HTML

1、WEB基础知识

1、web 与 Internet

1、Internet

Internet是一个全球性的计算机互联网络 ，中文名"因特网","网"

Internet 提供的服务：

Telnet,Email,WWW,BBS,FTP

基本实现技术：

1、分组交换原理

2、TCP/IP协议簇

2、Web

Web 是运行在 Internet 上的一种应用(网页应用)

Web 所谓的WWW

Web的规范是由W3C负责定制和推广

W3C : World Wide Web Consortium(万维网联盟)

Web能够将服务和信息关联到一起

服务：Email,BBS,...

信息：文字，图片，音视频，文件等

3、Web的工作原理

基于 B/S 结构模式的程序

B : Browser(浏览器)

S : Server(服务器)

C/S :

C : Client(客户端)

S : Server

由Web服务器，浏览器，通信协议 三部分组成

Web服务器：允许为其他用户提供服务的机器

通信协议：web 采用的是 http 协议

HTTP : Hyper Text Transfer Protocol

超文本传输协议

规范了数据是如何打包以及如何传递的。

4、Web服务器

1、主要功能

1、存储信息

2、响应浏览器的请求，执行服务器端程序

3、具备基本的安全功能

2、主要Web服务器产品

1、TOMCAT

2、IIS

3、APACHE

3、服务器端相关技术

1、php

2、JSP(Java Server Page)

3、ASP

4、ASP.NET

以上技术都具备访问数据库的能力

5、浏览器

1、主要功能

1、代理用户 提交请求(User Agent,UA)

2、作为HTML/CSS/JS解释器

3、以图形化方式显示页面

2、主流的浏览器产品

1、Microsoft IE

2、Mozilla Firefox

3、Google Chrome

4、Apple Safari

5、Opera

3、客户端技术

运行于浏览器

1、HTML

2、CSS

3、JavaScript(JS)

2、HTML快速入门(重点)

1、超文本

Web 是一个超文本文件的集合

2、什么是HTML

HTML：Hyper Text Markup Language(超文本标签语言)

超文本：具备一些特殊功能的文本

a : 英文字符的第一个

超文本 a : 在页面中具备超链接作用

b : 英文字符的第二个

超文本 b : 在页面中会加粗显示文本

标签：超文本的组成形式

超文本 a:<a></a>

超文本 b:<b></b>

语言：具备一定的语法结构

用HTML编写的网页文件，以.htm或.html作为文件的后缀(超文本文件)

运行环境：任何一款浏览器

开发工具：任意一款文本编辑工具都可以

3、HTML基础语法

1、标记语法

在html中，用于描述功能的符号，称为"标记"

<p>,<a>,<b>

标记在使用时，必须用<>括起来

标记分为

1、封闭类型的标记

也成为"双标记"，标记必须成对出现

语法：

<标记>内容</标记>

如果一对双标记中没有结束标记的话，网页中会产生意想不到的效果

ex:

<a>百度</a>

<b>加粗显示</b>

<p>段落文本</p>

2、非封闭类型的标记

也称为"单标记"或"空标记"

语法：

<标记>

或

<标记/>

ex:

<br> : 换行

<img>: 图片

2、元素

元素即标记

<a>百度</a>

1、元素的嵌套

HTML中通过嵌套的元素来形成更为复杂的页面结构

注意：

1、嵌套顺序

<p><a></a></p> 正确

<p><a></p></a> 错误

2、格式问题

被嵌套的东西要用 缩进来表示(Tab)

ex:

<p>

<b>

<a></a>

</b>

</p>

<p>

控制的内容

</p>

<p>

控制的内容。。。。

<b>

加粗的文本

</b>

</p>

2、属性 和 值

属性：是用来修饰元素的。

ex：想控制 p 标记里文本的 水平排列方式 为 右对齐

语法：

1、属性的声明必须位于开始标记里

2、属性与标记之间要用一个空格隔开

3、一个元素允许有多个属性，多属性之间用 空格 隔开， 并且排名不分先后

4、属性 和 值之间用 = 连接

ex:

文本水平排列方式：align

取值：left/center/right

<p align="right" id="p1">这是一段文本</p>

设置 id 属性值为 p1

标准属性：

html中所有元素都具备的属性称之为标准属性

1、id ：元素在页面中的独一无二的标识

2、title ：鼠标移入到元素上时所提示的文本

3、class ：指定元素引用的类选择器(样式中使用)

4、style ：定义元素的内联样式(样式中使用)

3、注释

不会被浏览器所编译的内容，称为注释

语法：

<!-- -->

注意：

1、注释不能嵌套

<!--

这是一段注释

-->

2、注释不能写在<>中

<div <!-- -->></div>错误

4、文档结构

1、HTML文档结构

任何网页都是由

1、文档类型声明

<!doctype html>

2、html页面

<html></html>

两部分组成

2、html 元素

包含两个子元素

1、<head></head>

用于表示网页头部信息,用于定义页面全局信息或页面的元数据

2、<body></body>

显示给用户去看的内容

3、head元素

允许包含的子元素:

1、<title>网页标题</title>

2、<meta>

<meta charset="utf-8">

告诉浏览器用什么样的编码解析网页内容

注意：

保证网页文件的 编码格式 也是 utf-8

3、文本标记

1、特殊文本

任意多的回车和空格，最终只会被解析成一个空格

在页面输出：<p>这是一个p标记</p>

在页面中，要通过"转义字符"来表示 空格，<,> 等一些字符

1、&nbsp;

表示一个空格

2、&gt;

表示一个 >

3、&lt;

表示一个 <

4、&copy;

表示一个 ©

5、&yen;

表示一个 ￥

2、文本标记

1、文本样式

<b></b> : 加粗

<i></i> : 斜体

<u></u> : 下划线

<s></s> : 删除线

<sup></sup> : 上标

<sub></sub> : 下标

2、标题元素

语法：

<h#></h#>

#:1~6

显示效果：

1、独占一行

2、文本以加粗方式出现

3、改变字号(1级最大，6级最小)

4、具备垂直空白距离

属性：

1、align

作用：控制当前文本的水平排列方式

取值：

left/center/right

3、段落元素

语法：<p></p>

效果：

1、独占一行

2、具备垂直空白

属性：

1、align

取值：left/center/right

4、换行元素

语法：<br>或<br/>

5、分割线元素

作用：在页面中绘制一条水平线

语法：<hr> 或 <hr/>

效果：独占一行

属性：

1、size

大小，以px(像素)作为单位

2、width

宽度，以px 或 % 作为单位

3、align

水平排列方式

4、color

6、预格式化元素

作用：保留源文档中的格式，即保留源文档中所有的回车和空格的效果

语法：<pre></pre>

7、块分区元素-div

作用：做布局

语法：<div>内容 </div>

效果：

1、独占一行

属性：

1、align

8、行内分区元素-span

作用：设置同一行文字内的不同格式

ex:既有 加粗，斜体，下划线的文本

<b>

<i>

<u>一段文本</u>

</i>

</b>

<span 此处定义样式>一段文本</span>

9、行内元素 与 块级元素

行内元素 与 块级元素在页面中的位置效果是不一样的。

块级元素：默认情况下，块级元素会独占一行，如：div,p,h1~h6。所有的块级元素，都可以做布局。

行内元素：默认情况下，元素是不会换行的.行内元素在页面中主要负责修改文本的样式。ex:span,i,b,s,u,sup,sub

扩展：

1、嵌套问题

1、p 标记 不能嵌套块级元素

<p>

<div></div>

</p> 错误!

2、尽量不要让行内元素嵌套块级元素

<span>

<div></div>

</span> 尽量不要出现

2、meta

1、定义关键字

面向搜索引擎

<meta name="keywords" content="关键字1,关键字2,...">

2、定义网页描述信息

面向搜索引擎

<meta name="description" content="描述信息">

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

1、图像和链接

2、表格

3、列表

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

1、图像和链接

1、URL

1、目录 & 目录结构

目录 ：文件夹

资源文件：网页中要用到的资源可以称之为资源文件

2、URL

URL(Uniform Resource Locator)，统一资源定位器，

主要用来表示网页中任何资源的"位置"URL，即路径，

在页面中主要有以下3中表现：

1、绝对路径

从文件所在的最高目录查找资源文件所经过的路径

1、网络资源

四部分组成：

1、协议名 : http / https

2、主机名(IP地址/域名) : www.baidu.com

3、目录路径 :

4、文件名 ：

2、本机资源

从盘符位置处开始

E:\test\jd\_logo.png

2、相对路径

从当前文件位置处开始查找资源文件所经过的路径

1、同目录，直接用

在同一目录下， 可以直接通过文件名称来引用资源文件

2、子目录，先进入

images/a.jpg

images/top/left/a.jpg

3、父目录，先返回

../ 返回上一级目录中

../logo.jpg

../images/logo.jpg

3、根相对路径

从网站所在的根目录出开始查找资源文件的位置

/ ：表示网站所在的根目录

ex:

/images/front/logo.jpg

2、图像

1、图像格式

1、jpeg

图形图像联合专家组

表现：\*.jpg

2、gif

图形接口格式

表现：\*.gif

特点：动态效果

3、png

可移植网络图形

表现：\*.png

特点：支持透明

2、图像元素

语法：<img> 或 <img/>

属性：

1、src

源,即要显示的图像路径

2、width

宽度，以 px 或 % 为单位

3、height

高度，以 px 或 % 为单位

注意：如果 width 和 height 只给其中一个属性赋值的话，

那么另外一个属性会跟着等比缩放

3、链接

1、链接元素

语法：<a>内容</a>

语法：

1、href

链接URL

2、target

目标,指定新网页的打开形式

取值：

1、\_self

在自身标签页中打开新网页,默认值

2、\_blank

在新标签页中打开新网页

2、链接的表现形式

1、目标为文档下载

<a href="xxx.rar/zip">内容</a>

2、电子邮件链接

<a href="mailto:zhaoxu@tedu.cn">联系我们</a>

3、返回页面顶部的空链接

<a href="#">返回顶部</a>

4、链接到Javascript

<a href="javascript:...">执行JS</a>

3、锚点

锚点：在文档中的某行位置处做记号

使用方式：

1、定义锚点

1、通过 a 标记的 name 属性

<a name="NAME1">内容</a>

2、任意标记的 id 属性

<ANY id="NAME2"></ANY>

2、链接到锚点

1、链接到本页锚点处

<a href="#锚点名称"></a>

2、链接到其它页锚点处

<a href="页面URL#锚点名称"></a>

2、表格

1、表格的作用

表格会按照一种特殊的结构来摆放页面数据

表格是由一些"单元格"，按照从左到右，从上到下的顺序排列到一起组合而成的。

主要作用：实现页面元素的布局

2、使用表格

1、创建表格

1、定义表格 ：<table></table>

2、创建表行 ：<tr></tr>

3、创建列(单元格) ：<td></td>

2、表格属性

1、table的属性

1、width ：设置表格的宽度

2、height：设置表格的高度

3、align ：表格的水平对齐方式

4、border：边框尺寸，以px为单位

5、bgcolor：背景颜色

6、cellpadding：设置单元格内边距(单元格边框与内容的距离)

