ ){

str+='※';

}

console.log(str);

练习：打印5行5列的※

for (var j=1;j<6 ;j++ )//循环体{

var str='';

for (var i=1;i<6 ;i++ ){

str+='※';

}

console.log(str);

}

练习：用※打印直角三角形

※

※※

※※※

//外层循环：循环的行数 i:1,2,3,4

for (var i=1;i<5;i++ ){ //内层循环：循环的列数 j:1,2,3,4

var str='';

for (j=1;j<=i;j++){

str+='※';

}

console.log(str);

}

练习：九九乘法表

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

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

for (var i=1;i<10 ;i++ ) //控制着行的情况

//外层循环：i控制着循环的行数

{ //控制着列的情况

//内存循环：j控制着循环的列数

var str='';

for (var j=1;j<=i ;j++ ){ //把行数乘以列数的值，并拼接到str中

str+=i+'\*'+j+'='+i\*j+' ';

}

console.log(str);

}

练习：打印一个倒着的九九乘法表

for(var i=9;i>=1;i--){ //内层循环：循环的列数j

var str='';

//每一行的初始值为当前的行号

for(var j=i;j>=1;j--){

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

}

//把内容拼接完以后，打印整行

console.log(str);

}

/\*

9\*9=81 9\*8=72 9\*7=63 9\*6=54 9\*5=45 9\*4=36 9\*3=27 9\*2=18 9\*1=9

8\*8=64 8\*7=56 8\*6=48 8\*5=40 8\*4=32 8\*3=24 8\*2=16 8\*1=8

7\*7=49 7\*6=42 7\*5=35 7\*4=28 7\*3=21 7\*2=14 7\*1=7

6\*6=36 6\*5=30 6\*4=24 6\*3=18 6\*2=12 6\*1=6

5\*5=25 5\*4=20 5\*3=15 5\*2=10 5\*1=5

4\*4=16 4\*3=12 4\*2=8 4\*1=4

3\*3=9 3\*2=6 3\*1=3

2\*2=4 2\*1=2

1\*1=1

\*/

## 函数

### 函数的概念

函数，有时也被称为方法（Function），或者过程（Procedure）。

是一段预定义好，并可以反复使用的代码块，其中可以包含多条可执行语句。

函数本质上是功能完整的对象。

分为系统函数和自定义函数

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

### 自定义函数

function：功能体，函数，可以接受若干数据，返回处理的结果。用于封装需要反复执行的代码。

#### 创建普通函数

function 函数名称(){

//函数体——要封装的反复执行的代码

}

调用

函数名称()

练习：封装10+20的结果，并把结果打印出来，调用3次。

function addNum(){

console.log(10+20);

}

addNum();

addNum();

addNum();

练习：封装1~100所有数字相加的和，打印和。

function getsum(){

var sum=0;

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

sum+=i;

}

console.log(sum);

}

getsum ()

#### 创建带有参数的函数

function 函数名称(参数列表){

函数体

}

调用

函数名称(实参列表)

参数列表：可以是0个或者多个数据，之间用逗号隔开。创建函数时的参数称为“形参”，只负责占位；在调用的时候，会被“实参”替换。

形参本质上就是一个变量。

例：带有参数的函数

function addNum(num1,num2){

console.log('参数1：'+num1); //参数1：3

console.log('参数2：'+num2); //参数2：5

}

addNum(3,5);

练习：创建函数getSum()，传递一个参数，得到1~任意数字之间的和

function getSum(num){

var sum=0;

for(i=1;i<=num;i++) {

sum+=i;

}

console.log(sum);

} //var num=100；

getSum(100); //结果：5050

练习：创建函数getRun，传递两个参数，打印任何两个年份之间所有的闰年

function getRun(start,end){

//循环start~end

var count=0;

for(var i=start;i<=end;i++){

//判断每一个年份是否为闰年

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

count++;

console.log(i);

}

}

//所有的年份都已经检测是否为闰年

console.log(count);

}

var num=getRun(2000,2008); //得到闰年的个数

#### 带有返回值的函数

function 函数名称(参数列表){

函数体

return 值;

}

调用

函数名称(实参列表)

函数的处理结果并非一定要“输出”，为避免该情况，只需“返回”执行结果即可--更便于函数的嵌套调用

例：计算3个数的和，并返回

function getSum(num1,num2,num3){

//console.log(num1+num2+num3); //画蛇添足

return num1+num2+num3; //不加return，会得到undefined

//return 10; //return后就是获取函数的值

}

//获取用户输入的值

//var a=prompt();

//获取函数的值返回结果

var sum=getSum(2,4,8);

console.log(sum);//打印效果

练习：计算1~10之间的乘积，并返回结果

function getJC(num){

var sumJC=1;

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

sumJC\*=i;

}

//把结果返回

return sumJC;

}

var b=getJC(10);

console.log(b);

注意：

1. 没有使用return或者return后没有值，返回undefined。
2. return可以返回任何数据类型。
3. return后所有的代码都不再执行。

例：计算3个数的和，并返回

function getSum(num1,num2,num3){

//console.log(num1+num2+num3); //画蛇添足

return num1+num2+num3; //不加return，会得到undefined

//return 10; //return后就是获取函数的值

}

//获取用户输入的值

//var a=prompt();

//获取函数的值返回结果

var sum=getSum(2,4,8);

console.log(sum); //打印效果

练习：计算1~10之间的乘积，并返回结果

function getJC(num){

var sumJC=1;

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

sumJC\*=i;

}

//把结果返回

return sumJC;

}

var b=getJC(10);

console.log(b);

练习：计算两个数相加

function addNum(num1,num2){

//console.log(num1+num2); //有return就不再需要了

return num1+num2;

}

var res=addNum(3,5);

console.log(res);

练习：声明函数getMax，传递两个参数，返回最大值。

法一：

function getMax(num1,num2){

if (num1>num2){

return num1;

}else{

return num2;

}

}

var a=getMax(35,33);

console.log(a);

法二：三目运算符 法

function getMax(num1,num2){

return num1>num2 ? num1 : num2;

}

var a=getMax(33,35);

console.log(a);

练习：声明函数isRun，传递一个参数（年份），是闰年返回true，否则返回false

法一：

function isRun(year){

var i=year;

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

return true;

}else{

return false;

}

}

var a=isRun(2018);

console.log(a);

法二：三目运算符 法

function isRun(year){

var i=year;

return (i%4==0 && i%100!=0 || i%400==0) ? 'true' : 'false';

}

var a=isRun(2018);

console.log(a);

##### 对比break，continue，return用法

1. break 结束循环。
2. continue 跳过本次循环，继续下一次循环。
3. return 终止函数的执行，并返回值，常用于函数中。

### 作用域

作用域：变量或者函数的可访问范围,他控制着变量与函数的可见性和生命周期。

在JavaScript中,变量和函数的作用域分为：

1. 函数作用域：只在当前函数内可访问
2. 全局作用域：一经定义，在代码中任何地方都能访问

注意：在函数内声明的变量，没有使用var关键字，自动变为全局变量。

#### 变量的作用域

函数作用域中的变量（局部变量）只在当前函数内可访问到，离开此范围就无法访问了。

全局作用域中的变量（全局变量）在代码的任何位置都可以访问到。

例：

var num2=3;

function fn(){

var num1=2;

console.log(num1); //只能访问函数以内的变量

//访问num2

console.log(num2); //可以访问函数以外的变量

}

fn();

console.log(num2);

|  |
| --- |
| **例**：  var num=2;  function fn(){  num=3; //全局变量可以在局部修改  }  fn();  console.log(num); //3 |
| **例**：  function fn(){  num=2; //若函数内的变量没有var声明，  } 可自动变为全局变量。前提是被调用。  fn();  console.log(num); //2 |

#### 变量声明提升

js程序执行前，使用var关键字声明的变量都会提升到所在作用域的最前面。但是赋值还是在原来的位置。

例：

//var num;

console.log(num); //undefined

var num=2; //num=2

console.log(num); //2

例：

function fn(){

//var num2; //声明提前，但赋值没有提前

console.log(num2); //undefined

var num2=3;

}

fn();

|  |
| --- |
| **例**：  var a=1;  function fn(a){ //把实参赋给形参,形参是局部变量  //var a=1; //全局函数影响局部函数的定义域的参数声明  a=a+3;  console.log(a); //4  }  fn(a);  console.log(a); //1 |
| **例：**  var a=1;  function fn(){  a=a+3; //是在访问全局变量a。  此处无var为全局变量；有var为局部变量  console.log(a); //4  }  fn();  console.log(a); //4 |
| **例：**  var a=1;  function fn(){  //var a; //undefined,声明未赋值  var a=a+3; //是在访问局部变量a。undefined+3=NaN  console.log(a); //NaN  }  fn();  console.log(a); //1 |

#### 变量的生存期

变量的生命期从它们被声明的时间开始。

局部变量会在函数运行以后被删除。

全局变量会在页面关闭后被删除。

#### 函数的作用域

函数和变量类似，也分为全局作用域和局部作用域。

全局作用域：全局作用域下创建的函数可以在任意位置访问。

局部作用域：局部作用域下创建的函数只能在局部访问。

|  |
| --- |
| **例**：全局作用域下创建函数  function fn(){  console.log(1);  }  //fn(); //1 全局函数  function addNum(){  fn(); //调用fn  }  addNum(); //1 全局函数在任何位置都可以调用 |
| **例**：在局部作用域下创建函数  function fn(){  function fn1(){  console.log(2);  }  fn1(); //2 局部函数  }  fn(); //2 无法在全局调用局部作用域的函数 |

例：

var num=3; <————————注释

function fn0(){

var num=0; <———————注释

function fn1(){

var num=1; <—————注释

function fn2(){

var num=2; <———注释

console.log(num); //2 //1 //0 //3 //报错

}

fn2();

}

fn1();

}

fn0();

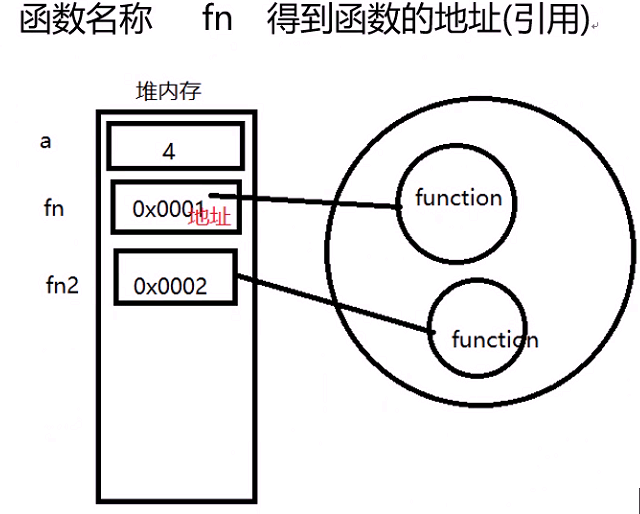
//当有var num=2时，结果为2；当没有var num=2时，其会向上一级寻找var num，得到1；以此类推，直至全局作用域均没有时，系统报错。

//上一级的局部作用域相当于下一级的全局作用域。

//注意要先调用才能将上一级的局部变量视为下一级的全局变量。

注意:

1. 函数调用：fn()，得到函数的返回值。
2. 函数名称：fn，得到函数的地址（引用）。



例：

function fn(){

console.log(1);

}

fn(); //表示：函数的调用

console.log(fn); //表示：函数的结构，属于引用类型，表示这函数的存储地址，即函数的引用

#### 函数声明的提升

和变量一样，是在程序运行前，把使用function声明的函数提升到程序的最前边。

例：

function fn(){

console.log(1); //2

}

fn();

function fn(){

console.log(2); //2

}

fn();

### 递归

在函数的内部调用自身。

例：执行3次

var i=0;

function say(){

i++;

alert('cong qian');

//当i为3时，结束

if (i==3){ //跳出的条件

return;

}

say();

}

say();

练习：创建函数getSum，传递参数，返回1~参数之间所有数值的和。

//1~n

function getSum(n){

//跳出条件 n=1时跳出

if (n==1){

return 1;

}

return n+getSum(n-1); //getSum(5)=5+getSum(4); getSum(4)=4+getSum(3); ...;getSum(1)=1

}

var res=getSum(5);

console.log(res);

练习：创建函数getJC，传递参数，返回1~参数之间所有数字的乘积。

//5\*4\*3\*...\*1，其顺序为倒着来

function getJC(n){

if (n==1){

return 1;

}

return n\*getJC(n-1);

}

var res=getJC(5);

console.log(res);

练习:斐波那契数列

1

1 2

3 5 8

13 21 34 55

89 144 233 377 610

………………

使用递归获取第n个月的个数

function fib(n){

//跳出条件：当第1,2个月的结果是1

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

return 1;

}

//当前月份的值=前两个月相加

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

}

var res=fib(12);

console.log(res);

/\*

fib(4)+fib(3)

fib(3)+fib(2)+fib(2)+fib(1)

fib(2)+fib(1)+fib(2)+fib(2)+fib(1)

\*/

### 匿名函数

#### 创建函数——函数表达式

var 函数名称=function(形参列表){

函数体

return 值； //返回值

}

调用方式：函数名称(实参列表)

##### 对比函数表达式和函数声明的区别

函数声明存在函数提升，在任何位置都可以调用。

函数表达式不存在提升，必须先创建再调用。

例：函数声明 创建函数

var fun=function (){

console.log(132);

}

fun(); //函数名称直接用变量来表示

例：函数表达式 创建函数

var fun=function (num1,num2){

console.log(num1+num2);

}

fun(3,4);

例：函数表达式 创建函数

var fun=function (num1,num2){

return num1+num2;

}

var res=fun(3,4);

console.log(res);

例：函数表达式不存在提升，必须先创建再调用，否则报错

var res=fun(3,4);

var fun=function (num1,num2){

return num1+num2;

}

console.log(res);

全局污染：在程序中使用变量创建函数，其余人再不能创建同名的全局函数，否则报错。

避免全局污染：创建函数作用域，不使用变量，也不使用函数名称

#### 匿名函数的自调用

创建一个局部（函数）作用域

(function (){

函数体

})();

例：

(function (){

var num1=1;

console.log(num1); //1

})();

(function (){

var num2=2;

console.log(num2); //2

})();

#### 回调函数

把匿名函数以实参的形式传递给形参

例：

function fn(a,b){

}

fn(function(){},function(){}) //回调函数

例：

function fn(a,b){

//匿名函数创建的 函数

//var a=function(){console.log('匿名函数1');} //计算机自动执行的操作

//var b=function(){console.log('匿名函数2');} //计算机自动执行的操作

//调用至var a=...和var b=...

a();

b();

}

fn(function(){

console.log('匿名函数1');

},function(){

console.log('匿名函数2');

}) //‘匿名函数1’‘匿名函数2’

例：使用回调函数实现两个数字的相加，并返回

function addNum(a,b){

//如果想要得到1，必须调用匿名函数

//a=function(){}

//var num1=a(); //调用函数 参数+（）

//var num2=b();

//console.log(num1);

//console.log(num2);

return a()+b();

}

例：

var res=addNum(function (){

//可以添加多行代码——意味着可以传递多行代码

return 1;

},function(){

return 3;

}); //传递一个匿名函数

console.log(res); //4

### 全局函数

例：//对中文编码

var url='http://www.codeboy.com/search?keyword=汽车';

console.log(url);//http://www.codeboy.com/search?keyword=汽车

//对URL进行编码

url=encodeURI(url);

console.log(url);//http://www.codeboy.com/search?keyword=%E6%B1%BD%E8%BD%A6

//对url进行解码

url=decodeURI(url);

console.log(url);//http://www.codeboy.com/search?keyword=汽车

encodeURI：对一个URI进行编码

decodeURI：对一个URI进行解码

eval：执行字符串表达式

parseInt：将数据转为整型

parseFloat：将数据转为浮点型

typeof：判断一个数据类型

isNaN：检查一个值是否为NaN,如果是->true , 否->false

isFinite：检查一个值是否为有限值，是->true 否->false

1/0->Infinity 无限的

练习：使用eval函数结合弹出提示框实现计算器效果，使用弹出警示框显示效果。

//弹出提示框

var str=prompt('input a expression');

var res=eval(str);

alert(res);

例：console.log(10/0);//infinity 无穷的

例：判断是否为有限值 -> 无限的false 有限的true

var num=1/0;

var res=isFinite(num);

console.log(res);//false

### 对象

对象是引用类型的数据，存储在堆内存以外。

对象：是一组属性（property）和方法（功能）（method）的集合。

#### JS中的对象

内置对象（相当于常量，已经定义好的）；

宿主对象（对执行环境有要求的对象，如只能在浏览器端或node服务器端运行的对象）；

组定义对象。

#### 创建自定义对象

对象字面量（直接量）

通过new Object()

自定义构造函数

#### 对象字面量创建对象

使用{}创建空对象

属性名和属性值之间用冒号隔开

多组属性之间用逗号隔开

属性名中可以加引号，如果出现特殊字符（空格、-）必须加引号

练习：创建一个员工对象，属性有编号，姓名，性别，生日，收入，电话

var emp={

id:001,

uname:'东山谢安石',

sex:1,

birthday:'575-3-6',

salary:5000,

tel:'029-85647321'

}

console.log(emp);

练习：创建一个商品对象，属性有编号pid，标题title，价格price，库存stockCount，上架时间shelfTime

var shop={

pid:15,

title:'驱蚊器二',

price:546,

stockCount:85,

shelfTime:'2018-9-13'

}

console.log(shop);

#### 使用new Object创建对象

创建一个对象 var book=Object();

给对象添加属性

如：

book.title='三国演义'; //竖着排

book['price']=88; //横着排

例：使用new Object创建对象

var book=new Object ();

//添加对象

book.bid=1501;

book.title='三国演义';

book.price=88.99;

book.author='罗贯中';

book['pubTime']

console.log(book);

练习：创建一个汽车对象，编号cid，品牌brand，颜色color,大小size,价格price

var car=new Object();

car.cid=131; //.指对象下面的

car.brand='bmw';

car.color='red';

car.size='small';

car.price=1e7;

console.log(car);

练习:创建一个手机对象,编号pid,品牌brand,

var phone=new Object();

phone['pid']=654;

phone['brand']='apple';

phone['color']='黑色';

console.log(phone);

//获取属性值

console.log(phone.brand);

console.log(phone['pid']);

例：

var person={

name:'tom',

age:18,

}

console.log(person);

//修改属性值

person.name='jerry';

//获取属性值

console.log(person.name);

console.log(person['age']);

#### 遍历（循环）属性

for-in

for(var key in emp){

//key 表示所有的属性

//in后表示要遍历的对象

//emp 表示要遍历的对象

//对象[key] 表示获取每一个属性的值

}

注意：不能遍历预定义（对象自带）的属性

例：

var emp={

eid:8,

ename:'tom',

sex:1,

birthday:'1998-3-1',

phone:'13912345678',

salary:5000

}

//遍历（循环）属性

//key 表示要遍历所有的属性名

//in后表示要遍历的对象

//对象[key]表示所有的属性值

for (var key in emp){

console.log(key+'---'+emp[key]);

}

练习：遍历之前一个商品对象

var product={

pid:654,

brand:'apple',

color:'黑色'

}

for (var key in product){

console.log(key+'---'+product[key])

}

//使用product ['pid']=654;格式会报错

#### 检查对象中是否含有某个属性

1. '属性' in 对象
2. 对象.hasOwnPropery('属性')
3. 对象.属性 === undefined

例：1.in关键字,可以判断某一对象是否存在，存在->true,不存在->false

console.log('salary' in person);

例：2.使用hasOwnProperty

console.log(person.hasOwnProperty('name'));

例：3.判断是否为undefined，如果true->不存在 false->存在

console.log(person.sex===undefined);//建议使用全等于，有时会有数字类型转换

#### 对象中的方法

var person={

say:function(){}

}

调用对象中的方法 person.say()

在方法中访问属性 this.属性名;this引用当前的对象

例：

var person={

name:'tom',

age:20,

say:function (){

console.log('我叫张三');

//this,代指函数名称

return 'my name'+this.name+',我今年'+this.age;

},

run:function(){

return 'running';

}

}

//调用对象中的方法

person.say();

var res=person.say();

console.log(res);

var res1=person.run();

console.log(res1);

例：创建手机对象，添加打电话call，发短信sendMsg，看电影movie

var phone={

call:function(){

return'calling...';

},

sendMsg:function(){

return'sending...';

},

movie:function(){

return'watching movie';

}

}

var res1=phone.call();

console.log(res1);

var res2=phone.sendMsg();

console.log(res2);

var res3=phone.movie();

console.log(res3);

练习：创建园对象circle，添加属性半径r、圆周率PI；添加方法计算圆的周长getLength和面积getArea。

var circle={

r:10,

PI:3.14,

getLength:function(){

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

},

getArea:function (){

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

}

}

var res=circle.getLength();

console.log(res);

var res1=circle.getArea();

console.log(res1);

例：

var person={

name:'tom',

age:20

}

//把person对象的地址复制给了p

//person和p两个指向同一个对象

var p=person;

p.name='jerry';

person.age=80;

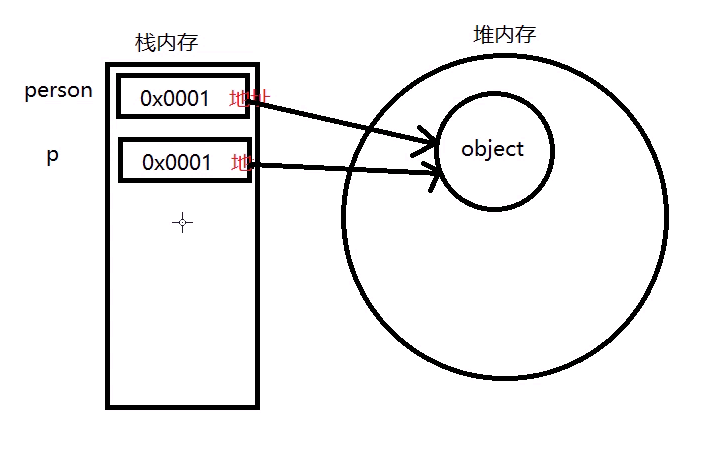
console.log(p); //{ name: 'jerry', age: 80 }

console.log(person); //{ name: 'jerry', age: 80 }

console.log(typeof(person)); //object

console.log(typeof(null)); //object，规定死的，没原因

**对象的存储**



### 数组

按照特定形式组合的数据，是数据集合。

#### 1.创建数组

##### 数组字面量

[元素1，元素2，元素3...]

数组中可以存放任何类型的数据

练习：创建汽车品牌数组，添加若干元素

var cars=['BMW','奇瑞','宝骏'];

console.log(cars);

##### 内置构造函数——new Array

new Array（元素1，元素2，....）

new Array(5) 创建数组，初始化长度为5

例：

var names=new Array(5); //表示初始化5个元素

//添加元素

names[0]='中锋';

names[1]='大前锋';

names[2]='小前锋';

names[3]='得分后卫';

names[4]='控球后卫';

names[5]='教练';

console.log(names);

练习：创建一个数组，初始化5个学科；添加对应的元素

var subjects=new Array(5);

subjects[0]='语文';

subjects[1]='数学';

subjects[2]='英语';

subjects[3]='物理';

subjects[4]='化学';

subjects[5]='生物';

console.log(subjects);

//获取任意一个数值

//获取下标为2的元素

console.log(subjects[2]);

console.log(subjects[6]); //undefined

#### 数组元素的访问

数组中每一个元素都有对应的下标，默认是数字，从0开始访问数组就可以通过下标

数组[下标]

练习：创建数组country保存4个国家名称，添加第5个国家，修改第3个国家；打印结果

var country=['中国','美国','英国','法国'];

country[4]='俄罗斯';

country[2]='瑞典';

console.log(country);

#### 获取数组的长度

数组.length

使用数组长度在最后添加新元素

数组[数组.length] = 值;

例：

var arr=[];

//数组的长度属性

//console.log(arr.length);

//使用长度来添加元素

arr[arr.length]='tom';

arr[arr.length]='jerry';

console.log(arr);

#### 遍历数组

for循环：只能遍历索引数组，本质是在循环数组元素的下标

for-in循环：即可以遍历索引数组，也可以遍历关联数组数据

for( var key in arr){

//key 遍历的数组下标

//获取数组元素 arr[key]

}

练习：使用for循环遍历一个汽车数组，把品牌为宝马的值改为BMW

var cars=['奔驰','宝马','奥迪','大众'];

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

//console.log(cars.[i]); //['奔驰','宝马','奥迪','大众']

//如果元素是宝马，改为BMW

if(cars[i]=='宝马'){

cars[i]='BMW';

}

}

console.log(cars); //['奔驰','BMW','奥迪','大众']

练习：创建一个员工数组，编号，姓名，性别，出生日期，工资

var emp=[]; //数组里不允许写冒号

emp['eid']=20;

emp['ename']='tom';

emp['sex']=1;

emp['birthday']='1991-2-2';

emp['salary']=5000;

console.log(emp);

例：遍历关联数组

var books=[];

books['title']='三国演义';

books['author']='罗贯中';

books['price']=100;

books['pubTime']='2000-10-1';

for (var key in books){

console.log(key+'----'+books[key]);

}

##### 遍历索引数组

练习：使用for-in遍历一个索引数组

var arr=['推荐','热点','新闻','娱乐','体育','科技'];

for (var key in arr){

console.log(key+'-----'+arr[key]);

}

练习：创建数组，包含员工的工资，获取员工的所有工资的和是多少？求平均工资是多少？

[3000,5000,4000,8000,12000]

//法一、for循环

var sum=0;

var arr=[3000,5000,4000,8000,12000];

for (var i=0;i<arr.length;i++ ){//i 0~4

arr[i];

sum+=arr[i];

}

console.log(sum);

console.log(sum/arr.length);

//法二、for-in循环

var sum=0;

var salary=[3000,5000,4000,8000,12000];

var i=0;

for (var key in salary){

i++;

sum+=salary[key];

}

console.log(sum);

console.log(sum/i);

练习：创建数组，包含多个城市，获取北京所在的下标值。

//for-in

var city=['北京','天津','南京','西安','成都','太原','沈阳','广州'];

for (var key in city){

if (city[key]=='北京'){

console.log(key);

}

}

//for

var city=['北京','天津','南京','西安','成都','太原','沈阳','广州'];

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

if (city[i]=='北京'){

console.log(i);

}

}

练习：将以上代码封装成函数getIndex，传递两个参数（1.遍历的数组；2.查找的元素）；返回找到的下标值。如果找不到返回-1

function getIndex(city,value){

//var city=['tom','jerry','david'];

//var value='jerry';

//进行遍历

for (var key in city){

//如果元素值是 形参value的值，就打印

if (city[key]==value){

//console.log(key);

return key;

}

//for循环结束，找不到，返回-1

return -1;

}

}

var res1=getIndex(['tom','jerry','david'],'jack');

console.log(res1);

练习：创建数组，包含多个姓名（有重复的），获取tom出现的次数。

//for-in循环

var names=['tom','jerry','david','tom','jack','tom'];

var count=0;

for (var key in names){

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

count++;

}

}

console.log(count);

//for循环

var names=['tom','jerry','david','tom','jack','tom'];

var count=0;

for (var i=0;i<names.length;i++ ){ //如果遍历的元素是'tom'，count加1

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

count++;

}

}

console.log(count);

练习：把以上代码封装成函数getCount，传递两个参数（1.要遍历的数组；2.要查找的值）；返回找到的数量

function getCount(array,value){

var count=0;

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

if (array[i]==value){

count++;

}

}

return count;

}

var res=getCount(['a','b','c','d','e','f'],'g');

console.log(res);

练习：创建数组，包含一组数字，获取最大值

//函数形式：创建函数，传递一个参数（数组），返回数组中的最大值。[50,30,27,80]

法一:

var num=[50,30,27,80];

for (var key in num){

var max=0;

if (num[key]>num[key-1]){

max=num[key];

}else{

max=num[key-1];

}

}

console.log(max);

法二:

function getMax(num){

var max=0;

for (var key in num){

if (num[key]>max ){

max=num[key];

}

}

return max;

}

var str2=getMax([50,90,27,80]);

console.log(str2);

法三:

function getMax(array){

//假设第一个元素就是最大值

var max=array[0];

for (var i=1;i<array.length;i++){

//如果元素值大于max,就把元素值放入到max中

if (array[i]>max ){

max=array[i];

}

}

return max;

}

var str=getMax([1,2,3,4,5]);

console.log(str);

例：

var arr1=['tom','herry','jacky'];

var arr2=arr1;

arr2[1]='scott';

console.log(arr1); //[ 'tom', 'scott', 'jacky' ]

console.log(arr2); //[ 'tom', 'scott', 'jacky' ]

console.log(typeof(arr1)); //object

//同“对象”的存储一样

#### 数组的分类

数组分为索引数组和关联数组

索引数组：以整数作为下标；

关联数组：以字符串作为下标；

例：创建数组——以字符串作为下标（关联数组）

var arr=new Array(3); //索引数组

arr['name']='tom'; //关联数组

arr['age']=20;

//console.log(arr); //[ <3 empty items>, name: 'tom', age: 20 ],前3个以数字为下标，后2个以字符串作为下标

#### 数组中的方法（API）

API->Application Programming Interface

toString() 将数组转为字符串 （加（）是方法，不加（）是属性）

join() 按照指定的分隔符转为字符串 "|"用指定的字符拼接

concat(arr1,arr2,...) 拼接两个或者更多的数组

slice(start,end) 截取数组元素；star开始，end结束，不包含结束；如果没有end，将截取到最后。

splice(start,count,value1,valu2,...) 删除数组中元素；start开始的下标，count删除的数量，value1...要插入的元素

reverse() 翻转数组，数组中的元素前后颠倒顺序

sort() 对数组的元素排序，默认按照Unicode码排序

sort(function (a,b){

return a-b; //数字从小到大

return b-a; //数字从大到小

})（return只能出现一个）

push() 在数组的最后添加元素，返回数组的长度

pop() 删除数组的最后一个元素，返回删除的元素

unshift() 在数组的开头添加一个元素，返回数组的长度

shift() 删除数组的第一个元素，返回删除的元素

例：

var arr=['tom','jerry','kate','King','cris'];

console.log(arr.toString()) //将数组转为字符串

console.log(arr.join('|')) //按照指定的分隔符转为字符串；"|"用指定的字符拼接

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

var arr2=['richard','cris'];

console.log(arr.concat(arr1)); //拼接多个数组

console.log(arr1.concat(arr,arr2)); //拼接多个数组

console.log(arr.slice(1)); //截取从1开始到最后

console.log(arr.slice(1,3)); //截取从1开始到3结束（不包括3）

console.log(arr.slice(-3));

console.log(arr.slice(-3,-1));

练习：截取b,c,d三项和g,h两项；拼接成一个新数组，打印出来。

var arr4=['a','b','c','d','e','f','g','h','i'];

var a1=arr4.slice(1,4);

var a2=arr4.slice(-3,-1);

console.log(a1);

console.log(a2);

console.log(a1.concat(a2));

例：

var arr=['tom','jerry','kate','King','cris'];

console.log(arr.splice(2,1)); //将删除的元素打印,从第2个之后开始，删除1个

console.log(arr.splice(2,2,'scott')); //删除后，在原位置插入1个（可写多个元素）

console.log(arr.splice(2,0,'scott','lucy'));

console.log(arr); //原数组元素被删除

练习：删除'热点'，在'娱乐'后添加'新闻'和'体育'；替换'推荐'为'汽车'

var arr=['推荐','热点','娱乐','科技'];

arr.splice(1,1);

arr.splice(2,0,'新闻','体育');

arr.splice(0,1,'汽车');

console.log(arr); //['汽车', '娱乐', '新闻', '体育', '科技']

例：翻转

var arr=['tom','jerry','kate','King','cris'];

console.log(arr.reverse()); //数组翻转;[ 'cris', 'King', 'kate', 'jerry', 'tom' ]

console.log(arr.sort()); //默认按照Unicode码排序

例：排序

var arr=[52,34,29,17,60];

console.log(arr.sort(arr)); //按照首字母排序

例：排序

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

//return a-b; //从小到大

return b-a; //从大到小

})) //固定用法

例：

var arr=['推荐','热点'];

console.log(arr.push()); //2

console.log(arr.push('新闻')); //3,表示数组的元素长度

console.log(arr.pop()); //默认删除最后一个元素，并返回删除的元素

console.log(arr);

练习：使用变量翻转数组

['tom','jerry','jacky'] -> ['jacky','jerry','tom']

var arr=['tom','jerry','jacky'], a='';

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

a += arr[i] + ' ';

}

console.log(a);

练习：**冒泡排序**

[1,2,3,4,5] 由大到小排序

法一:自编

var num = [1,2,3,4,5];

var temp;

for (var i = 0; i < num.length - 1; i++){ //总共有5个数，相当于有5组，每组分别对比一次则需要循环4次

for (var j = 0; j < num.length - 1 ; j++){ //每个数之间对比一次后，才能确定一个数的位置，所以为4次

if (num[j] < num[j + 1]){

temp = num[j + 1];

num[j + 1] = num[j];

num[j] = temp;

}

}

console.log(num); //[5,2,3,4,1]

}

法二:官方

function fn(array){

var length = array.length, temp;

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

for (var j = length - 1; j >=1;j--) {

if (array[j] > array[j - 1]){

temp = array[j];

array[j] = array[j - 1];

array[j - 1] = temp;

}

}

console.log(array);

}

}

var str=fn([1,2,3,4,5]); //[5,2,3,4,1]

例：

var arr=['tom','jarry'];

//在开头添加元素

arr.unshift('jone');

console.log(arr.unshift('jone')); //4

//在开头删除元素

console.log(arr.shift()); //jone

console.log(arr); //[ 'jone', 'tom', 'jarry' ]

#### 二维数组

在数组中，每一个元素也是数组

var arr=[ [ ], [ ], [ ] ];

获取二维数组中的元素 arr[下标][下标]

例：二维数组的创建

var arr=['北京','天津','河北'];

var city=[

['西城','东城','海淀'],

['河东','和平','蓟县'],

['石家庄','秦皇岛','唐山']

];

//二维数组的获取

console.log(city[0]); //[ '西城', '东城', '海淀' ]

console.log(city[0][2]); //海淀

### 字符串操作

#### 包装对象

目的就是让原始数据类型也可以像引用类型，具有属性和方法。

JS提供了3中包装类型：String、Number、Boolean

例：

var str1='hello';

console.log(typeof(str1)); //string

console.log(str1+' world'); //hello world

//包装成字符串对象

var str2=new String('hello');

console.log(str2); //[String: 'hello']

console.log(typeof(str2)); //object //对象

console.log(str2+' world'); //hello world

var str3=String('hello');

console.log(str3); //hello

console.log(typeof(str3)); //string //字符串

#### 转义字符——**\**

把普通的字符，转成具有特定意义的字符

\n 换行

\' 普通引号'

\t 制表符（合tab键作用一样）

\\ 产生一个普通的\

例：转义字符

var str='hello';

console.log(str.length); //5

例：转义字符

var str='hello\nworld'; // \,转义字符

console.log(str); //hello 换行 world

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

练习：打印出现 welcome to chi\na

var str='welcome to chi\\na';

console.log(str); //welcome to chi\na

#### 字符串中的常用方法

toUpperCase() 英文字符转大写

toLowerCase() 英文字符转小写

charAt(n) 获取指定下标的字符

charCodeAt() 获取一个字符的Unicode码

indexOf(value,start) 查找某个字符对应的下标；value要查找的字符串，start开始查找的下标，默认是0；找不到返回-1。

lastIndexOf(value,index) 倒着查找某个字符对应的下标；value要查找的字符串，index表示查找结束的下标，开始是0开始；找不到返回-1。

slice(start,end) 截取字符串，start开始的下标，end结束的下标，不包括end本身，如果end为空，截取到最后，允许使用负值；

substring(start,end) 截取字符串，和slice用法一致；区别在于不支持负值，如果是负值，自动转成0；

substr(start,length) 截取指定长度的字符串，start开始的下标，length，返回字符串中的字符数，如果length为空，则返回从开始位置到结束的所有数值。

split(sep) 按照指定的格式(sep)分隔字符串,返回一个数组。

例：大小写转换

var str='hOw aRE YoU';

console.log(str.toUpperCase()); //HOW ARE YOU,全部转大写

console.log(str.toLowerCase()); //how are you,全部转小写

练习：初始化4个英文字符保存在变量中，循环弹出提示框，输入4个字符，如果输入正确结束循环。（不区分大小写）

var code='FtMz';

while (true){

//弹出提示框

var str=prompt('input the code');

//判断输入的是否正确

//都转成大写或小写进行比较

code=code.toLowerCase();

str=str.toLowerCase();

if (str==code){

break;

}

}

练习：使用for循环遍历字符串

var str='welcome',a;

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

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

}

例：

var str='welcome',a;

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

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

}

查看字符的Unicode码

console.log('ab'.charCodeAt()); //97,只显示a的Unicode码

console.log(str.indexOf('s')); //找不到返回-1

console.log(str.indexOf('e',2)); //6,查找e字符，从编号为2的字符开始

console.log(str.lastIndexOf('w',0)); //0

练习：检查一个字符串是否为一个合法的邮箱格式；只需要检查是否含有@；如果有，返回true，否则返回false

var email='tom@163.com';

//检查字符串中是否含有@，如果有返回下标 >=0

//console.log(email.indexOf('@'));

email.indexOf('@')>=0 ? console.log(true) : console.log(false);

例：截取字符串——slice

var str='javascript';

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

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

console.log(str.slice(-5,-2)); //cri

练习：将身份证号中的年月日取出来；打印\*\*\*\*年\*\*月\*\*日；性别

倒数第2位 奇数-男 偶数-女; 110230199508053579

var id='110230199508053579';

var sNum,year,month,date,sex;

var sNum=id.slice(-2,-1);

year=id.slice(6,10);

month=id.slice(10,12);

date=id.slice(12,14);

sNum%2==0 ? sex='女' : sex='男';

console.log (year+'年'+month+'月'+date+'日'+'；'+'性别：'+sex);// 1995年08月05日；性别：男

练习：根据邮箱（jerry@sina.com.cn）格式，获取邮箱的用户名和邮箱的注册域名

var str='jerry@sina.com.cn';

//查找到@的下标

var index=str.indexOf('@');

//console.log(index); //5

//截取用户名：从0开始到@前

var uname=str.slice(0,index);

console.log(uname);

//截取域名:从@下标的下一位

var domain=str.slice(index+1);

console.log(domain);

练习：将一个单词的首字母转为大写，其他都小写。‘hello’

var str='hello';

var frist=str.slice(0,1);

frist=frist.toUpperCase();

//console.log(frist); //h

//截取第二个字符到最后

var last=str.slice(1);

last=last.toLowerCase() ;

console.log(frist+last); //Hello

例：substring

var str='javascript';

console.log(str.substring(4,8)); //scri

console.log(str.substring(0)); //javascript

console.log(str.substring(-6)); //javascript

例：数组转为字符串

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

console.log(arr.toString()); //tom,jerry,kate

console.log(arr.join('|')); //tom|jerry|kate

例：split（sep）

//字符串转为数组

var str='tom,jerry,kate';

//按照逗号(,)将字符串切割为3部分——返回数组

var arr=str.split(',');

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

var str1='tom@jerry@kate';

var arr1=str1.split('@');

console.log(arr1); //[ 'tom', 'jerry', 'kate' ]

练习：使用split获取邮箱（jerry@sohu.com）的用户名和域名

var str='jerry@sohu.com';

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

//console.log(arr); //[ 'jerry', 'sohu.com' ]

console.log('用户名：'+arr[0]); //用户名：jerry

console.log('域名：'+arr[1]); //域名：sohu.com

练习：把一句英文的所有单词首字母转为大写，非首字母转为小写。'wE aRe faMILY'-> We Are Family

var str='wE aRe faMILY';

//字符串转为数组

var arr=str.split(' ');

//遍历数组，获取每一个元素

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

//arr[i]代表每一个元素（单词）

//首字母转为大写，其余转为小写

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

first=first.toUpperCase(); //字首转大写

var last=arr[i].slice(1);

last=last.toLowerCase(); //后面转小写

console.log(first+last);

//把转换的结果替换当前的元素

arr[i]=first+last;

}

console.log(arr); //[ 'We', 'Are', 'Family' ]

//数组转换为字符串

var res=arr.join(' ');

console.log(res); //We Are Family

### 匹配模式

作用：用于查找、替换字符。

replace(value1,value2) 用于查找字符串value1，使用value2的值替换。value1可以使用字符串形式，也可以使用匹配形式/china/ig

i ->ignore 忽略大小写

g ->global 全局查找

match(value) 用于查找匹配的字符串，返回一个数组，里边可以使用i和g

search(value) 用于查找匹配的字符串，返回满足条件的第一个字符的下标，如果找不到返回-1，里面可以使用i。

例：var str='china是世界上人口最多的国家，China的面积很大有960万平方公里，CHINA的互联网高度发达，china有很多的程序员';

//替换china为中国

console.log(str.replace('china','中国'));

console.log(str.replace(/china/g,'中国')); //加上g后，表示全局查找，可以查找不只一个

console.log(str.replace(/china/i,'中国')); //加上i后，表示忽略大小写

console.log(str.replace(/china/gi,'中国')); //不分先后，相互匹配可全部替换

//match 查找满足条件的字符串——返回数组

//查找china出现的次数

var arr=str.match(/china/gi);

console.log(arr.length); //4

//查找满足条件的第一个字符的下标

console.log(str.search(/china/i));

var str2='tom@163.com';

var res=str2.search(/@/); //类似于indexOf

console.log(res); /3

### Math对象

Math对象不需要使用new来创建，直接使用

Math.PI 取圆周率

Math.abs() 取一个数字的绝对值

Math.round() 四舍五入取整

Math.ceil() 向上取整

Math.floor() 向下取整

Math.max() 取多个数字的最大值

Math.min() 取多个数字的最小值

Math.pow(x,y) 取x的y次幂

Math.random() 取随机数，范围0~1之间

例：

//获取圆周率

console.log(Math.PI); //3.141592653589793

//绝对值

console.log(Math.abs(-1)); //1

//取整

console.log(parseInt(9.9));

//四舍五入取整

console.log(Math.round(3.4999)); //3

//进一法取整（向上取整）

console.log(Math.ceil(17/8)); //3

//去一法取整（向下取整）

console.log(Math.floor(5.9)); //5

//比较多个数，获取最大值

congsole.log(Math.max(7,3,9)); //9

//比较多个数，获取最小值

console.log(Math.min(7,3,9)); //3

//次方根

console.log(Math.pow(8,2)); //64,次方根

//随机数

console.log(Math.rand()); //小于1大于0的数

练习：创建数组，保存10个人的姓名；每次随机获取一个人的姓名。（随机取0~9之间的一个整数）

//随机取4个,放入到数组中

var str=['a','b','c','d','e','f','g','h','i','j'];

var newArr=[];

for (var i=0;i<4 ;i++ ){

var num=Math.random()\*10;

num=Math.floor(num);

newArr.push(str[num]);

}

console.log(newArr);

练习：使用随机数，获取4个随机验证码；范围在a-z,0-9；要求返回数组

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

var newArr=[];

for (var i=0;i<4 ;i++ ){

var num=Math.random()\*36;

num=Math.floor(num);

newArr.push(str[num]);

}

console.log(newArr);

练习:模拟彩票摇奖，随机获取1~30之间的7个数字；要求返回数组

//有重复版

var str=['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30'];

var newArr=[];

for (var i=0;i<7 ;i++ ){

var num=Math.random()\*30;

num=Math.floor(num);

newArr.push(str[num]);

}

console.log(newArr);

//无重复版

var arr=[];

for (i=1;i<=30 ;i++ ){

arr.push(i);

}

//console.log(arr);

var newArr=[];

//从arr中随机取7个元素

//取完后，在arr中删除这个元素splice(strat.count)

for (var i=0;i<7 ;i++ ){

//获取随机数——随机获取的下标

//arr.length可取长度有变化的数组

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

num=Math.floor(num);

newArr.push(arr[num]);

console.log(arr[num]);

//取完后，删除这个元素

arr.splice(num,1);

}

console.log(newArr);

例：在a~i中加入0~9，并随机取4个

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

//0~9加入到数组

for (var i=0; i<10; i++){

arr.push(i);

}

//随机取4个字符

var newArr=[];

for(var i=0;i<4;i++){

//获取随机数\*数组的长度

var num=Math.random()\*arr.length;//0.\*~19.\*

//向下取整

num=Math.floor(num);

//console.log(arr[num]);

//获取的随机元素加入到新数组中

newArr.push(arr[num]);

}

console.log(newArr);

### Date对象

用于对日期和时间进行存储和计算。

#### 实例化Date对象

new Date(2018,8,18,10:30:20) //第2个参数月份0~11

new Date('2018-9-18 10:30:20')

如果没有设置时分秒，默认是8:0:0

new Date('2018/9/18 10:30:20')

如果没有设置时分秒，默认数0:0:0

new Date() 获取当前系统时间的对象

new Date(1000\*60\*60\*24) 获取距离计算机元年的毫秒数对应的对象

例：

//月份的存储范围是0~11;0->1月;11->12月

//时间显示的是格里尼治时间

var d1=new Date(2018,9,18,10,0,0);

var d2=new Date('2018-9-18 10:23:30'); //不能用‘\’，可以用‘-’、‘/’

var d3=new Date(); //获取当前系统时间

var d4=new Date(0); //1970-01-01T00:00:00.000Z;表示时间的起始

var d5=new Date(1000\*60\*60\*24); //是距离1970年1月1日的时间差

练习：创建2019-1-1 0:0:0的Date对象

var d1=new Date(2019,0,1 ,8,0,0);

console.log(d1);

var d2=new Date(2019,0,1 ,0,0,0);

console.log(d2.toLocaleString()); //2019-1-1 00:00:00;转为本地时区

console.log(d2.toLocaleDateString()); //2019-1-1

console.log(d2.toLocaleTimeString()); //00:00:00

#### 本地化时间格式

toLocaleString() 2018-9-18 10:45:20

toLocaleDateString() 2018-9-18

toLocaleTimeString() 10:45:20

#### 获取时间信息

getFullYear() 年

getMonth() 月(0~11)

getDate() 日

getHours() 小时

getMinutes() 分钟

getSeconds() 秒钟

getMilliseconds() 毫秒

getDay() 星期(0~6)

getTime() 距离计算机元年的毫秒数(1970年1月1日0:0:0:0)

例：

var d1=new Date('2018-9-18 10:30:50');

//获取具体的年月日...

var year=d1.getFullYear(); //2018

var month=d1.getMonth(); //0~11之间//8

var date=d1.getDate(); //18

var hour=d1.getHours(); //10

var minute=d1.getMinutes(); //30

var second=d1.getSeconds(); //50

var milli=d1.getMilliseconds(); //毫秒//0

var day=d1.getDay(); //2//周日~周六 -> 0~6

var time=d1.getTime(); //1537237850000//距离计算机元年1970年1月1日的时间，后三位为毫秒

练习：今天是2018年9月18日 11点20分30秒 星期二 AM

var d1=new Date('2018-9-18 11:20:30 ');

var year=d1.getFullYear();

var month=d1.getMonth()+1;

var date=d1.getDate();

var hour=d1.getHours();

var minute=d1.getMinutes();

var second=d1.getSeconds();

var day=d1.getDay();

var arr=['日','一','二','三','四','五','六'];

hour>12 ? str='PM' : str='AM';

console.log('今天是'+year+'年'+month+'月'+date+'日'+' '+hour+'点'+minute+'分'+second+'秒'+' '+str+' '+'星期'+arr[day]);

练习：计算当前时间距离2018年圣诞节还有多少天？

//圣诞节距离计算机元年毫秒 - 当前距离计算机元年毫秒数 = 两个时间相差的毫秒数

var target=new Date('2018/12/25');

var now=new Date();

//var d=target.getTime()-now.getTime();

//两个对象直接相减，返回时两个对象的相差的毫秒数

var d=target-now;

//把毫秒数转成秒数

d=Math.floor(d/1000);

//秒数转成天数

day=d/(60\*60\*24);

day=Math.floor(day);

console.log(day); //97天

//距离圣诞节还有97天\*小时\*分钟\*秒

var h=d%(60\*60\*24); //结果是秒

hour=h/(60\*60); //把秒转成小时

hour=Math.floor(hour);

var m=d%(60\*60); //结果是秒

var minute=m/60; //把秒转成分钟

minute=Math.floor(minute);

var second=m%60; //结果是秒钟

console.log('距离圣诞节还有'+day+'天'+hour+'小时'+minute+'分钟'+second+'秒');

#### 设置时间信息

setFullYear() 年

setMonth() 月

setDate() 日

setHours() 小时

setMinutes() 分钟

setSeconds() 秒

setMilliseconds() 毫秒

setTime() 设置距离计算机元年的毫秒数；一旦使用后，年月日时分秒都会受其影响。

例：设置Date中的属性

var d=new Date(2018,11,25,0,0,0);

var d1=new Date(d); //以实参的形式存到new Date中

//设置为2019年

d.setFullYear(2019);

d.setMonth(2);

d.setDate(40);

d.setHours(24);

console.log(d.toLocaleString()); //2019-4-10 00:00:00

console.log(d1.toLocaleString()); //2018-12-25 00:00:00

练习：创建2018-10-1对象，计算10天以后是周几

//创建一个新对象，不能在原来基础之上修改

var d1=new Date('2018-10-1');

var d2=new Date(d1); //2018-10-1

//1天以后，在当前的基础之上+10

var date2=d2.getDate()+10

d2.setDate(date2);

console.log(d1.toLocaleDateString()); //2018-10-1

console.log(d2.toLocaleDateString()); //2018-10-11

console.log(d2.getDay()); //4

例：

var d1=new Date('2018/9/18');

var d2=new Date(d1);

d2.setFullYear(2019);

d2.setTime(1000); //会覆盖所有其他时间设置方式

console.log(d1.toLocaleString()); //2018-9-18 00:00:00

console.log(d2.toLocaleString()); //1970-1-1 08:00:01

### Number对象

创建Number对象

new Number（值）

创建一个数字对象，并且把值转为数值型。

Number.MIN\_VALUE 获取计算机能存储的最小值

Number.MAX\_VALUE 获取计算机能存储的最大值

toString() 数字转为字符串

toFixed(n) 获取小数点后n位

例：

var num1=1;

var num2=new Number(2);

console.log(num2); //[Number: 2]

console.log(typeof(num2)); //object

var num3=Number('15.2a');

console.log(num3); //NaN

console.log(typeof(num3)); //number

var num4=new Number(5);

console.log(num4===5); //false

var num5=6;

//数字转为字符串

var res=num5.toString();

console.log(typeof(res)); //string

var res2=0.1+0.2;

//获取小数点后两位

console.log(res2.toFixed(2)); //0.30 //value表小数点后几位

//计算机存储的最大/最小数值

console.log(Number.MAX\_VALUE); //1.7976931348623157e+308

console.log(Number.MIN\_VALUE); //5e-324

### Boolean对象

new Boolean(值) 本质上就是将值转为布尔型（强制转为布尔型）

Boolean(值) （强制转为布尔型）

！！值 隐式转为布尔型

例：

//0 NaN '' undefined null,以上出false

var b1=true;

var b2=new Boolean(1);

var b3=Boolean(1); //强制转为布尔型

var b4=!!null;

console.log(b4); //false

console.log(typeof(b4)); //boolean

练习：查看空数组和空对象的布尔型结果。

var b5=Boolean('');

console.log(b5); //false

console.log(typeof(b5)); //boolean

var b6=Boolean([]);

console.log(b6); //true

console.log(typeof(b6)); //boolean

## 错误处理

SyntaxError 语法错误，错误的使用了中文、缺少括号等。

ReferenceError 引用错误，错误的使用了未声明的变量

TypeError 类型错误，错误的使用了括号

RangeError 范围错误

引用错误、类型错误、范围错误会影响后续代码的执行。

语法错误会影响所有代码的执行。

例：

console.log(1);

var a=1； //将; -> ；后，回报SyntaxError，语法错误

console.log(a); //ReferenceError,引用错误

var a=1;

a(); //TypeError,类型错误，错误的使用了类型

console.log(2);

## 异常处理

try{

尝试执行的代码，可能出现错误（异常）

}catch ( err ) {

err 捕获的错误信息

处理错误的方案

}

例：

var a=7;

console.log(3);

//如果遇到可能出现错误的代码，尝试执行

try{

//尝试执行

console.log(a);

}catch (err){ //err：若果有错误，收集到错误

//console.log(err); //开发过程中使用

//处理方案

console.log(5); //当有错误是，返回5；无错误时，不返回5

}

console.log(4);

## ES6新特征

### 块级作用域

使用let声明的变量，只能在块级作用域下访问，不能被外部访问。

块级作用域：{ }、for、while、do-while等都是块级作用域。

例：

var i=0;

while (i<5){

let a=2;

console.log(a); //返回5个2

i++;

}

console.log(a); //报错

### 箭头函数

箭头函数是回调函数的另一种写法，但不完全一致。

sort((a,b)=>{

return a-b;

})

如果箭头函数中，只有一行代码，并且含有return；

可以简写为 sort((a,b)=>a-b)

例：箭头函数

var arr=[4,18,29,8];

console.log(arr.sort(function(){

return a-b;

}));

//简化

console.log(arr.sort((a,b)=>{ //[ 29, 18, 8, 4 ]

return b-a;

}));

//再简化

console.log(arr.sort((a,b)=>a-b)); //[ 4, 8, 18, 29 ]

### 函数中的参数

ES6中允许为形参设置默认值，如果没有实参传递，自动使用形参的默认值。

例：

function fn(a,b,c=0){

console.log(a+b+c);

}

fn(2,3); //5

fn(3,8,9); //20

### 模板字符串

`之间可以写任何形式的代码 ${变量}`

模板字符串之间可以放回车、引号、分号等...

例：2018年9月18日 17：56

var year=2018;

var month=9;

var date=17;

var hour=17;

var minute=56;

var arr=['星期一','星期二','星期三'];

//模板字符串

console.log(`

${year}年${month}月${date}日 ${hour}：${minute}

${arr[2]}

`);

练习：计算员工的入职信息

员工的入职时间2015/6/12

合同有效期3年，求合同的到期时间。

合同到期前一个月，需要续签合同，如果遇到周末，提前到上一个周五，求续签的时间

var start=new Date('2015/6/12');

var end=new Date(start);

console.log('入职时间为：'+start.toLocaleDateString());

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

console.log('合同到期时间为：'+end.toLocaleDateString());

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

var i=end.getDay();

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

if (i==0){

console.log('原合同续签时间为：'+end.toLocaleDateString()+'为周'+week[i]);

end.setDate(end.getDate()-2);

console.log('合同续签时间为：'+end.toLocaleDateString());

}else if (i==6){

console.log(`原合同续签时间为：${end.toLocaleDateString()}为周${week[i]}`);

end.setDate(end.getDate()-1);

console.log('合同续签时间为：'+end.toLocaleDateString());

}

例：不同解释器不同效果会有所区别

var d1=new Date('2018-10-01');

var d2=new Date('2018/10/01'); //推荐使用该形式，并完整，避免不同解释器造成的偏差

var d3=new Date(2018,9,1,8,0,0);

console.log(d1); //2018-10-01T00:00:00.000Z

console.log(d2); //2018-09-30T16:00:00.000Z

console.log(d3); //2018-10-01T00:00:00.000Z

console.log(d1.toLocaleString()); //2018-10-1 08:00:00

console.log(d2.toLocaleString()); //2018-10-1 00:00:00

console.log(d3.toLocaleString()); //2018-10-1 08:00:00

# NodeJavaScript

## nodeJS概述

### NODEJS运行

#### 交互模式 node

exit 退出

Ctrl+C 两次 退出

#### 脚本模式

cd + C:/xampp..../nodejs/day01

node 01.js

shift + 桌面空白区域鼠标右键，选择Powershell窗口

### Node.js的特点

#### 优势

1. 简单，避免过度设计
2. 单线程逻辑处理
3. 非阻塞的异步I/O处理
4. 事件驱动编程
5. 无锁机制，不会产生死锁
6. 支持数万个并发连接
7. 适合搭建以IO操作为主，响应速度快，易于扩展的网络应用
8. 基于社交网络的大规模Web应用

#### 劣势

1. 不适合CPU密集型应用
2. 深层次的嵌套和递归
3. 复杂加密和解密算法
4. 高可靠性运算
5. 严格内存管理
6. 数据挖掘和数据分析

总结：

NodeJS是服务器端语言，和Java，PHP，.net性质一样。

## Node.js全局对象

全局作用域下的变量，可以通过全局对象来访问。

global是NODEJS下的全局对象

NODEJS每一个js文件都属于局部作用域，里边声明的变量属于局部变量，不能使用global来访问。

window是JS下的全局对象

JS下每一个js文件下都属于全局作用域，里边声明的变量属于全局变量，可以使用window来访问。

例：在浏览器中打开

var a=1;

//使用js的全局对象window来访问

console.log(a); //1

console.log(window.a); //1

例：在node中打开

var a=1;

console.log(a); //1

//a不是全局作用域下的变量，不能使用global访问

console.log(global.a); //undefined//global,全局对象

### 全局对象

#### console对象

console.log() 打印消息

console.info() 打印消息

console.warn() 打印警告消息

console.error() 打印错误消息

例：//在node里面没区别，在浏览器中有明显区别

console.log(123); //123//log打印

console.info(123); //123//info信息

console.warn(123); //123//warn警告

console.error(123); //123//error错误

#### 计算程序的运行时间

console.time('**loop**'); //开始计时

检测的程序

console.timeEnd('**loop**');

例：

console.time('for循环运行时间');

var sum=0;

for(var i=0;i<10;i++){

sum+=i;

}

console.log(sum);

console.timeEnd('for循环运行时间'); //可显示程序运行时间，括号内内容需完全相同

练习：比较while循环和do-while执行10万次，哪个时间更短

console.time('while时间');

var i=0;

var sum=0;

while (i<100000){

sum+=i;

i++;

}

console.log(i) ;

console.timeEnd('while时间');

console.time('do-while时间');

var i=0;

var sum=0;

do{

sum+=i;

i++;

}

while (i<100000);

console.log(i) ;

console.timeEnd('do-while时间');

#### process对象（了解）

arch 查看CPU的架构类型

platform 查看操作系统类型

version 查看当前nodejs的版本号

env 查看当前计算机的环境变量有哪些

例：

process.arch //cup架构

'x64'

process.platform //系统

'win32'

process.version //node版本

'v8.11.1'

process.env //环境变量

#### Buffer对象

Buffer:缓冲区,值一块专用于存储数据的内存区域,可用于存储读写的文件数据、网络上要传输的数据等等。

Buffer对象的实例，可直接构造，也可以通过数据读写获得。

Buffer实例中，不仅能存储字符数据，也可以存储二进制的字节数据。

注意：不同版本的Node.js中，Buffer对象的API变化较大，请参考当前版本的Node.js手册。

### 全局函数

定时器

#### 一次性定时器

开启

var time=null;

timer=setTimeout(参数1,参数2);

参数1:回调函数，间隔的时间到了，执行的命令。

参数2:间隔的时间，单位毫秒

清除

clearTimeout(timer);

例：

//开启一次性定时器

var timer=null;

timer=setTimeout(function(){

console.log('嘀嘀嘀');

},3000);

//清除一次性定时器

clearTimeout(timer);

#### 周期性定时器

使用方式和一次性定时器完全一致。

setInterval()/clearInterval()

练习：查看周期性定时器函数的使用

var timer=null;

timer=setInterval(function(){

console.log('lalala');

},2000);

clearInterval(timer);

#### process.nextTick(回调函数)

例：

process.nextTick(function(){

console.log('滴答');

});

#### 开启

var timer=null;

setImmdiate(回调函数)

清除 clearImmdiate(timer);

例：

var timer=null;

timer=setImmediate(()=>{

console.log('hello');

});

clearImmediate(timer);

#### 同步和异步

同步：程序按照顺序执行，后边的程序始终等待前边的程序执行完，才会执行。——阻塞

异步：程序运行过程中，遇到一些耗时较久的程序，把这些程序放到整个程序的最后。——非阻塞

回调函数多为异步

parseInt/ parseFloat/ eval/ isNaN/ isFinite/ encodeURI...