7、cellspacing：设置单元格外边距(单元格与单元格之间的距离)

2、tr的属性

1、align

设置当前行内容的水平对齐方式

2、valign

设置当前行内容的垂直对齐方式

取值：top/middle/bottom

3、bgcolor

设置当前行的背景颜色

3、td的属性

1、width

2、height

3、align

4、valign

5、bgcolor

6、colspan

设置单元格跨列

7、rowspan

设置单元格跨行

3、表格标题

语法：<caption>标题内容</caption>

注意：

1、<caption>最多只能有一个

2、必须是<table>下的第一行代码

3、表格的复杂应用

1、行分组

为表行(tr)进行分组

1、表头行分组

表格中第一行分组的话，可以放在表头行分组

<thead></thead>

2、表尾行分组

表格中最后一行要分组的话，可以放在表尾行分组中

<tfoot></tfoot>

3、表主体行分组

表格中，除第一行和最后一行外其他的行，可以放在表主体行分组中

<tbody></tbody>

2、不规则表格

通过 td 的跨行或跨列属性，实现不规则表格的创建

1、跨列

又称为：合并列

从指定单元格位置处开始，横向向右合并几个单元格(包含自己)，被合并掉的单元格要删除出去。

属性：colspan="列数"

2、跨行

又称为：合并行

从指定单元格位置处开始，纵向向下合并几个单元格(包含自己)，被合并掉的单元格要删除出去。

属性：rowspan="列数"

3、表格嵌套

被嵌套的表格，必须放在td中

<table>

<tr>

<td>

<table>

<tr></tr>

</table>

</td>

<td></td>

</tr>

</table>

https://ss0.bdstatic.com/5aV1bjqh\_Q23odCf/static/superman/img/logo/bd\_logo1\_31bdc765.png

协议名：https

域名：ss0.bdstatic.com

目录路径：5aV1bjqh\_Q23odCf/static/superman/img/logo

文件名：bd\_logo1\_31bdc765.png

1、列表

1、作用

按照一定的结构来显示数据

按照从上到下的顺序来显示数据

所有的列表都由两部分组成

1、列表类型(有序或无序)

2、列表项

2、使用列表

1、有序列表

1、语法

列表类型：<ol></ol> -- Order List

列表项：<li></li> -- List Item

2、属性

列表类型属性

1、type

取值：

1、1 ：默认值，数字

2、a ：小写字符显示

3、A ：大小字符显示

4、i ：小写罗马字符

5、I ：大写罗马字符

2、start

类型的起始编号

2、无序列表

1、语法

列表类型：<ul></ul> --- Unorder List

列表项：<li></li>

2、属性

1、type

取值：

1、disc ：默认值，实心圆

2、circle ：空心圆

3、square ：实心矩形

4、none ：没有

3、定义列表

1、作用

定义列表往往用于要给出一类事物的定义的情形，如名词解释等...

2、语法

<dl></dl> : 定义一个定义列表

<dt></dt> : 定义 列表中的一个术语

<dd></dd> : 定义 列表中对术语的解释

3、使用场合

图文混排

2、结构标记

1、作用

专门搭建网页结构，用于提升标记的可读性

以后可以通过结构标记来取代做布局的div

2、详解

1、<header>元素

作用：定义网页文档的页眉(顶部信息)

语法：<header></header>

取代：<div id="header"></div>

2、<nav>元素

作用：定义页面中的导航链接部分

语法：<nav></nav>

取代：<div id="nav"></div>

3、<section>元素

作用：定义文档中的小节，在页面中可以表示主体内容部分

语法：<section></section>

取代：<div id="main">

4、<article>元素

作用：通常用于描述 论坛帖子，报纸的文章，博客信息或微博条目

语法：<article></article>

取代：<div id="article"></div>

5、<aside>元素

作用：描述页面中的边栏信息

语法：<aside></aside>

取代：<div id="left\_side"></div>

6、<footer>元素

作用：定义页面中底部的信息

语法：<footer></footer>

取代：<div id="footer"></div>

3、表单(重难点)

1、表单的作用

用于收集信息并将信息提交给服务器

表单的两个基本部分

1、实现数据交互的可见界面元素-表单控件

2、表单提交后的处理

2、表单元素

表单：收集用户数据，并且提交给服务器

语法：

1、标记

<form></form>

注意：只有放在form中的表单控件才能被提交

2、属性

1、action

要提交给服务器处理程序的地址

默认提交给本页

2、method

提交方式,默认值是 get

取值：

1、get

1、以明文方式提交数据

2、安全性较低

3、大小限制为2KB

4、向服务器要数据时，使用get

2、post

1、隐式提交

2、安全性较高

3、无大小限制

4、传递数据给服务器进行处理时，使用post

3、... ...

3、enctype

指定表单数据的编码方式即指定什么样的数据可以提交给服务器

1、application/x-www-form-urlencoded

默认值，允许将普通字符，特殊字符提交给服务器

2、multipart/form-data

允许将 文件 提交给服务器

3、text/plain

允许将普通字符提交给服务器。特殊字符，文件则不可以

3、表单控件

能与用户交互，并且提供可视化外观的HTML元素

表单控件，只有放在表单中，数据才可以被提交

表单控件分类：

1、input元素

2、textarea元素-多行文本域

3、select 和 option 元素 - 选项框

4、其他元素

表单控件详解：

1、input元素

1、作用 ：用于收集用户的信息

2、语法 ：<input>

3、属性 ：

1、type

根据type属性值，创建各种不同的输入控件

2、name

指定控件的名称，一般提交给服务器时使用

采用的是匈牙利命名法(控件缩写+作用)

3、value

控件的值，提交给服务器使用

4、disabled

禁用控件。

该属性无值

4、input 详解

1、文本框与密码框

文本框：<input type="text">

秘密框：<input type="password">

特有属性：

1、name

缩写：txt

ex:

txtUsername : 用户名称的文本框

txtUserpwd : 用户密码

2、maxlength

限制输入的字符数

ex:

<input maxlength="15">

3、readonly

只读，只能看不能写

无值

2、单选按钮和复选框

单选按钮：<input type="radio">

复选框：<input type="checkbox">

属性：

1、name

单选按钮缩写：rdo

复选框缩写：chk

注意：name属性除定义名称外，还能分组，如果属于一组的单选按钮或复选框，名称必须相同

2、value

3、checked

设置默认被选中

3、按钮

1、提交按钮

<input type="submit">

作用：功能预定义，负责将当前表单中的数据提交给服务器

2、重置按钮

<input type="reset">

作用：功能预定义，将表单中的数据恢复成初始化的状态

3、普通按钮

<input type="button">

作用：无功能，可以由用户自定义的编写功能

属性：

value ：定义按钮上的文本

name ：缩写 btn

4、隐藏域 和 文件选择框

1、隐藏域：<input type="hidden">

作用：想提交给服务器，但不想让用户看到的数据，可以保存在 隐藏域中。

缩写：txt

2、文件选择框

<input type="file">

缩写：txt

注意：

1、method 必须设置为 post

2、enctype 必须为 multipart/form-data

2、textarea元素

称为：多行文本域

语法：

1、标记

<textarea></textarea>

2、属性

1、name

定义名称，缩写 txt

2、cols

指定文本域显示的列数，变相的指定宽度

ex:

cols="50"

英文字符：显示50个

中文字符：显示25个

3、rows

指定文本域显示的行数，变相的指定高度

4、readonly

只读

3、select 和 option 元素

称为：选项框

1、语法

1、<select></select>

作用：创建选项框(下拉，滚动)

属性：

1、name

控件名称，缩写 sel

2、size

指定选项框，默认显示选项的数量。

大于1的话，则为滚动列表，否则就是下拉选项框

3、multiple

设置多选

无值

2、<option>显示的数据</option>

选项

属性：

1、value

提交给服务器的数据

2、selected

预选中

4、其他元素

1、label元素

作用：关联 文本 和 表单控件的。关联之后，单击文本，就如同单击 表单控件 一样

语法

<label>文本</label>

属性

for : 表示要与该文本关联的控件的ID值

2、为控件分组

<fieldset></fieldset> : 为控件分组

<legend></legend> : 为分组指定标题

4、其他标记

1、浮动框架

1、什么是浮动框架

可以在一个浏览器窗口中能够同时显示多个HTML文档内容

2、语法

<iframe>文本内容</iframe>

属性：

1、src

浮动框架中引入的页面url

2、width

宽度

3、height

高度

4、frameborder

边框，如果不想要边框，则将frameborder改为0

2、摘要与细节

1、作用

可以通过用户的点击操作，来显示/隐藏某一部分内容

2、语法

<details></details> : 定义摘要和细节

<summary></summary> : 定义允许被用户点击的标题文本

注意：summary 必须是 details 中的第一对子元素

3、度量元素

1、语法

<meter></meter>

作用：表示度量元素，比如：投票的比例，电池的电量... ...

2、属性

1、min ：定义度量范围的最小值，默认为0

2、max ：定义度量范围的最大值，默认为1，通常情况会将其改为 100

3、value ：当前显示在度量元素上的值

4、时间元素

1、作用

关联时间的不同表现形式

2、语法

<time>文本</time>

属性：

datetime : 表示关联的日期和时间,如果同时关联日期和时间的话，中间用 T 分隔

5、高亮文本显示

1、作用

以突出的背景色显示文本

2、语法

<mark>文本</mark>

CSS

1、CSS概述

1、问题

1、控制页面中所有的文本颜色为 红色

2、控制页面中所有div的文本颜色 为蓝色

3、将 所有div 的颜色 变为 黄色

HTML元素的不足：

1、要完成相同的效果，使用的是不同的属性

<body text="red">

<font color="blue">

2、没有提升 html属性的 可重用性和可维护性

可重用性：重复的操作 不用重复的写

可以通过 CSS 解决以上两个问题

2、什么是CSS

CSS (Cascading Style Sheets),层叠样式表，级联样式表，简称：样式表。专门用于定义页面元素的样式的。

以统一的方式定义页面元素的样式

CSS实现了内容与表现相分离

提升了代码的可重用性 和 可维护性

3、CSS 与 HTML之间的关系

HTML : 专门用于构建网页结构

CSS : 专门构建HTML网页的样式

HTML属性效果 与 CSS样式效果相冲突的话：

优先使用 CSS 定义页面元素的效果

2、CSS语法

1、使用CSS样式表

1、内联方式

又称为 ：行内样式

将样式定义在某个html元素中

语法：

<ANY style="样式声明;样式声明;"></ANY>

样式声明：

由 属性名称 和 属性值 来组成，属性名称和属性值之间用 : 来关联

ex:

属性名称 属性值

color red,blue,yellow...

background-color 合法颜色值即可..(同上)

font-size 以px，pt，为单位的数值

控制以下 文本颜色为 红色，背景颜色为 黄色，文字大小为 24px

<p>静夜思</p>

2、内部样式表

将样式定义在网页的一块独立位置处

在 <head></head> 中 增加一对 <style></style>元素，在 <style> 元素中 定义元素的样式,具体语法如下：

<head>

<title></title>

<style>

样式规则1

样式规则2

...