例：打印结果：1,2,123；与时间设定无关,是因为非阻塞

console.log(1);

setTimeout(()=>{

console.log(123);

},1000);

console.log(2);

例：打印结果：1,2,4

console.log(1);

process.nextTick(()=>{

console.log(4);

});

console.log(2);

例：打印结果1,2,3

console.log(1);

setImmediate(()=>{

console.log(3);

});

console.log(2);

## 模块

### 模块的分类

nodejs下分为核心模块（nodejs提供）、第三方模块（个人组织写好的）、自定义模块（自己写的）。

### 模块封装

在nodejs下每一个模块文件的本质——是被一个构造函数所包含的

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

程序员写的代码

})

红色内容是nodejs自动为每一个文件添加的

### 相关函数

\_\_filename: 当前模块的完整路径和文件名

\_\_dirname: 当前模块的完整路径

require: 用于引入一个模块，返回一个对象

module: 指代当前的模块；

module.exports 当前模块导出的对象，可以供其他的模块使用。

exports: 等同于module.exports

|  |  |  |
| --- | --- | --- |
|  | 以路径开头 | 不以路径开头 |
| 文件模块 | require('./circle.js')  通常用于用户自定义的模块 | require('querystring')  用于引入官方提供的核心模块 |
| 目录模块 | require(./02\_1);  在目录下存储一个index.js文件，或者使用package.json文件来指定引入的文件 | require('2');  要求把目录2放入当前目录下的node\_modules中；  通常用于引入第三方模块 |

例：var res=require('./06\_1.js'); //同级一定要加./，否则会报错

例：

console.log(\_\_filename);//文件的完整路径和文件的名称//C:\xampp\htdocs\sxs\03\_NODEJS\day01\06\06\_1.js

console.log(\_\_dirname); //文件的完整路径//C:\xampp\htdocs\sxs\03\_NODEJS\day01\06

例：

console.log(module.exports==exports);

module.exports={

a:1,

addNum:function(num1,num2){

return num1+num2;

}

}

例：//核心模块

console.log('这是主模块');

//导入功能模块

//var res=require('./06\_1.js');

//console.log(res.a);

//console.log(res.addNum(3,5));

//需要计算长方形的周长和面积

var rect=require('./06\_2.js');

console.log(rect.getLength(3,2));

console.log(rect.getArea(3,2));

//功能模块06\_1

console.log('这是功能模块1');

var a=1;

console.log(module.exports==exports);

module.exports={

a:1,

addNum:function(num1,num2){

return num1+num2;

}

}

//功能模块06\_2

//公开计算长方形的周长和面积两个方法

module.exports={

getLength:function (w,h){

return 2\*(w+h);

},

getArea:function(w,h){

return w\*h;

}

}

//导入主模块，可得

练习:创建主模块，创建一个功能模块，用于计算圆的周长和面积。

//主模块

console.log('这是主模块');

var rect=require('./01\_1.js');

console.log(rect.getLength(2));

console.log(rect.getArea(2));

//功能模块

//创建一个功能模块，用于计算圆的周长和面积。

module.exports={

getLength: function(r){

return 3.14\*2\*r;

},

getArea: function(r){

return 3.14\*Math.pow(r,2);

}

}

练习：创建一个03\_1.js，分别引入03\_2目录模块（有index.js），引入03\_3目录模块（有rect.js）

//创建03\_1.js文件

require('./03\_2');

require('./03\_3');

//在同文件内建立03\_2和03\_3文件夹

//在03\_2文件夹内建立index.js

console.log('./03\_2/index.js');

//在03\_3文件夹内建立rect.js和package.json文件

//在rect.js文件内

console.log('./03\_3/rect.js');

//在package.json文件内

{

"main":"rect.js"

}

//导入03\_1.js文件可得

注：.jiso 文件要求属性名必须加双引号

例：//带路径的模块

//建立main.js文件

require('2');

//在同文件内建立node\_modules文件夹

//在node\_modules文件夹内建立2文件夹

//在2文件夹内建立index.js文件

console.log('./node\_modules/2/index.js');

//导入main.js文件可得

练习：引入目录模块circle，里边文件名称circle\_1.js

//不带路径的模块

//建立新文件夹

//建立main.js文件

require('circle');

//在同文件内建立node\_modules文件夹

//在node\_modules文件夹内建立circle文件夹

//在circle文件夹内分别建立circle\_1.js和package.json文件

//在circle\_1.js文件内

console.log('circle\_1.js');//输出文件

//在package.json文件内

{

"main":"circle\_1.js"//关联circle\_1.js文件

}

//导入main.js文件可得

### 包和npm

（1）包：就是一个目录模块，里边含有多个文件，其中有一个命名为package.json的包说明文件。

（2）npm -> node package manage

cd切换到要下载的目录

npm install 包的名称

### 工具模块

#### querysting模块

查询字符串模块

'lid=20&name=tom'

parse 将查询字符串转为对象（服务器端解析传递的数组）

stringify 将对象转为查询字符串（浏览器发送数据）

例：

//nodejs提供一个模块

const querystring=require('querystring'); //一般核心模块就作为常量名称

var str='lid=5&name=tom&age=18';

//把查询字符串解析为对象

var obj=querystring.parse(str);

console.log(obj); //{ lid: '5', name: 'tom', age: '18' }

//把对象转换为查询字符串

var product={

pid:25,

name:'dell',

price:4000

}

var str=querystring.stringify(product);

console.log(str)

//pid=25&name=dell&price=4000;

练习：把百度网址的查询字符串解析为对象；

ie=utf-8&f=8&rsv\_bp=0&rev\_idx=1&tn=baidu&wd=手机

var baiduStr='ie=utf-8&f=8&rsv\_bp=0&rev\_idx=1&tn=baidu&wd=手机';

console.log(querystring.parse(baiduStr));

#### URL模块

parse 将路径解析为对象

protocol 协议

hostname 主机名（域名）

port 端口

pathname 文件在服务器的路径

query 查询字符串

format 将对象转换成路径

注意：query对应的是一个对象

一个网址包含：协议，域名，端口，路径，查询对象，这5个部分

例：//引入url模块

const url=require('url');

var str='http://www.codeboy.com:80/course/1.html?lid=10&name=tom&age=18';

//使用URL模块下的方法解析路径

var obj=url.parse(str);

console.log(obj);

/\*

Url {

protocol: 'http:',

slashes: true,

auth: null,

host: 'www.codeboy.com:80',

port: '80',

hostname: 'www.codeboy.com',

hash: null,

search: '?lid=10&name=tom&age=18',

query: 'lid=10&name=tom&age=18',

pathname: '/course/1.html',

path: '/course/1.html?lid=10&name=tom&age=18',

href: 'http://www.codeboy.com:80/course/1.html?lid=10&name=tom&age=18' }

\*/

console.log(obj.query); //lid=10&name=tom&age=18

练习：将一个路径中查询字符串解析为对象

//http://www.tmooc.cn:8080/images/list.html>pid=10&pname=dell&price=5000

//用url模块解析出查询字符串

var obj=url.parse(str);

//通过url对象获取查询字符串

var qs=obj.query;

//通过querystring解析查询字符串

var obj2=querystring.parse(qs);

console.log(obj2);

例：将一个对象再转换为路径

const url=require('url');

var obj2={

protocol:'http',

hostname:'www.baidu.com',

port:3000,

pathname:'/products/list.html',

//search: 'lid=5&name=tom'

query:{lid:5,name:'tom'}

}

var res=url.format(obj2);

console.log(res);

例：

引入url模块

const url=require('url');

const querystring=require('querystring');

var str='http://www.tmooc.cn:8080/images/list.html>pid=10&pname=dell&price=5000';

//用url模块解析出查询结果

var obj=url.parse(str);

//通过url对象获取查询字符串

var qs=obj.query;

console.log(qs);

var obj2=querystring.parse(qs);

console.log(obj2);

练习：如何使用对象转成路径

//http://www.tmooc.cn:8080/images/list.html?pid=10&pname=dell&price=5000

const url=require('url');

var obj={

protocol:'http',

hostname:'www.tmooc.cn',

port:8080,

pathname:'/images/list.html',

query:{pid:10,pname:'dell',price:5000}

}

var str=url.format(obj);

console.log(str);

#### Buffer对象

buffer 缓冲器

创建buffer

Buffer.alloc(size,fill)

size，buffer空间的大小；fill，填充的内容。

buffer数据转为普通字符

toString()

例：//创建一个缓冲区

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

console.log(buf); //<Buffer 61 62 63 61 62>

//将buffer数据转为普通字符

console.log(buf.toString()); //abcac

练习：创建一个buffer空间，存储'hello world';分别打印buffer数据和普通字符形式。

var buf=Buffer.alloc(11,'hello world');

console.log(buf); //<Buffer 68 65 6c 6c 6f 20 77 6f 72 6c 64>

console.log(buf.toString()); //hello world

#### 文件系统模块

fs.stat (path,callback) 查看文件的状态

path 文件的路径；

callback 回调函数；

fs.startSync(path)

path 文件路径

sync表同步

##### 对比同步方法和异步方法的区别

同步方法会阻止后续的执行，是通过返回值来得到结果。

异步方法不会阻止后续的执行，是通过回调函数来得到结果

例：1. 查看文件的状态

//引入文件系统模块

const fs=require('fs');

//查看文件的状态

法1

fs.stat('05.js',function(err,stats){

//err，如果出现错误，会存到这个里边

if (err){

//抛出错误，一旦有错误，后续的代码将无法再执行

throw err;

}

console.log('执行成功');

//stats，文件的状态信息

//如何查看是否为 文件

console.log(stats.isFile());

//如何查看是否为 目录

console.log(stats.isDirectory());

});

console.log(2);

//为异步函数

法2

同步方法：是通过返回值来获取结果

const fs=require('fs');

var stats=fs.statSync('05.js'); //同步函数的方法是把变量放到返回值

console.log(stats.isFile());

console.log(stats.isDirectory());

console.log(2);

#### FS模块

fs模块，使用前需要引入fs模块（var fs= require(“fs”);）

fs.kldir 创建目录

fs.rmdir 删除目录

fs.readdir 读取目录，读取的数据是数组的形式

fs.readFile 读取文件，读取的文件类型是buffer形式数据

fs.writeFile(path,data,callbank) 写入文件，如果文件不存在，则会创建；如果文件已存在，清空里边的内容；然后写入；

data 是要写入的数据

fs.appendFile(path,data,callback) 写入文件，如果文件不存在，则会创建；如果文件已经存在，在文件的末尾写入数据。

data 是要写入的数据

fs.unlink 删除文件

fs.exists**Sync** 判断文件是否存在；存在true，不存在false

fs.rename(oldPath,newPath,[callback(err)]) 修改文件名称，可更改文件的存放路径

oldPath 原路径

newPath 新路径

callback 回调，传递一个err异常参数

例：2.创建目录

法1

const fs=require('fs');

fs.mkdir('files',function(err){

if (err){

throw err;

}

console.log('目录创建成功');

}); //创建名为'files'的文件夹

法2

const fs=require('fs');

var res=fs.mkdir('files2');

console.log(res);

例：3.删除目录

const fs=require('fs');

fs.rmdir('files',function(err){

if (err) throw err;

console.log('删除成功');

});

例：4.读取目录

const fs=require('fs');

fs.readdir('03\_3',function(err,files){

if(err) throw err;

console.log(files); //为对象形式

}); //[ 'package.json', 'rect.js' ]

例：5.读取文件

const fs=require('fs');

fs.readFile('05.js',function(err,data){

if (err) throw err;

console.log(data);

});

例：6.写入文件——清空写入

const fs=require('fs');

fs.writeFile('num.txt','hello',function(err){

if(err) throw err;

console.log('写入成功');

});

例：7.写入文件——追加写入

const fs=require('fs');

fs.appendFile('num.txt','world\n',function(err){

if(err) throw err

console.log('ok');

}); //不能用记事本打开，用编辑器打开才有换行的效果

例：8.删除文件

const fs=require('fs');

fs.unlink('num.txt',function(err){

if (err) throw err;

console.log('ok');

});

例：9.判断文件是否存在————同步方法

const fs=require('fs');

var res=fs.existsSync('num.txt');

console.log(res);

例:10.删除文件————判断文件num.txt是否存在，如果存在就删除，否则打印“此文件不存在”。

const fs=require('fs');

if (fs.existsSync('num.txt')){

fs.unlink('num.txt',function(err){

if(err) throw err;

console.log('删除成功');

});

}else{

console.log('此文件不存在');

}

练习：计数器：把数字保存在num.txt中，每次运行读取文件中的数字。

1. 判断文件是否存在如果不存在创建num.txt，初始值为0；
2. 读取num.txt中的内容，将数字加1（num++），并打印出来
3. 把加1的结果再次写入到num.txt中

const fs=require('fs');

if (fs.existsSync('num.txt')){

fs.readFile('num.txt',function(err,file){

if(err) throw err;

fs.writeFile('num.txt',parseInt(file.toString())+1,function(err){

if(err) throw err;

});

console.log(Number(file.toString('utf8'))+1);

});

}

else{

fs.writeFile('num.txt','0',function(err){

if(err) throw err;

console.log('写入成功');

});

}

官方答案

const fs=require('fs');

//计数器

//1.初始化保存数字的文件——0

//用到同步

//判断是否有num.txt，如果没有则创建文件，初始值0

var res=fs.existsSync('num.txt');

if (!res){ //!false

fs.writeFileSync('num.txt',0); //必须用同步方法

}

//2.读取文件中的数字，打印出来

//把数字加1，再次保存到文件中

fs.readFile('num.txt',function(err,data){

var num=data.toString();

num=parseInt(num);

num++;

console.log(num);

//把num写入到文件中，先清除之前的内容再写入

fs.writeFile('num.txt',num,function(err,data){

if (err) throw err;

});

});

### HTTP

#### HTTP协议

HTTPS协议是在HTTP协议的基础上就安全进行了升级。

HTTP：用于浏览器和Web服务器之间的通信协议。翻译为，超文本传输协议。

##### 通用头部General

Request URL 请求的URL

Request Method 请求的方法

get 用于从服务器端获取内容（浏览网页...）

post 向服务器端发送内容（注册、登录...）

Status Code 响应的状态码

1\*\*系列 请求-响应继续进行

2\*\*系列 成功的响应

3\*\*系列 响应的重定向

4\*\*系列 客户端错误

5\*\*系列 服务器端错误

Remote Address 请求的远程地址和端口

##### 请求的头信息Request headers

Accept 接受的文件类型

Connection：keep-alive 告诉浏览器持久连接

User-agent 使用的浏览器类型

Accept-Language 浏览器所接受的语言

##### 响应的头信息Reponse headers

Content-Type 响应主体类型是什么

text/html、text/css...

Connection：keep-alive 开启持久连接

#### HTTP模块

既可以模拟浏览器向服务器发请求，也可以创建web服务器。

##### 作为客户端浏览器

http.get(url,callback)

get 以get方法发请求

url 请求的url

callback 以回调函数形式获取服务器的响应

res.statusCode 获取相应的状态码

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

使用事件来获取服务端响应的数据

数据是分段传输，格式为buffer数据

例：

//1.使用http模块，模拟浏览器

const http=require ('http');

//使用get方法向服务器发请求

http.get('http://www.codebody.com',function(res){

//res 服务器端做出的响应————对象

console.log(res.statusCode); //statusCode，响应的状态码

//获取响应的内容

//通过一个事件

//on('data')//当有数据传输时 //on，表示当有事件时

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

console.log(buf.toString());

});

});

练习：使用http模块模拟浏览器向以下url发请求，并打印响应的内容。

http://www.codeboy.com/products.html

const http=require('http');

http.get('http://www.codeboy.com/products.html',function(res){

console.log(res.statusCode);

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

console.log(buf.toString());

});

});

##### 创建web服务器

createServer 创建web服务器

listen 分配端口，监听端口的变化

server.on('request',function(req,res){ })

当有浏览器发起请求，调用事件

req 请求的对象

method 请求的方法 get/psot

url 请求URL，显示端口后边的路径

http://127.0.0.1/login

res 响应的对象

writeHead(statusCode,对象) 设置响应的头信息

对象中Location实现跳转

write（文本） 响应文本内容（显示在浏览器页面上）

end() 结束响应

练习：使用http创建web服务器，端口是3001；使用事件获取客户端的请求

const http=require('http');

var server=http.createServer();

server.listen(3000,function(){

console.log('服务器创建成功');

});

//接收请求

server.on('request',function(req,res){

console.log('收到1个请求');

//响应的对象——res

/\*

res.write('this is web 1808');

res.write('hello world');

\*/

//重定向到其他的URL

//跳转到学子商城的首页

res.writeHead(302,{

Location:

'http://www.codeboy.com',

'Content-Type':'text/html;charset=utf-8'

});

//响应结束

res.end();

});

练习：使用http创建web服务器，根据浏览器请求的URL执行不同的操作。

/login 响应一行文本 'this is login page' 到浏览器

/reg 响应一行文本 'this is reg page' 到浏览器

/home 转到学子商城首页，http://www.codeboy.com

const http=require('http');

//创建web服务器

var server=http.createServer();

server.listen(3000);

//接收请求

server.on('request',function(req,res){

//console.log('收到请求');

//获取请求的URL

var url=req.url;

console.log(url);

//根据url的不同，来响应不同的内容——路由

switch (url){

case '/':

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

break;

case '/login':

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

break;

case '/reg':

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

break;

case '/home':

res.writeHead(302,{

Location:'http://www.codeboy.com'

});

break;

default:

res.write('404 NOT FOUND');

}

//结束响应

res.end();

});

练习：在node中创建服务器

之后在浏览器内输入127.0.0.1:3000

输入127.0.0.1:3000/ //this is home page

输入127.0.0.1:3000/login //this is login page

输入127.0.0.1:3000/reg //this is reg page

输入127.0.0.1:3000/home //跳转到http://www.codeboy.com（学子商城）

输入127.0.0.1:3000/\* //404 NOT FOUND

//node会显示favicon.ico，其表示默认搜索小图标

const http=require('http');

var server=http.createServer();

server.listen(3000);

server.on('request',function(req,res){

var url=req.url;

console.log(url);

switch(url){

case '/':

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

break;

case '/login':

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

break;

case '/reg':

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

break;

case '/home':

res.writeHead(302,{

Location:'http://www.codeboy.com'

});

break;

default:

res.write('404 NOT FOUND');

}

res.end();

});

#### Express

##### Express概述

Express是基于nodejs，用于构建web服务器的框架。

官网：www.expressjs.com.cn

注：包，如果下载时不经过特殊处理，若移动位置，会造成包无法使用

##### 使用Express创建服务器

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('服务器创建成功');

});

##### 路由

客户端发来请求，服务器程序会根据请求的URL和请求的方法（Method）来确定响应的内容。

路由：请求的URL，请求的方法，响应的内容

（浏览器端进行请求；服务器端负责响应）

###### 请求的对象req

req.method 请求的方法 get/post...

req.url 请求的URL /login...

req.headers 请求的头信息

JSON.stringify() 对象转换为字符串

JSON.parse() 将对象字符串解析为对象

req.query 请求的查询字符串，格式为对象

例：

//使用express创建web服务器

const express=require('express');

var app=express(); //创建

app.listen(3000,function(){

console.log('服务器创建成功');

});

//使用get方法，请求url为/login

//只有两个条件都满足，才会做出以下的响应

app.get('/login',function(req,res){

//app:代表服务器端

//get：代表请求法方法

//login：代表请求的url

console.log('接收到了请求为get，url为/login');

});

练习：创建请求方法为get，url为/reg的路由

创建请求方法为get，url为/home的路由

创建请求方法为post，url为/buy的路由

const express=require('express');

var app=express();

app.listen('3000',function(){

console.log('服务器创建成功');

});

app.get('/reg',function(req,res){

console.log('接收到了get，url为/reg');

});

app.get('/home',function(req,res){

console.log('接收到了get，url为/home');

});

app. post ('/buy',function(req,res){

console.log('接收到了get，url为/bug');

});

例：

//创建服务器

const express=require('express');

var app=express();

app.listen(3000);

app.get('/login',function(req,res){

//console.log('这是login');

//请求的方法，请求的url

console.log('请求的方法：'+req.method);

console.log('请求的URL：'+req.url);

//console.log('请求的头部信息'+JSON.stringify(req.headers));

console.log('请求中的查询字符串：'+JSON.stringify(req.query));

});

//JSON.stringify 将json对象转为字符串

//JSON.parse 将json字符串解析为对象

app.get('/list',function(req,res){

//响应一行文本

//不允许出现多个send

// res.send('这是商品的列表页面');//发送一行文本

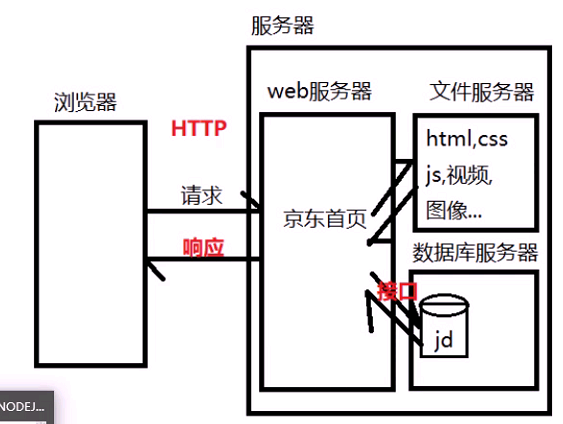
//响应文件

//当前文件的完整目录 \_\_dirname

//console.log(\_\_dirname);

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

});



练习：使用express创建web服务器，创建以下路由

get /index 发送文本 'this is index page'

get /shopping 发送文件

post /login 发送文本 '登录成功'

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

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

res.send('this is index page');

});

app.get('/shopping',function(req,res){

res.sendFile(\_\_dirname+'/01\_homework.html')

});

app.get('/login',function(req,res){

res.send('登陆成功');

});

###### 响应的对象res

res.send 响应文本；注意：只能用有一个send

res.sendFie 响应文件；注意：使用绝对路径\_\_dirname

res.send() 响应文本 里边的参数若为数字，表示状态码；若状态码不存在，则报错

res.redirect(url) 响应的重定向 url可以跳转到其他的地址

##### 获取浏览器端请求的数据

###### POST请求方法的数据

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

//获取的数据是bufer形式

buf.toString()

//转换后的数据是查询字符串，需要借助查询字符串模块，解析为对象

});

###### GET请求方法的数据

以查询字符串的形式发送服务器

req.query //结果自动将查询字符串转为对象

###### 对比POST和GET方法

POST请求安全程度相对比较高，但速度较慢，常用于注册、登录等；

GET请求安全程度相对较低，但速度较快，常用于搜索、检索等。

例：//建立02\_post.js

//使用express创建web服务器

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

})

//创建路由

//请求方法:get;url:/login;

//响应内容:login.html

app.get('/login',function(req,res){

res.sendFile(\_\_dirname+'/02\_login.html');

});

//因为浏览器发送了请求方法为post，

//请求URL为/mylogin

app.post('/mylogin',function(req,res){

res.send('登陆成功');

});

//建立02\_login.html

<h2>登录页面</h2>

<!--

在form表单中，设置请求的方法和请求的url

method：请求的方法

action：请求的url

-->

<form method="post" action="/mylogin">

用户：<input type="text" name="uname"><br><!--uname可以自定义命名-->

密码：<input type="text" name="upwd"><br>

<input type="submit">

</form>

练习：

//路由：请求的方法：get；请求的url：/reg

//请求的内容：/reg.html

//在reg.html中发送post的请求（用户、密码、邮箱、手机），请求的URL：/nyreg

//要求点击按钮，响应的内容是注册成功

//在02\_post.js中

const querystring=require('querystring');

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

app.get('/reg',function(req,res){

res.sendFile(\_\_dirname+'/02\_1\_reg.html');

});

app.post('/myreg',function(req,res){

//获取浏览器请求的数据

//事件:当有数据从浏览器传输到服务器的时候会触发事件

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

var body=buf.toString();

//转化后的数据是查询字符串形式

var obj=querystring.parse(body);

console.log(obj);

});

res.send('注册成功');

});

//建立02\_1\_reg.html

<h2>注册页面</h2>

<form method="post" action="/myreg">

用户：<input type="text" name="uname"><br>

密码：<input type="text" name="upwd"><br>

邮箱：<input type="text" name="email"><br>

手机：<input type="text" name="phone"><br>

<input type="submit">

</form>

例：

//在02\_post.js中，增加

//路由：请求方法get，请求url：/myreg2

app.get('/myreg2',function(req,res){

//获取请求的查询字符串，并自动解析为对象

console.log(req.query);

res.send('使用get注册成功');

}); //数据会记录在缓存中，可在安全要求不高的地方使用

##### 路由传递参数

###### 配置路由

app.get('/detail/**:lid**',function(){

//lid是用于接收的参数名称

req.params //获取传递参数，是对象形式的数据

});

###### 请求URL

http://127.0.0.1/detail/30

30表示传递的参数

例：

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

//路由

//接受浏览器请求的参数

app.get('/detail/:lid/:pname',function(req,res){

//把所有要传递的参数都放入到req.params下

console.log(req.params);

res.send('这是详情');

});

练习：创建路由，请求的方法：get，请求的URL：/book,传递图书的价格和图书的名称，把两个值响应在浏览器中。

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

app.get('/book/:price/:bname',function(req,res){

res.send(`

图书的价格为：${req.params.price}

<br>

图书的名称为：${req.params.bname}

`);

});

##### 路由器

把同一模块下的路由归纳到一起进行管理。

###### 使用步骤

1. 创建路由器模块（js文件）
2. 引入express模块
3. 调用express下的路由器功能，创建路由器
4. 在路由器下添加所有的路由
5. 导出路由器
6. 在服务器模块中，引入路由器模块
7. 把路由器挂载到具体某个URL

**app.use('/product',productRouter)**

访问形式 /product/delete

例:

//创建文件app.js作为服务器模块

//创建服务器

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

//引入创建好的路由器

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

//路由器只能用于web服务器

//把导入的路由放到(挂载)/product下

//最终的访问形式 /product/list product/delete

app.use('/product',productRouter);

//创建routes文件夹作为所有路由器的集合

//创建product.js文件作为产品路由器模块

//商品路由器——所有商品模块下的路由

//属于express下的一个功能

const express=require('express');

//使用路由器功能——默认是一个空路由

var router=express.Router();

//在路由器中添加路由

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

res.send('这是商品列表');

});

router.get('/delete',function(req,res){

res.send('这是删除商品');

});

//创建好的路由器公开出去——让服务器调用

//导出

module.exports=router;

//在node中导入app.js文件,启动服务器;在在浏览器中输入127.0.0.1:3000/product/list和127.0.0.1:3000/product/delete可分别显示

练习:创建用户模块路由器(user.js)

//在app.js文件输入以下内容

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

//把导入的用户路由器挂在到user下

// /user/list /user/update

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

//在routes文件夹中建立user.js文件作为用户路由器

const express=require('express');

//创建了一个空路由器

var router=express.Router();

//往空路由器中添加路由

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

res.send('这是用户列表页面');

});

router.get('/update',function(req,res){

res.send('这是用户修改的页面');

});

//导出路由器router

module.exports=router;

//在node中导入app.js文件,启动服务器;在在浏览器中输入127.0.0.1:3000/user/list和127.0.0.1:3000/user/update可分别显示

##### 引入第三方模块

require('express');

首先在当前目录下查找node\_module中是否有express。

若没有就会到上一级目录查找，直到顶级目录找不到才会报错。

##### express中间件

###### 中间件概述

中间件的作用是为主要的业务逻辑所服务，就是express下的函数。

必须要配合其他的中间件或者路由共同实现一个业务逻辑。

app.use(fn) 应用于(拦截)所有的路由

app.use('/list',fn) 应用于(拦截)特定的路由

例:

const express=require('express');

var app=express();

app.listen(3000);

//创建了一个中间件函数

function fn(req,res,next) {

console.log('执行了验证');

//res.send('中间件拦截了请求');//send只能出现一次,此次只是测试

//继续调用下一个中间件

next();

}

function fn1(req,res,next) {

console.log('查看了权限');

next();

}

//在中间件中使用函数——把函数名称传递进去

app.use(fn);

//创建中间件——查看权限

app.use(fn1);

//路由

app.get('/reg',function(req,res,next){

res.send('注册成功');

next();

});

app.get('/login',function(req,res,next){

res.send('登录成功');

next();

});

//后置中间件

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

console.log('打印的日志')

next();

});

例:

const express=require('express');

var app=express();

app.listen(3000);

//应用于指定的路由

//给/list添加中间件

app.use('/list',function(req,res,next){

console.log('列表的前置中间件');

next();

});

app.get('/detail',function(req,res){

res.send('这是商品详情');

});

练习,给/detail分别添加前置和后置中间件

//在上述例子的/detail处添加

app.use('/detail',function(req,res,next){

console.log('详情前置中间件');

next();

});

app.get('/detail',function(req,res,next){

res.send('这是商品详情');

next();

});

app.use('/detail',function(req,res){

console.log('详情后置中间件');

});

练习:使用Web服务器创建网页计数器。

a.使用中间件初始化num为0

b.在路由中,nun加1,把num的值响应到浏览器

提示:发送式,数字转字符串

const express=require('express');

var app=express();

app.listen(3000,function(){

console.log('ok');

});

var num=0;

app.use('/num',function(req,res,next){

num++;

console.log(num);

next();

});

app.get('/num',function(req,res){

res.send('计数:'+num.toString());

});

###### 中间件分类

**应用级的中间件** 拦截路由 app.use（'/list',fn）;

**路由级的中间件** 用于调用路由器;

**内置的中间件** 托管静态文件到某一个目录下;

app.use(express.static('要托管的目录'));

可以托管多个目录,按照托管的顺序来查找文件,如果第一没有,会继续往后查找

###### **第三方的中间件**

先下载安装模块

body-parser 中间件使用,作用:将post请求的数据解析为对象

app.use(bodyParser.urlencoded({

extended:false //使用querystring解析数据

}));

获取数据: req.body

###### 错误级的中间件

例://内置中间件

const express=require('express');

var app=express();

app.listen(3000);

//把所有的静态文件(html、css、浏览器端的js、图像)托管到public目录下

//如果浏览器请求某一个静态资源,不需要再使用路由,而是自动到public下查找

app.use(express.static('public')); //express.static,使用express包下的static表托管静态文件

//浏览器端输入http: //127.0.0.1:3000/login.html,可直接登录网页

练习:把静态文件托管到files目录下,查看如果和public下存在相同名称的文件,最终显示哪一个

//接上个程序

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

//显示哪一个与程序的书写顺序有关,当第一个文件中没有时,才会访问第二个文件

练习:创建Web服务器,托管静态文件到public目录下;在login.html中实现用户登录(用户名,密码,提交按钮)

//创建03\_bodyParser.js文件

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

const express=require('express');

var app=express();

app.listen(3000);

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

//使用中间件body-parser,解析post请求的数据

//urlencoded 将post请求数据解析为对象

app.use(bodyParser.urlencoded({

extended:false

//false 使用querystring模块解析为对象

//true 使用第三方模块qs解析为对象

}));

//路由

app.post('/mylogin',function(req,res,next){

//获取get请求的数据

//req.query

//获取post请求的数据——数据

/\*

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

console.log(buf);

console.log(buf.toString());

//使用查询字符串模块将数据解析为对象

});

\*/

//若是使用了第三方的中间件,获取数据

console.log(req.body);

res.send('登陆成功');

});

//在login.html中输入以下代码

<form method="post" action="/mylogin">

用户名:<input type="text" name="uname"><br>

密码:<input type="text" name="upwd"><br>

<input type="submit">

</form>

###### npm使用

npm init -y //查看安装npm包的日志

npm install 包名 //安装npm包

例:npm install express

用npm包的时候现装现用,正常的不得拷贝不得移动位置

拷贝package.json文件

在node中转入package.json文件所在文件夹,输入npm install,可自动安装

###### req对象

req.method 请求的方法

req.url 请求的URL

req.query 获取GET请求的查询字符串,并解析为对象

req.params 获取路由中传递参数,解析为对象

req.body 获取POST请求的数据,并解析为对象

### MySQL模块

增 INSERT INTO emp VALUES(NULL...)