样式规则n

</style>

</head>

样式规则，由两部分组成

1、选择器

选择(规范)页面中哪些元素能够使用声明好的样式

比如：元素选择器，由元素名称来充当选择器，控制某标记对应所有页面元素的样式

2、若干样式声明

样式规则 语法结构：

选择器{

样式声明1;

样式声明2;

...

样式声明n;

}

ex:

p{

color:red;

background-color:yellow;

font-size:24px;

}

3、外部样式表

将样式定义在外部的"样式表文件(\*.css)"中

1、创建 .css 文件，编写样式规则

2、在要使用 样式表的 html文件中进行引入

<head>

<link rel="stylesheet" href="样式表文件url">

</head>

2、CSS样式表特征

1、继承性

大部分样式属性是可以被继承

2、层叠性

可以为一个元素定义多个样式规则

如果 属性 不冲突时，多个样式规则可以层叠为一个(即都可以应用到当前元素上)

3、优先级

在层叠性基础上，如果 样式定义 冲突的话，那么就会按照不同使用方式的优先级，来选择性应用样式

低 ：浏览器默认设置

中 ：外部样式表 和 内部样式表

就近原则

后定义优先

高 ：内联方式

4、!important 规则

显示调整样式属性的优先级

语法：

属性名:值 !important;

谨慎使用!

3、选择器(重点)

1、作用

选择(规范)页面中哪些元素能够使用声明好的样式

为了匹配页面的元素

2、详解

1、通用选择器

作用：匹配页面中所有的元素

语法：\*{}

缺点：效率较低，尽可能少用,可以通过 使用 body 的继承性 来替代 \*

2、元素选择器

作用：匹配某一标记所对应的页面中所有的元素

语法：元素/标签 {}

ex:

div{/\*匹配页面中所有div\*/}

span{/\*匹配页面中所有span\*/}

li{/\*匹配页面中所有li\*/}

3、类选择器

作用：事先定义好样式，可以由任意元素通过class属性进行引用

语法：

定义：.类名{}

引用：<ANY class="类名"></ANY>

多类选择器的引用方式：

让一个元素引用多个类选择器，中间用 空格 隔开即可

分类选择器的声明方式：

将元素选择器和类选择器结合到一起进行声明，实现对某种元素中的不同样式细分控制

ex:想要控制 class为important的div元素，文字大小为 48px

语法：元素选择器.类选择器{}

4、id选择器

作用：匹配 指定id值的元素的 样式

ex:

<div id="nav"></div>

语法：#ID值{}

5、群组选择器

群组选择器的声明以一个以 , 隔开的选择器列表

语法：

选择器1,选择器2,...{}

ex:

#title,p.redColor,.important,span{

color:red;

}

6、后代选择器

后代：

多于 一级 层级关系的元素，只能叫后代

只具备 一级 层级关系的 也可以叫后代

语法：

选择器1 选择器2{}

语义：匹配 在选择器1 中的 所有后代元素 选择器2

ex:

#content span{}

7、子代选择器

子代：

只具备 一级 层级关系的 称之为 子代

语法：

选择器1>选择器2{}

语义：匹配 在选择器1 中的 子级元素 选择器2

8、伪类选择器

伪类：专门匹配元素不同状态的选择器

分类：

1、链接伪类

2、动态伪类

3、目标伪类

4、元素状态伪类

5、结构伪类

6、否定伪类

语法：

以 :作为开始的选择器

1、链接伪类

1、:link

适用于未被访问的超链接

2、:visited

适用于访问过的超链接

2、动态伪类

1、:hover

适用于鼠标悬停在html元素时

2、:active

适用于html元素被激活时

3、:focus

适用于html元素获取焦点时

3、尺寸与边框

1、CSS单位

1、尺寸单位

1、px ：像素

2、pt ：磅 (1pt=1/72in)

3、in ：英寸，1in=2.54cm

4、% ：百分比，占据 当前元素 或 父元素 对应属性的 占比

5、em ：倍数，多数用于 文字大小的描述

6、mm ：毫米

7、cm ：厘米

注意：css中尺寸单位是不能省略的。

2、颜色单位(取值)

1、rgb(r,g,b)

r : red , 范围 0-255

g : green,范围 0-255

b : blue, 范围 0-255

ex：

background-color:rgb(0,0,0);黑色

background-color:rgb(255,0,0);红色

background-color:rgb(255,255,255);白色

2、rgb(r%,g%,b%)

3、rgba(r,g,b,alpha)

alpha : 透明度 0-1之间的数字

0 : 完全透明

1 : 完全不透明

0.5 : 半透明

4、#rrggbb

通过6位16进制数字表示一个颜色

每位数字范围：0-9 A-F

#00ff00 : 绿色

#ff0000 : 红色

#1a2b3c : ?

5、#rgb

#rrggbb 每两位数字相同时，可以使用#rgb简化

#00ff00 -> #0f0

#aabbcc -> #abc

#accdda -> ×

6、颜色的英文表示

red,green,blur,orange,yellow...

2、尺寸属性

1、作用

用于设置元素的宽度和高度

单位一般为 px 或 %(父元素对应属性的占比)

2、语法

1、宽度

width

min-width : 最小宽度

max-width : 最大宽度

2、高度

height

min-height : 最小高度

max-height : 最大高度

注意：

页面中所有元素的高度，在不指定的情况全部是自适应(以内容为主)。

页面中所有块级元素的宽度，默认占据父元素的100%

页面中所有行内元素的宽度，默认自适应(以内容为主)

3、页面中允许修改 尺寸 属性的元素

1、所有块级元素都允许修改

2、大部分的行内块元素

所有的表单元素都是行内块元素

除radio、checkbox之外，其他元素允许修改尺寸

3、本身html元素就具备width 和 height的

img，table 可以修改尺寸

4、行内元素(除具备width和height之外)是不能修改尺寸

span,b,a,i,s,b ... 不能修改尺寸

4、溢出

1、什么是溢出

使用尺寸属性限制元素大小时，如果内容所需要的空间大于元素本身空间，则会导致内容溢出

2、溢出处理属性

属性：

overflow

overflow-x : 横向溢出处理

overflow-y : 纵向溢出处理

取值：

1、visible

默认值，溢出可见

2、hidden

隐藏

3、scroll

滚动，让元素出现滚动条，溢出时，滚动条可用

4、auto

自动，溢出时显示滚动条并可用，非溢出时，不显示滚动条

3、边框属性

1、边框

1、简写方式

border:width style color;

作用：设置元素上下左右四个边框的宽度，样式以及颜色

width:边框的宽度(尺寸)

style:边框的样式

取值：

1、solid ：直线

2、dotted ：虚线(点状)

3、dashed ：虚线(线状)

color:边框的颜色,可以取值为 transparent(透明)

ex:

div{

border:1px solid red;

}

2、单边定义

border-方向:width style color;

方向可以被以下关键字所取代

top : 上

right : 右

bottom : 下

left:左

ex:

border-bottom:2px dashed #00ff00;

3、单属性定义

border-属性:值;

属性可以被以下关键字取代：

width:宽度

style:样式

color:颜色

ex:

border-width:5px;

border-style:solid;

border-color:#f00f00;

4、单边单属性定义

border-方向-属性:值;

方向：top/right/bottom/left

属性：width/style/color

ex:

border-top-color:yellow;

border-right-style:dotted;

屏幕截图、取色工具：FastStoneCapture

1、边框

1、边框属性

边框是由四个三角形或梯形组成的

2、边框倒角

将边框的四个直角倒换成圆角

属性：

border-radius

取值：

1个值：四个角的倒角半径

2个值：...

3个值：...

4个值：左上 右上 右下 左下 倒角半径

取值单位：

1、px ：具体数值

2、% ：使用元素的宽度和高度的占比来作为倒角半径

如果 元素本身是 正方形，border-radius 指定为 50%的话，则当前元素为圆形

单独定义：

border-top-left-radius:左上角倒角半径

border-top-right-radius:右上

border-bottom-left-radius:左下

border-bottom-right-radius:右下

3、边框阴影

属性：box-shadow

取值：由多个值组成的值列表

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

1、h-shadow ：(必须)

阴影的水平偏移距离

取值为正，则向右偏移

取值为负，则向左偏移

2、v-shadow ：(必须)

阴影的垂直偏移距离

取值为正，则向下偏移

取值为负，则向上偏移

3、blur

可选值，模糊距离

4、spread

可选值，阴影大小

5、color

可选值，颜色

6、inset

可选值，将默认的外阴影改为内阴影

4、轮廓

1、什么是轮廓

轮廓是绘制于元素周围的一条线，位于边框之外

2、属性

outline:width style color;

outline-width:

outline-style:

outline-color:

outline:none;或outline:0; 取消轮廓

border:none;或border:0;取消边框

2、框模型

1、什么是框模型

框模型(Box Model),也称为"盒模型"，定义了元素框处理元素内容，内边距，边框和外边距的方式

框：页面元素皆为框

cellpadding:边框内边距 - 边框与内容之间的距离

cellspacing:边框外边距 - 单元格与单元格之间的距离

如果为元素设置了框模型属性的话，那么元素最终所占据的区域尺寸会发生改变，但不会影响内容区域尺寸

住人：300平米(内容区域尺寸)

院子：500米 (内边距)

搭院子砖头宽度：5米(边框)

院子外：周围 1公里(外边距)

我家最终占地面积：300平米 + 500米 + 5米 + 1公里

住人：300平

元素占地尺寸(宽度) = 左右外边距 + 左右边框 + 左右内边距 + width

元素占地尺寸(高度) = 上下外边距 + 上下边框 + 上下内边距 + height

2、外边距

1、什么是外边距

围绕在元素边框周围的空白区域就是外边距

默认情况下，是不允许被其它元素所占据的

2、语法

属性：

margin:value;/\*上下左右四个方向的外边距值\*/

margin-top/right/bottom/left:value;

取值：

1、px ：具体数值

2、% ：宽和高的占比作为外边距

3、auto ：自动，只对左右外边距有效,控制块级元素水平居中对齐的。(前提是必须设置元素宽度)

4、取值为负数 ：移动元素

简便写法：

margin:value;/\*四个方向外边距值\*/

margin:v1 v2;/\*v1:上下 v2:左右\*/

margin:v1 v2 v3;/\*v1:上 v2:左右 v3:下\*/

margin:v1 v2 v3 v4;/\*上 右 下 左\*/

ex:

margin:0 auto;

margin:5px 10px 3px;

3、具备默认外边距的元素

可以通过 css重写 的方式，更改具备默认外边距的元素

4、特殊问题

1、外边距合并

当两个垂直外边距相遇时，它们将形成一个外边距，合并后外边距的高度等于两个元素中外边距较大的值。

2、外边距溢出

在某些特殊情况下，给子元素增加垂直外边距时，有可能会作用到父元素上

1、父元素没有上下边框时

2、为第一个子元素(上外边距)或最后一个子元素(下外边距)设置外边距时

解决方案：

1、为父元素增加边框(透明)

弊端：会额外增加元素的高度

2、通过为父元素设置上内边距取代子元素的上外边距

弊端：会额外增加元素的高度

3、为父元素再增加一个子元素在第一个位置处

<table></table>

4、... ...