删 DELETE FROM emp WHERE eid=3;

改 UPDATE emp SET sex=1,salary=800 WHERE eid=10;

查 SELECT \* FROM emp;

#### 普通连接

var connection=mysql.createConnect({ }) 创建连接

connection.connect() 执行连接

connection.query(sql.callback) 执行SQL语句,通过回调函数来获取SQL语句的执行结果。

例://使用MySQL模块连接数据库

//引入MySQL模块

const mysql=require('mysql');

//mysql -h127.0.0.1 -p3306 -uroot -p

//使用数据库 use tedu;

//连接:主机,端口,用户名,密码

var connection=mysql.createConnection({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu',//连接成功后所使用的数据库

});

//执行连接

connection.connect();

//执行SQL语句

connection.query('select\*from emp', function( err,result ){

//err 如果查询失败

//result 查询的结果

if (err) throw err;

console.log(result);

});

#### 使用连接池

var pool=mysql.createPool() 创建连接池,使用connectionLimit设置连接池的数量。

pool.query(sql,callback) 执行SQL语句,通过回调函数来获取SQL语句的执行结果。

SQL语句中可以使用占位符（？），可以防止SQL注入。

例://使用连接池——在创建MySQL连接的时候，设置连接的个数

const mysql=require('mysql');

//直接连接

var pool=mysql.createPool({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu',

connectionLimit:15 //设置连接池的数量

});

//执行SQL语句

pool.query('select \* from dept',function(err,result){

if(err) throw err;

console.log(result);

});

练习:往部门表中插入2条记录,最后打印插入的结果

const mysql=require('mysql');

//直接连接

var pool=mysql.createPool({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'tedu',

connectionLimit:15 //设置连接池的数量

});

pool.query('insert into dept values (50,"人事部"),(60,"行政部") ',function(err,result){

if(err) throw err;

console.log(result);

});

var obj={

did:70,

dname:'后勤部'

}

pool.query('insert into dept SET ? ',obj,function(err,result){

if (err) throw err;

console.log(result);

});

练习:使用对象的形式往员工表插入1条记录

var obj={

eid:null,

birthday:'1995-3-15',

salary:7500,

ename:'zhangsan',

sex:1,

deptId:20

}

pool.query('insert into emp SET ? ',obj,function(err,result){

if (err) throw err;

console.log(result);

});

练习:查询工资大于5000,性别为男的员工

var s=5000;

var n=1;

pool.query(`select \* from emp where salary>? and sex=?`,[s,n],function(err,result){ //?表示占位符,可以防攻击

if (err) throw err;

console.log(result);

});

练习:删除员工表中编号为5的记录。更改员工编号为6的,工资8888,生日为2000-1-1

//删除编号为5的记录

pool.query('DELETE FROM emp WHERE eid=?',5,function(err,result){

if (err) throw err;

console.log(result);

});

//更改编号为6的记录

pool.query('UPDATE emp SET salary=?,birthday=? WHERE eid=?',[8888,'2000-1-1',6],function(err,result){

if (err) throw err;

console.log(result);

});

练习:

//1.创建Web服务器

const mysql=require('mysql');

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

const express=require('express');

var app=express();

app.listen(3000);

//连接mysql数据库

var pool=mysql.createPool({

host:'127.0.0.1',

port:3306,

user:'root',

password:'',

database:'tedu',

connectionLimit:15

});

//2.托管静态文件到public下

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

//使用body-parser中间件,解析post请求的数据

app.use(bodyParser.urlencoded({

extended:false

}));

//3.在public下,创建add.html

//4.路由:请求的方法post 请求的URL:/dept

//获取浏览器post请求的数据,格式:对象 body-parser

app.post('/dept',function(req,res){

//console.log(req.body); //测试代码

//把post请求的数据,插入到MySQL数据库中

//执行SQL语句,如果插入成功,响应给浏览器"部门添加成功"

pool.query('insert into dept set ?',req.body,function(err,result){

if (err) throw err;

//console.log(result); //会出affectedRows的值

//如果结构中affectedRows大于0,说明插入成功,否则插入失败

if (result.affectedRows>0){

res.send('部门添加成功');

}else{

res.send('部门添加失败');

}

});

});

//add.html中的程序

<form method="post" action="/dept">

部门编号:<input type="text" name="did"><br>

部门名称:<input type="text" name="dname"><br>

<input type="submit">

</form>

## 连接错误

1. connect ECONNREFUSED 127.0.0.1:3306 没有启动MySQL服务器
2. You have an error in your SQL syntax SQL中存在语法错误
3. Access denied for user ''@' localhost' 连接错误,用户名或密码错误
4. Duplicate entry '50' for key 'PRIMARY' 插入的记录出现了重复内容

## 项目一

# HTTP

李然 qq:1535212067

## 导论

### 课程安排

1. 课程安排 20天
2. HTML5 Basic(2天)

搭建网页的结构

静态网页——数据写死，不与服务器做数据交互

动态静态——与服务器做数据交互

1. AJAX异步数据交互（3天）

异步完成前后端数据交换

1. HTML5+AJAX完成项目（2天）

mysql、js、nodejs、html5 ajax

1. CSS3（4天基础+2天项目）

美化html网页

简单易懂，代码量大，单词量大

1. Bootstrap框架（4天）

CSS的框架 做响应式开发

1. Bootstrap完成学子商城的项目（3天）

### 第二阶段知识特点

1. 知识点极为细碎
2. 单词量巨大
3. 知识点之间没有逻辑调理

### 学习要求

1. 单词
2. 键速
3. 笔记，不要每一个字都抄
4. 加大练习量

## web基础知识

### web和internet

#### web简介

万维网，即WWW。

各种类型的信息，和服务器无缝连接，并提供可视化的界面。

#### Internet简介

全球性计算机网络。

主要服务：www、email、Telnet、BBS、FTP

基本实现技术：分组交换原理、TCP/IP协议

#### web和internet

web是运行在Internet上的程序（运行在网络上的程序）

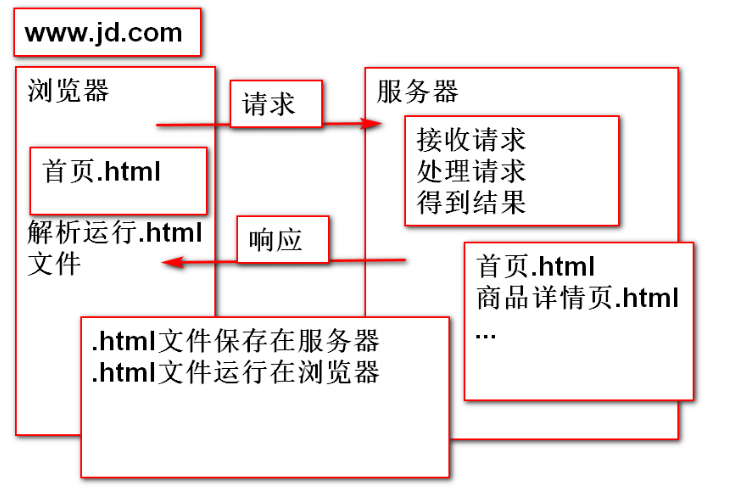
internet为web提供了网络运行环境

### Internet上的应用程序

|  |  |
| --- | --- |
| 安装在设备上的程序，客户端web程序 | 网页程序，不需要安装客户端运行 |
| C/S  C:Client 客户端  S:Server 服务器 | B/S  B:Browser 浏览器  S:Server 服务器 |
| CS需要升级客户端 | BS不需要升级 |

主要工作：主要开发B/S结构的web程序

## web运行原理



总结：

1. 网页存储在服务器
2. 网页运行在浏览器
3. web程序：运行在internet上的一种B/S结构的应用程序，俗称：网站
4. web的工作原理

基于浏览器和服务器，以及通信协议来实现的数据传输和展示

通信协议：规范了数据是如何打包和传递的

### 服务器

#### 功能

1. 存储数据
2. 接收用户的请求并给出响应
3. 提供程序的运行环境
4. 具备一定的安全功能

#### 服务器的产品

1. TOMCAT Java
2. Apache php
3. IIS .net

#### 写服务器的技术:

1. .php
2. .java
3. .net
4. .node.js
5. .python

### 浏览器

#### 功能

1. 代表用户向服务器发送请求
2. 作为HTML、CSS、JS的解析器
3. 以图形化的解码展示给用户

#### 浏览器的技术

HTML、CSS、JS

#### 浏览器的产品

内核：内部的核心代码

1. Chrome
2. Safari
3. FireFox
4. Oprea
5. IE--edge

## HTML快速入门

### 什么是HTML

HTML：Hyper Text Markup Language

超文本标记语言

### HTML的特点

1. 以.html或.htm为后缀
2. 由浏览器取解析执行
3. 用带有尖括号的'标记'来标识
4. 在网页中可以执行JS脚本语言

### HTML基础语法

#### 标记

标记，又称“元素”、“标签”、“节点”。

<关键字></关键字>

#### 标记的分类

1. 双标签/封闭标签

<关键字></关键字>

双标签必须成对出现，有开就有关

ex：<a>内容啊</a>

1. 单标签/非封闭标签（空标签）//占时先无视空标签

<关键字>（HTML4.0要求）

或者<关键字/>（XHTML要求）

ex：<br> 或者 <br/>

练习：创建文本文档，将名称改为first.html。然后在文档中按下面要求写出标记

写一对html标记

写一对head标记

写一对title标记

写一对body标记

<html></html>

<head></head>

<title></title>

<body></body>

#### 元素

元素即标记。

每一对尖括号包围的部分，元素就像小标签，用于表示网页文档的不同部分。

元素可以包含文本内容和其他元素，也可是空的，如：空标记。

#### 标签的嵌套

##### 什么是嵌套

在一对标签中出现其他的标签，从而形成了功能的叠加

##### 语法

<标记1>

</标记2>

<标记3/>

</标记2>

<标记1>

注意嵌套顺序和代码缩进。

例:常用写法

<i>

<b>

<s>演示内容</s>

</b>

</i>

不推荐写法

1.<i><b><s>演示内容</s></b></i>

2. <i>

<b>

<s>

演示内容

</s>

</b>

</i>

错误写法

<i>

<b>

<s>演示内容</b>

</i>

</s>

#### 属性和值

允许通过属性和值对标签进行修饰

语法:

<关键字˽属性="值"></关键字>

<关键字˽属性1="值1"˽属性2='值2'></关键字>

例:

<p align='center' style="color:red" title='脑瓜疼,脑瓜疼,脑瓜疼'>唉呀妈呀脑瓜疼!</p>

例:

<b align='center' title='周一放七天'>今天周五,明天不放假</b>

//align不产生效果,是为p标签的专有属性,所以在b标签中不起作用

#### 公有属性

所有标签都支持的属性

id:定义元素在页面中唯一标识

title:鼠标悬停在元素上,所显示的文本

style:为CSS中,定义行内样式的

class:为CSS中,引用类选择器

总结:学习html究竟是学习什么?

1. 记忆固定的关键字,及其代表的功能
2. 记忆固定的属性,
3. 记忆固定的嵌套关系

#### 注释

不被浏览器的解析运行的代码

<!--注释内容-->?

|  |
| --- |
| 注意:   1. 注释不能嵌套注释 2. 不能放在尖括号里 |

## HTML的文档结构

### 文档类型的声明

<!doctype html>

告诉浏览器, 解析这篇代码时,使用html5的标准

### 网页的结构

<html>

网页的内容

</html>

作用:代表一个网页的开始和结束

一个html文件中只能有一对html标签

在html中包含头和主体

<head></head> 网页的头部,定义全局信息

<body></body> 网页的主体,展示的内容

一个网页的标准结构

<!doctype html>

<html>

<head></head>

<body></body>

</html>

### head元素

head元素，是其他头元素的容器

<title></title> 网页的标题

<meta/> 定义全局的信息

<meta charset='utf-8'>

<meta name="description" content="描述的内容">

<style></style> 定义网页的内部样式

<link> 引用外部CSS文件

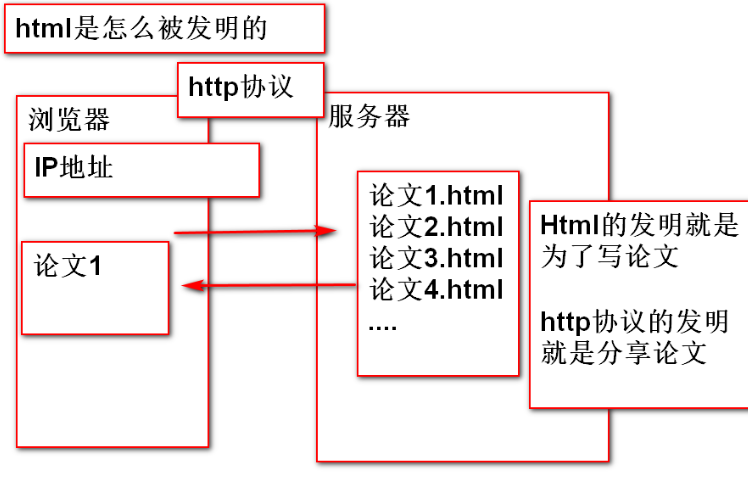
<script></script> 定义或者应用js代码

### body元素

指定网页的主体

1. 定义网页背景颜色 bgcolor="black"
2. 定义网页字体颜色 text="yellow"

### 文本标记



#### 标题元素

作用:在页面中以醒目的方式显示文本

语法:

<h1> </h1>

<h2> </h2>

<h3> </h3>

<h4> </h4>

<h5> </h5>

<h6> </h6>

特点:

1. 字体大小有变换;h1最大,h6最小
2. 字体加粗
3. 标签独占一行,上下有垂直空白间距

属性:align

作用:设置标签内容水平对齐方式

取值:left/ center/ right

#### 段落标记 p

<p></p>

作用:突出表示一段文字

##### 特点

###### 独占一行

###### 文本上下有空白间距

属性:align

取值:left/ center/ right

<p>lorem+tab</p> 假文,用于测试段落

###### 换行

<br>或者<br/>

###### 水平线

<hr>或者<hr/>

属性

size 粗细,取值为px的数值

width 水平线的宽度,取值为px或者%的数值

align 水平对齐的方式,取值left/center/right

color 水平线的颜色,取值合法的颜色值

###### 预格式化

<pre></pre>

作用,保留html代码中的回车和空格效果

因为html存在空格折叠现象

(空格折叠现象,因为所有空格和回车在页面上都会显示成一个空格)

###### 特殊字符(实体)

&nbsp; 空格

&lt; <

&gt; >

&copy; ©

&yen; ¥

&reg ®

&times ×,关闭x号

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

<p>Copyright&nbsp;&copy; 2004 - 2018 &lt;京东JD.com&gt;版权所有</p>

###### 文本样式

<b></b> 字体加粗

<strong></strong> 字体加粗

<i></i> 字体斜体

<em></em> 字体斜体

<u></u> 下划线

<s></s> 删除线

<del></del> 删除线

|<sup></sup> 上标

<sub></sub> 下标

##### 分区元素(逻辑分区)

###### 块分区

<div></div>

用于页面中的布局效果

特点:独占一行

###### 行分区

<span></span>

处理同一行文本中的不同样式

特点:多个span标签共用一行

##### 级元素和行内元素(行级元素)

###### 块级元素

在网页中,独占一行的元素,就是块级元素

默认从上往下顺序排列

常见块级元素如下:

1. 标题元素 h1-h6
2. 段落元素 p
3. div
4. 结构元素

###### 行内元素

在网页中多个元素在同一行中显示

默认从左往右顺序排列

常见的行内元素如下：

span b sub等,文本标签都是行内元素

###### 行内块元素

表现方式,如行内元素一样。但具备块级元素的一些特性。

###### table

宽度和高度，默认是通过内容决定的

表格是一种单独的表现方式

### 图像和链接

#### 图像的使用

<img src="url">

属性

src="url" url指图片资源的路径

### URL

#### 目录结构

文件目录:文件夹的嵌套结构

#### URL

##### 目录结构

文件目录：文件夹的嵌套结构

##### URL

URL: Uniform Resource Locator

统一资源定位器--->路径

##### 路径的表现形式(url)

###### 绝对路径

完整路径

通信协议+主机名称+文件目录结构+文件名称

使用网络资源

优点:不占用本地存储空间

缺点:不稳定

###### 相对路径

使用本服务器的图片资源

参照物,当前的.html文件

1. 资源和.html在同一个目录下,是兄弟关系

直接使用图片的文件名称

<img src='08.png'>

1. 资源在.html同级的文件夹下

<img src='image/08.png'>

1. 资源在父级目录

<img src="../09.png">

#### 属性

src source源,设置图像的路径

alt 图片出错时,显示的文本

title 鼠标悬浮在图片,显示的文本

width 设置图片宽度,单位为px的数值

height 设置图片高度,单位为px的数值

解决图片的失真,只设置宽高中的一个

#### 链接

<a href='http://www.tmooc.cn/'>跳转到tmooc</a>

属性

href 链接的路径(需要由协议http)

target 指定打开链接的方式

\_self 默认值,在当前标签页打开新网页

\_blank 在新的标签页打开新网页

<a href='http://www.tmooc.cn/' target='\_blank'>用新标签跳转到tmooc</a>

##### a标签的其他表现形式

###### 调用js代码

头部:

<a href='javascript:showMsg()'>调用js代码</a>

主体:

<script>

function showMsg(){

alert("我是a标签调用的js代码")

}

</script>

###### 下载资源——不重要

<a href="01.zip">下载zip</a>

###### 连接到电子邮件的客户端——不重要

(仅适用于有windows自带邮件客户端)

<a href="mailto:xxxx@tedu.com">发送邮件</a>

###### 返回页面顶部——重要

<a href="#">返回到顶部</a>

##### a标签的锚点

使用a标签链接到其他页面，跳转新页面

使用a标签链接到本页面的其他位置锚点

a也可以跳转到其他页面的某个锚点去

###### 定义锚点

1. 使用a标签定义锚点

<a name="锚点名称"></a>

1. 任何非a定义锚点

<any id="锚点名称"></any>

###### 跳转到锚点

<a href="#锚点名称">跳转锚点</a>

###### 跳转到另外一个页面的某个锚点

<a href="03\_a.html#No1">jeep</a>

练习:创建网页03\_ex.html,在网页上方使用a标签写出3个书名.30个br之后,分别使用假文充当3本书的内容.点击每本书的a标签,跳转到响应书的内容(1本书使用a设定锚点,)

<head>

<title></title>

<meta charset='utf-8'>

</head>

<body>

<a href="#No1">《论语》</a>

<a href="#No2">《孟子》</a>

<a href="#No3">《荀子》</a>

<a href="03\_a.html#No1">jeep</a>

<!--1.使用a标签的方式定义锚-->

<a name="No1">book1 Lorem</a>

<any id="No2">book2 Lorem</any>

<any id="No3">book3 Lorem</any>

</body>

练习:创建01\_ex.html,在网页中显示3张呆呆兵的图片,要求，分别以相对路径的三种形式引用图片,同时设置文字属性

<!doctype html>

<html>

<head>

<title></title>

<meta charset='utf-8'>

</head>

<body>

<img src='0-2.png' alt="兵死了" title='蓝帽'>

<img src="image/01.png" alt="绿帽兵" title='绿帽' width='200'>

<img src="../10.png" alt="红帽兵" title='红帽' width='200px' height='400'>

</body>

</html>

### 表格

#### 语法

<table>

<tr>

<td></td>

...

</tr>

</table>

tr----table row----行

td----table data----列

#### 属性

##### table的属性

border 设置表格的边框

width/ height 宽高

align 水平对齐 left/ center/ right

bgcolor 背景颜色

cellspacing 设置单元格内边距,内容与边距之间的距离

cellpadding 设置单元格外边距,边框与边框之间的距离

练习:设置表格400\*400,背景颜色、表格内容自拟。表格边框1px，水平距离显示，内边距5px，外边距10px

<table border="1px" bgcolor="#00ffff" align="center" cellspacing="5px" cellpadding="10px">

<tr>

<td>序号</td>

<td>姓名</td>

</tr>

<tr align='center'>

<td>1</td>

<td>啦啦</td>

</tr>

</table>

##### tr的属性

align 设置当前内容,水平对齐方式 left/ center/ right

valign 设置当前内容,垂直对齐方式 top/ middle/ bottom

bgcolor 设置当前行的背景

##### td的属性

width/height 宽高;一个td的高度会影响同行所有的高度

align 设置当前内容,水平对齐方式 left/ center/ right

valign 设置当前内容,垂直对齐方式 top/ middle/ bottom

bgcolor 设置当前行的背景

colspan 跨列

rowspan 跨行

#### 不规则的表格使用

1. 跨列

colspan:从指定的单元格开始,横向向右合并几个单元格(包含自己)

被合并的单元格要删除

1. 跨行

rowspan:从指定的单元格开始,纵向向下合并几个单元格(包含自己)

#### 可选标签

1. 表格标题(自动居中)

<caption></caption>

如果要设置表格的标题,必须将<caption></caption>放在table标签之后

1. 行/列标题

<th></th> 可直接代替<td></td>存在

加粗并且居中

#### 表格复杂的应用

##### 行分组(分区)

1. 表头<thead></thead>

表格中最上面几行放在thead中

1. 表主体<tbody></tbody>

将若干行放在tbody中统一管理

1. 表脚<tfoot></tfoot>

表格中最后一行放入tfoot

##### 表格嵌套

表格中所有的嵌套,都只能放在td中

### 列表(重点)

#### 列表的作用

默认按照从上到下(或从左到右)显示所有的数据;

并可以在数据之前添加标识

#### 列表组成

列表类型------有序列表 ol /无序列表 ul

列表项 li

1. 有序列表 ol(order list) li(list item)

<ol>

<li>美国超级英雄</li>

<li>中国超级英雄</li>

<li>日本超级英雄</li>

</ol>

1. 无序列表 ul(unorder list)

<ul>

<li>火影忍者</li>

<li>海贼王</li>

<li>妖精的尾巴</li>

<li>死神</li>

</ul>

#### 列表属性

##### 有序列表的属性

type 作用,指定标识项的类型

取值

1:默认,数字

a:小写字母

A:大写字母

i:小写罗马字母

I:大写罗马字母

start 指定起始编号

##### 无序列表的属性

type 作用,指定列表标识类型

取值

disc:默认,实心圆

circle:空心圆

square:实心方块

none 不显示标识

#### 列表嵌套

##### 列表嵌套列表

嵌套的内容,必须在li中写

例:

<li>

<ol>

<li>qwe</li>

<li>asd</li>

<li>zxc</li>

</ol>

</li>

练习:外层使用有序列表.内层使用无序列表.

1.前端技术

html、css、js

2.后台技术

java、nodeJS

3.前端框架

VueJS、Angular、React

<ol>

<li>前端技术

<ul>

<li>html</li>

<li>css</li>

<li>js</li>

</ul>

</li>

<li>后台技术

<ul>

<li>java</li>

<li>nodeJS</li>

</ul>

</li>

<li>前端框架

<ul>

<li>VueJS</li>

<li>Angular</li>

<li>React</li>

</ul>

</li>

</ol>

##### 列表嵌套其他元素

li中嵌套其他标签 a img

<ul type="none">

<li>

<h3>

<a href="#">买家帮助</a>

</h3>

</li>

<li><a href="#">新手指南</a></li>

<li><a href="#">服务保障</a></li>

<li><a href="#">常见问题</a></li>

</ul>

#### 定义列表

<dl></dl> 定义定义列表

<dt></dt> 表示定义列表中要解释说明的名词

<dd></dd> 表示定义列表中对名词的解释内容

一般用于图文混排

bordercolor 边框颜色

### 结构标记<div></div> 语义化标签

经常使用<div>元素设计页面的大致布局，如页头，导航栏，侧边栏，主要内容部分，页脚等。

<strong>比<b>——可读性强——搜索时优先级高（理论上）

#### 结构标记的作用

用于描述整个网页的结构,取代div做布局

提示标记的语义性

#### 常用的语义标签

1. <header></header>

定义网页的头部或某个区域的头部内容

1. <nav></nav>

定义网页的导航链接部分

1. <section></section>

定义网页的主体内容

1. <aside></aside>

定义网页的侧边栏的信息

1. <footer></footer>

定义网页的底部内容

1. <article></article>

定义与文字相关的内容（纯文字）

比如:论坛的帖子,微博的条目,用户的评价

|  |  |
| --- | --- |
| 练习：制作右侧网页 | 01_ex |

<!doctype html>

<html>

<head>

<title>欢迎来自此网址</title>

<meta charset='utf-8'>

<p>

<strong>欢迎来自此网址

<img src="../web第二阶段练习效果图示\day03\_ajax\_01作业\img\jd\_logo.png" alt="">

</p>

<hr>

</head>

<body>

<nav>

<a href="">京东超市</a>

<a href="">服装</a>

<a href="">宠物</a>

<a href="">母婴</a>

<a href="">生鲜</a>

</nav>

<aside align="left">

<ul type="none">

<li>路由器</li>

<li>手机</li>

<li>笔记本</li>

<li>大哥大</li>

<li>BB机</li>

<li>交换机</li>

<li>服务器</li>

</ul>

</aside>

<section align="center">

<img src="../web第二阶段练习效果图示\day03\_ajax\_01作业\img\login.png" alt="" height="300px">

<article align="left">

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptas beatae sunt omnis corporis animi vero aperiam. Distinctio, esse, doloribus, iure, debitis rem explicabo quis minus possimus illo quam tempore expedita.

</article>

</section>

<footer align="center">

Copyting&copy;2004 - 2018 &lt;京东JD.com&gt;版权所有

</footer>

</body>

</html>

### 表单(重点&难点)

#### 作用

1. 提供可视化的输入控件
2. 收集用户信息并提交给服务器

#### 表单的组成

##### 前端部分

提供表单的控件,与用户交互的可视化控件

##### 后端服务

对提交的数据进行处理

##### 表单

<form></form>

属性:

1. action

定义表单被提交时,发送的URL

通常写的是服务器处理程序的地址

默认提交给本网页

1. method

指定表单数据的提交方式

method取值:

|  |  |
| --- | --- |
| get(默认值) | post |
| 1.提交的数据有大小限制，限制为2KB | 1.提交数据没有大小限制 |
| 2.明文提交，待提交的内容会显示在浏览器的地址栏上 | 2.隐式提交，提交的内容不会显示在地址栏中 |
| 3.向服务器要数据的使用（查询，删除） | 3.要传递数据给服务器的时候，使用post |
| method其他取值（了解）  delete  put | |

##### enctype编码方式

指定表单数据的编码方式

允许将什么样的数据提交给服务器

1. application/x-www-form-urlencoded

默认,允许将任意字符提交给服务器(文件除外)

1. rext/plain

只能将普通字符提交给服务器如:123、abd、中文等

注意,如果在传输字符中有特殊符号,form的enctype属性,不能使用text/plain。必须使用application/x-www-form-urlencoded(使用ajax的时候,如果用post传递参数,需要更改此处)

1. multipart/form-data

运行将文件提交给服务器

multipart把文件转换成二进制,传递给服务器

#### 表单控件(在表单中可以输入数据的标签)

能够与用户进行交互的可视化元素(标签)

##### 分类

1. input<input type="text">
2. textarea多行文本域
3. select和option下拉选
4. 其他元素

##### input控件

在页面中提供各种各样的输入控件

比如:文本框,密码,单选按钮,复选框等

<input type="类型">

##### 属性

###### type

指定input输入的类型

text 文本框

password 密码框

radio 单选框

checkbox 多选框

###### name

为控件定义名称,提供给服务端使用(必须写)

###### value

控件的值,提交给服务器使用

###### disabled

禁用的,如果操作,也不能提交

可以disabled="disabled" 也可以不给值

##### input详解

###### 文本框和密码框

<input type="text">

<input type="password">

属性:

maxlength 限制输入的最大字符数

readonly 只读,只能看不能写,但是允许被提交

该属性没有值,或者值是其属性本身

placeholder 占位符,默认显示在控件上的文本

练习:完成登录页面,创建02\_form.html

用户名:文本框——提示输入用户名

密码:密码——提示输入密码

验证码：文本框——只读

提交按钮

要求,可以正确使用get方法提交数据

<body>

<form action="" method="post">

<p>用户名:

<input type="text" name="uname" placeholder="输入用户名">

</p>

<p>密码:

<input type="password" name="upwd" placeholder="输入密码">

</p>

<p>验证码:

<input type="text" name="checking" readonly value="abcd">

</p>

<input type="submit" value="提交">

</form>

</body>

###### 按钮

提交按钮

<input type="submit" value="提交">

将表单中的数据提交给服务器

重置按钮

<input type="reset" value="重置">

将表单的内容回复到初始化的状态

普通按钮

<input type="button" value="普通按钮">

没有任何功能,但是可以调用js的方法

属性:value,设置按钮上的文本

###### 单选框

<input type="radion" name="gender" value="mm" checked>

属性

name 除了定义控件名称.还起到的了分组作用

value 必须写

checked 默认被点,无值属性

###### 复选框

<input type="checkbox" name="hobby[]" value="eat">

属性

name 除了定义控件名称,还起到了分组的作用

必须使用数组

value 必须写

checked 默认被选中,无值属性

###### 隐藏域

想提交给服务器,又不想被用户看见

<input type="hidden" value="1000123" name="id">

###### 文件的选择框

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

method 必须使用post

enctype 必须是multipart/form-data

属性

multiple 多重的,可以一次选择多个文件

##### 非input标签

###### 文本域

允许录入多行数据的文本框

属性

name readonly

cols 列数(一行显示多少个英文字符)

rows 行数

###### select和option

<select name="add">

<option>西安</option>

<option>北京</option>

<option>佳木斯</option>

<option>大连</option>

</select>

1. option没有定义value属性

提交的时候,select的value为选中的那个option的内容

1. option有value值

提交的时候,select的value为选中的那个option的value

<select>属性

name

size 定义显示选项的数量,默认值1,下拉选

如果取值大于1的数字

元素表现为滚动列表

multiple 设置多选,无值属性

按住Ctrl可以多选,只有滚动列表支持多选

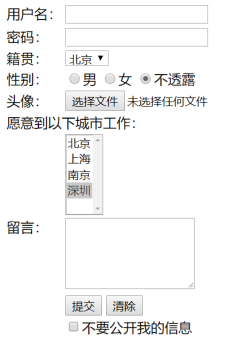
<option>属性

value

selected 默认选中,无值属性

练习:完成右侧网页

<form action="" method="post">

<div>

<label>用户名:</label>

<input type="text" maxlength="5">

</div>

<div>

<label>密码:</label>

<input type="password" maxlength="10">

</div>

<div>

<label>籍贯:</label>

<select name="address" id="">

<option>北京</option>

<option>上海</option>

<option>广州</option>

<option>深圳</option>

</select>

</div>

<div>

<label>性别:</label>

<input id='m' type='radio' name='gender' value='m'><label for='m'>男</label>

<input id='f' type='radio' name='gander' value='f'><label for='f'>女</label>

<input id='x' type='radio' name='gander' value='x' checked><label for='x'>保密</label>

</div>

<div>

<label>头像:</label>

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

</div>

<div>

<label>愿意到以下城市工作:</label>

<select name="add" size="" multiple>

<option value="bj">北京</option>

<option value="sh">上海</option>

<option value="gz">广州</option>

<option value="sz">深圳</option>

</select>

</div>

<div>

<label>留言:</label>

<textarea name ="submit" cols="10" rows="3"></textarea>

</div>

<div>

<input type="submit" value="提交">

<input type="reset" value="清除">

</div>

<div>

<input id='mi' type="checkbox" name="mi"><label for='mi'>不要公开我的信息</label>

</div>

</form>

#### 其他控件

##### 关联文本域表单控件

<label></label>

label使用for属性,与表单控件的id值一致,进行绑定

<input id='m' type="radio" name='gender' value="m">

<label for='m'>男</label>

##### 为控件分组的标签

<fieldset></fieldset>为控件分组

<legend></legend>为分组定义标题

##### 浮动框架

允许在一个网页中,引入另外一个网页进来

<iframe></iframe>

src引入网页的资源路径

frameborder浮动框的边界,默认是1

width/ height

#### 新表单元素

HTML5版本中新提出的表单控件

##### 电子邮件

<input type='email' name='email'>

表单提交时,会验证数据是否符合email的规范(@)

##### 搜索框

<input type="search" name='search'>

提供了快速清除的功能

##### URL类型的文本框

<input type="url" name="url">

提交时,验证数据是否符合url规范(http://)

##### 电话号码文本框

<input type="tel" name="tel">

仅在移动设备中显示拨号键盘的效果

##### 数字类型文本框

<input type="number" name="num" max='25' min='20' step='2'>

属性

min 允许接收的最小值

max 允许接收的最大值

step 每次调整数字的步数

##### 范围选择

给用户提供一个滑块组件,允许用户指定选取的范围

<input type="range" max='60' min='50' step='2' value='55'>

属性

value 初始值

min 允许接收的最小值

max 允许接收的最大值

step 每次调整数字的步数

##### 颜色拾取器

<input type="color" name="range">

提供一个颜色拾取器

##### 日期类型

<input type="date" name="date">

提供了一个选择日期的控件

##### 月份类型

<input type="month" name="month">

提供了一个选择月份的控件

##### 周类型

<input type="week" name="week">

提供了一个选择星期的控件

练习:

<body>

<form action="" method='post'>

<!-----用户基本信息表----->

<fieldset>

<legend>用户基本信息</legend>

<table align='left'>

<tr>

<td><label>用户名称:</label></td>

<td>

<input type="text" name="uname" placeholder="请输入您的用户名称">

</td>

</tr>

<tr>

<td><label>用户密码:</label></td>

<td>

<input type="password" name='password' placeholder="输入密码">

</td>

</tr>

<tr>

<!--用户性别-->

<div>

<td><label>用户性别:</label></td>

<td>

<input id='m' type="radio" name="gender" value="m"><label for='m'>男</label>

<input id='f' type="radio" name="gender" value="f"><label for='f'>女</label>

<input id='x' type="radio" name="gender" value="x" checked><label for='x'>薛定谔状态</label>

</td>

</div>

</tr>

</table>

</fieldset>

<!--复选框-->

<div>

<label>用户爱好:</label>

<input id='chi' type="checkbox" name="hobby[]"><label for='chi'>吃</label>

<input id='he' type="checkbox" name="hobby[]"><label for='he'>喝</label>

<input id='wan' type="checkbox" name="hobby[]"><label for='wan'>玩</label>

<input id='le' type="checkbox" name="hobby[]"><label for='le'>乐</label>

<br>

</div>

<div>

<!--选择头像-->

<span>选择头像:</span>

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

<br>

<br>

</div>

<div>

<!--自我介绍-->

<span>自我介绍:</span>

<textarea name="submit" id="" cols="30" rows="10"></textarea>

<br>

<br>

<span>工作城市:</span>

<select name="address" size="3" step='2'>

<option value="xa">西安</option>

<option value="bj">北京</option>

<option value="dl">大连</option>

<option value="sh">上海</option>

<option value="gz">广州</option>

<option value="wh">武汉</option>

<option value="sz">深圳</option>

</select>

</div>

</form>

<br>

<div>

<input type="submit" value='登录'>

<input type="reset" value='重填'>

<input type="button" value='普通按钮'>

</div>

</body>

练习:

<head>

<title>作业2</title>

<meta charset='utf-8'>

<p>

<!--插入头部图片-->

<img src="..\web第二阶段练习效果图示\day01\_html\_01作业\素材\jd\_logo.png" alt="图挂了">

<!--设置头部字体大小-->

<font size='5'>

<strong>欢迎登陆</strong>

</font>

</p>

</head>

<body>

<!--设置表格-->

<div>

<table align='center' border="2px" height="100px" cellpadding='0px' cellpadding='10px'>

<tr>

<td rowspan='8' align='center'>

<img src="..\web第二阶段练习效果图示\day01\_html\_01作业\素材\login.png" alt="" width="400px">

</td>

<td width="300px">

<label>用户名/邮箱:</label>

</td>

</tr>

<tr>

<td>

<input type="text" name='uname'>

</td>

</tr>

<tr>

<td>

<label>密码:</label>

</td>

</tr>

<tr>

<td>

<input type="password" name='password' maxlength='6'>

</td>

</tr>

<tr>

<td>

<input type="checkbox" name='hobby' id='auto'><label for="auto">自动登陆</label>

</td>

</tr>

<tr>

<td align='center'>

<input type="submit" value='登陆'>

<input type="reset" value='重填'>

</td>

</tr>

<tr>

<td>

<label>使用京东合作账号登陆</label>

</td>

</tr>

<tr>

<td align='right'>

<input type="radio" name='fr' value='qq' id='qq'><label for="qq">QQ</label>

<input type="radio" name='fr' value='jd' id='jd'><label for="jd">京东钱包</label>

</td>

</tr>

<tr>

<td colspan='2' align="right">

<input type="submit" value='免费注册&gt;&gt;' style='background:#66ff66;color:#ffffff'>

</td>

</tr>

</table>

</div>

<br><br><br>

<!--底边广告-->

<footer>

<nav align='center'>

<a href="">关于我们</a>&brvbar;

<a href="">联系我们</a>&brvbar;

<a href="">人才招聘</a>&brvbar;

<a href="">商家入驻</a>&brvbar;

<a href="">广告服务</a>

</nav>

</footer>

</body>

|  |
| --- |
| <div id='d2' style="color:red"> </div>  属性和值之间用等号；样式和值之间用冒号。 |

### HTTP协议

#### URL

协议+主机名称+目录结构+文件名称

http://www.jd.com:80/index/login.html#no1

上面的URL写法不完整，一些默认信息被忽略掉。

完整写法如下：

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

<scheme>://——http:// 协议

<host>——www.jd.com 主机名称

:<port>——:80 端口号

/<path>;——/index 路径

<params>——login.html 文件名

?<query>——查询字符串

#<frag>——#no1 锚点

scheme 方案,协议,以哪种方式获取服务器资源

不区分大小写

**常见的协议(了解)**

|  |  |  |
| --- | --- | --- |
| 常见协议 | 默认端口号 | 协议的基本作用 |
| FTP | 21 | 文件的上传下载 |
| SSH | 22 | 安全的远程登录 |
| TELNET telnet | 23 | 远程登录 |
| SMP | 25 | 邮件的传输 |
| DNS | 53 | 域名解析 |
| HTTP | 80 | 超文本传输 |
| POP3 | 110 | 邮件的接收 |
| HTTPS | 443 | 加密传输的HTTP |

<user>:<pwd>@<host>:<port>/<path>

user:用户名

pwd:密码

host:主机名,localhost(127.0.0.1)

port:端口号

path:路径,资源在服务器上具体存放的位置,具体到文件名

<params>?<query>#<frag>

params:参数,服务器端使用 cookie/session

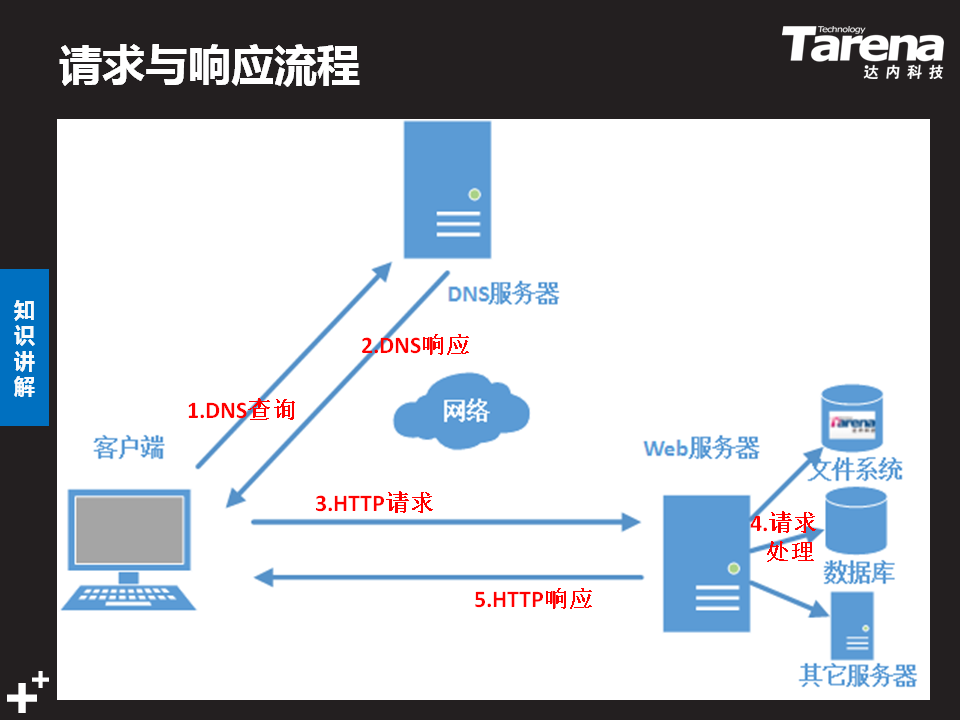
query:查询字符串

frag:锚点

#### HTTP

HTTP 更新历史,现在使用http协议版本是HTTP1.1,可以保持持久连接

**DNS服务器的详解**



#### HTTP详解

message 消息/报文

HTTP协议 对消息有特殊的格式要求

1. 请求消息(请求的起始行,请求头,请求主体)
2. 响应消息(响应的起始行,响应头,响应主体)

##### 请求消息

###### 请求的起始行

1. 请求的方法

get 客户端向服务器要数据的时候使用

特点:靠地址栏传递查询的字符串,无请求主体

post 客户端向服务器提交数据时使用

特点:请求主体

delete 表示客户端删除服务器的内容(一般禁用)

put 表示客户端想要放数据到服务器上(一般禁用)

connect 连接测试

trace 追踪请求路径

option 选项

head 表示客户端只获取响应的消息头

1. 请求url
2. 协议的版本号PPT/1.1

###### 请求头

1. HOST

例:Host:127.0.0.1:3000

告诉服务器,请求的是哪个主机

1. Connection:keep-alive

告诉服务器进入持久连接

1. User-Agent(用户代理)

告诉服务器自己浏览器,操作系统的类型

1. Accet-Encoding:gzip,deflate

告诉服务器自己能接收的压缩文件的类型

1. Accept-Language:zh-CN,zh

告诉服务器自己能接收的自然语言

1. Referer 引用/推荐人

告诉服务器请求来自哪个网页

###### 请求主体 FORM DATA

##### 响应消息

服务器端发送给客户的消息

###### 响应起始行

1. 协议版本号 HTTP/1.1
2. 响应状态码
3. 原因断句,对状态码进行解释

###### 响应头

1. Date 告诉浏览器,服务器响应的时间
2. Connection:keep-alive 告诉浏览器启动了持久连接
3. Content-Type 响应主体的类型是什么

ext/html 响应回来的数据是html文本

text/plain 响应回来的是普通文本(不能有特殊符号)

text/css 样式文件

application/x-javesctipt js脚本代码

application/xml xml格式的字符串

application/json json格式的字符串

images/jpg images/png 图片

###### 响应主体

##### 响应状态码

作用,告诉浏览器,服务器的响应状态是什么

取值:

1xx:100-199 提示信息

2xx:成功响应

200 ok

3xx:

301 永久重定向

302 临时重定向

304 请求未被修改,命中了缓存

4xx:

404 NOT Found 请求资源不存在

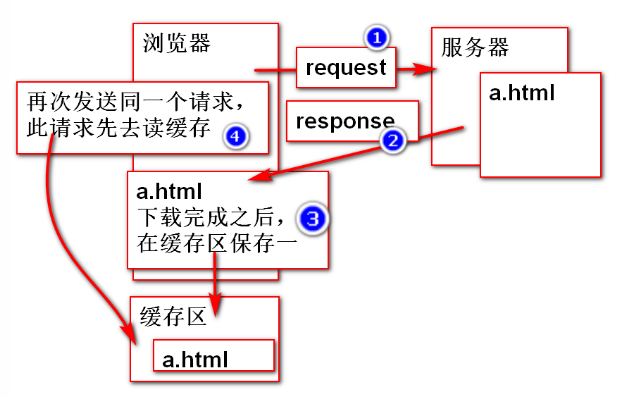
403 Forbidden 权限不够

405 Method not Allowed 请求方法不被允许

5xx: 无服务器允许错误

500 服务器内部错误

##### 缓存

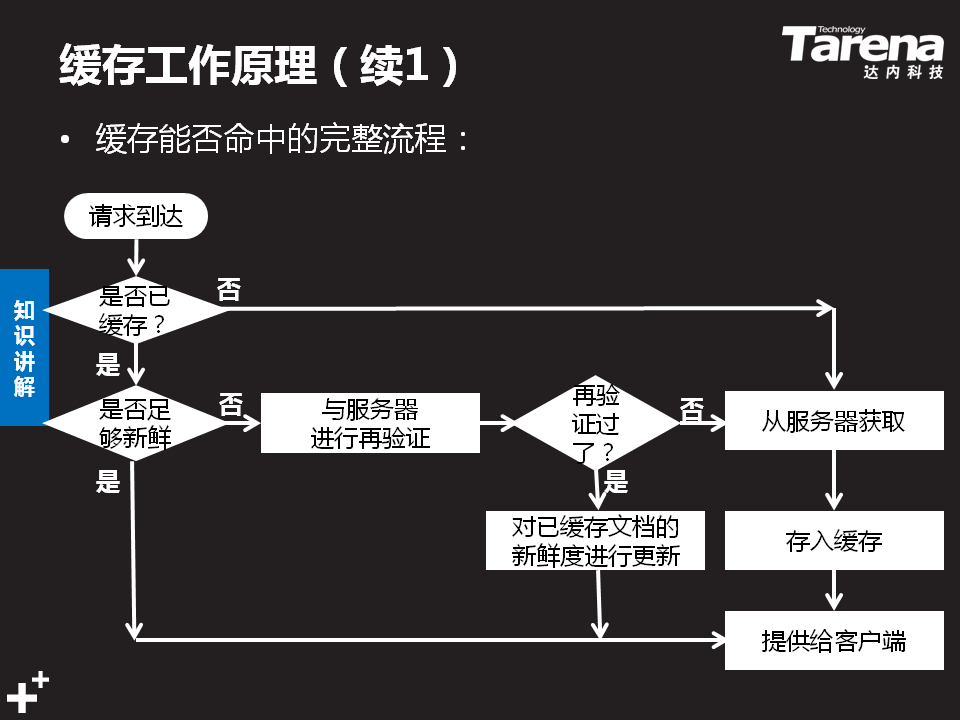


客户端,将服务器响应回来的数据,进行自动的保存；当再次访问的时候,直接使用保存好的数据。

优点:

1. 减少冗余数据传输,节省客户端浏览
2. 节省服务器带宽
3. 降低了对服务器资源的消耗和运行要求
4. 降低了由远距离传输造成的延时加载

###### 缓存完整流程



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

##### 与缓存相关的消息头

###### Cache-Control http/1.1的用法

从服务器将文件传到客户端时起，此文档处于新鲜的秒数，是一个相对时间。

语法：Cache-Control:max-age=秒数（）

Cache-Control:no-Cache; 不缓存

###### Expires http/1.0用法

制定过期的确切时间

Expire是：Month，08 Oct 2018 08:18:48 GMT

##### 在网页中添加缓存

<meta http-equiv"消息头" content="值">

例:<meta http-equiv="Cache-Control" content="max-age=3600">

##### HTTP的性能优化

###### HTTP连接过程

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

###### HTTP连接性能的优化

1. 减少连接的创建次数(开启持久连接)
2. 减少请求的次数
3. 提高服务器端的运行速度
4. 尽可能的减少响应数据的长度

###### 安全的http协议(了解)

HTTPS协议,安全版本的HTTP

SSL,为数据通讯提供了安全支持

1. 客户端发送请求消息时,在SSL层加密

服务器接收到加密文件,在SSL层解密

1. 服务器发送响应消息时,在SSL层加密

客户端接收到加密文件,在SSL层解密,得到响应明文,解析响应内容

### DOM操作

#### 完整的JavaScript组成

1. JS核心:ECMA Script
2. DOM:doctumento bject model 文档对象模型

PS:在html页面中找标签,并往标签中放数据

1. BOM:操作浏览器

#### 使用js获取页面上的某个元素

1. 元素要设置id属性

<div id="d1">我是做实验用的div</div>

1. 在js中,通过document的getElementById("d1");
2. 获取/设置某个元素的内容

DOM的对象有一个属性,innerHTML

表示的就是标签内所有的内容

console.log(ele.innerHTML);

ele.innerHTML="快下课了,开心吗";

练习:在网页中新建一个div元素,元素的内容是假文。添加一个按钮,点击时,获取div中的文字,并打印在控制台；再添加一个按钮,点击后将div的内容变为,"我的第一个dom操作"

<body>

<div id='a'>

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quasi, reiciendis voluptates saepe excepturi pariatur sequi architecto sunt omnis odio nihil quis hic quo laudantium! Quos, atque assumenda velit temporibus sint!

</div>

<input type="button" value="出现吧" onclick="show()">

<input type="button" value="变换吧" onclick="change()">

<script>

function show(){

var ele=document.getElementById('a');

console.log(ele);

}

function change(){

ele=document.getElementById('a');

console.log(ele.innerHTML);

ele.innerHTML="我的第一个dom操作";

}

</script>

</body>

|  |
| --- |
| Request Message 客户端向服务器发送的请求消息  Response Message 服务器向客户端返回的响应消息 |

|  |
| --- |
| 双标签内的信息叫"内容"  单标签内的信息叫"值" |

#### 获取/设置表单控件的数据

DOM中,表单控件的数据是通过DOM对象的value属性来获取的

1. 通过表单控件的id获取表单控件对象
2. 通过控件对象value属性获取、设置表单控件的值

<input type="text" id="t1" value='dangdang'>

<input type="button" value="改变内容" onclick="doChange()">

function doChange(){

var oText=document.getElementById("t1");

oText.value="dingding";

}

练习:在02\_ex.html中新建一个input输入框,添加一个按钮;点击按钮时,将input输入框中的值获取,把此值显示到按钮上

<input type="text" id="d1">

<input type="button" value="获取" onclick="doChange()">

<div id='d1'>input在我这显示</div>

<script>

function doChange(){

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

console.log(oText.value);

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

d1.innerHTML=oText.value;

}

</script>

总结:

1. 通过id获取标签对象

var ele=document.getElemerntById("id");

1. 通过对象的innerHTML属性获取非表单标签的内容

ele.innerHTML="";

1. 如果是表单元素,通过value属性

ele.value="";

#### 获取/设置表单控件的数据

##### 关于方法名的要求

1. 只能包含 字母,数字, \_ $
2. 只能用字符 \_ $ 开头
3. 不能使用系统关键字

##### 简化

创建一个common.js文件,写下如下代码

在每个想使用这个方法的html文件,导入这个js

function $(id){

var ele=document.getElementById(id);

return ele;

}

导入js文件

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

#### HTML的事件

##### 什么是事件

允许通过用户的行为来激发的操作

##### 要学习的事件

onclick="fn()" 按钮单击事件

onblur="fn()" 文本框/密码框 失去焦点的事件

onfocus="fn()" 文本框/密码框 获取焦点时的事件

onload="fn()" 页面加载时要执行的操作

练习:

<p>用户密码：

<input type="password" id="upwd" onblur="upwd\_blur()">

<span id="showUpwd"></span>

</p>

密码框获取焦点--->span显示“用户密码为8-12位”

密码框失去焦点-->验证密码框是否为空

为空，span显示"密码不能为空"

不为空，span显示"通过"

<body onload='page\_focus()'>

<p>用户名:

<input type="text" id='uname' onblur='btn\_blur()'>

<span id='showUname'>显示用户名</span>

</p>

<p>用户密码:

<input type="password" id='upwd' onblur="upwd\_blur()">

<span id='showUpwd'>显示用户密码</span>

</p>

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

<script>

//页面加载完成,马上调用

function page\_focus(){

$("showUname").innerHTML="用户名6-12位";

$("showUpwd").innerHTML="用户密码8-12位";

}

//失去焦点

function btn\_blur(){

//验证文本框中是否输入了用户名

//获取文本框的值

var uname=$('uname').value;

//验证值是否为空

if (uname==''){

//1.为空,显示"用户名不能为空"

$("showUname").innerHTML='用户名不能为空';

}else{

//2.不为空,span显示"通过"

$("showUname").innerHTML="通过";

}

}

//失去焦点

function upwd\_blur(){

var upwd=$('upwd').value;

//验证值是否为空

if(upwd==''){

//1.为空,显示"用户名不能为空"

$('showUpwd').innerHTML='密码不能为空';

}else{

//2.不为空,span显示"通过"

$('showUpwd').innerHTML='通过';

}

}

</script>

</body>

## AJAX

### 名词解释

#### 同步(Synchronous)

在一个任务进行中,不能开启其他的任务.

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

出现场合:

1. 地址栏输入网址,访问网页(网速不好时,很明显)
2. a标签跳转的时候
3. submit表单提交

(跳转之后,前面的表单页面不能做任何事情了)

#### 异步(Asynchronous)

在一个任务开启时,可以进行其他的任务.

异步的访问:浏览器在想服务器发送请求时,不耽误用户在页面上做其他操作.

出现场合:

1. 百度搜索的自动提示
2. 股票走势图
3. 聊天室
4. 用户验证

#### 什么是AJAX

AJAX:Asynchronous JavaScript And Xml

异步的 JS 和 xml

本质:使用js提供的异步对象(XMLHttpRequest)

异步的向服务器发送请求,

并且接受响应回来的数据

ajax中可以无刷新的效果更改页面的局部内容

### 异步请求的步骤

1. 创建异步对象 XMLHttpRequest
2. 绑定监听事件——状态码
3. 创建请求，打开链接
4. 发送请求

#### 创建异步对象

1. 标准创建异步对象

var xhr=new XMLHttpRequest();

1. IE8以下版本,创建异步对象

var xhr=new ActiveXObject("Microsoft.XMLHttp");

1. 通过一个全局的属性,可以判断浏览器是否支持标准创建

window.XMLHttpRequest

1. 若浏览器不支持标准创建,window.XMLHttpRequest是null

if(window.XMLHttpRequest){

var xhr=new XMLHttpRequest();

}else{

var xhr=new ActiveXObject("Microsoft.XMLHttp");

}

注:对于js 0,'',null 均为false

练习:在common.js创建一个createXhr()函数,函数内容,判断浏览器是否支持标准异步对象创建;如果支持标准,返回一个xhr的对象;如果不支持标准创建,返回一个ie8以下的异步对象

function createXhr(){

var xhr=null;

if(window.XMLHttpRequest){

xhr=new XMLHttpRequset();

}else{

xhr=new ActiveObject("Microsoft.XMLHttp");

}

return xhr;

}

#### 异步对象的属性

##### readyState属性

用于表示xhr对象的请求状态

0：请求尚未初始化

1：已经打开服务器的连接，请求正在发送中...

2：接收响应头信息

3：接收响应主体

4：接收响应数据成功

注意：只有当readyState的值为4的时候，才表示所有数据接收完毕

##### status属性

表示服务器的响应状态

值：200

当status的值为200的时候，服务器已经正确处理了请求，并且给出了响应

##### onreadystatechange——事件

当xhr的readyState属性值发生改变时，自动激发的操作

xhr.onreadystatechange=function(){

//什么时候响应正确的能够接收到

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

//处理响应

var result=**xhr.responseText**;

}

}

注：监听，为系统自动做出的反应

function(){},为匿名函数

加粗处是有可能根据需求而改变的地方

#### 创建请求,打开链接

xhr.open(merhod,url,isAsyn);

作用:打开链接,提交请求

method:请求方式 string类型(get/post)

url:请求的地址 string类型

isAsyn:Boolean类型 指定采取同步false,还是采取异步true请求

#### 发送请求

xhr.send(body);

body:请求主体

如果没有请求主体,body位置放null(get方法)

如果有请求主体,就放到body的位置(post方法)

例:127.0.0.1:3000/demo/myAjax响应输出一句话"这是我的第一个ajax练习"

1.在nodejs服务器中，routes文件中，创建demo.js

2.在demo.js中

//1.ajax\_demo

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

res.send("这是我的第一个ajax程序");

});

3.在主文件中app.js中添加路由

const demo=require("./routes/demo.js");

//测试案例的路由挂载

app.use("/demo",demo);

4.开启服务器，在浏览器地址栏测试

http://127.0.0.1:3000/demo/myAjax

5.创建文件夹myEx文件夹，在app.js中托管静态资源

6.在myEx中，创建01\_ajax\_demo.html

function getMsg(){

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var result=xhr.responseText;

console.log(result);

}

}

xhr.open("get","/demo/myAjax",true);

xhr.send(null);

}

#### ajax使用步骤总结

##### 创建异步对象 XMLHttpRequest

var xhr=null;

if(window.XMLHttpRequest){

xhr=new XMLHttpRequest();

}else{

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

}

##### 绑定监听事件

xhr.onreadystatechange=function(){

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

var result=xhr.responseText;

//操作响应数据

}

}

##### 创建请求，打开链接

xhr.open(method,url,isAsyn);

##### 发送请求

xhr.send(body);

练习:使用上练习的demo.js,在myEx创建02\_ex.html,创建一个按钮,点击按钮获得接口myAjax

把接受回来的响应结果,放入一个div显示

<input type="button" value='异步请求服务器' onclick="getMsg()">

<div id='d1'>还是测试用的</div>

<script src='common.js'></script>

<script>

function getMsg(){

//1.

varxhr=createXhr();

//2.

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

$('d1').innerHTML=res;

}

}

//3.

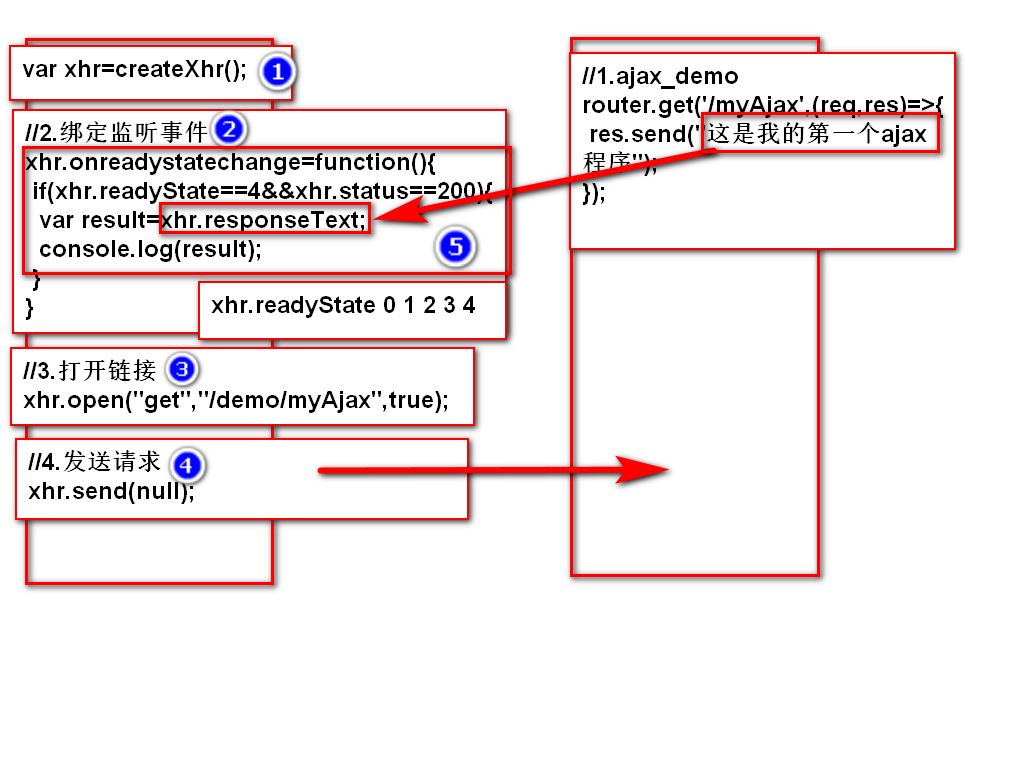
xhr.open('get','/demo/myAjax',true);

//4.

xhr.send(null);

}

</script>



##### 带参数的get请求

//3.打开链接

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

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

//console.log(uname+"...."+upwd);

xhr.open("get","/demo/login?uname="+uname+"&upwd="+upwd,true);

## 项目二

### 准备工作

#### 建立服务器

创建pool.js文件,输入如下代码:

//创建MySQL连接

const mysql=require('mysql');

var pool=mysql.createPool({

host:'127.0.0.1',

port:'3306',

user:'root',

password:'',

database:'xz',

connectionLimit:15

});

//将来要被其他的模块使用

module.exports=pool;

#### 建立路由器

在routes文件夹,建立mypro.js文件,输入如下代码

//引入MySQL连接模块

//哪一个模块中操作SQL语句,就引入连接

const pool=require('../pool.js');//../表上一级

//express下的路由器

const express=require ('express');

//创建路由器

var router=express.Router();

//往路由器中添加路由

......

//导出路由器

module.exports=router;

#### 建立挂载路由文件

建立app.js文件,输入如下代码

//导入路由器

const express=require('express');

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

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

//创建Web服务器

var app=express();

app.listen(3000,function(){

console.log('启动服务器');

});

//使用body-parser中间件

app.use(bodyParser.urlencoded({

extended:false

}));

//托管静态文件到public目录

app.use(express.static('mypro'));

//把路由器挂载到 /mypro下

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

#### 建立通用js工具库

判断用户浏览器所使用的标准

在mypro文件夹建立common.js文件,输入如下代码:

function createXhr(){

var xhr=null;

if(window.XMLHttpRequest){

xhr=new XMLHttpRequest();

}else{

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

}

return xhr;

}

### 建立登录界面

接收前端传递过来的用户名和密码

将用户和密码 到数据库查询

返回提示 登录成功/用户名或密码错误

#### 建立前端文件

在网页中添加用户名和密码输入框及按钮

点击按钮时 异步向后端发起请求，完成登录 post方法

在mypro文件夹建立01\_login.html文件,输入如下代码:

<body>

<p>登录:

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

</p>

<p>密码:

<input type="password" id="upwd">

</p>

<p>

<input type="button" value='登录' onclick='login()'>

</p>

<script src='common.js'></script>

<script>

function login(){

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

console.log(res);

}

}

xhr.open('post','/mypro/login',true);

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

var uname=$('uname').value;

var upwd=$('upwd').value;

var formData="uname="+uname+"&upwd="+upwd;

xhr.send(formData);

}

</script>

</body>

#### 建立后台文件(在myPro.js中定义接口login)

在mypro.js文件,输入如下代码:

//1.login

router.post('/login',(req,res)=>{

var $uname=req.body.uname;

if (!$uname){

res.send({code:401,msg:'用户名不存在'});

return;

}

var $upwd=req.body.upwd;

if (!$upwd){

res.send({code:402,msg:'用户密码不存在'});

return;

}

var sql='select \* from xz\_user where uname=? and upwd=?';

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

if (err) throw err;

if (result.length>0){

res.send("right");

}else{

res.send('error');

}

});

});

### 建立用户列表界面

#### 设立基本功能

##### 建立前端页面

在mypro文件夹中，创建网页02\_list.html

网页加载时发起异步请求，

获取list接口响应回来的数据

以表格的形式，显示在网页中

在02\_list.html中

<body onload="loadUsers()">

<table border="1px" width="800px" align="center">

<thead>

<th>登录名称</th>

<th>电子邮箱</th>

<th>联系方式</th>

<th>用户名称</th>

<th>用户性别</th>

<th>操作</th>

</thead>

<tbody id="tbody"></tbody>

</table>

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

<script>

function loadUsers(){

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

var users=JSON.parse(res);

//console.log(users);

var html='';

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

html+="<tr>"

html+="<td>"+users[i].uname+"</td>"

html+="<td>"+users[i].email+"</td>"

html+="<td>"+users[i].phone+"</td>"

html+="<td>"+users[i].user\_name+"</td>"

//对性别做逻辑判断,不能向用户展示出1和0

var gender='';

if (users[i].gender==1){

gender="男";

}else if(users[i].gender==0){

gender="女";

}else{

gender="不详";

}

html+="<td>"+gender+"</td>"

html+="<td>"

html+="<a href="">删除 |</a>"

html+="<a href=""> 修改</a>"

html+="</td>"

html+="</tr>"

}

$("tbody").innerHTML=html;

}

}

xhr.open("get","/mypro/list",true);

xhr.send(null);

}

##### 建立后台文件(在mypro.js中定义接口list)

在mypro.js文件,输入如下代码:

//2.查询用户表的所有数据

查询用户表的所有数据，并响应给前端

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

var sql="select \* from xz\_user";

pool.query(sql,(err,result)=>{

if (err) throw err;

res.send(result);

});

});

#### 设立删除功能

在mypro.js中获取前提交的uid(用get方法),并将uid作为条件拿到数据库中完成删除用户功能

##### 修改前端页面

接上步,在02\_list.html中,输入如下代码:

在删除<a>标签处,修改为如下代码(标红处):

建立连接端口

html+="<a href='javascript:deleteUser("+users[i].uid+")'>删除 |</a>"

在程序末尾添加如下代码:

//删除

function deleteUser(uid){

//alert(uid);

//1.先创建异步对象

var xhr=createXhr();

//2.绑定监听事件

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

if (res=='1'){

alert("删除成功!");

loadUsers();//删除成功后重新加载

}else{

alert("删除失败!");

}

}

}

//3.打开链接

xhr.open('get','/mypro/deleteUser?uid='+uid,true);

//4.发送请求

xhr.send(null);

}

##### 在后台文件(在mypro.js中定义接口deleteUser)

在mypro.js文件,输入如下代码:

//3.删除用户

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

var $uid=req.query.uid;

if (!$uid){

res.send('不能为空');

return;

}

var sql="DELETE FROM xz\_user WHERE uid=?";

pool.query(sql,[$uid],(err,result)=>{

if (err) throw err;

res.send('1');//成功删除

});

});

#### 设立修改功能

建立连接端口

在修改<a>标签处,修改为如下代码(标红处):

html+="<a href='03\_selUpdate.html?uid="+users[i].uid+"'> 修改</a>"

跳转页面见下部

### 建立修改页面

#### 建立基本功能

##### 建立前端页面

新建一个03\_selUpdate.html网页，在当前网页中创建如下控件,控件用于显示用户的信息（根据uid查询当前用户信息，并绑定给以下控件），当点击“修改”按钮时，将修改后的信息，提交给服务器完成修改功能。

登录名称：uname

登录密码：upwd

用户邮箱：email

联系方式：phone

真实姓名：user\_name

用户性别：gender

uid:不用显示给用户看设置type="hidden"

新建03\_selUpdate.html文件,输入如下代码:

<body onload="getInfo()">

<!--body中没见过的编码-->

<form>

<p>用户姓名：

<input type="text" name='uname' id='uname'>

</p>

<p>用户密码：

<input type="text" name='upwd' id='upwd'>

</p>

<p>用户邮箱：

<input type="text" name='email' id='email'>

</p>

<p>联系电话：

<input type="text" name='phone' id='phone'>

</p>

<p>真实姓名：

<input type="text" name='user\_name' id='user\_name'>

</p>

<p>用户性别：

<input type="radio" name='gender' id='man' value='1'><label for="man">男</label>

<input type="radio" name='gender' id='woman' value='0'><label for="woman">女</label>

<input type="radio" name='gender' id='other' value='2'><label for="other">保密</label>

</p>

<p>

<input type="hidden" name="uid" id="uid">

</p>

<p>

<input type="submit" value="修改">

</p>

</form>

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

<script>

function getInfo(){

//获取url中的参数uid的值

var urlPara=new URLSearchParams(location.search);//urlPara自定义,和下面对应;URLSearchParams(location.search)固定用法

var uid=urlPara.get('uid');

//console.log("用户的id值:"+uid);//测试代码

//1.创建异步对象

var xhr=createXhr();

//2.绑定监听事件

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

console.log(typeof(res));//string类型

//解析json字符串

var user=JSON.parse(res);

$("uname").value=user.uname;

$("upwd").value=user.upwd;

$("email").value=user.email;

$("phone").value=user.phone;

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

if (user.gender==1){

$("man").checked=true;

}else if (user.gender==0){

$("woman").checked=true;

}else{

$("other").checked=true;

}

//用户id

$("uid").value=user.uid;

}

}

//3.打开链接

xhr.open("get",'/mypro/selUser?uid='+uid,true);

//4.发送器请求

xhr.send(null);

}

</script>

</body>

##### 建立后台文件(在mypro.js中定义接口selUser)

在mypro.js文件输入如下代码:

//完成跳转显示该用户信息功能

//在02\_list.html文件点击任意用户"修改"按键,可跳转至03\_selUpdate.html页面,显示对应用户信息(uid=?)