3、为行内元素和行内块元素设置上下外边距的异常

行内元素：对上下外边距无效

行内块元素：设置上下外边距时，整个一行都跟着变。

3、内边距

1、什么是内边距

内容区域和边框之间的距离

注意：内边距会扩大元素边框所占区域

2、语法

属性：

padding:value;/\*四个方向内边距值\*/

padding-top/right/bottom/left:value;

取值单位：

1、px

2、%

3、auto

简写方式：

padding:value;/\*四个方向内边距\*/

padding:v1 v2;/\*上下 左右\*/

padding:v1 v2 v3;/\*上 左右 下\*/

padding:v1 v2 v3 v4;/\*上 右 下 左\*/

3、注意问题

1、为行内元素和行内块元素增加上下内边距时，只影响当前元素自己，并不影响其他元素

4、box-sizing

重新指定框模型数据的计算模式

默认情况：

元素最终宽度=左右外边距+左右边框+左右内边距+w;

元素边框以及边框内宽度=左右边框+左右内边距+w;

ex:

div{

width:150px;

padding:15px 10px 3px 8px;

border:5px solid red;

}

边框内宽度：150+10+8+10 = 178px

属性：box-sizing

取值：

1、content-box

默认值

元素的padding和border会额外的加载width,height上

2、border-box

给定的width(height)值，会包含 padding以及border;

3、背景属性

1、背景色

属性：background-color

取值：合法颜色值

注意：背景色是从边框位置处开始绘制

2、背景图像

属性：background-image

取值：url(图像地址)

ex:

background-image:url(images/a.jpg);

3、背景重复(平铺)

属性：background-repeat

取值：

1、repeat

默认值，垂直和水平两个方向都平铺

2、repeat-x

只在 水平方向 平铺

3、repeat-y

只在 垂直方向 平铺

4、no-repeat

不平铺

4、背景图片尺寸

属性：background-size

取值：

1、value1 value2

value1：宽度

value2：高度

2、value1% value2%

采用元素宽和高的占比，决定背景图大小

3、cover

覆盖

将背景图进行等比扩大，直到覆盖到元素所有区域为止。

4、contain

包含

将背景图进行等比扩大，直到右边或下边有一个边缘碰到元素位置

5、背景图片固定

背景图不会随着滚动条而发生位置的改变，永远都在可视化范围的固定位置处。

属性：background-attachment

取值：

1、scroll

默认值

2、fixed

固定

6、背景图像定位(位置)

属性：background-position

取值：

1、x y

x：背景图片水平偏移距离

取值为正，图片右移动

取值为负，图片左移动

y：背景图片垂直偏移距离

取值为正，图片下移动

取值为负，图片上移动

2、x% y%

采用元素的宽和高以及图片的宽和高计算出来的偏移量

3、关键字

x ：left/center/right

y ：top/center/bottom

ex:

background-position:left 15px;

7、背景属性

通过 background 属性设置所有有关背景的属性值

属性：background

取值：color url() repeat attachment position;

常用方式：

background:url() repeat position;

background:url(a.jpg) no-repeat 25px -15px;

8、CSS Sprites(雪碧图，精灵图)

将若干幅小图合并到一副大图中，根据实际使用需求，改变图像的位置，从而显示出向看的图片

如何显示 雪碧图 中的某一块图像???

1、根据要显示的图像创建一个一模一样大小的元素

2、为元素设置背景图，并且移动图像到显示区域

1、渐变

1、什么是渐变

两种或多种颜色之间平滑过渡的效果

2、渐变分类

1、线性渐变

按照一条直线的方式填充渐变色

2、径向渐变

按照原型的方式向外放射性填充渐变颜色

3、重复渐变

将线性渐变和径向渐变重复实现多次

3、色标

每个渐变颜色 出现的 "位置" 及其 "颜色值"

4、语法

1、属性 & 取值

属性：background-image

取值：

1、线性渐变

linear-gradient()

2、径向渐变

radial-gradient()

3、重复渐变

repeating-linear-gradient()

repeating-radial-gradient()

2、线性渐变

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

1、angle ：线性渐变的方向或角度

取值：

1、关键字

1、to top : 从下向上填充

2、to right : 从左向右填充

3、to bottom : 从上向下填充

4、to left : 从右向左填充

2、角度

可以取值为 0deg ~ 360deg

to top -> 0deg

to right -> 90deg

to bottom -> 180deg

to left -> 270deg

2、color-point ：色标

由渐变的颜色 以及 出现的位置 组成

ex:

1、red 0px

开始的位置是红色

2、green 50px

按照填充的方向，50px的位置处，变为绿色

3、blue 100%

按照填充的方向，结束的位置处，变为蓝色

注意：色标中，可以省略位置不写，如果省略位置的话，css会按照元素的大小及填充方向平均分配渐变色

3、径向渐变

background-image:radial-gradient([size at position,]color-point,color-point);

1、size at positioin

可以省略，如果省略的话，css会默认分配渐变色，圆心在元素的中间位置处

size ：圆的半径，以px为单位

at ：关键字，不能省略

position ：圆心位置

4、浏览器兼容性

主流版本浏览器支持性比较好

对于不支持的版本，通过增加浏览器前缀的方式实现兼容性

Firefox ：-moz-

Chrome & Safari ：-webkit-

Opear ：-o-

浏览器前缀加在什么位置?????

1、如果浏览器不支持属性的话，则将前缀加到属性名称前

animation:....;

-moz-animation:...;

-webkit-animation:....;

-o-animation:....:

2、如果浏览器支持属性，但不支持属性值的话，则将前缀加到属性值的前面

background-image:linear-gradient();

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

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

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

2、文本格式化

1、文本格式化

字体属性：大小，加粗，系列等

文本格式：文本颜色，线条样式，... ...

2、字体属性

1、指定字体系列

属性：font-family

取值：value1,value2,...

注意：如果字体中包含中文以及特殊符号的话，要用""引起来。

ex：

font-family:"微软雅黑",Arail,Times;

font-family:"microsoft yahei";

2、指定字体大小

属性：font-size

取值：以px或pt或em 为单位的数值;

3、指定字体加粗

属性：font-weight

取值：

1、normal

正常体

2、bold

加粗

3、value

400 ~ 900 之间的整百倍数

400 : normal一样

900 : bold一样

4、指定字体样式

属性：font-style

取值：

1、normal

正常体

2、italic

斜体

5、小型大写字母

作用：将元素中的所有小写英文字母转换为大写，但是大小与小写字符一样

属性：font-variant

取值：

1、normal

正常显示

2、small-caps

小型大写字符

3、文本属性

1、文本颜色

属性：color

取值：合法颜色值

2、文本排列方式

属性：text-align

取值：left/right/center

3、文字修饰

属性：text-decoration

取值：

1、none

无任何线条显示

2、underline

显示下划线

3、overfline

显示上划线

4、line-through

显示删除线

4、行高

作用：一行数据的高度。如果行高的高度高度数据本身高度，那么该行数据将在指定的行高范围内垂直居中。

属性：line-height

取值：px 为单位的数值

使用场合：

1、解决一行数据的垂直居中问题

2、拉长多行数据之间的间距-行间距

5、首行文本缩进

属性：text-indent

取值：px为单位的数值

6、文本阴影

属性：text-shadow

取值：h-shadow v-shadow blur color;

3、表格

1、表格常用样式

1、边距属性

padding(table,td)

margin(table),不适用于td

2、尺寸

width,height

3、文本格式化属性

font-\*

text-\*

4、背景属性(颜色，图片，渐变)

5、border

6、vertical-align

内容垂直对齐方式

属性：vertical-align

取值：top/middle/bottom

2、表格特有样式

1、边框合并

属性：border-collapse

取值：

1、separate

默认值，分离边框模式

2、collapse

合并边框模式，table和td将合并成一个边框

2、边框边距

作用：设置单元格与表格，单元格与单元格之间的距离(cellspacing)

属性：border-spacing

取值：

1个值：水平和垂直间距相同

2个值：第1个值 表示水平间距，第2个值 表示垂直间距。两个值之间用空格隔开

注意：border-collapse属性必须是 separate时，边框边距才有效，否则无效

3、标题位置

属性：caption-side

取值：

1、top

默认值，标题位于表格之上

2、bottom

标题位于表格之下

4、显示规则

作用：指定浏览器如何显示或布局一张表格，即设置单元格，行，表格的算法规则

属性：

table-layout

取值：

1、auto

自动表格布局，默认布局方式，单元格的尺寸实际上是由内容决定的。

2、fixed

固定表格布局，单元格的尺寸由设定的数据值来决定的，不由内容决定

自动表格布局 VS 固定表格布局

1、自动表格布局(默认布局方式)

1、单元格的大小由内容来决定

2、表格复杂时，加载会比较慢(缺点)

3、适用于不确定每列大小时使用

4、虽然效率较低，但比较灵活(特点)

2、固定表格布局

1、单元格的大小由设定的值来决定，与内容无关

2、会加速显示表格，不需要每行数据都进行计算(特点)

3、适用于确定每列大小时使用

4、虽然效率较高，但不够灵活(缺点)

1、浮动定位

1、定位

简单点说，定位就是改变元素在页面上的默认位置

2、定位的分类

按照定位的效果，可以分成以下几类：

1、普通流定位(默认方式)

2、浮动定位

3、相对定位

4、绝对定位

5、固定定位

3、定位方式-普通流定位

普通流定位，又称之为"文档流定位"，页面中所有元素默认定位方式

所有元素默认都是在其父元素的左上方开始出现

页面中所有的块级元素，按照从上到下的方式排列，每个元素独占一行

页面中所有的行内元素，按照从左到右的方式排列，多个元素在一行内显示，显示不下自动换行

问题：多个块级元素想在一行内显示 - 浮动定位

4、定位方式-浮动定位

1、浮动定位的概述 & 特点

如果将元素的定位方式设置为浮动定位的话，那么元素将具备以下特征

1、元素会排除在文档流之外即脱离文档流，不占据页面空间(但可以显示在页面中)。其他剩余未浮动元素要上前补位

2、浮动元素会停靠在父元素的左边或右边，或者其他已浮动元素的边缘上

3、浮动只在当前行浮动

4、浮动元素依然位于包含框之内

5、浮动元素处理的问题-让多个块级元素在一行内显示

2、语法

属性：float

取值：

1、none

默认值，即无浮动效果

2、left

左浮动，停靠在父元素的左边，或挨着左侧已有的浮动元素

3、right

右浮动，停靠在父元素的右边，或挨着右侧易用的浮动元素

3、浮动引发的特殊效果

1、当父元素的宽度显示不下所有的已浮动子元素，那么最后一个将换行，但，有可能被卡

2、元素一旦浮动起来之后，宽度将变成自适应(由内容来决定宽度)

3、元素一旦浮动起来之后，那么将变成块级元素，尤其对行内元素影响最大

块级元素：能修改尺寸

行内元素：不能修改尺寸

4、文本，行内元素，行内块元素时采用环绕的方式排列的，是不会被浮动元素压在底下的，而会巧妙的避开浮动元素

4、清除浮动

元素浮动起来之后，会对后续的元素带来位置的影响。如果后续元素不想受到任何影响的话，可以通过清除浮动的方式来解决。清除浮动后，元素不会上前占位。

属性：clear

取值：

1、none

默认值，不做任何清除操作

2、left

清除当前元素前一个元素左浮动带来的影响

3、right

清除当前元素前一个元素右浮动带来的影响

4、both

清除当前元素前一个元素左或右浮动带来的影响

5、浮动元素对父元素高度的影响

由于浮动元素会脱离文档流，所以会导致元素不会占据父元素的控件，会对父元素的高度带来一定的影响。父元素高度是以没有浮动子元素的高度为准。

解决方案：

1、直接设置父元素高度

弊端：必须知道父元素的高度是多少

2、设置父元素也浮动

弊端：不是任何时候父元素都需要浮动，元素浮动后会影响后续元素的位置

3、为父元素设置overflow属性

取值为：hidden 或 auto

弊端：如果内容中有东西要溢出显示(如弹出框)那么也会一同被隐藏

4、父元素中，追加空块级子元素，并设置其clear属性为both

5、... ...

2、显示

1、显示方式

显示方式决定了元素在页面中默认的定位方式

属性：display

取值：

1、none

让生成的元素不显示，并且脱离文档流

2、block

让元素表现的和块级元素一样

3、inline

让元素表现的和行内元素一样

4、inline-block

让元素表现的和行内块一样

多个行内块在一行内显示，并且允许修改尺寸(radio,checkbox除外)

2、显示效果

1、visibility

作用：可见性

属性：visibility

取值：

1、visible

默认值，可见的

2、hidden

隐藏，依然占据页面空间

3、collapse

用在表格中，删除一行或一列时，不影响表格整体布局

Visibility:hidden 与 display:none的区别

display:none; 脱离文档流，所以不占页面空间

visibility:hidden; 没有脱离文档流，所以占页面空间，不显示内容而已

2、opacity

作用：指定元素的透明度

属性：opacity

取值：0(完全透明) - 1(完全不透明) 之间的数字

3、vertical-align

作用：垂直居中对齐,控制的是当前元素两边的文本或行内元素相对于当前元素的垂直对齐方式

注意：用在 td 或 行内块元素中 或 img中

取值：

1、top ：顶部对齐

2、middle ：中间对齐

3、bottom ：底部对齐

4、baseline ：基线对齐

基线对齐：

1、行内块元素的基线

基线是在最后一行文本的底端

2、图像

基线是在图像底部再向下3px处

3、光标

作用：改变鼠标的显示状态

属性：cursor

取值：

1、default

默认

2、pointer

小手，与a标记移入时一样

3、crosshair

+

4、text

I

5、wait

等待

6、help

帮助

3、列表

1、列表项标识

属性：list-style-type

取值：

1、none

无标记

2、disc

3、square

4、circle

... ...

2、列表项图像

属性：list-style-image

取值：url();

3、列表项位置

默认位置：标识默认列表项内容区域之外

属性：list-style-position

取值：

1、outside

默认值

2、inside

将标识放置于列表项区域之内

4、列表属性

属性：list-style

取值：type url() position;

常用方式：list-style:none;

4、定位

1、定位属性

1、定位方式

属性：position

取值：

1、static ：静态定位，默认定位

2、relative ：相对定位

3、absolute ：绝对定位

4、fixed ：固定定位

注意：将position设置为relative/absolute/fixed 任意一种定位方式的话，那么该元素就称之为 "已定位元素"

2、偏移属性

属性：

1、top ：以元素顶端作为基准点，上下移动元素

2、bottom ：以元素底端作为基准点，上下移动元素

3、left ：以元素左边作为基准点，左右移动元素

4、right ：以元素右边作为基准点，左右移动元素

3、堆叠顺序

作用：堆叠效果中，显示调整堆叠顺序

属性：z-index

取值：没有单位的数字

2、定位-相对定位

1、什么是相对定位

元素会相对于它原来的位置偏移某个距离

实现元素位置的微调

2、语法

属性&值：

position:relative;

配合偏移属性实现位置的微调

top/bottom/left/right

3、定位-绝对定位

1、什么是绝对定位

1、绝对定位的元素会脱离文档流，不占据页面空间

2、会相对于它 最近的 已定位的 祖先元素去实现位置的初始化

3、如果当前元素没有 已定位的祖先元素，那么就会相对于最初的包含框实现位置初始化，比如body或html

4、绝对定位的元素会变成块级元素

2、语法

属性 & 值:

position:absolute;

配合着 偏移属性 实现元素的定位(位置移动)

3、使用场合

1、弹出菜单一律采用绝对定位的方式

4、堆叠顺序

1、什么是堆叠顺序

产生堆叠效果之间的元素的排列顺序

2、语法

属性：z-index

取值：没有单位的数值

3、注意问题

1、必须是已定位元素才能使用z-index

2、默认顺序是 后来者居上

3、父子关系之间无法用z-index调整堆叠顺序，永远都是 子压在父上

1、固定定位

1、什么是固定定位

将元素固定在网页的可视化的某个位置处，不会随着滚动条而发生位置的改变

2、属性 & 值

position:fixed;

通过偏移属性实现位置的调整

top/right/bottom/left

3、注意

1、固定定位元素会脱离文档流

2、固定定位元素会变成块级元素

3、固定定位元素永远都是相对于body实现位置的初始化

1、复杂选择器

2、内容生成

3、多列

4、CSS Hack

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

1、复杂选择器

1、兄弟选择器

兄弟：平级元素

兄弟选择器：只能向后找元素，不能向前找元素

<div>

<p></p>

<span></span>

<div></div>

</div>

1、相邻兄弟选择器

1、作用

紧紧挨着的元素称之为相邻元素

2、语法

选择器1+选择器2{}

ex:

1、div+span

匹配 紧紧挨着div元素后面的span元素

2、#d1+span

匹配id为d1的元素后紧紧挨着的span元素

3、#d1+.important

匹配 紧紧挨着 id为d1的元素 后面的 class为important的元素

2、通用兄弟选择器

1、作用

通用:向后匹配所有

2、语法

选择器1~选择器2{}

ex:

1、#d1~span

匹配 #d1 后面 所有的 span元素

2、#d1~.important

匹配 #d1 后面 所有class为important元素

2、属性选择器

1、作用

允许使用元素所附带的属性 及其 值 ，进行元素的匹配

ex:匹配页面中所有的文本框

2、语法

[属性]

1、[attr]

attr:表示任意属性名称

作用：匹配页面中所有附带attr属性的元素

ex:

1、[id]

匹配页面中所有附带id属性的元素

2、[title]

匹配页面中所有附带title属性的元素

2、elem[attr]

elem:表示任意元素

作用：匹配页面中所有附带attr属性的elem元素

ex:

1、p[id]

匹配页面中所有附带id属性的p元素

2、img[title]

匹配页面中所有附带title属性的img元素

3、[attr1][attr2]

作用:匹配即具备attr1属性同时也具备attr2属性的元素

ex:

1、div[id][class]

匹配即具备id属性同时也具备class属性的div元素

4、[attr=value]

[attr="value"]

[attr='value']

匹配 具备attr属性并且值为value的元素

ex:

1、匹配页面中所有的文本框

input[type="text"]

5、[class~=value]

多类选择器中使用

class属性值是以空格隔开的值列表，value是值列表中的独立的值

只要class属性值中，包含value选择器，就能够匹配出来

6、[attr^=value]

^= : 以...作为开始

匹配 attr 属性值 是以 value 字符作为开始的元素

<div class="col-5"></div>

<div class="col-3"></div>

ex:

1、[class^="col-"]

匹配 class 属性值 是以 col- 字符作为开始的元素

7、[attr$=value]

匹配 attr 属性值 是以 value 字符作为结束的元素

ex:

1、[class$="users"]

匹配 class 属性值 是以 users 字符作为结束的元素

8、[attr\*=value]

\*=:包含

匹配 attr 属性值中 包含 value 字符的元素

ex：

1、[class\*="cols"]

3、伪类选择器

1、目标伪类

1、作用

匹配 被激活的 html锚元素(锚点)

2、语法

:target 或 选择器:target

2、元素状态伪类

1、作用

匹配元素可用，禁用，被选中的状态

2、语法

1、:enabled

匹配页面中每个已启用的元素(表单元素)

2、:disabled

匹配页面中每个被禁用的元素(表单元素)

3、:checked

匹配每个被选中的元素(radio,checkbox)

3、结构伪类

1、:first-child

匹配 属于其父元素中的 首个子元素

2、:last-child

匹配 属于其父元素中的 最后一个子元素

3、:nth-child(n)

匹配 属于其父元素中的 第n个子元素

ex:

td:nth-child(2)

nth-child(1) == :first-child

4、:empty

匹配没有子元素(包括文本)的元素

ex:

<div>Hello World</div> : 非空

<div> </div>:非空

<div>

</div>:非空

<div></div>:empty

5、:only-child

匹配属于其父元素中的唯一子元素

<div>

<p></p> :only-child

</div>

4、否定伪类

匹配 非指定 选择器的每个元素

语法：

:not(selector)

将 满足 selector 选择器的元素 排除在外

ex:

1、控制 除第一行以外的 所有行的 第一列的背景颜色 为 红色

#tbl tr:not(:first-child){}

4、伪元素选择器

1、伪类 VS 伪元素

伪类：匹配到的是完整的元素

伪元素：匹配到的是元素中的某部分内容

2、语法

1、:first-letter 或 ::first-letter

匹配某元素中的首字符

2、:first-line 或 ::first-line

匹配某元素中的首行

3、::selection

匹配被用户选取的部分

3、: 与 ::

1、:

在css中表示的是 伪类选择器

在css2中还可以表示 伪元素选择器

2、::

在css3中，:只表示为类选择器

在css3中，::只表示伪元素选择器

2、内容生成

1、作用

允许通过css的方式，动态的像某元素的内容区域生成或插入一部分数据

2、语法

1、伪元素选择器

1、:before 或 ::before

匹配某元素的内容区域之前

<div>(:before)This is a div</div>

2、:after 或 ::after

匹配某元素的内容区域之后

<div>This is a div(:after)</div>

2、属性

属性:content

取值:

1、字符串 ：普通文本

2、url() ：插入一副图像

3、计数器

3、计数器

1、属性 ：counter-reset

作用：声明/复位 一个计数器

counter-reset:名称 值;

注意：

1、值可以省略，省略的话，默认为0

2、允许一次性声明多个计数器以及值

counter-reset:名1 值1 名2 值2;

ex:

counter-reset:c1 c2;

counter-reset:c1 0 c2 10;

3、在哪声明计数器？

不能放在使用计数器的元素中声明

2、属性：counter-increment

作用：用于设置某个计数器每次出现的增量值

语法：

counter-increment:名 增量值;