//4.根据uid查询当前用户的所有信息

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

var $uid=req.query.uid;

if (!$uid){

res.send('uid 不存在');

return;

}

var sql="select \* from xz\_user where uid=?";

pool.query(sql,[$uid],(err,result)=>{

if (result.length>0){

res.send(result[0]);

}else{

res.send("无用户信息");

}

});

});

#### 建立修改功能

点击修改按钮,依托form表单将数据提交给服务器端,在mypro.js中创建一个updateUser接口(post方法),获取前端网页提交的用户信息,并通过sql完成修改功能。

##### 修改前端页面

接上步,在03\_selUpdate.html中,输入如下代码:

在<form>标签处,修改为如下代码(标红处):

建立连接端口

<form action="/mypro/updateUser" method="post">

##### 建立后台文件(在mypro.js中定义接口updateUser)

//完成修改功能

//5.修改成功

router.post('/updateUser',(req,res)=>{

var $uid=req.body.uid;

var $uname=req.body.uname;

var $upwd=req.body.upwd;

var $email=req.body.email;

var $phone=req.body.phone;

var $user\_name=req.body.user\_name;

var $gender=req.body.gender;

//验证不能为空

if (!$uid){

res.send({code:400,msg:'uid required'});

return;

}

if (!$uname){

res.send({code:401,msg:'uname required'});

return;

}

if (!$upwd){

res.send({code:401,msg:'upwd required'});

return;

}

if (!$email){

res.send({code:402,msg:'email required'});

return;

}

if (!$phone){

res.send({code:403,msg:'phone required'});

return;

}

if (!$user\_name){

res.send({code:404,msg:'user\_name required'});

return;

}

if (!$gender){

res.send({code:405,msg:'gender required'});

return;

}

var sql="UPDATE xz\_user SET uname=?,upwd=?,email=?,phone=?,user\_name=?,gender=? WHERE uid=?";

pool.query(sql,[$uname,$upwd,$email,$phone,$user\_name,$gender,$uid],(err,result)=>{

if (err) throw err;

//res.send('修改成功');//提示修改成功

res.send("<script>alert('修改成功');location.href='http://localhost:3000/02\_list.html';</script>");//可自动调回原页面

});

});

### 注册功能

#### 设立前端页面

新建一个04\_register.html,在网页中添加如下控件：

登录名称：

登录密码：

用户邮箱：

联系方式：

真实姓名：

用户性别：

注册按钮：

新建04\_register.html,输入如下代码:

<p>登录名称:

<input type="text" id="uname" onblur="checkUname()">

<span id="showUname">请输入注册用户名</span>

</p>

<p>登录密码:

<input type="password" id='upwd'>

</p>

<p>确认密码:

<input type="password" id='cpwd' onblur="checkPwd()">

<span id="showPwd">请输入一致的密码</span>

</p>

<p>联系邮箱:

<input type="email" id='email' onblur="checkEmail()">

<span id="showEmail">请输入正确的邮箱</span>

</p>

<p>联系电话:

<input type="text" id='phone'>

</p>

<p>真实姓名:

<input type="text" id='user\_name'>

</p>

<p>用户性别:

<select value id="gender">

<option value="1">男</option>

<option value="0">女</option>

<option value="2">其他</option>

</select>

</p>

<p>

<input type="button" value="注册" onclick="register()">

</p>

#### 用户名验证

登录名称输入框失去焦点时验证名称是否已存在，用异步的方式将用户名提交给服务器端，在服务器端myPro.js中创建一个checkUname接口，在接口中获取前端提交的用户名称，并去数据库中查询当前用户名称是否已存在。

##### 修改前端页面

接上步,在04\_register.html,输入如下代码

<script src='common.js'></script>

<script>

var isRegister=false;//防止输入栏为空报true

var isReEmail=false;

var isRePwd=false;

function checkUname(){//用户名查重

//console.log(isRegister);

//异步请求4步

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

if (res=="1"){//可以注册

$("showUname").innerHTML="可以注册";

isRegister=true;

}else if (res=="0"){//不可以注册

$("showUname").innerHTML="用户名被占用";

isRegister=false;

}else{

$("showUname").innerHTML="用户名不能为空";

}

console.log(isRegister);

}

}

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

xhr.open('get','/mypro/checkUname?uname='+uname,true);

xhr.send(null);

}

##### 建立后台文件(在mypro.js中定义接口checkUname)

在mypro.js文件输入如下代码:

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

var $uname=req.query.uname;

if (!$uname){

res.send("用户名不能为空");

return;

}

var sql='select \* from xz\_user where uname=?';

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

if (err) throw err;

if (result.length>0){

res.send("0");//用户名已存在

}else{

res.send("1");//用户名可用

}

});

});

#### 用户注册

当点击注册按钮时需要判断用户名称是否已存在，如果已存在，不让注册并提示“请检查注册信息”，否则允许注册并提示“注册成功”。

##### 修改前端页面

接上步,在04\_register.html,输入如下代码

function register(){//进行注册

if(isRegister&&isReEmail&&isRePwd==true){//判断是否注册的前提条件,开始注册

//发送异步请求

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

alert(res);

}

}

xhr.open('post','/mypro/register',true);

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

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

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

var email=$("email").value;

var phone=$("phone").value;

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

var gender=$("gender").value;

var formdata="uname="+uname+"&upwd="+upwd+"&email="+email+"&phone="+phone+"&user\_name="+user\_name+"&gender="+gender;

xhr.send(formdata);

}

}

##### 建立后台文件(在mypro.js中定义接口register)

在mypro.js文件输入如下代码:

router.post('/register',(req,res)=>{

var $uname=req.body.uname;

if (!$uname){

res.send('uname err');

return;

}

var $upwd=req.body.upwd;

if (!$upwd){

res.send('upwd err');

return;

}

var $email=req.body.email;

if (!$email){

res.send('email err');

return;

}

var $phone=req.body.phone;

if (!$phone){

res.send('phone err');

return;

}

var $user\_name=req.body.user\_name;

if (!$user\_name){

res.send('user\_name err');

return;

}

var $gender=req.body.gender;

if (!$gender){

res.send('gender err');

return;

}

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

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

if (err) throw err;

res.send('success register');

});

});

### 邮箱验证功能

#### 修改前端页面

接上步,在04\_register.html,输入如下代码

function checkEmail(){//邮箱的查重

var xhr=createXhr();

xhr.onreadystatechange=function(){

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

var res=xhr.responseText;

if (res=='0'){//可以注册

isReEmail=true;

$("showEmail").innerHTML="可以注册";

}else if (res=='1'){//不可以注册

$("showEmail").innerHTML="不可以注册";

isReEmail=false;

}

console.log(isReEmail);

}

}

var email=$("email").value;

xhr.open('get','/mypro/doubleEmail?email='+email,true);

xhr.send(null);

}

#### 建立后台文件(在mypro.js中定义接口doubleEmail)

在mypro.js文件输入如下代码:

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

var $email=req.query.email;

if (!$email){

res.send("email err");

return;

}

var sql='select \* from xz\_user where email=?';

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

if (err) throw err;

if (result.length>0){

res.send('1');//已被占用

}else {

res.send('0');//可以使用

}

});

});

### 两次密码输入一致验证功能

#### 修改前端页面

接上步,在04\_register.html,输入如下代码

function checkPwd(){//两次密码验证

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

var cpwd=$("cpwd").value;

if (cpwd==upwd){

isRePwd=true;

$("showPwd").innerHTML="两次输入一致";

}else{

isRePwd=false;

$("showPwd").innerHTML="两次输入不一致，请重新输入";

}

console.log(isRePwd);

}

# CSS3

## CSS3概述

### 名词解释

CSS:Cascaing style sheets

层叠样式表,级联样式表。简称样式表

### 作用

设置HTML与CSS的关系

提高代码的重用性和可维护性

实现了将内容与表现分离

### HTML与CSS的关系

HTML:负责网页的搭建，内容显示

CSS:负责网页的修饰，样式的构建

### HTML与CSS属性的使用原则

w3c建议我们尽量使用CSS的方式来取代HTML属性

1. 实现将内容与表现分离
2. 若为HTML所特有的属性，则使用HTML属性

## CSS3 基础

### CSS3概述

#### 名词解释

CSS:Cascaing style sheets

层叠样式表，级联样式表。简称样式表

#### 作用

设置HTML与CSS的关系

提高代码的重用性和可维护性

实现了将内容与表现分离

#### HTML与CSS的关系

HTML:负责网页的搭建，内容显示

CSS:负责网页的修饰，样式的构建

#### HTML与CSS属性的使用原则

w3c建议我们尽量使用CSS的方式来取代HTML属性

1. 实现将内容与表现分离
2. 若为HTML所特有的属性，则使用HTML属性

### CSS3的语法概述

#### CSS的使用方式

1. 内联样式

样式定义在单个的HTML元素中

1. 内部样式

样式定义在HTML页的头元素中

1. 外部样式

将样式定义在一个外部的CSS文件中（.css文件）

由HTML页面引用样式表文件

##### 行内样式\内联样式

<any style="样式的声明"></any>

样式的声明:由样式属性和值组成

样式属性和值之间用冒号(:)连接

多个样式声明，用分号(;)分隔

<p style="color:red;font-size:30px">小了白了兔</p>

(标签属性="样式属性1:值1;样式属性2:值2")

###### 常见样式属性

1. 设置文本颜色

样式属性:color

取值:合法的颜色值(颜色的英文单词)

1. 设置字体大小

样式属性:font-size

取值:以px为单位的数字

1. 背景颜色

样式属性:background

取值:合法的颜色

###### 内联样式的特点

1. 不能被重复使用
2. 内联样式优先级最高

element.style{}

项目中，基本不使用内联样式

只有在学习和测试时才使用

##### 内部样式

在head标签中添加style标签，在style中定义该网页的样式

语法:

<head>

<style>

样式规则:

</style>

</head>

样式规则:

选择器{样式属性:值;}

选择器:规范了页面中哪些元素能够使用定义好的样式。

就是把声明好的样式匹配给页面目标元素。

样式属性:包含一个或者多个属性/值

<head>

<style>

div {

width:300px;

height:100px;

background:yellow;

font-size:30px;

color:red

}

</style>

</head>

<body>

<div>被内部样式所改变的内容</div>

</body>

总结:

项目中，内部样式使用不多

只能在同一个页面中使用

多在学习和测试中

##### 外部样式

独立于任何网页的位置

声明一个样式表文件.css为后缀

在.css中声明样式

使用步骤:

1. 创建样式表文件\*.css
2. 在css文件中编写样式
3. 在网页头部引用css文件

<head><link rel="stylesheet" href="css文件的路径"></head>

注意:

<style>(内部样式)在<link>(外部样式)的上面或下面，产生的效果不同

其效果的产生和与<body>标签的远近有关，离<body>标签近的优先级高。

练习:

新建一个网页，02\_ex.html，页面添加3个h3，内容随意;

1. 是引用外部样式定义h3的字体颜色为蓝色blue;背景颜色为橙色orange，观察页面效果
2. 使用内部样式，编写h3的字体颜色为红色red
3. 使用内联样式，编写第一个h3的背景颜色为黄色yellow

02\_ex.html代码如下:

<head>

<style>

h3{

color:red;

}

</style>

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

</head>

<body>

<h3 style="background:yellow">测试一</h3>

<h3>测试二</h3>

<h3>测试三</h3>

</body>

my.css代码:

h3{

color:blue;

background:orange;

}

效果:测试一黄底红字;测试二、测试三橙底红字

测试可知:

内联样式优先级最高，且当样式冲突时，显示优先级高的，当样式不冲突时相互间不影响，均显示;

外部样式与内部样式，其优先级与哪个在下面有关(最靠近<body>标签的优先级高，即“就近原则”)，且优先级高的样式会完全覆盖优先级低的，不管是否有样式效果的冲突，均只显示高优先级的样式。

#### CSS语法规范

1. 内联样式：由样式表声明组成
2. 样式表(内部样式或者外部样式)

由多个样式规则组成

每个样式规则有两个部分:选择器和样式声明

#### CSS的特性

##### 继承性

大部分的css效果可以直接被子元素继承

必须是父子结构(有层级嵌套)，子继承父

##### 层叠性

可以为一个元素定义多个样式规则

当样式属性没有发生冲突，这些样式都可以作用到当前元素

### 优先级的问题

如果多个样式作用同一个元素，样式属性发生冲突，按照样式规则的优先级，优先级高的发生作用，优先级低的无效

1. 浏览器默认样式--最低
2. 外部和内部样式--就近原则
3. 内联样式--最高

相同样式，如果重复定义，以最后一次的定义为准。

### 调整优先级

!important (可调至最高优先级)

放在样式值之后，与值之间用空格隔开

作用，调整样式的优先级

调整过后的优先级，比内联样式还高

h3{

color:red !important;

}

练习:

03\_ex.html页面中添加一个p，用内部样式设置文字颜色为蓝色;字体大小为24xp，用外部样式设置字体大小为40px。字体颜色为红色。将外部样式引入到网页中。

查看网页效果，然后改变内部样式为外部样式的位置。

再次查看页面效果。并尝试使用!inportant来调整样式优先级

<link>就近，则显示外部样式;

<style>就近，则显示内部样式;

使用!important，则所在属性内的该条样式提升。

### CSS的基础选择器(重点)

#### 选择器的作用

规范了页面中哪些元素能够使用定义好的样式。

匹配元素！！

#### 通用选择器 通配符

可以与页面中任意元素匹配

\*{样式声明}

效率低，尽量少用。

一般情况下，项目中只会有一种情况使用

|  |
| --- |
| \*{margin:0;padding:0}  所有标签内外边距清零  注:浏览器自带边距，且不同浏览器的距离不同 |

#### 元素选择器

设置页面中某种元素的默认样式

标记名称{样式声明}

div{} p{} span{} header{} footer{}

#### 类选择器

定义页面上某一类或者某些个元素的样式

(公共的样式、谁想用谁拿走)

特点:通过元素class属性来引用

<style>

.my-fontsize60{

font-size:60px;

}

</style>

<body>

<div class="my-fontsize60">小了白了兔</div>

</body>

注:标红的my-fontsize60为自定义名，只需上下一致，但要注意选择器前的"."。

总结：

1. 声明的时候，点不能省略
2. class中没有点
3. 类名不能以数字开头
4. 不能包含特殊字符（除了-和\_）

#### 特殊用法

##### 多类选择器

让元素同时引用多个类的选择器

<any class="类名1 类名2 类名3"></any>

<div class="my-fontsize60 my-colorblue">小了白了兔</div>

##### 分类选择器

将元素选择器和类选择器联合使用

元素选择.类选择器{样式声明}

特点:

优先级变高了

指向更精确

<style>

p.my-coclor{

color:yellow;

}

</style>

<body>

<p class="my-color">测试1</p>

<span class='my-color'>测试2</span>

</body>

p标签实现了样式，span没有实现

只因标红处

练习:

新建一个网页04\_ex.html。网页中添加div和p，内容随意；用类选择器为所有元素设置字体颜色为red；用类选择器为p元素设置背景颜色为黑色

<style>

.my-fontcolor{

color:red;

}

</style>

<body>

<div class="my-fontcolor">样式一</div>

<p class="my-fontcolor" style="background:black">样式二</p>

</body>

样式一和样式二均字体均变红，样式二因有内联样式，故其背景色为黑

#### id选择器

设置指定的id元素样式

仅作用于id属性的值

语法：

选择器前面需要有一个#号

选择器本身则为文档中某个元素的id属性的值

#id{样式声明}

<span id='s1'></span>

#id{color:red}

一个页面中，id值唯一

练习:

在04\_ex.html，添加一对h2标记，内容随意;用id选择器设置第一个h2文本颜色purple;字体为斜体font-style:italic;

<style>

#s2{

color:purple;

font-style:italic;

}

</style>

<body>

<h2 id='s2'>体颜色</h2>

</body>

若有外部样式，其效果同内联样式与外部样式的情况相同。

#### 群组选择器

选择器声明以逗号隔开的选择列表

将一些相同的规则作用于多个元素

式1:

#content,.my-fontcolor,div{

color:red;

}

式2:

#content{

color:red;

}

式3:

.my-fontcolor{

color:red;

}

式4:

div{

color:red;

}

相当于式1(公共选择器)=式2(id选择器)+式3(类选择器)+式4(元素选择器)的结果

#### 后代选择器

通过元素的后代关系（其位置的上下关系），匹配目标元素

后代:一级或者多级都算后代

语法:选择器1 选择器2 选择器3{样式声明}

使用空格来分隔

.d1 .p1 span{

color:purple;

}

<div class="d1">

<span>我是div中的span</span>

<p class="p1">

<span>我是p中的span</span>

</p>

<p class="p1">我是div中的p</p>

</div>

只有符合.d1-.p1-span层级关系的标签样式效果改变。

#### 子代选择器

通过元素的子代关系匹配元素

语法:选择器1>选择器2>选择器3{样式声明}

使用>来分隔

div>span{

background:pink;

}

<div>

<span>测试一</span>

<div>

<span>测试二</span>

</div>

<p>

<div>

<span>测试三</span>

</div>

</p>

</div>

凡是符合div>span这种层级关系的都可改变其样式属性。

#### 伪类选择器

作用:匹配元素不同状态下的选择器

语法:

所有的伪类选择器都是以 : 开头

选择器:伪类选择器{样式声明}

##### 链接伪类

a:link{

color:#f00;/\*red\*/

}

a:visited{

color:#ff0;/\*yellow\*/

}

:link 匹配元素访问之前的状态

:visited 匹配元素访问之后的状态

##### 动态伪类

a:active{

color:#f0f;/\*pink\*/

}

input:focus{

background-color:#ff0;/\*yellow\*/

}

img:hover{

width:500px;

}

:active 匹配元素被激活的状态

:focus 匹配元素获取焦点时的状态

:hover 匹配鼠标悬停在元素上时的状态

练习:02\_ex.html，页面中添加一个a标签，内容随意，设置样式如下:

1. 未被访问的字体为黑色，没有下划线text-decoration:none;
2. 鼠标悬停时，文本颜色为红色
3. 被激活时，颜色为绿色
4. 访问后，颜色为橙色

页面添加input输入框，默认字体为灰色gray，字体为斜体font-style:italic

1. 被激活时，字体非斜体fon-style:normal
2. 获取焦点时，文本颜色为purple

#### 选择器权值

权值:表示当前选择器的重要程度，权值越大优先级越高

!important>内联>id>class>tag(标签)>\*>继承

|  |  |
| --- | --- |
| 选择器 | 权值 |
| !important | >1000 |
| 内联样式 | 1000 |
| id选择器 | 100 |
| 类选择器和伪类选择器 | 10 |
| 元素选择器 | 1 |
| 通用选择器\*{} | 0 |
| 继承的样式 | 无 |

##### 权值的计算（重点）

1. 当一个选择器中包含多个选择器时，需要将所有的选择器权值进行相加。将结果进行比较，权值大的优先级高；
2. 选择器的计算不会超过最大的数量级(阶级问题)；

10个id选择器的优先级，永远不可能达到1000

1. 群组选择器，各算各的，不会相加；
2. 分组选择器，优先级变高；
3. 优先级相同，使用就近原则；
4. ！important，获得最高优先级，

内联样式不能添加！important属性

### 尺寸和边框

#### 尺寸属性

##### 作用

改变元素的宽度和高度

##### 属性

宽度 width:500px;

最大宽度 max-width:800px;

最小宽度 min-width:300px;

取值:px或者%(父级元素的百分比)

高度 height:100px;

最大高度 max-height:

最小高度 min-height:

取值:px或者%

响应式开发用的较多

##### 页面允许设置尺寸的元素

1. 所有的块级元素都可以设置

div p h1-h6 结构标记 列表

1. 行内块可以设置

表单控件元素（单选按钮和多选按钮除外）

1. 本身具有width和height属性的元素，可以设置

img table

1. 大部分行内元素不允许设置修改

#### 尺寸单位(附加知识点)

1. px像素
2. cm厘米
3. in英寸 1in=2.54cm
4. pt磅值 多数用于表示字体大小 1pt=1/72in
5. mm毫米
6. em 相对父级元素乘以的倍数(2em)
7. rem 相对于根元素乘以倍数(body、html)
8. % 相对父级的百分比

#### 合法的颜色(附加知识点)

1. 英文单词 red blue yellow black....
2. 以#号开头，6位16进制数组 #rrggbb 格式
3. 以#号开头，#aabbcc--->#abc #f00

注:

印染三原色:红（品）、黄（黄）、蓝（青）

光学三原色：红、绿、蓝

1. rgb(0-255,0-255,0-255) rgb(0,0,0)
2. rgb(r%,g%,b%)
3. rgba(0-255,0-255,0-255,alpha) alpha=0-1透明度

#### 溢**出的处理**

当内容过多，元素区域过小的时候，就会产生溢出

默认纵向溢出

属性：overflow，overflow-x，overflow-y

取值：

1. visible 溢出部分可见，默认值
2. hidden 溢出部分隐藏
3. scroll 显示滚动条，溢出时滚动条可用(纵向\横向滚动条)
4. auto 自动，溢出时显示滚动条并且可用(只有默认方向的)

#### 边框属性

属性：border:width style color;

width:边框的宽度

style:边框的样式

取值:

solid 实线

dotted 虚线(点型)

dashed 虚线(线型)

double 双实线

color:边框的颜色

取值:合法颜色值 transparent透明色

border:none/0 取消边框

练习:创建一个div id为container 尺寸为200\*200，设置4个方向的边框，1px，点虚线，orange颜色

<style>

#container{

width:200px;

height:200px;

border:1px dotted orange;

}

</style>

<div id="container"></div>

##### 边框的单边设置

border-top/boder-right/border-bottom/border-left:

width style color;

##### 单属性设置

border-width/border-style/border-color 同时设置4条边某个属性

只设置1条边某一个属性

border-top-width

border-bottom-style

border-right-color

练习:制作4色正方形

#d1{

width:0px;

height:0;

border-top:30px solid #f00;

border-right:30px solid #0f0;

border-bottom:30px solid #00f;

border-left:30px solid #f0f;

}

<div id='d1'></div>

练习:制作灰色三角形

#d2{

width:0px;

height:0px;

border-top:30px solid #ccc;

border-right:30px solid transparent;

border-left:30px solid transparent;

border-bottom:30px solid transparent;

}

<div id='d2'></div>

##### 边框倒角

将直角倒成圆角

属性:border-radius

取值:

1. 以px为单位的数组;
2. % 设置圆形(>=50%)

单角设置，先写上下，再写左右

border-top-left-radius

border-top-right-radius

border-button-left-radius

border-button-right-radius

##### 边框阴影

属性:box-shadow

取值:

h-shadow v-shadow blur spread color inset

1. h-shadow:水平方向阴影的偏移，必须值

取值为正，阴影往右;取值为负，阴影往左

1. v-shadow:垂直方向阴影的偏移，必须值

取值为正，阴影往下;取值为负，阴影往上

后面的值是可选值，但要按顺序

1. blur：阴影模糊距离。可选值，值越大模糊距离越明显。
2. spread：阴影的尺寸。可选值，指定要在基础阴影上扩出来的大小。
3. color：阴影颜色，可选值，默认黑色。
4. inset：将默认的外阴影改为内阴影。

练习：设置一个圆形的外发光（太阳）

#sun{

width:200px;

height:200px;

border:3px solid #ffd230;

margin:30px auto;

border-radius:50%;

background:#ffa225;

box-shadow:0px 0px 103px 64px #f7db7c;

}

<div id='sun'></div>

##### 轮廓

轮廓：绘制与边框外的线条——边框的边框

outline:width style color;

取消边框：outline:none/0;

练习:设置一个乒乓球拍的效果图

### 框模型

#### 什么是框模型

页面元素都是框

框模型：box model 又称盒子模型

定义了元素框，处理元素的内容，内边距，外边距以及边框的一种计算方式（计算一个元素在页面上真正占据的空间大小）。

##### 框模型的默认计算方式

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

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

内边距：元素内容区域和边框之间的空白距离

外边距：围绕在元素空间外的空白距离（元素与元素之间的距离）

#### 外边距

margin：定义了4个方向的外边距

定义某一个方向的外边距

margin-top、margin-right、margin-bottom、margin-left

取回：

1. 以px为单位的数字
2. %
3. 取值为负数——让元素往相反的方向移动

margin-left

取正值，让元素往右移动；

取负值，让元素往左移动。

margin-top

取正值，让元素往下移动；

取负值，让元素往上移动。

1. auto 自动计算块级元素的外边距

控制块级元素在水平方向居中对齐

##### 简写方式

margin:value;4个方向的外边距

margin:v1 v2; v1上下外边距 v2左右外边距

margin:0 auto; 块级元素水平的写法

margin:v1 v2 v3; v1上外边距；v2左右外边距；v3下外边距

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（顺时针方向）

练习：05\_ex.html页面中创建两个div;都是300\*300;设置背景颜色，自拟

设置两个div之间的间距为50px;第二个div水平居中

##### 自带外边距的元素

blockquote，body，button，dd，dl，dt，fieldset，form，h1，h2，h3，h4，h5，h6，hr，input，legend，li，ol，p，pre，td，textarea，th，ul

##### 外边距的特殊效果(重要)

1. 外边距合并

当两个垂直的外边距相遇时，他们将合并成一个外边距，最终取值为两个外边距中较大的那个值。

1. 行内元素的外边距

行内元素垂直外边距无效（img除外）

水平外边距相遇时，两个值会相加

练习：05\_ex.html，创建两个div;id为d3，d4;尺寸都是200\*200;设置背景颜色，自拟;设置两个div之间的间距为50px.第二个div水平居中

#d3{

width:200px;

height:200px;

background:#f1f;

}

#d4{

width:200px;

height:200px;

background:#1f1;

}

#d5{

width:100px;

height:100px;

background:#357;

margin-top:50px

}

<div id='d3'></div>

<div id='d4'>

<div id='d5'></div>

</div>

1. 外边距溢出——上面练习会出现上边距溢出的效果

|  |
| --- |
| 在特殊条件下，为子元素设置外边距时，会作用于父元素  特殊条件：1.父元素没有上边框  2.第一个子元素设置上边距  解决方案：1.对父元素添加上边框  弊端：影响父元素的高度  2.为父元素添加上内边距  弊端：影响父元素的高度  3.在父元的第一个子元素的位置处，添加一对空的table标签  table有内容则显示，无内容则不显示 |

#### 内边距

不会影响其他元素，但是会改变自己的占地尺寸，元素变大

##### 语法

属性:padding:v1;4个方向的内边距都设置

padding-top

padding-right

padding-bottom

padding-left

取值:以px为单位的数字，或者%

##### 简写方式

padding:v1; 4个方向的内边距

padding:v1 v2; v1:上下 v2:左右

padding:v1 v2 v3; v1:上 v2:左右 v3:下

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

##### box-sizing

作用:指定框模型的计算方式

默认的方式:

实际占地宽度=左外边距+左边框+左内边距+内容宽度+右内边距+右边框+右外边距

取值:

box-sizing:content-box;默认值，采取默认计算方式

box-sizing:border-box;内容区域的尺寸，会包含内边距和边框的尺寸。

计算方式变为：

元素实际占地宽=左外边距+内容区域+右外边距

### 背景

#### 背景图片

background-image：

取值：url(资源路径); 加不加引号都行

ex: background-image:url(../image/08.png);

#### 背景颜色

background-color:

取值:合法颜色值 transparent透明色

注意:背景颜色和背景图片默认都是从**边框内部**开始填充

#### 背景图片的平铺

background-repeat:

取值:

repeat 默认值，背景图平铺

no-repeat 不平铺，背景只显示1个

repeat-x 水平方向上的平铺

repeat-y 垂直方向上的平铺

#### 背景图的定位

background-position:

取值:

1. x y以px为单位的数字

x:水平的位移，正数往右，负数往左

y:垂直的位移，正数往下，负数往上

1. x% y%

0% 0% 左上角

50% 50% 中心

100% 100% 右下角

1. 关键字

x:left/center/right

y:top/center/bottom

#### 背景图尺寸

background-size:

取值:

1. width height 以px为单位数字
2. w% h%
3. cover 将背景图等比放大，直到背景图完全覆盖到元素区域为止(背景图依据数值大的那条边，停止放大)
4. contain 将背景图等比放大，直到背景图碰到元素某一个边为止(背景图依据数值小的那条边，停止放大)

#### 背景图固定

将背景图固定在网页的某个区域

background-attachment:

取值:

1. scroll;默认值，网页滚动时，图片不固定，跟着滚动条跑

特殊情况:当元素内容过多，元素尺寸比较小，会出现内容溢出，如果元素设置了overflow:scroll，那么背景图会被相对元素固定，拖动元素滚动条，图片不动，拖动网页滚动条，图片还是会跑

1. fixed与background-position协作，把背景图片固定在网页某个位置，不管怎么拖动浏览器的滚动条，背景图片都不会动

#### 背景属性的简写方式

在一个属性中，指定多个背景属性

background

取值:

color url() repeat attachment position

例:background:no-repeat url(../image/15.png) #666;

### 渐变

#### 什么是渐变

渐变指多种颜色平缓变化的一种现实效果

#### 渐变的主要因素

色标:一种颜色及其出现的位置

一个渐变效果至少有两个色标

#### 渐变色的分类

1. 线性渐变

以直线的方向来填充渐变色

1. 径向渐变

以圆形的方式实现渐变色

1. 重复渐变

将线性渐变或者径向渐变重复几次出现

#### 线性渐变

background-image:

取值:

linear-gradient(angle,color-point1,color-point2,...)

angle:表示渐变的方向或者角度

1. 关键字

to top/to right/to tottom/to left

1. 角度值

0deg------>to top

90deg---->to right

180deg--->to buttom

270deg--->to left

color-point:

取值:颜色，颜色的位置%

background-image:linear-gradient(0deg,#f00 0%,#0f0 50%,#00f 100%);

颜色 颜色位置的数字px

#### 径向渐变

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

size:半径，以px为单位的数字

position:圆心所在的位置

取值:x y具体数字

x% y% 元素宽高占比

关键字: x--->left/center/right

y--->top/center/bottom

background-image:radial-graient(100px at center center,#f00 0px,#ff0 50%,#0ff);

#### 重复渐变

##### 重复线性渐变

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

当最后一个色标没有充满整个容器，从第一个色标开始重复显示，直至填满。

##### 重复径向渐变

background-image:repeating-radial-gradient(szie at position,color-point1,color-point2...);

注意:在重复渐变中，位置一定要给绝对值(数字px)，不要给%

#### 浏览器的兼容性

各个浏览器新版本都支持渐变属性

Opera12.1、IE10、Firefox16、Chrome26、Safari6.1以上版本

对于不支持的浏览器版本(各个浏览器的旧版本)

可以通过添加浏览器内核前缀的方式，让浏览器兼容

Chrome&Safari

新:线性渐变，不加前缀，不能使用top/ right/ bottom/ left关键字，必须使用to top/ to right/ to bottom/ to left

如果就想使用top/ right/ bottom/ left关键字，需要添加前缀

Firefox：-moz-

IE：-ms-

Opera：-o-

：-webkit-

浏览器兼容性问题：www.caniuse.com

### 文字格式化(重点)

#### 字体属性

font-family

取值：用逗号隔开的多个字体类型

font-family：黑体，微软雅黑，宋体

#### 字体大小

font-size

取值：pt px em rem

#### 字体加粗

font-weight

取值：

bold加粗

normal正常

无单位的整百数字:400-900

lighter变细

#### 字体样式

font-style

取值：

italic斜体

normal正常

#### 小型大写字母

font-variant

取值:small-caps;小型大写字母

normal 正常

#### 字体属性简写模式

font

取值:style、variant、weight、size、family

注意：使用简写方式，最少写size和family，否则无效。

练习：05\_ex.html;定义一个div，内容随意，最好中英文结合，设置一下样式;字体，微软雅黑;文字大小，32px;加粗并斜体;将所有小写字符变为小型大写字符

div{

font:italic small-caps bold 32px '微软雅黑';

}

<div>

啥的洒地ing el alias.

</div>

### 文本格式化

#### 文本颜色

color:合法的颜色值

#### 文本的水平方向排列

text-align

取值:center、right、left、justify(两边对齐)

#### 文本的垂直方向排列

line-height

如果行高的高度大于字体本身大小，那么该行文本将在指定的行高内，呈现垂直居中的效果.

利用这个效果，把行高设置为容器高度，可以让文本在容器中垂直居中.

取值:

以px为单位的数字

不带单位的数字，是字体的倍数

#### 线条修饰

text-decoration

取值:

none无线条修饰

overline上划线

underline下划线

line-through删除线

#### 首行缩进

text-indent

取值:以px为单位的数字

#### 文本阴影

text-shadow

取值:

h-shadow水平偏移量

v-shadow垂直偏移量

blur阴影模糊距离

color颜色

### 表格的样式属性

#### 表格的常用样式属性

1. border
2. padding(margin无效)
3. width、height
4. font-\*、text-\*、line-height
5. background- image color gratient
6. vertical-align

作用：指定单元格数据的垂直对齐方式

取值：top/middle/bottom

练习：在01\_table.html中，创建一个4\*4的表格，表格高宽的都是400px，内容随意.