注意：

1、增量值可以省略，默认为1

2、增量值可以是正，也可以是负

正：递增

负：递减

3、可同时设置多个计数器增量，空格隔开

counter-increment:c1 1 c2 -1;

4、在哪声明计数器的增量？？？

哪个元素使用计数器，在哪个元素中声明增量

3、函数：counter()

作用：使用计数器生成的值

语法：

配合着 内容生成 一起来用

content:counter(计数器名);

content:"第"counter(名)"章、";

3、多列

1、分隔列

将一段文本拆成几列

属性：column-count

取值：数字(无单位)

ex:

column-count:3;

2、列间隔

指定每两个列之间的间隔距离

属性：column-gap

取值：距离值

ex:

column-gap:15px;

3、列规则

指定每两个列之间的分隔线的 宽度，样式，颜色

属性：column-rule

取值：width style color;

4、浏览器兼容性

IE10+,Opera 支持多列

Firefox : -moz-

Chrome & Safari : -webkit-

4、CSS Hack

1 、原则 & 原理

原则：根据不同的浏览器编写不同的css

原理：根据样式的优先级 来实现兼容性的处理

2、CSSHack 的解决方式

1、CSS类内部Hack

1、实现

通过 css 属性 增加前缀或为属性值增加后缀的方式来匹配不同的浏览器

2、选择器Hack

3、HTML头部引用Hack

1、CSS Hack

1、头部引用Hack

通过 HTML 条件注释 的方式，判断浏览器的类型及版本，选择性的执行某些内容

1、语法

<!--[if 条件 IE 版本]>

条件满足时则执行里面的内容

<![endif]-->

1、版本

6 ~ 10

如果省略版本，则判断是否为IE浏览器

2、条件

1、省略不写

判断是否为IE，或是否为指定版本的IE浏览器

ex:

1、<!--[if IE]><![endif]-->

判断是否为IE浏览器

2、<!--[if IE 7]><![endif]-->

判断是否为IE7浏览器

2、gt

判断 浏览器是否大于指定版本

ex:

1、<!--[if gt IE 6]><![endif]-->

判断是否 大于IE 6浏览器

3、gte

判断 浏览器是否大于等于指定版本

ex:

1、<!--[if gte IE 6]><![endif]-->

判断是否 大于等于IE6版本

4、lt

判断 浏览器是否小于指定版本

5、lte

判断 浏览器是否小于等于指定版本

6、!

判断 浏览器是否是条件以外的其他版本

<!--[if ! IE 6]><![endif]-->

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

1、转换

2、过渡

3、动画

4、CSS优化

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

1、转换

1、转换的简介

1、什么是转换

改变元素在页面中的形状，位置，大小，角度的一种效果

在页面中 允许 实现

1、2D转换

允许元素在x轴 和 y轴上做转换效果

2、3D转换

允许元素在x轴，y轴和z轴上做转换效果

2、转换属性

1、transform

属性：transform

取值：指定一组转换函数,多个转换函数之间用空格隔开

2、转换的原点

属性：transform-origin

取值：数值/百分比/关键字

两个值：原点在 x轴 和 y轴的位置

三个值：原点在 x轴，y轴，z轴的位置

2、2D转换

1、2D-位移

改变元素在页面中的位置

语法：

属性：transform

取值：

translate(x)

改变元素在x轴的位置

取值为正，右移

取值为负，左移

translate(x,y)

x : 改变元素在x轴的位置

y : 改变元素在y轴的位置

取值为正，下移

取值为负，上移

translateX(x)

改变元素在x轴的位置

translateY(y)

改变元素在y轴的位置

2、2D-缩放

改变元素在页面的默认大小

语法：

属性：transform

取值：

1、scale(value)

指定x轴和y轴的缩放比例(倍数)

2、scale(x,y)

分别指定x轴和y轴的缩放比例

3、scale(x)

指定x轴缩放比例

4、scale(y)

指定y轴缩放比例

函数值：

默认 1

缩小：0 - 1

放大：大于1

3、2D-旋转

改变元素在页面中的角度

语法：

属性：transform

取值：rotate(ndeg)

根据转换的原点实现角度的旋转

n 取值为正，则顺时针旋转

n 取值为负，则逆时针旋转

注意：

1、转换原点

2、坐标轴也跟着旋转

4、2D-倾斜

改变元素在页面中的倾斜角度，即形状

语法：

属性：transform

取值：

skew(xdeg)

向x轴产生倾斜效果

实际是改变y轴的倾斜角度

取值为正，y轴逆时针倾斜

取值父负，y轴顺时针倾斜

skew(xdeg,ydeg)

xdeg ：向x轴产生倾斜效果

ydeg ：向y轴产生倾斜效果

实际是噶变x轴的倾斜角度

取值为正，x轴顺时针倾斜

取值为负，x轴逆时针倾斜

skewX(xdeg) :

skewY(ydeg) :

3、3D转换

1、perspective 属性

1、作用

模拟人眼到投射平面的距离

2、注意

该属性要添加在3D转换元素的父元素上，其子元素允许实现3D的转换效果

3、语法

perspective : 数值;

Chrome & Safari : -webkit-perspective

2、3D旋转

1、作用

允许使得元素在x轴，y轴，z轴上进行旋转的操作

2、语法

属性：transform

取值：

1、rotateX(xdeg)

以 x轴 作为中心轴去旋转元素

2、rotateY(ydeg)

以 y轴 作为中心轴去旋转元素

3、rotateX(zdeg)

以 z轴 作为中心轴去旋转元素

4、rotate3d(x,y,z,ndeg)

x,y,z 如果取值大于0，则说明该轴参与旋转

ndeg ， 旋转角度是多少

rotate3d(1,0,0,45deg)

rotate3d(1,1,1,45deg)

3、3D位移

1、作用

改变元素在z轴上的位置

2、语法

属性：transform

取值：

1、translatez(z)

2、translate3d(x,y,z)

属性：transform-style

作用：指定子元素的排列方式

取值：

1、flat

默认值，不保留其子元素的3d位置

2、preserve-3d

保留其子元素的3d位置,即允许子元素呈现出3d的位置排列

2、过渡

1、什么是过渡

使得CSS属性值，在一段时间内平缓变化的效果

2、过渡的基本要素 & 属性

1、指定过渡的属性

1、作用

指定哪个属性值在变化的时候使用过渡的效果

属性：transition-property

取值：

1、none ：不使用过渡，默认

2、all ：所有属性全部使用过渡

3、property ：指定属性名称

允许使用过的效果的属性：

1、颜色属性

2、取值为数值的属性

3、转换属性

4、渐变

5、visibility属性

6、阴影属性

ex:

1、transition-property:background;

2、trnasition-property:transform;

2、指定过渡时间

1、作用

指定一个过渡效果在多长时间内完成

注意：必须指定过渡时间，否则没有效果

2、语法

属性：transition-duration

取值：以 s | ms 作为单位的数值

3、ex

transition-duration:300ms;

3、指定过渡的速度时间曲线函数

1、作用

指定变化速率

2、属性

transition-timing-function

取值：

1、ease

默认值，慢速开始，快速变快，慢速结束

2、linear

匀速

3、ease-in

慢速开始，加速结束

4、ease-out

快速开始，减速结束

5、ease-in-out

慢速开始和结束，中间先快后慢

3、ex

transition-timing-function:linear;

4、过渡延迟

1、作用

用户激发过渡操作后，等待多长时间后在显示效果

2、语法

属性：transition-delay

取值：以 s | ms 为单位的数值

5、过渡的简写方式

transition:property duration timing-function delay;

ex:

transition:background 1s linear,border-radius 2s linear;

3、动画

1、什么是动画

使元素从一种样式逐渐变化为另一种样式的效果

动画是复杂的过渡效果

2、实现动画的步骤

1、声明动画

是一个独立的规则

包含动画的名称以及关键帧

关键帧：

元素在 某个时间点上的状态

2、为元素调用动画

让元素调用指定的动画并且指定时间等参数

3、声明动画

语法

@keyframes 动画名称{

0% | from{

动画开始时，元素的样式

}

...

50%{

动画执行到一半时，元素的样式

}

...

100% | to{

动画结束时，元素的样式

}

}

@-webkit-keyframes 动画名称{}

@-moz-keyframes 动画名称{}

@-o-keyframes 动画名称{}

4、动画属性

注意：浏览器兼容性，使用之前加浏览器前缀

1、指定调用动画名称

属性：animation-name

2、指定动画的执行时长

属性：animation-duration

取值：s | ms

3、指定动画的速度时间曲线函数

属性：animation-timing-function

取值：ease,linear,ease-in,ease-out,ease-in-out

4、指定动画的延迟

属性：animation-delay

取值：s | ms

5、指定动画播放次数

属性：animation-iteration-count

取值：

1、具体数值

2、infinite ：无限次播放

6、指定动画的播放方向

属性：animation-direction

取值：

1、normal

正常，从0% 到 100%

2、reverse

逆向播放，从100% 到 0%

3、alternate

轮流播放

奇数次数从 0 - 100

偶数次数从 100 - 0

7、动画的简写属性

属性：animation

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

8、指定动画的填充模式

1、作用

指定动画播放前后，关键帧的状态

2、属性

animation-fill-mode

取值：

1、none

默认值，无填充

2、forwards

动画完成后，保持在最后一个帧状态上

3、backwards

动画播放前(延迟时间内)，元素会保持在第一帧的状态上

4、both

动画播放前后分别显示在第一帧和最后一帧状态上

9、指定动画播放状态

指定动画是播放还是暂停

属性：animation-play-state

取值：

1、paused

暂停

2、running

播放

JS

1、JS 概述

1、什么是JS

JS：Javascript，是一种运行于JS解释器/引擎中的解释型脚本语言

JS解释器/引擎：

JS的运行环境，有以下两种

1、独立安装的JS解释器(NodeJS)

2、嵌入在浏览器内核中的JS解释器

解释型：

编译型：运行程序前，会先将其编译，并且能够检查语法错误

解释型：直接将代码交给解释器运行，无需编译

PC，平板，手机，机顶盒全部支持JS

2、JS的发展史

1、1992年时，开发了一款脚本语言 - ScriptEase,可以嵌入在网页中的脚本语言

2、1995年，Netscape(网景)为Navigator2.0 浏览器设计了一款脚本语言LiveScript,更名为 JavaScript

3、1996年，Microsoft在IE3.0中发布了一款JavaScript的克隆版本，称为 JScript

4、1997年，JavaScript1.1 提交给了 ECMA(欧洲计算机制造商协会)，各个厂家合力推出 ECMA-262的标准,并且将核心定义为ECMAScript

3、JS的组成

完整的JS是由三部分组成：

1、JS核心(ECMAScript)

2、文档对象模型(DOM,Document Object Model)

W3C定义的规范,允许Javascript能够与网页进行交互

3、浏览器对象模型(Browser Object Model)

允许JavaScript能够与浏览器进行交互

4、JS的特点

1、JS编写简单

2、无需编译

3、弱类型语言

由数据 决定 数据类型

4、面向对象

5、JS用途

1、面向客户端的JS

1、客户端的数据计算

2、表单输入的验证

3、事件的触发和处理

4、网页特效制作

5、服务器的异步数据提交(AJAX)

2、面向服务器端的JS

1、分布式运算

2、实时服务器

3、窗口应用

4、网络应用

2、JS 基础语法

1、使用JS

1、浏览器内核

内核主要负责页面的渲染

内核主要由两部分组成

1、内容排版引擎

解析HTML/CSS

2、脚本解释引擎

解释JS

浏览器 内核 排版引擎 解释引擎

IE Trident - Chakra

Firefox Gecko - Monkey系列

Safari Webkit Webcore Nitro

Chrome Webkit Webcore V8

Opera Presto - Carakan

Opera Webkit Webcore V8

(2013~)

2、搭建JS运行环境

1、独立安装的JS解释器

要求：安装NodeJS

开始-> 搜索输入 cmd

操作：向控制台中输出一句话

console.log("你好世界!");

console.log('Hello World');

2、嵌入在浏览器内核中的JS解释器

1、直接在浏览器Console中输入脚本并执行

2、将JS脚本嵌入在HTML页面中执行

1、通过 html 元素事件

onclick 事件 ：在鼠标单击时要执行的操作

<button onclick="JS脚本"></button>

<div onclick="JS脚本"></div>

...