设置每个单元格的尺寸100\*100

设置表格和单元格的边框2px solid #f00

尝试给每个td设置上外边距20px

单元格小，内容多，自动撑开

内容小，单元格尺寸大，按单元格尺寸显示

#### 合并边框

border-collapse 合并

取值：

collapse合并边框

separate默认值

#### 边框边距

设置单元格之间的距离

border-spacing

取值：

给一个值，水平和垂直间距相同。

给两个值，第一个为水平 第二个为垂直。

注意：此属性想生效，border-collapse值是separate

#### 标题的位置

caption-side

取值：top、bottom

#### table的显示规则

table-layout

作用：用来告诉浏览器如何布局一张表格（就是告诉浏览器，td尺寸的计算方式）

取值：

auto默认值，自动布局表格，td尺寸由内容决定

fixed固定表格布局，td尺寸由设置决定尺寸

注意：所谓固定，是指width的固定。表格内容的过多，会按照每个单词为单位，向下撑开高，但宽固定不变。

#### 自动布局与固定布局对比

|  |  |
| --- | --- |
| 自动布局auto | 固定布局fixed |
| 单元格大小会适应内容 | 单元格的尺寸取决于设定的值 |
| 缺点：表格复杂时，加载较慢 | 优点：任何情况下会加速加载表格 |
| 优点：自动布局比较灵活 | 缺点：固定布局不够灵活 |
| 适用于不确定每列的大小，并且表格不复杂时 | 适用于确定每列尺寸使用 |

### 定位

#### 什么是定位

改变元素在页面中的位置

#### 分类

1. 普通流定位（默认文档流）
2. 浮动定位
3. 相对定位
4. 绝对定位
5. 固定定位

#### 普通流定位

又称文档流定位，页面中元素们的默认显示方式

1. 每个元素在页面都是有自己的空间
2. 每个元素都是从父元素的左上角开始显示
3. 块级元素都是按照从上到下的方式逐个排列，每个元素独占一行
4. 行内元素是多个元素在一行中显示，从左往右逐个排列

#### 浮动定位

##### 什么是浮动&浮动的特点

1. 元素一旦浮动起来，脱离了文档流，不再占用页面空间，其他未浮动元素会上前占位；
2. 浮动元素会停靠在父元素的左边或右边，或跟在其他浮动元素的后面
3. 浮动解决多个块元素在一横行显示的问题

##### 语法

float

取值：

left，让元素停靠在父元素的左侧，或者挨着左侧已有的浮动元素

right，让元素停靠在父元素的右侧，或者挨着右侧已有的浮动元素

##### 浮动引发的特殊情况

1. 当父元素显示不下所有的浮动元素时，最后一个元素将换行(如图1)；但有时也会出现“卡住”的情况(即第一个浮动元素的长度大于第二个浮动元素时)(如图2)。
2. 当元素浮动起来之后，宽度以内容为准（元素本身没定义宽度）(如图3)。
3. 当元素一旦浮动，将变为块级元素，允许修改尺寸，可以设置垂直外边距。
4. 文本行内元素是不会被浮动元素压在下面的，而是巧妙地避开，环绕着浮动元素显示(如图4)。

|  |  |
| --- | --- |
| 图1 | 图2 |
| 图3 | 图4 |

##### 清除浮动

元素不被其他浮动元素覆盖

|  |
| --- |
| clear  取值：  left：当前元素不被左浮动的元素覆盖  right：当前元素不被右浮动的元素覆盖  both：当前元素不被左和右浮动的元素覆盖  none：默认值，不做任何清除 |

##### 子元素浮动（高度坍塌）

块级元素的高度，如果不写，默认被内容撑起。

内部元素浮动，脱离文档流，文档流中的父辈元素不认为自己还有内容了。

所以父元素高为0，发生了高度坍塌。

|  |
| --- |
| 解决方案：   1. 直接给父元素写高度   弊端：不是每次都能知道父元素应该多高   1. 设置父级元素也浮动   弊端：父级元素，会影响其他元素的布局   1. 为父元素设置overflow属性，取值：hidden/auto   弊端：如果内部有内容需要溢出显示，那么也会被一同隐藏  在父元素中追加一个空块级元素，不写宽高，设置clear:both。 |

### 显示方式

#### 什么是显示方式

决定了元素在网页中的表现形式(块级、行内块)

#### 语法

display

取值：

1. none 不显示元素，隐藏状态
2. block 让元素表现的与块级元素一致
3. inline 让元素表现的与行内元素一致
4. inline-block 让元素表现的与行内块元素一致

特点：多个元素在同一行显示，但是可以设置宽高，垂直外边距

1. table 让元素表现的与table一致

特点：尺寸以内容为准，每个元素独占一行，允许修改尺寸

注意：一般讲行内元素修改显示为行内块，便于修改尺寸

练习：页面中添加5个div，分别设置div，呈现不同显示状态，同时设置尺寸，背景，内外边距，观察元素显示效果

##### 行内元素、块级元素、行内块元素对比

|  |  |
| --- | --- |
| 行内元素 | span、img、a、lable、input、abbr（缩写）、em（强调）、big、cite（引用）、i（斜体）、q（短引用）、textarea、select、small、sub、sup，strong、u（下划线）、button（默认display：inline-block）  特征：   1. 设置宽高无效 2. 对margin仅设置左右方向有效，上下无效； 3. padding设置上下左右都有效，即会撑大空间； 4. 不会自动进行换行； 5. 不会被浮动元素（float-left/right；）遮盖，而是避开； 6. 弹性布局（display:inline-flex；）子元素会变成行内元素； |
| 块级元素 | div、p、nav、h1~h6、article、address、ul、ol、dl、li、dd、table、hr、blockquote、table、menu、pre，HTML5新增的header、section、aside、footer等  特征：   1. 能够识别宽高，高度可不写，由内容撑起； 2. margin和padding的上下左右均对其有效； 3. 可以自动换行； 4. 多个块状元素标签写在一起，默认排列方式为从上至下，每个元素独占一行； 5. 浮动（float-left/right；）解决块级元素一行显示多个的问题； 6. 绝对定位（position：absolute；）的元素都将变成块元素； 7. 固定定位（position：fixed；）的元素会变成块元素； 8. 弹性布局（display：flex；）子元素会变成块级元素； |
| 行内块状元素 | 行内块状元素综合了行内元素和块状元素的特性，但是各有取舍。  特征：   1. 不自动换行； 2. 能够识别宽高,宽度默认由内容决定； 3. 默认排列方式为从左到右； 4. 元素间默认有间距； 5. 支持宽高、外边距、内边距的所有样式的设置； |
| table | 特征：   1. 允许设置页面尺寸； 2. 会向下撑开高，宽度不变； |
| **行内元素的间距问题**  两个行内元素在一起，会出现一定的间距，即使将border、padding、margin都设置为零也无济于事。  解决方法：（方法很多，此处只列举简单易用的）   1. 不换行，标签之间无空格   格式如下：<span>12</span><span>12</span>   1. 重设字体   将行内元素的直接父级设置font-size=0px;再给行内元素设置字体大小就可以解决。   1. 借助HTML注释   在两个行内元素之间加入HTML注释<!---->，告诉浏览器这中间不是换行也不是空格，而是连接在一起的，这样也可以解决。 | |

#### 显示效果

visibility

取值：

1. hidden 隐藏，元素不可见
2. visible 可见的，默认值

|  |
| --- |
| 面试题：  display:none和visibility:hidden的区别  display:none隐藏，会脱离文档流，不占页面空间  visibility:hidden隐藏，不会脱离文档流，依然占据页面空间 |

#### 透明度

opacity

取值：0-1 越小越透明

rgba(0-255,0-255,0-255,0-1) 越小越透明

|  |
| --- |
| 面试题：  opacity和rgba的区别  opacity作用于元素，只要和元素相关的颜色都会跟着改变透明度  rgba作用于元素的颜色或其背景颜色，其透明度会发生改变 |

#### 垂直对齐方式

vertical-align

控制图片与两边文字的垂直对齐方式

使用场合：

1. 表格中使用
2. 图片中使用

取值：

1. baseline基线对齐，默认值
2. top、middle、bottom

注意：编写网页时，通常都会讲所有图片垂直对齐方式更改为除“基线对齐”以外的方式。

#### 光标

cursor

改变鼠标悬停在元素上的状态

取值：

1. default 箭头，默认
2. pointer 小手
3. crosshair +
4. text 工
5. wait 等待，转圈
6. help 帮助，箭头+？

### 列表

#### 列表项标识

list-style-type

取值：

1. none 无标识
2. disc 圆点(实心)
3. circle 圆点(空心)
4. square 方块(实心)

#### 列表项使用图像

list-style-image

取值：url()

#### 列表项的位置

list-style-position

将默认的列表项标识位置，放到li中

取值：

1. outside 默认值，将标识放到li外面
2. inside 将标识放到li里面

#### 简写方式

list-style

取值：typle url() position

常用方式：list-style:none;

注:有外边距，有内边距，使用\*{margin:0;padding:0}

### 定位

#### 定位属性

position

取值：

1. relative 相对定位
2. absolute 绝对定位
3. fixed 固定定位
4. static 静态，默认值

注意：将元素的position属性定义为relative、absolute、fixed，意味着该元素被称为已定位元素

#### 偏移属性

top、bottom、left、right

以px为单位数字

注：使用position后，偏移属性才可以使用

#### 定位方式

##### 相对定位relative

元素相对于他自己原来的位置，偏移某个距离；

只要元素定义position：relative相对定位，该元素脱离文档流，但原来位置不被其他文档中的元素占用

position：relative

必须配合偏移属性使用

使用场合：在做元素微调的时候使用

##### 绝对定位absolute

若离开文档流，不占据页面空间位置

会相对于最近的已定位祖先元素去位移

如果没有已定位的祖先元素，那么就相对于最初的body位移

属性

position:absolute 配合偏移属性使用

注意：绝对定位的元素，都将变成块级元素

margin除了auto不生效，其他都正常

元素设置绝对定位，脱离默认文档流，原来位置，被其他没有脱离文档流的元素占用。

###### 绝对定位的参照物

1. 如果该元素的父元素是已定位元素，从父元素的左上角为基准，相对父元素的左上角偏移
2. 如果该元素的父元素不是已定位元素，向上再找一层，直到找到定位元素的“父元素”，以该“父元素”的左上角为基准偏移。
3. 该元素没有是定位元素的父元素，以body的左上角为基准偏移

position：absolute;配合偏移量使用

所有的层叠效果，都要使用absolute

dat32

##### 固定定位fixed

将元素固定在页面的某个位置，位置不会随滚动条的滚动发生变化。位置一直固定在可视区域。

position：fixed，配合偏移属性

注意：

1. 固定定位始终相对于body去实现位置变化
2. 固定定位的元素会变成块级元素
3. 固定定位元素不占页面空间

##### 堆叠顺序

一旦将元素变为已定位元素，该元素有可能出现堆叠效果

z-index

取值：无单位的数字，数字越大元素越靠上。

理论上最大值2^31-1，一般使用1000以内。

没有默认值，写的比不写的大(即使写1)；同级别，后写的比先写的大。

注意：

1. 父子元素之间，z-index无效，永远是子元素压在父元素上
2. 只有已定位的元素，才能使用z-index

|  |
| --- |
| 写样式的思路(个人总结)   1. 从外往里，从上往下，从左往右写 2. 清除默认公有样式 3. 确定尺寸 4. 确定大体位置 5. 解决所有的背景，图片，边框 6. 处理文字属性，文本对齐 7. 微调(包括4、5、的微调，包括位置的微调（偏移属性，内外边距）) |

## CSS3 Core核心

### 复杂选择器

#### 兄弟选择器

兄弟元素：具有相同父级元素的平级元素之间称为兄弟元素。

注意：兄弟选择器只能往后找，不能往前找。

##### 相邻兄弟选择器

获取紧挨在某元素之后的兄弟元素

选择器1+选择器2{}

##### 通用兄弟选择器

获取某元素后的所有满足条件的兄弟元素。

选择器1~选择器2{}

#### 属性选择器

id class style title name

允许通过元素所附带的属性及其属性值来匹配页面元素

语法：

1. [attr] attr表示任意属性

作用：匹配页面中所有附带attr的标签

ex：[id]{} [class]{} [title]{}

1. elem[attr] elem表示任意元素

作用：匹配页面中所有附带attr属性elem的元素

ex：div[title]{}

1. [attr1][attr2]

匹配同时附带多个属性的元素

ex：div[id][title] p[class][title]

1. [attr=value]

匹配attr属性值为value元素

ex：[class="c"]{}

1. [attr^=value]

匹配attr属性值，以value开头的元素

1. [attr$=value]

匹配attr属性值，以value结尾的元素

1. [att~=value]

匹配attr属性值中含有该单词的元素

ex：[title~="will"] 匹配title中，有will单词的元素(需要是一个完整的单词)

1. [att\*=value]

匹配attr属性值中含有该字符的元素

ex：[title\*="wi"] 匹配title中，有wi字符的元素(不需要是一个完整的单词)

#### 伪类选择器

链接伪类：:link :visited

动态伪类：:hover :active :focus

##### 目标伪类

突出显示活动(被激活，被点击)的HTML锚点元素，匹配被激活的锚点

选择器：target{}

ex：h2：target{}

练习：5张呆呆兵，做锚点，写五个名字，点击谁的名字，显示谁的图片，其他都隐藏

div>a{

width:255px;

height:255px;

border:2px dotted #f00;

}

div>img{

display:none;

}

div>img:target{

display:block;

}

<div>

<a href="#d1">呆1</a>

<a href="#d2">呆2</a>

<a href="#d3">呆3</a>

<a href="#d4">呆4</a>

<a href="#d5">呆5</a>

</div>

<div>

<img id='d1' src="../../04\_HTML/img-素材/image/01.png" alt="">

<img id='d2' src="../../04\_HTML/img-素材/image/02.png" alt="">

<img id='d3' src="../../04\_HTML/img-素材/image/03.png" alt="">

<img id='d4' src="../../04\_HTML/img-素材/image/04.png" alt="">

<img id='d5' src="../../04\_HTML/img-素材/image/05.png" alt="">

</div>

##### 结构伪类

通过元素结构关系匹配相应的元素

1. :first-child

匹配的元素是属于其父元素第一个子元素(大哥)

1. :last-child

匹配的元素是属于其父元素最后一个子元素(小弟)

1. :nth-child(n)

匹配的元素是属于其父元素第n个子元素

注意:tr:nth-child(2) td:nth-child(3){}表示第二行的第三列

练习:创建一个4行4列的表格，宽高400\*400px，内容随意；通过结构伪类选择器；设置第一行的背景颜色为#00f；最后一行的背景颜色为#ff0；设置第二行，第三列的背景颜色为#990066，字号24px；设置第三行，第二列的背景颜色为#660099，字号30px

p:first-child{

color:#f00;

}

p:last-child{

color:#369;

text-indent:20px;

}

p:nth-child(3){

color:#ff0;

}

table{

width:400px;

height:400px;

border:2px solid #f00;

}

table td{

border:2px solid #f00;

}

tr:first-child{

background:#00f;

}

tr:last-child{

background:#ff0;

}

tr:nth-child(2) td:nth-child(3){

background:#906;

font-size:24px;

}

tr:nth-child(3) td:nth-child(2){

background:#609;

font-size:30px;

}

<table>

<tr>

<td>01</td>

<td>02</td>

<td>03</td>

<td>04</td>

</tr>

<tr>

<td>05</td>

<td>06</td>

<td>07</td>

<td>08</td>

</tr>

<tr>

<td>09</td>

<td>10</td>

<td>11</td>

<td>12</td>

</tr>

<tr>

<td>13</td>

<td>14</td>

<td>15</td>

<td>16</td>

</tr>

</table>

1. :empty{} 选择空元素

匹配没有子元素的空元素

ex:<div></div> 为没有任何东西的空标签

ex:div:empt{}

单标签叫做空标记

1. :only-child 选择是唯一子元素的元素

匹配属于其父元素的唯一子元素

<div>

<p>123</p>

</div>

p:only-child{

width:100px;

height:100px;

background:#996600;

}

1. :not(selector) 否定伪类

将满足指定条件的选择器元素排除在外

注：在使用否定伪类时，若配套结构伪类，不要使用:first-child和:last-child，应使用:nth-child(1)和:nth-child(n)（n表示总数），否则不会出结果。

练习:在上面的表格中，除了第三行，其他都变红

tr:not(:nth-child(3)){

background:#f00;

}

#### 伪类选择器——修饰标签内容

##### 匹配某个元素的首字符

::first-letter 或者 :first-letter （一个或两个冒号均可）

##### 匹配某个元素的首行

::first-line 或者 :first-line

如果首字符和首行样式冲突，优先使用首字符的样式

##### 匹配用户选取的内容

::selection

注意：只能修改文本颜色及其背景颜色

练习：创建07\_ex.html.添加一个div#d1，内容随意

将d1中首字符设置为加粗，红色 20px 右外边距为15px；首行文本设置为橙色并倾斜；用户选取内容时，文字颜色黄色，背景为红色

#d1::first-letter{

color:#f00;

font-size:20px;

font-weight:bold;

margin-right:15px;

}

#d1::first-line{

color:orange;

font-style:italic;

}

#d1::selection{

color:#ff0;

background:#f00;

}

<div id="d1">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Neque, reprehenderit, ducimus, doloremque, illum repudiandae eos harum quaerat vel maiores pariatur maxime veritatis magnam cum corporis minus consequatur adipisci error dolor?</div>

##### 内容生成content

使用css动态向某元素中插入一段内容

###### 匹配到某元素的内容区域之前

::before 或者 :before

#d1:before{

content:"柯南";

}

<div id="d1">(:before)真相只有一个!</div>

###### 匹配到某元素内容区域之后

::after 或者 :after

<div id='d1'>真相只有一个!(:after)</div>

###### 内容属性

content

向匹配的位置添加内容

取值:

字符串，使用 " " 包裹

图片 url(路径)

计数器

###### 解决的问题

|  |
| --- |
| 1. 子元素外边框溢出   父元素:before{  content:"";  display:table;  }  步骤:  在父元素的第一个子元素位置处添加(:before);  添加是一个空内容，这个空内容使用table的特性去显示;  相当于添加了一个<table></table>   1. 子元素浮动之后，父元素高度坍塌   解决方案:  在父元素末尾追加一个空的块级元素，把这个跨级元素设置为clear:both  #parent::after{  content:"";  display:block;  clear:both;  } |

课后练习：

1. 预习弹性布局和动画
2. 复习复杂选择器，内容伪类、定位（浮动、相对、绝对、固定）
3. 使用css所学的知识，完成下图



### 弹性布局(重点)

#### 弹性布局是什么

弹性布局是一种布局方式，主要解决某元素中子元素的布局方式。

为布局提供最大的灵活性。

把父元素设置成弹性布局display:flex;--------容器；

父元素变成了容器，子元素就称之为项目。

#### 弹性布局的相关概念

##### 弹性布局的容器

要布局的子元素的父级元素

##### 弹性布局的项目

要布局的子元素称之为项目

##### 主轴

项目们（子元素们）排列方向的一根轴，叫主轴

如果项目横向排列，主轴就是x轴；

如果项目纵向排列，主轴就是y轴。

##### 交叉轴

与主轴垂直相交的轴

#### 语法

将元素变为弹性容器后，他内部所有子元素将变为弹性项目。所用的项目都允许按照弹性布局的方式排列

父元素（容器）中写属性

display:flex;

注意：

1. 项目的float/clear/vertical-align属性都失效
2. 项目允许修改尺寸
3. 容器的对齐方式text-align失效

#### 容器的属性

##### 指定容器的主轴及其排列的方向

flex-direction

取值：

1. row 默认值，主轴是x，起点在左边
2. row-reverse 主轴是x轴，起点在右边
3. column 主轴是y，起点在右边
4. column-reverse 主轴是y，起点在下边

##### 换行

当一个主轴排列，放不下所有的项目，可以设置换行显示

flex-wrap

取值：

1. nowrap 默认值，即使控件不够，也不换行，项目自动缩小
2. wrap 换行
3. wrap-reverse 换行，反转

##### 属性简写

flex-flow

是flex-direction和flex-wrap的缩写

取值：

1. row nowrap 默认值
2. direction wrap

##### 定义项目在主轴对齐方式

justify-content

作用：该属性定义了项目在主轴（横向）上的对齐方式。

取值：

1. flex-start 在主轴起点对齐
2. flex-end 在主轴终点对齐
3. center 在主轴居中对齐
4. space-between 两端对齐
5. space-around 每个间距相同

作业讲解：见01\_homework.html

##### 项目在交叉轴上的对齐方式

align-items

作用：该属性定义了项目在交叉轴上的对齐方式。

取值：

1. flex-start 在交叉轴上对起点对齐
2. flex-end 在交叉轴上对终点对齐
3. center 在交叉轴上对中间对齐
4. baseline 在交叉轴上对基线对齐
5. stretch 如果未设置项目尺寸，在交叉轴上，占满所有空间

##### 项目在多根轴上的对齐方式

align-content

作用：该属性定义项目在多根轴线的对齐方式。如果项目只有一条轴线，该属性不起作用。

取值：

1. flex-start 与交叉轴的起点对齐
2. flex-end 与交叉轴的终点对齐
3. center 与交叉轴的中间对齐
4. space-between 与交叉轴两端对齐，轴线之间的间隔平均分布
5. space-around 每个轴线的两侧的间隔都相等
6. stretch 默认值，轴线占满整个交叉轴

#### 项目属性

该组属性只能设置在一个项目中，是不影响容器以及其他项目。

##### 定义项目的排序

order:

值越小就越靠近起点。默认值都为0。

取值，整数数字，无单位。

##### 定义项目的放大比例

flex-grow:

取值：无单位整数。

如果容器有足够大的剩余空间，项目将按照比例放大。默认值为0。

值越大，占据的剩余空间越多。

##### 定义项目的缩小比例

flex-shrink:

取值：默认值为1，无单位整数。

空间不足时，等比缩小。取值越大，缩小越快。

##### 定义某一个项目在交叉轴上的对齐方式

align-self

取值：

1. flex-start 在交叉轴上对起点对齐
2. flex-end 在交叉轴上对终点对齐
3. center 在交叉轴上对中间对齐
4. baseline 在交叉轴上对基线对齐
5. stretch 如果未设置项目尺寸，在交叉轴上，占满所有空间
6. auto 基础容器的align-items的效果

中午作业

把学子首页1楼，使用弹性布局完成

见02\_ex.html

### CSS hack（见拓展视频）

由于不同的浏览器，对CSS的解析认识不一样。

针对不同浏览器写不同CSS的过程，就叫做CSS hack。

有 **拓展视频** CSS3CORE 01视频

#### 相关概述

IE6之前，不同浏览器之间没有兼容性；IE6之后浏览器厂家开始寻求统一标准，但仍允许向前兼容，于是产生了不同的运行模式。

#### 浏览器运行模式分类

混杂模式（Quirks Mode）

标准模式（Standard Mode）

准标准模式（Almost Standard Mode）

#### 触发相关模式

不同模式下，浏览器对CSS（尤其涉及盒模型）和JS解析会有不同效果。

其他浏览器会根据DOCTYPE自动进行模式选择，IE除此之外还可以手动选择运行模式。

##### 触发混杂模式

不声明DOCTYPE

##### 触发准标准模式

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

##### 触发标准模式

<!DOCTYPE HTMLPUBLIC"-..W3C..DTD GTML 4.01//EN" "http://www.3w.org/TR/html4/strict.dtd">

<!DOCTYPE HTMLPUBLIC"-..W3C..DTD GTML 1.0//EN" "http://www.3w.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html>

#### 什么是CSS Hack

浏览器的类型及版本的不同会造成CSS效果不尽相同，不同厂家的浏览器；同一厂家的不同版本的浏览器。

需要针对不同的浏览器编写不同的CSS，实现浏览器兼容；也可以为不同版本的浏览器定制编写不同的CSS效果。

#### Hack的原理

##### CSS类内部Hack

在CSS样式属性名前加上一些只有特定浏览器才能识别的hack前缀

##### 选择器Hack

在CSS选择器前加上一些只有某些特定浏览器才能识别的前缀。

##### HTML头部引用Hack

即IE条件注释：

条件注释的基本结构和HTML的注释（<!---->）相同

IE根据if条件来判断是否解析条件注释里的内容

### 转换（重点）

改变元素在页面中的位置、大小、角度及形状的一种方式。

2D转换，只在x轴和y轴上，发生转换效果。

3D转换，增加了z轴的转换，z轴是模拟出来的。

#### 转化属性

##### 转换属性

transform

取值：

1. none 默认值，无任何转换效果
2. transform-function 表示一个或者多个转换函数，如果是多个转换函数，使用空格分开。

##### 转换原点

transform-origin

取值：数字/%/关键字

默认值：center center

默认情况下，转换的原点在元素的中心点。

1个值表示所有轴的位置

2个值表示原点在x和y轴上的位置

3个值表示原点在x，y，z轴上的位置

#### 2D转换

##### 位移

改变元素在页面中的位置

属性transform

取值：数值、百分比，也可以是负值；

1. translate（x） 元素在x轴上位移的距离；+ 元素右移，- 元素左移。
2. translate（x,y） x：同上；y：在y轴上的位移；+ 元素下移，- 元素上移。
3. translateX（x）只能设置x轴的位置。
4. translateY（y）只能设置y轴的位置。

##### 缩放

改变元素在页面中的尺寸

属性：transform

取值：

1. scale(valute)

valute 横向和纵向的缩放比

无单位整数，默认值为1。

>1 放大

0< <1 缩小

-1< <0 水平和垂直都翻转180deg，缩小

<-1 水平和垂直都翻转180deg，放大

1. slcor(x,y)

x 横向缩放比例

y 是纵向缩放比例

1. scaleX(x) 单独设置水平缩放
2. scaleY(y) 单独设置竖直缩放

##### 旋转

改变元素在页面中的角度

属性：transform

取值：

rotate(ndeg)

n取正,是顺时针；n取负,是逆时针

注意：

1. 转换原点会影响最后的旋转效果。
2. 旋转是连同坐标一起旋转的，会影响旋转后的位移效果

##### 倾斜

改变元素在页面中的形状

属性：transform

取值：

1. skew()等同skewX()

让元素向着x轴发生倾斜效果，x轴角度不变。

真正改变角度的是y轴。

+是逆时针；-是顺时针。

1. skewY()

让元素向着y轴发生倾斜效果，y轴角度不变。

真正改变角度的是x轴。

+是顺时针；-是逆时针。

1. skew(x,y)

一般不用。

#### 3D转换

##### 透视视距

模拟人的眼睛,到3D转换元素之间的距离。

视距越小，人的眼睛，离3D元素越近，3D效果越强烈；距离越大，3D效果越小。

属性：perspective

取值：带px的数字

该属性只能加载转换元素的父级上，只影响3D转换元素本身。

##### 3D旋转

属性：transform

取值：

1. rotateX(xdeg)

以x轴为中心，旋转元素的角度

1. rotateY(ydeg)

以y轴为中心，旋转元素的角度

1. rotateZ(zdeg)

以z轴为中心，旋转元素的角度

1. rotate3D(x,y,z,deg)

x，y，z取值大于0的数字，表示参与旋转；

取值为0的，表示不参与旋转。

##### 3D缩放

属性：transform

取值：

1. scaleZ(z)
2. scale3d(x,y,z)

### 过渡（重点）

#### 什么是过度

使CSS属性值，在一段时间内平缓的变化。

指定4个要素：

1. 指定过渡属性；
2. 指定过渡所需时间；
3. 指定过渡函数；
4. 指定过渡延迟时间。

触发过渡：

1. 由用户的行为触发；
2. 由元素的状态变化触发；
3. 有JavaScript代码触发。

#### 语法

##### 指定过渡属性

transition-property

取值：

1. all 默认值，能够使用过渡元素的属性，一律用过渡来体现
2. none 没有属性会获得过度效果
3. property 定义应用过度效果的CSS属性名称列表，列表以逗号分隔。

能使用过渡效果的CSS属性

颜色

取值为数字的属性

转换效果

阴影效果

渐变效果

visibility

##### 指定过渡的时长

transition-duration

取值：以s或者ms为单位的数字 1s=1000ms

注意：必须设置transition-duration属性，否则时长为0，不会产生过渡效果。

##### 指定过渡时间曲线函数

transition-timing-function

取值:

1. ease 默认值,慢速开始，中间加速，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，慢速结束
5. ease-in-out 慢速开始和结束，中间先加速再减速

##### 指定过渡的延迟时间

transition-delay

取值：以s或者ms为单位的数字。

##### 过渡属性的编写位置

将过渡属性放在元素声明的样式中，有去有回；

将过渡元素放在触发属性（hover），只管去不管回。

##### 过渡的简写

transition：property duration timing-function delay

为过渡子属性设置多个值，多个值之间用逗号隔开。

练习：马不停蹄的滚

06\_ex.html

在页面中，创建一个100\*100的div，设置背景颜色和倒角50%（00FFBB39图片），滚

1.鼠标悬停，5秒之内，向右偏移500px，同时旋转3圈

2.鼠标移出之后，还能滚回来

### 动画（重点）

#### 什么是动画

是元素从一种样式逐渐变成另外一种样式，将多个过渡效果放在一起。

#### 动画是通过“关键帧”，来控制动画的每一步

关键帧

1. 动画的执行时间
2. 在这个时间点，样式是什么

#### 关键帧的使用步骤

##### 声明动画

@keyframes 动画名称{

/\*定义关键帧\*/

0%{

动画开始时的样式

}

......过程中的关键帧

100%{

动画结束时的样式

}

}

##### 调用动画

1. 调用的动画名称

animation-name:change;

1. 指定动画播放的周期

animation-duration:5s;

1. 指定动画播放曲线函数

animation-timing-function:linear;

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

1. 指定动画延迟播放的时间

animation-delay:5s;

1. 动画的播放次数

animation-iteration-count:

取值：

具体的参数

infinite无限次

1. 指定动画的播放方向

animation-direction

取值：

normal 默认值，表正常播放0-100%

reverse 逆向播放，100%-0

alternate 表轮流播放，即动画会在奇数次数正向播放，在偶数次数逆向播放。

1. 指定动画在播放前和结束时的状态

animation-fill-node

取值：

none 默认值，动画开始前及结束后都消失

forwards 动画完成后保存在最后一个动画的状态

backwards 动画播放前，第一帧显示

both forwards和backwards的结合

1. 简写方式

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

1. 动画播放状态

animation-play-state

取值：

running 默认值，指定正动画运行

paused 指定动画暂停

#### 动画的兼容性

如果要兼容低版本浏览器，需要在动画声明时，添加前缀

@keyframes 动画名{定义关键帧}

@-webkit-keyframes动画名{定义关键帧}

@-o-keyframes动画名{定义关键帧}

@-ms-keyframes动画名{定义关键帧}

@-moz-keyframes动画名{定义关键帧}

### CSS优化

#### 目的

减少服务器压力

提升用户体验

#### CSS优化原则

尽量减少HTTP请求个数

页面顶部引入CSS文件

将CSS和JS文件单独放到外面的独立文件夹下

#### 代码优化

合并样式

缩小样式文件

减少样式的重写

避免出现空的src和href

选择更优的样式属性值

代码压缩(css在线压缩工具)

## Bootstrap

### Bootstrap概述

#### 响应式布局

Responsive web page 响应式/自适应网页

网页可以根据浏览器设备的不同，自动的更改布局，图片，文字效果，不会影响用户的浏览体验。

#### 响应式布局的要求（重要）

1. 布局：不能固定宽度，必须是流式布局；
2. 文字和图片大小随着容器的大小改变；
3. 媒体查询技术（根据不同的设备，页面显示不同的）

##### 响应式布局存在问题

增加网页的复杂程度；

代码的复杂程度极大的网站，不建议使用响应式布局。

#### 如何测试响应式网页

1. 使用真实设备

好处：真实可靠

缺点：测试任务量较大

1. 使用第三方的模拟软件

好处：无需太多的真实设备，测试方便

缺点：测试效果有限，有待进一步验证

1. 使用Chrome等浏览器自带模拟器

好处：简单方便

缺点：测试效果有限，有待进一步验证

#### 如何编写响应式网页（重点）

##### meta

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0, shrink-to-fit=no, viewport-fit=cover">