<ANY 事件名称="JS脚本"></ANY>

2、在<script>标记中 编写JS脚本

<script>

document.write("<h1>Hello</h1>");

</script>

3、使用外部脚本文件，编写JS脚本

1、创建脚本文件

\*\*\*.js

编写JS脚本

2、在使用的页面中引入JS文件

<script src="文件URL"></script>

3、JS调试

代码出现问题，结束本块运行，但不会影响后续块

2、语法规范

1、语句-会被JS解释器所执行的代码

由表达式、关键字、运算符组成

严格区分大小写(重要)

每条语句都由 ; 表示结束

中英文符号：

1、, 和 ，

2、. 和 。

3、; 和 ；

4、: 和 ：

5、" 和 “

6、() 和 （）

2、注释

单行注释：只能注释一行内容

语法：//

多行数值：一次性注释多行内容

语法：/\* \*/

3、变量 和 常量

1、变量

1、什么是变量

内存：保存程序运行过程中所需要用到的"数据"

空间：

位 ：bit 8bit = 1byte

字节：byte 1024byte = 1kb

千字节：kb 1024kb = 1mb

兆字节：mb 1024mb = 1gb

G : gb 1024gb = 1tb

T : tb

变量：表示的就是内存中的一段存储空间

生活中：

1、宾馆的房间

2、超市储藏箱

变量值：保存在变量中的数据

变量名：标识数据在内存中的名字

2、变量的声明

到内存中 声明一段空间，并且起个别名再保存数据

语法：

1、声明变量

var 变量名;

2、为变量赋值

变量名=值;

简写：

var 变量名=值;

3、一条语句声明多个变量

var 名1,名2,...,名n;

ex:

var stuName="张三丰",age=18,gender,score;

3、变量名的规范

1、不允许使用JS的关键字或保留关键字

2、允许包含字母、数字、下划线(\_)以及$

var stuName;正确

var $stuName;正确

var stu-name;错误，不能出现 -

3、不能以数字开头

var 1name;错误

var name1;正确

4、不能重复

5、尽量见名知意

var a;

var b,c,d,e,f,g;

var studentName;

6、推荐使用 "匈牙利命名法","驼峰命名法","下划线命名法"

驼峰命名法：

1、变量名由一个单词组成，采用全小写

2、变量名由多个单词组成，第一个单词采用全小写，其余的每个单词首字符变大写

var studentName;

4、变量的使用

1、未经初始化的变量

1、变量声明之后，没有被赋值就直接使用

var stuName;

console.log(stuName);

结果为：undefined

2、变量未被声明过，而直接使用

console.log(stuAge);

2、声明变量不使用var

在JS中，允许声明变量时不使用var关键字

如果省略var关键字，声明的就是"全局变量"

但，并不推荐此写法

stuGender="男";

console.log(stuGender);

3、可以对变量进行存取操作

1、设置变量的值-存值操作(SET操作)

将数据保存在变量中

特点：只要变量出现在赋值符号(=)的左边，一律是存执操作

var stuName="张三丰";

stuName="张无忌";

var newName=stuName;

2、获取变量的值-取值操作(GET操作)

将数据从变量中取出来使用

特点：只要变量"没"出现在赋值符号的左边，一律都是取值操作

var stuAge=18;//存值操作

console.log(stuAge);//在控制台上打印输出stuAge的值

document.write(stuAge);//在页面上打印输出stuAge的值

var newAge = stuAge;

针对newAge 是存值操作

针对stuAge 是取值操作

stuAge = stuAge + 10;

存值 取值

套餐名:香辣鸡腿堡套餐

汉堡：香辣鸡腿堡

配餐：烤翅

饮料：可乐

2、常量

1、变量与常量

变量：是一个可以变化的数据

常量：是一个不能变化的数据

常量的特点：

在程序中，一旦被声明好，在程序运行过程中就不允许被修改

2、语法

const 常量名=值;

注意：

1、声明常量时必须赋值

2、通常，常量名称采用全大写形式(规范)

3、练习

声明一个变量 用来 保存一个圆的 半径值

计算 该圆的周长，并打印输出 2\*π\*半径

计算 该圆的面积，并打印输出 π\*半径\*半径

4、数据类型

1、作用

决定了数据在内存中所占的空间大小

2、数据类型分类

1、原始类型(基本类型)

1、number类型

数字类型

可以表示32位的整数，也可以表示64位的浮点数

整数：32位空间(4字节)

浮点数：小数，64位空间(8字节)

整数：

1、保存十进制数字

由0-9共10个数字组成，逢10进1

var age = 35;

2、保存八进制数字

由0-7共8个数字组成，逢8进1

JS中，八进制数字值，由0开始

var num=010;

3、保存十六进制数字

由0-9，A-F 共 16个数字组成，逢16进1

JS中，十六进制数字，由0x开始

var num=0x10;

浮点数：

小数

支持小数点计数法：35.5,76.8,31.00004

支持指数记录法：4.3e2

2、string类型

字符串类型

表示一系列的文本字符

由Unicode字符，数字，标点符号组成的序列

Unicode : 编码方式之一，可以由一组十六进制的数字(Unicode码)来表示一个字符。

每个字符在计算机内存中，占2个字节

如何查看字符的Unicode码？？？

字符.charCodeAt(); 十进制表示方式

字符.charCodeAt().toString(2); 二进制

字符.charCodeAt().toString(16);十六进制

张：24352(十进制)

张：0101111100100000(二进制)

张：5f20(十六进制)

如何将 Unicode码 转换为字符？？？

使用 \u 将 十六进制的Unicode码转换为字符

var str = "\u5f20";

中文汉字的范围：

从 \u4e00 开始 到 \u9fa5 结束

string在声明的时候需要使用"" 或 '' 引起来

var num1 = 456;//number

var num2 = "456";//string

特殊字符需要转义:

ex：

想在字符串中 打印输出 ""

\n : 换行

\t : 一个制表符

\" : "

\' : '

\\ : \

3、boolean类型

布尔类型

在程序中只表示真 或 假 的状态值(条件中用的多)

该类型 仅有两个值

true ：真

false ：假

在实际运算中true当做1运算，false当做0运算

4、undefined

语义：访问的数据不存在

2、引用类型

1、数据类型转换

2、运算符 和 表达式

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

1、数据类型转换

弱类型语言：由数据决定数据类型

var stuName = "张三丰";//string

stuName = 35; //number

stuName = true;//boolean

函数：var 变量=typeof(变量);

1、隐式转换

不同数据类型之间的数据在计算过程中会自动进行转换

1、数字 + 字符串 ：将数字转换为字符串，整体结果为字符串

2、数字 + 布尔类型：将布尔类型转换为数字，整体结果为 数字

3、字符串 + 布尔类型：将 布尔类型转换为 字符串，整体结果为 字符串

4、布尔类型+布尔类型：将 布尔类型转换为 数字，整体结果为 数字

注意：字符串和任意数据类型的数据相加运算时，结果都为字符串

2、转换函数

1、toString()

作用：将任意类型的数据转换为字符串类型

返回值：转换后的结果

语法：var 变量=变量.toString();

2、parseInt(参数)

作用：将任意类型的数据转换为 "整数"

参数：待转换的数据

返回值：转换后的结果(number类型)

特点：

1、碰到第一个非数字字符就停止转换

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

isNaN(参数) : 判断 参数 是否为 非数字

返回值为 false，则为 数字

返回值为 true ，则为 非数字

3、parseFloat(参数)

作用：将 参数 转换为 小数(浮点数)

参数：待转换的数据

返回值：转换后的结果(number类型)

4、Number(参数)

作用：将一个 string 类型的参数 转换为 number

参数：待转换的数据

返回值：转换后的结果(number)

注意：如果包含非法字符，则返回 NaN

2、运算符 和 表达式

运算符：实现数据运算的一组符号

操作数：辅助运算符进行运算的数据

15 + 10

表达式：由运算符 和 操作数 组成的式子

var num = 15;

注意：每一个表达式都会有运算结果

1、算术运算符

1、作用

完成数据的 +-\*/ 等基础运算

2、运算符

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

由算术运算符组成的表达式称之为算术表达式

+ :

两个数字的相加运算

字符串的连接运算

% : 取余操作，俗称 模

获取两个数字的 "余数"

5 % 2 结果:1

10 % 3 结果：1

3 % 10 结果：3

10.5 % 3 结果：1.5

-10 % 3 结果：-1

10 % -3 结果：1

-10 % -3 结果：-1

使用场合：

1、判断奇偶性 或 倍数的问题

2、获取 某数字的 最后几位

获取1234的最后一位

1234 % 10 结果 4

获取1234的最后两位

1234 % 100 结果 34

注意：

除 + 外，其他的运算符如果两边的操作数不是number类型的话，会自动使用Number()进行转换然后再做运算

任何数据在与NaN做运算时，结果一定是NaN

3、优先级

除 ++，--外

运算顺序：从左向右，先 \*/%，后+-

var result = 35+2\*8+6%3\*4;

想提升优先级 ，使用 ()

var result = (35+2)\*(8+6)%3\*4; 结果 8

var result = (35+2)\*((8+6)%3)\*4; 结果 296

4、++，--

++ ：自增运算符,在自身的数值基础上做+1操作

-- ：自减运算符,在自身的数值基础上做-1操作

var num = 10;

num++;

做后缀，先使用num的值，然后再进行自增操作

ex:

1、var num = 10;

var newNum = num++;

1、先将num的值赋值给newNum

2、再将num的值进行自增

2、var num = 10;

console.log(num++);

1、先 打印输出 num的值

2、在将num的值进行自增

++num;

做前缀，先将num的值进行自增，然后再使用num的值

ex:

1、var num = 10;

var newNum = ++num;