1. 在meta中使用viewport源标签（手机适配）

viewport：早期3G手机把窗口缩小，导致文字图片都变小。

iOS提出“视口”，视口用于盛放网页的内容，超过的部分，可以滑动显示。

1. content

width=device-width 表示视口的宽度就是设备的宽度

initial-scale=1.0 表示视口的宽度能否缩放，1.0不缩放

maximum-scale=1.0 表示允许缩放的最大倍数

user-scalable=0 是否允许用户手动缩放

取值：yes/no/1/0

iOS专用，隐藏导航头和地址栏

minimal-ui:ios ui设置

一般设置：宽度为设备宽度，设置缩放

<meta name='viewport' content='width=device-width, initial-scale=1'>

##### 数值

所有内容/文字/图片，都使用相对尺寸，不要使用绝对数值

.container{

/\*width:100px;\*/

width:100%;

/\*font-size:16px;\*/

font-size:1rem;

}

##### 布局

流式布局+弹性布局,再搭上媒体查询技术

实现方式：float/flex + Media Query

##### 使用条件

使用CSS3 media query技术---制作响应式网页的必备条件

media：媒体，指浏览网页的设备

如：screen（PC/pad/phone） TV（电视剧） print（打印机）等等

Media Query媒体查询，网页可以自动根据当前浏览网页设备的不同，有选择的执行部分CSS样式，忽略其他一般分CSS样式。

#### 使用media query技术,做媒体查询的方式

1. 做多个css文件，根据link中的查询条件，加载相应的css文件

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

注意：此方式，有缺陷，不用了。

1. 根据媒体查询的结果,，执行同一个css文件中不同的代码块

总结语法：

@media screen and (min-width:XXX) and (max-width:XXX){

选择器{样式}

}

注：

PC最小宽度为992px；

pad最大宽度为991px，最小宽度为768px；

phone最大宽度为767px。

### Bootstrap框架

1. 如何使用boot
2. 全局css样式
3. 组件
4. js插件
5. 定制(scss)

#### 如何使用boot

http://www.bootcss.com/

下载boot包

在自己的项目中创建css和js文件夹；

在html页面中引入css文件和js文件；

同时不要忘记写viewport元数据。

制作boot模板。

#### 全局css样式

##### BTN

###### 控制按钮颜色

btn 基本按钮

btn-success 成功--绿色

btn-danger 危险--红色

btn-warning 警告--黄色

btn-info 信息--藏蓝

btn-primary 主要--亮蓝

btn-secondary 次要--灰色

###### 按钮边框

btn btn-outline-dark 红色边框

btn btn-outline-primary 亮蓝色边框

btn btn-outline-info 藏蓝色边框

作业：

1.使用媒体查询完成，



2.重新创建一遍使用boot的模板

3.完成web第二阶段练习效果图示\day14\_boot\_01练习作业\课后作业



###### 3种尺寸

默认尺寸，不用写

<button class="btn btn-lg btn-success">大按钮</button>

<button class="btn btn-sm btn-danger">小按钮</button>

###### 其他颜色

<button class="btn btn-secondary">次要按钮</button>

<button class="btn btn-dark">深色要按钮</button>

<button class="btn btn-light">浅色要按钮</button>

<button class="btn btn-link">链接</button>

###### 图片相关的css

rounded 圆角

rounded-0 去除圆角

rounded-circle 椭圆图片

img-thumbnail 缩略图,有边框,弹性布局

img-fluid 响应式布局，图片根据屏幕大小缩放，最大值不超过图片本身宽度

###### 文字相关的css

h1~h6 不同字体大小

.text-danger/warning/info/primary/success 字体颜色

.text-left/center/right/justify 文字对齐

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

###### 列表相关的class

1. ul

list-unstyled 去掉点，左内边距清空

list-group 列表组 弹性布局，主轴为Y

1. li

list-group-item 列表组中的列表项

1. 颜色

list-group-item-success/danger/warning/info/primary

active 激活状态

disabled 禁用列表项

###### table的class

1. 写在table父容器中的class

table-responsive 响应式表格，用于table的父容器

1. table的class

table 每列带上边框的表格

table-bordered 带边框的表格

table-striped 隔行变色

table-hover 带鼠标悬停效果

table-success/danger/warning/info/primary/dark/light

#### 辅助类名---大部分标签都可以使用的class

##### 基本类名

###### 边框

border 边框

border-success/danger/warning/info/primary/dark/light

rounded 边框圆角

rounded-0 去掉圆角

###### 浮动

float-\*-left \*:xl/lg/md/sm/xs 使用媒体查询

clearfix 解决高度坍塌

###### 显示

invisible 隐藏

visible 显示

###### 背景颜色

bg-success/danger/warning/info/primary/dark/light

##### 内外边距

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| m-0 | m-1 | m-2 | m-3 | m-4 | m-5 |
| 0 | 0.25rem | 0.5rem | 1rem | 1.5rem | 3rem |
| p-0 | p-1 | p-2 | p-3 | p-4 | p-5 |

##### 各个方向

|  |  |  |  |
| --- | --- | --- | --- |
| mt-0 | mr-1 | mb-2 | ml-3 |
| margin-top | margin-right | margin-bottom | margin-left |
| pt-0 | pr-0 | pb-0 | pl-0 |
| padding-top | padding-right | padding-bottom | padding-left |

##### 高度

|  |  |  |  |
| --- | --- | --- | --- |
| h-25 | h-50 | h-75 | h-100 |
| height:25% | height:50% | height:75% | height：100% |

#### 栅格样式布局(重点)

##### web页面布局方法有三种

|  |  |  |  |
| --- | --- | --- | --- |
|  | 名称 | 优点 | 缺点 |
| 1 | table布局 | 简单，容易控制 | 语义错误  渲染效率极低 |
| 2 | div+css布局 | 语义正确，渲染效率高 | 控制起来比较麻烦 |
| 3 | Bootstrap中栅格布局 | 简单，容易控制，语义正确，渲染效率高，支持响应式 | 如果页码非常复杂，使用栅格不适合 |

##### 使用栅格布局

1. 最外层需要使用容器 div class="container"
2. 容器中声明行 div class="row"
3. 行中声明列div class="col col-1" 每一列要指定宽度占比 n/12

##### 针对不同屏幕需要使用不同的比例

<!--大屏幕 1:1:1-->

col-lg-4

<!--中屏幕 1:1-->

col-md-6

<!--小屏幕 1-->

col-sm-12

xl:Extra large 超大屏幕 w>=1200px

lg:large 大屏幕 1200px>w>=992px

md:medium中级的 992px>w>=768px

sm:samll 小屏幕 768px>w>=576px

xs:Extra small 超小屏幕 w<576px

注意：只是用.col不添加数字，自动处理布局；

每一行，每个列宽度相等；

两列 每个占50%

四列 每个占25%

练习：每行一列；大中小屏幕，都只占一半；

##### 注意不同屏幕下，列的适应性问题

col-xl-\* 只能在xl屏幕下有效

col-lg-\* 能在xl/lg屏幕下有效

col-md-\* 能在xl/lg/md屏幕下有效

col-sm-\* 能在xl/lg/md/sm屏幕下有效

col-xs-\* 能在xl/lg/md/sm/xs屏幕下有效

总结：col-\*列可以在当前屏幕，以及更大屏幕下有效；

小尺寸可以适配大尺寸，减少代码冗余。

##### 列偏移

可以使用列偏移实现指定列及其后的列向右偏移

offset-md-8 在中屏幕下向右偏移8个

列偏移的本质，就是修改当前列的左外边距

##### 响应式

一个div可以指定在不同的屏幕下，显示不同的宽占比。

<div class="col-xl-1 col-lg-4 col-md-3 col-sm-1 col-xs-1">

##### 栅格嵌套

在一个列中嵌套其他的行

<div class="row">

<div class="col-md-6">

<div class="row m-3 p-3">

<div class="col-md-6">文本内容1</div>

<div class="col-md-6">文本内容2</div>

</div>

</div>

<div class="col-md-6"></div>

</div>

第一个col占6份，嵌套之后，在6份的基础上，再分12份。内部每个col占6份。

#### 弹性布局

##### 使用dflex/d-inline-flex 类创建一个弹性容器

.flex-row 设置主轴方向，默认效果，x轴

.flex-row-reverse 设置主轴起点为右,终点在左边

.flex-column 设置主轴方向y轴

.flex-column-reverse

.justify-content-\* 设置项目在主轴上的对齐方式

(\*：end start center between around)

##### 响应式flex类

根据不同的屏幕，设置flex类

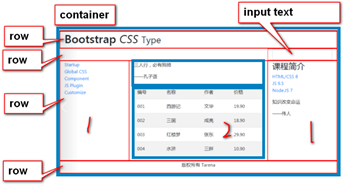
.d-md-flex 在中屏幕及以上的屏幕是弹性

.flex-md-column 在中屏幕及以上的屏幕主轴为y

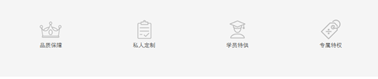
justify-content-\*-arouind 在不同屏幕的对齐方式

作业：

1.完成栅格布局练习



2.使用栅格 布局完成学子首页footer



3.把脑图自己整理一下

### 公共CSS

#### 表单

##### form的样式

form-group 下边距

form-inline 弹性布局，元素会横向显示

##### input的样式

form-control 基础样式

form-check 独占一行，往右移动0.25rem

form-check-inline 弹性布局,在一行显示

form-check-input 位置往左移动1.25rem

form-text 添加上外边距

col-form-label 添加上下内边距

练习：使用栅格布局



<div clas="container">

<form action=""class="form-group">

<div class="form-inline m-3">

<div class="col-md-2">用户名</div>

<div class="col-md-6">

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

</div>

<div class="col-md-4">

用户名包含字符数字3-8位

</div>

</div>

<div class="form-inline m-3">

<div class="col-md-2">密码</div>

<div class="col-md-6">

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

</div>

<div class="col-md-4">

密码包含字符数字3-8位

</div>

</div>

<div class="form-inline m-3">

<div class="col-md-12 text-center">

<button class="btn btn-success">登陆</button>

<button class="btn btn-danger">取消</button>

</div>

</div>

</form>

</div>

##### 总结row和form-inline类似

|  |  |
| --- | --- |
| row | form-inline |
| display: flex;  flex-wrap: wrap; | display: flex;  flex-flow: row wrap;  align-items: center;  增加了一个交叉轴的对齐方式 |

结论，row和form-inline都可以作为栅格布局的行容器；

form-inline除了可以放在form元素，还可以放在div中。

### 组件

#### 下拉菜单

##### 外层有容器dropdown

##### 容器中包含两部分

1. 下拉按钮

样式dropdown-toggle----------作用:触发元素上添加小箭头

属性:data-toggle='dropdown'

1. 下拉菜单

放到一个容器中,样式dropdown-menu

每一个菜单项,样式dropdown-item

总结：display:none;写在dropdown-menu类中，点击了按钮之后，boot依赖js文件，把display的值改掉，改成block。

###### 下拉菜单的3级结构

<div class='dropdown'>

<button data-toggle='dropdown'>

触发元素

</button>

<ul class='dropdown-menu'>

<li class='dropdown-item'></li>

...

</ul>

</div>

#### 按钮组

在几个按钮外，定义一个容器，此容器class='btn-group'，就可以变为按钮组。

注意：btn-group是基础形式，必须写；

btn-group-lg/sm并没有样式，作用是按钮组中，每一个btn的父级条件，完成子代选择器用的，对每一个btn做样式条件，改变btn大小.btn-group-sm > .btn

使用btn-group-vertical创建垂直按钮组

#### 信息提示框

alert

颜色，alert-danger/warning/info/dark...

data-dismiss="alert" 事件触发 关闭alert

close是修饰X样式，父级中alert-dismissible是修饰X的样式。

#### 导航

Bootstrap提供了三种导航：水平，选项卡，胶囊

##### 水平导航

创建一个简单的水平导航

1. 在ul上添加 nav
2. 在li上添加 nav\_item
3. li中的a标签上添加nav\_link

注意：

1. 通过ul的justify-content-\* 设置导航的对齐方向
2. nav-justified设置导航等宽显示
3. flex-column设置主轴方向Y

练习：完成bootcss.com的导航



##### 选项卡导航

选项卡导航有两部分，ul是导航栏，div是选项卡内容

1. ul使用nav-tabs变成选项卡导航
2. ul>li nav-item,ul>li>a nav-link
3. 选项卡的div tab-content
4. 选项卡div>div tab-pane

至此，样式完成

事件触发：

1. a中，data-toggle="tab"
2. 把每个选项卡和对应的tab绑定

tab写id，选项卡的href="#id"（类似锚点的写法）

1. 在a中和对应tab写类名active，设置默认激活选项卡

##### 胶囊导航

胶囊导航有两部分，ul是导航栏，div是选项卡内容

1. ul使用nav-pills变成选项卡导航
2. ul>li nav-item，ul>li>a nav-link
3. 选项卡的div tab-content
4. 选项卡div>div tab-pane

至此，样式完成

事件触发：

1. a中，data-toggle="pill"
2. 把每个选项卡和对应的tab绑定

tab写id，选项卡的href="#id"（类似锚点的写法）

1. 在a中和对应tab写类名active，设置默认激活选项卡

#### 导航栏

需要ul外层添加容器

1. 容器使用 navbar navbar-expand-sm/lg/xl/md 创建响应式

（大屏幕展开，小屏幕垂直）

1. ul 添加navbar-nav
2. li 添加 nav-item a 添加nav-link

注意：在外层容器可以设置背景颜色 bg-\*

#### 折叠

1. button必须写data-toggle="collspse" data-target="#d1"
2. div必须写id，class="collapse"，刷新完是隐藏状态

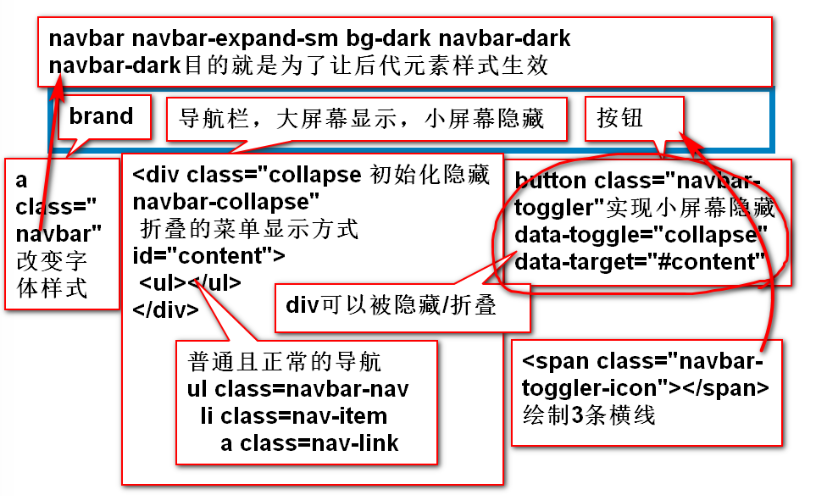
总结：控制内容隐藏与显示。

在a或button元素上设置data-toggle="collapse"属性

button------------>data-target="#id"

a------------------>href="#id"

#### 折叠导航栏



#### 卡片

通过card与card-body创建一个卡片

card-header设置卡片的头部

card-body设置卡片的主体

card-footer设置卡片的底部

#### 媒体对象（重点）

Bootstrap提供很多类来处理媒体对象和内容的布局(图片、音频、视频)

容器元素添加media，然后将多媒体内容放到容器中。

多媒体

子容器（文字） media-body

可以使用align-self-\* 控制多媒体的显示位置

作业：

1.今天demo全部在完成一遍

2.折叠导航栏

3.提高折叠+卡片完成手风琴效果（提高题）



作业：手风琴效果

注意：使用data-parent属性，确保容器内所有的折叠元素，只显示一个。（一个折叠显示，其他隐藏）

#### 轮播

1. 外包裹carousel id=demo
2. 轮播图carousel-inner

每一种图片要包裹在一个div中 carousel-item

可以添加标题carousel-caption

1. 轮播指示标志

ul--carousel-indicators

li--data-target="#demo" data-slide-to='0'图片的顺序

1. 左右箭头

a class='carousel-control-next' data-slide='next'

a class='carousel-control-prev' data-slide='prev'

绘制箭头：

span class='carousel-control-next-icon'

span class='carousel-control-prev-icon'

#### 模态框

模态框modal是覆盖在父窗体上的子窗体，可以在不离开父窗体的情况下，与用户有一些互动,提交一些交互信息

<!--用于打开模态框的控件-->

<button class="btn btn-info" data-toggle="modal" data-target="#myModal">点点我试试</button><!--模态框的内容-->

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

<div class="modal-dialog">

<div class="modal-content">

<div class="modal-header"></div>

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

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

</div>

</div>

</div>

### 其他组件

#### 徽章

badge badge-danger badge-pill

#### 巨幕

jumbotron

#### 分页

1. ul class="pagination" 控制大小pagination-lg/sm
2. li class="page-item" 激活 active 禁用 disabled
3. a class="page-link"

#### 面包屑

1. ul class="breadcrumb"
2. li class=""

#### 进度条

1. 外包裹div class='progress'
2. 内部div是真正的进度条，div class="progress-bar bg-\* w-\*"
3. 条纹进度条 内部div class="progrss-bar-striped"
4. 动画进度条 内部div class="progress-bar-animated"
5. 一个外包裹可以放多个内部div，分区显示进度条

**Bootstr中重要的知识：媒体查询技术+栅格布局+scss**

## SCSS动态样式语言

scss不能被浏览器解析,必须转译成css文件

### css的缺点

语法不够强大，没有变量和合理的复用机制。

动态样式语言为css赋予了动态语言的特性。

极大地提高了样式语言的可维护性。

#### 常见的动态样式语言

1. Scss/Sass，scss兼容sass，scss更接近css的语法
2. stylus
3. Less

以上3种都属于css预处理器

预处理器：用一种专门编辑语言，进行web页面样式设计。

再通过编辑器转化为正常的css文件，以供项目使用。

### scss是什么

SCSS是一款强化css的辅助工具。

在css语法的基础上，增加了变量，嵌套，混合，导入，函数等高级功能，这些扩展令css更加强大与优雅。

### scss的作用

有助于更好地管理样式文件，以及更高效率的开发项目。

### 使用SCSS

#### 在服务器端使用

1. 安装nodejs解释器
2. 安装scss编译程序

在线安装npm install -g node-sass

##### 测试是否安装成功

在webstorm中打开命令行alt+f12

检查scss版本node-sass -v

注意：需要node版本8.11以上

1. 在命令行编译scss文件，转化为css文件

首先，要确认，路径是在项目中

C:\xampp\htdocs\CL\BootProject\day04>

##### 执行转换命令

转换单文件

node-sass (scss文件夹)/01.scss

转换多文件

node-sass scss -o css

监听单文件

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

监听多文件

node-sass -w scss -o css

手动刷新ctrl+alt+y

#### SCSS的基础语法

##### 变量

使用$标识变量

变量名规范遵循css中选择器的命名规范。

可以包含 - \_ 让名称有语义，见名知意

注意：

1. 声明变量时，变量值可以引用其他变量值
2. 变量定义在{}规则块外部

整个样式文件都可以使用

定义到{}内部，只能在当前规则块中使用

1. 声明重复变量，后声明会覆盖前面的变量。

尽量避免重复声明

！default

如果变量已经声明赋值，那就使用它声明的值

否则，就使用整个默认值

$jd\_red:#006699 !default;

##### 嵌套规则

嵌套写法

a{

color:#ff0000;

&:hover{color:#f00000;}

}

在嵌套规则中，有时需要使用嵌套外层父选择器，可用&代替规则外层的父选择器。

##### 群组选择器嵌套

SCSS的群组嵌套写法

nav,aside,footer{

a{color:#f00;}

}

生成css

nav a, aside a, footer a {

color: #f00;

}

##### 属性嵌套

SCSS中属性嵌套

div{

border:{style:solid;width:2px;color:#eee;}

}

生成的CSS

div {

border-style: solid;

border-width: 2px;

border-color: #eee;

}

##### 导入SCSS文件

在scss中，局部文件名以下划线开头

编译软件在编译生成css文件时，不会编译下划线开头的scss文件。而只是把整个文件用作导入。

一个scss导入其他局部scss文件的时候

1. @import导入
2. sass的局部文件可以不写全名，省略开头下划线和.scss后缀

##### 混合器

###### 低版本浏览器，需要做兼容

box-shadow:0 0 50px 20px #f00;

-ms-box-shadow:0 0 50px 20px #f00;

-webkit-box-shadow: 0 0 50px 20px #f00;

###### 声明混合器(封装)

@mixin 混合器名称{

box-shadow:0 0 50px 20px #f00;

-ms-box-shadow:0 0 50px 20px #f00;

-webkit-box-shadow: 0 0 50px 20px #f00;

}

###### 使用混合器(调用)

div{

@include 混合器名称;

}

练习:

a标签,文字为灰色无下划线,用列表布局 ul>li>a

在scss中编写样式,边框圆角用混合器的方式变写

鼠标悬停在啊时,元素颜色变为京东红

课后作业：

1.仿写boot中文官网主页，折叠navbar

2.手风琴效果

3.复习今天所有内容

4.定义一个带参数混合器，抽离出字体大小，样式和粗细

在div和p元素中使用混合器，传入不同的参数查看效果

(使用scss生成css)----提高

###### 带参数的混合器

声明：

@mixin 混合器(参数1,参数2,......){

样式属性1:参数1;

样式属性2:参数2;

......

}

使用：

@include 混合器(实参1,实参2,...)

##### 继承

继承是一个选择器可以继承另一个选择器的样式

@extend

p.v1{

color: #fff;

border-color: #000;

}

p.v2{

font-size: 12px;

background: #000000;

@extend p.v1;

}

练习:在选择器.myborder中定义1px、实现、透明边框、宽度为400px，距离左边距有20px外边框

在选择器.my\_bottom\_border中，继承.myborder。并设置底边为红色，当鼠标悬停时，底边为蓝色。

#### 运算

##### 数字运算

+ - \* / %

注意：

1. 加法

+ 可以用于连接字符串

如果用有引号字符串去连接无引号的字符串，结果是有引号的；

相反，用无引号的字符串去连接有引号的字符串，是无引号

1. 除法

在scss中，通常 / 起到的是分隔数字的作用。

同时也具备除法运算的功能。

在以下情况，视为除法：

1. 符号左右是变量，或者函数返回值（同单位相除，单位消失）
2. 如果值被小括号包围
3. 符号前后有其他算数表达式

避免除数带单位

1. 运算表达式与其他值的连用，中间用空格连接
2. 在带引号的字符串中，可以添加运算表达式

需要使用#{}包裹运算表达式

##### 颜色

要按照rgb去相加。

关于rgba的运算，两个rgba运算，不会运算到a。

且，必须要求，两个值alpha必须相等。

练习：定义两个变量分别为10px和20px。两个变量相加以后乘以2，将结果赋值给p的width

#### 函数

##### 从css调用的函数

scss定义了多种函数,有一些函数可以用css语句直接调用

###### 颜色函数

color:rgba(red,green,blue,alpha);

hsl(hue,saturation,lightness);

hue：色调0~360，3个色调，每120一个

saturation：饱和度，0.0%~100.0%

lightness：亮度，0.0%~100.0%

###### 数字函数

round($value); 四舍五入

floor($value); 向下取整

ceil($value); 向上取整

min($value1,$value2,$value3...) 取最小值

max($value1,$value2,$value3...) 取最大值

random(); 0-1的随机数

###### 字符串函数

unquote($string) 删除字符串的引号

quote($string) 加上字符串的引号

to-upper-case() 转为大写字母

top-lower-case() 转为小写字母

##### 自定义函数

@function getWidth($n){

$max\_width:20px;

@return $max\_width - $n;

}

@function定义函数;@return返回值;

@function 函数名($n...){

@return 返回值

}

$width:函数名($n);

练习:

定义一个带参的函数get\_mywidth($n)

函数内定义两个变量分别取值60px和10px

函数最终返回&n乘以上面两个变量的最大值，加上$n-1乘以最小值

将结果赋值给div元素的width属性

#### 控制指令

当@if的表达式返回值

表达式成立,执行{}内代码

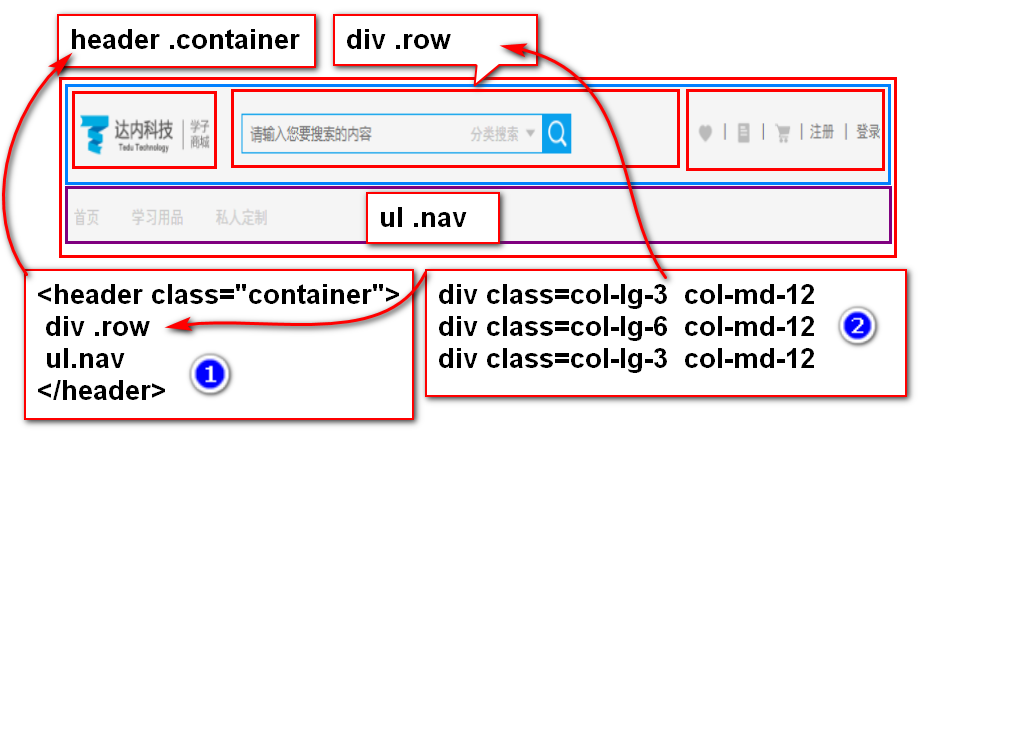
@if后面可以跟多个@else if，或者跟1个@else if。

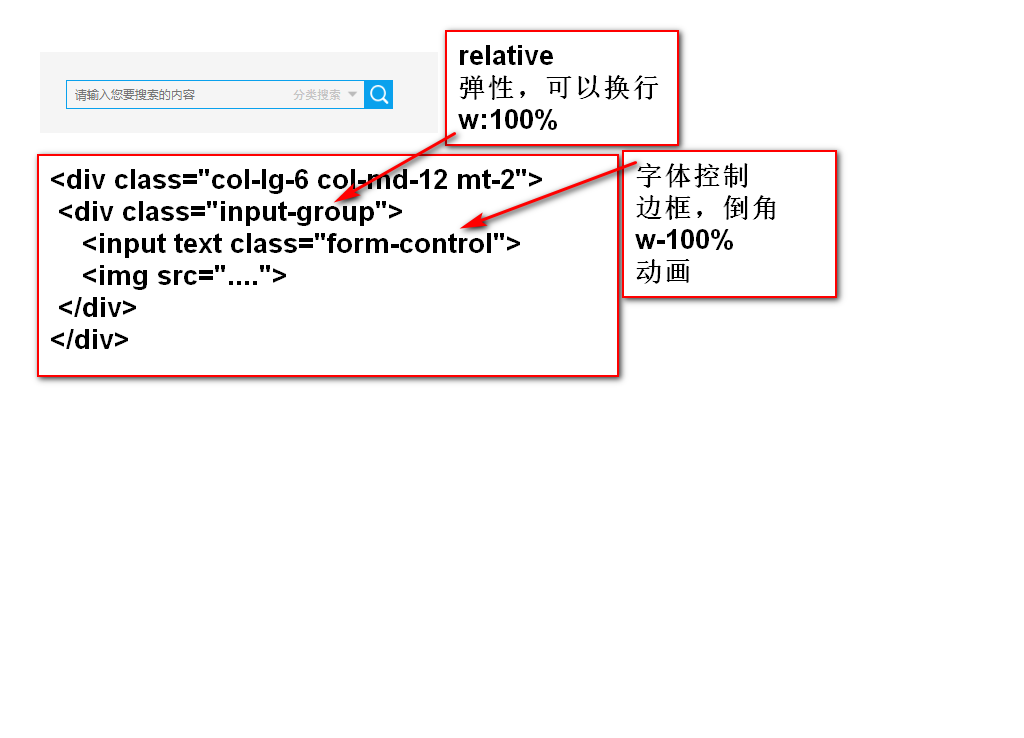
## 项目阶段

### 完成头部标签

header.html----->boot创建

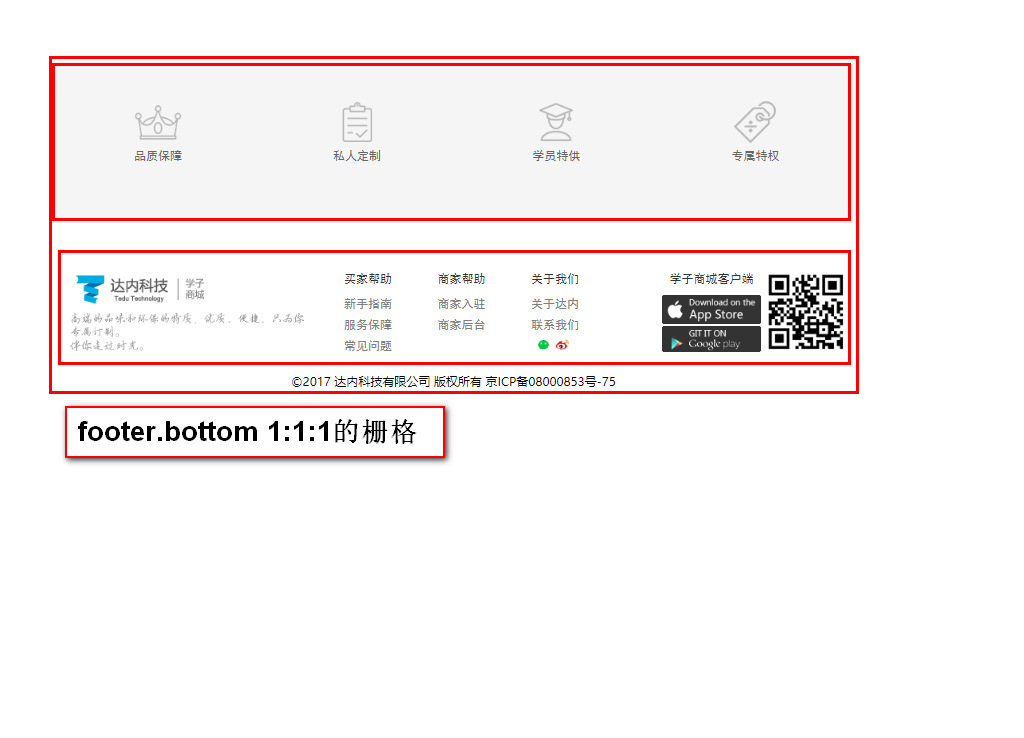
header.css







### 完成底部标签



### 一楼的结构

要求，重新创建index\_1.html

1. 把header.html footer.html 正确的轮播图，都放入这个页面
2. 自己尝试完成1楼第一行标题和第一行内容

总结：

|  |  |  |
| --- | --- | --- |
| 弹性布局 | 媒体查询 | 栅格布局 |
| 在同一个屏幕尺寸范围内(991-768)  屏幕做伸缩,弹性布局的项目,会跟着伸缩 | 不同尺寸的屏幕,对应不同媒体查询下的样式 | 在一个row,几个元素的宽度占比是不变的  col-lg-3 col-md-6 col-sm-2 |

bootstrpit中，1rem=16px

### 商品列表页

#### 思路

1. html结构----设计，选择你想用的技术，弹性，栅格，响应式（考虑小屏幕下的布局效果）
2. 基本样式--->boot提供
3. 特殊样式--->自己去css写
4. 位置和距离的控制--->配合前面的弹性/栅格，修改boot提供的类自带的内外边距
5. 调整不同分辨率情况下的布局问题
6. 在媒体查询中写具体的边框情况

#### 步骤

1. 引用header.html footer.html
2. 头部下发添加广告大图
3. html结构布局
4. 左侧产品列表div的细节

标题

列表具体内容 card

分页

1. 右侧商家公告
2. 购物车

### 商品详情

引入头部尾部 product\_details.html/css