//newNum : 11

//num : 11

1、先将num进行自增操作，变为11

2、在将num的值(11)赋值给 newNum

2、var num=10;

num++;

console.log(num);

ex

1、var num = 10；

console.log(num--);??10

console.log(--num);??8

var newNum = num--;

console.log(newNum);??8

console.log(num);??7

2、var num = 2;

2 (3)3 3(4) (5)5 5 5(6)

var result = num + ++num + num++ + ++num + num + num++;

result : 23

num : 6

2、关系运算符

1、作用

比较数据之间的大小，相等关系的

由关系运算符所组成的表达式称之为"关系表达式"

关系表达式的结果都是 boolean 类型

2、关系运算符

>,<,>=,<=,==(判断等于),!=(不等于),===(全等)，!==(不全等)

ex:

console.log(3 > 5); // false

console.log(5 >= 5);// true

console.log(3 == 5);// false

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

console.log(5 != 3);// true

console.log("张三丰">"张无忌");

console.log("10">3);

console.log("10">"3");

注意：

1、运算符两边如果有一个是number类型的话，另一个非number类型的数字会通过Number()隐式转换为 number 在进行比较

2、如果运算符两边都是 string 类型的话，那么比较的就是每位字符的Unicode码，不相同的字符比较出大小，Unicode码的胜出

1、"张三丰">"张无忌"

张三丰

张：24352

三：19977

张无忌

张：24352

无：26080

2、"10">"3"

"10"

"3"

=== : 全等，既要判断数值，还要判断数据类型，数值和类型都相等的情况下，才为真，否则就为假

输入一个数字：25

是奇数吗?true

输入一个数字：50

是奇数吗?false

输入一个价格：6000

降价了吗?true

输入一个价格：9000

降价了吗?false

3、逻辑运算符

1、作用

关联多个条件

ex: 年龄必须在 18岁 到 65岁之间

var age = prompt("输入年龄:");

判断1：age > 18

判断2：age < 65

2、逻辑运算符

!:非(取反)

对现有条件的结果进行取反操作

非真即假

非假即真

&&:逻辑与

作用：关联两个条件

如果两个条件的值都为真的话，则整个逻辑结果就为真

如果两个条件中，有一个条件值为假，则整个逻辑结果就为假

var result = 条件1 && 条件2;

条件1：true，条件2：true，result：true

条件1：false,条件2：true，result：false

条件1：true，条件2：false,result：false

条件1：false,条件2：false,result：false

|| : 逻辑或

关联的两个条件中，只要有一个结果为真，那么整个的结果就为真

var result = 条件1 || 条件2;

条件1：true，条件2：true，result：true

条件1：false,条件2：true，result：true

条件1：true，条件2：false,result：true

条件1：false,条件2：false,result：false

优先级：

!>&&>||

比较运算符 > 逻辑运算符

5>3 && 1<2

练习2：

请输入一个字符：a (input)

是英文字符吗?true

是否在 a ~ z 或 A~Z之间

var r = input >= 'a' && input <= 'z'

如果r为真，说明是 小写字符

是数字吗?false

是汉字吗?false

3、短路逻辑

1、短路 与

条件1 && 条件2

如果条件1的结果为假的话，那么就"不会判断执行条件2"，整个表达式的值，就为假

如果条件1的结果为真的话，那么则"继续判断执行条件2"，并且以条件2的结果作为整个逻辑表达式的结果

2、短路 或

条件1 || 条件2

如果条件1的结果为真的话，那么久"不会判断执行条件2"，整个表达式的值，就为真

如果条件1的结果为假的话，那么则"继续判断执行条件2"，并且以条件2的结果作为整个逻辑表达式的结果

1、运算符

1、位运算符

作用：

针对数字操作

将数字先转换为二进制再进行运算操作

二进制：

由0,1组成,逢二进一

十进制 二进制

0 0

1 1

2 10

3 11

4 100

5 101

6 110

... ...

运算符：

1、&(按位与)

语法：num1 & num2

作用：将两边的操作数转换为二进制，每位数字进行按位比较,对应位置的数字都为1的时候，该位结果才为1，否则为0

ex:

3 & 5 : 1

3:011

5:101

=========

001

12 & 7 : 4

12:1100

7:0111

==========

0100

1、15 & 1 : 1

15:1111

1:0001

======

0001

2、18 & 1 : 0

3、37 & 1 : 1

4、26 & 1 : 0

5、12345 & 1 : 1

6、78966 & 1 : 0

场合：判断一个数字的奇偶性，num & 1,如果结果为1，num则为奇数，否则为 偶数

2、|(按位或)

语法：num1 | num2

作用：将两边的操作数转换为二进制，每位数字进行按位比较,对应位置的数字只要有一个为1的时候，该位的结果就是1

ex:

1 | 3

1:01

3:11

=====

11 = 3

5 | 7

5:101

7:111

=====

111 = 7

5:101

3:011

=====

111 = 7

场合：将小数做向下取整操作

3、^(按位异或)

语法：num1 ^ num2

作用：将两边的操作数转换为二进制，按位比较，对应位置只有一个1的时候才返回1，否则返回0

ex :

5 ^ 3 = 6

5 : 101

3 : 011

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

110

场合：在不借助第三方变量时交换两个数字的位置

ex:

var num1 = 5;

var num2 = 3;

num1 = num1 ^ num2; //num1 = 6

5 : 101

3 : 011

=======

110 = 6

num2 = num2 ^ num1;//num2 = 5

3 : 011

6 : 110

========

101 = 5

num1 = num1 ^ num2;//num1 = 3

6 : 110

5 : 101

=======

011 = 3

4、<<(按位左移)

语法：num<<n

作用：将num转换成二进制，并且向左移动n位，右侧以0补齐

ex: 2<<1 = 4

2

0000 0000 0000 0000 0000 0000 0000 0010

2<<1 = 4

0000 0000 0000 0000 0000 0000 0000 0100

2<<2 = 8

0000 0000 0000 0000 0000 0000 0000 1000

5、>>(按位右移)

语法：num>>n

作用：将num转换成二进制，并且向右移动n位，左侧以0补齐

ex:8>>2

8

0000 0000 0000 0000 0000 0000 0000 1000

8>>2 = 2

000000 0000 0000 0000 0000 0000 0000 10

7>>1

7

0000 0000 0000 0000 0000 0000 0000 0111

7>>1 = 3

00000 0000 0000 0000 0000 0000 0000 011

2、赋值运算

赋值运算符：=

变量 = 值;

注意：=的左侧必须是变量

扩展的赋值表达式：

+=,-=,\*=,%=,/=,^=,... ...

a+=b; 等价于 a=a+b;

ex:

var num = 10;

num += 5;//num=num+5;

console.log(num);

num = 10;

num \*= 3;//num = num \* 3

console.log(num); // 30

var a = 5;

var b = 3;

a = a ^ b; // a^=b;

b = b ^ a; // b^=a;

a = a ^ b; // a^=b;

3、条件运算符(三目运算符)

目：操作数

单目运算符：只有一个操作数，++,--,!

双目运算符：有两个操作数,+,-,\*,...

三目运算符：要求有三个操作数

条件运算符：

? :

表达式1 ? 表达式2 : 表达式3;

表达式1 ：是一个条件，结果为boolean类型

如果表达式1的值为true，则执行表达式2

如果表达式1的值为false，则执行表达式3

输入一个考试成绩，如果成绩大于60分，输出及格，否则输出 不及格

2、函数

1、没有函数的弊端

1、代码直接写在script中，网页加载时就运行

2、代码不够通用

2、什么是函数

函数(function),也被称为方法(method)或者过程(procedure)

函数是一段预定义好，并可以被反复执行的代码块

预定义好：

事先被定义，但不会马上被执行。可以由任意元素的事件来调用(调用时被执行)

反复执行：

同一个函数，可以被多次调用

提升了代码的可重用性

代码块：

允许包含多条可执行的语句

函数的本质也是功能完整的对象

3、语法

1、普通函数的 声明 和 调用

1、声明

function 函数名(){

若干条语句;

}

2、调用

任何 JS 的位置处 ，可以通过

函数名();

进行调用

2、带参数的函数的声明 和 调用

parseInt("35.5");

parseInt(46.8);

参数：由函数外部传递到函数内部进行辅助操作的数据

声明语法：

function 函数名(参数列表){

函数体/方法体

}

参数列表：

可以为空，为空的话称之为无参函数

也可以由一个或多个参数名称组成，多参数的话，中间用 ，隔开

参数名称：与变量名称一致，不需要使用var来声明

定义函数时定义的参数称之为 "形参(形式参数)"

调用带参数函数

任意位置处，通过

函数名(参数列表);

调用函数

参数列表:要传递的值的列表，多个值之间用,隔开

调用函数时传递的参数称之为 "实参(实际参数)"

3、带返回值的函数的声明 和 调用

返回值：执行完函数最后得到的一个数据

var r1 = parseInt(35.5);

var r2 = parseFloat("37.8");

... ...

语法：

声明：

function 函数名(参数列表){

函数体 ...

return 值;

}

调用：

var 变量 = 函数名(参数列表);

注意：

1、return 后面只能返回一个值

2、return 后面的语句不会被执行

4、作用域

1、什么是作用域

作用域指的是变量或函数的可访问范围

在JS中，变量和函数的作用域分为：

1、函数作用域，只能在定义的函数内访问

2、全局作用域，一经定义，任何位置都能用

2、函数作用域中的变量

又称为"局部变量"

function fun1(){

var num = 15; // 局部变量,只能在fun1中使用

console.log(num);

}

3、全局作用域中的变量

又称为"全局变量"，一经声明，JS的任何位置处都能使用。全局变量 会声明在 独立于 任何 function 外围位置处

<script>

var num = 15; // 全局变量

function fun1(){

console.log(num); //15

}

console.log(num);//15

</script>

注意：

声明变量时，不加var关键字也是全局变量

4、声明提前

JS在正式运行前，会将var声明的变量以及function声明的函数，预读到所在作用域的顶部。

但是对变量的赋值仍然保留在原位置处

5、按值传递

基本数据类型作为参数时，传参时，实际是将实参的值复制一个副本传递给函数

6、函数的作用域

1、全局函数

在script中定义的函数，相互之间独立，没有所属关系，一经定义，任何位置都能使用

2、局部函数

在 function 中声明的，有所属关系，只能在声明的函数内使用

function fun1(){

function son(){

console.log("fun1中的函数son");

}

}

3、ECMAScript 提供的全局函数

1、parseInt()

2、parseFloat()

3、Number()

4、isNaN()

5、eval()

作用：计算并执行以字符串表示的js代码

6、encodeURI()

URI : Uniform Resource Identifier

按照 URI 的规则 解析一个字符串(编码)

https://www.baidu.com/s?wd=减肥

7、decodeURI()

对一个URI格式的字符串进行解码，并返回解码后的字符串

8、encodeURIComponent()

9、decodeURIComponent()

3、分支结构

1、程序的流程

程序 = 数据 + 算法

任何复杂的算法都可以通过 "顺序","分支","循环" 三种结构来表示

2、什么是分支结构

在不同的情况下，可以选择不同的语句来执行

当 条件 满足时，执行某些语句 - if结构

当 条件 满足时，执行某些语句

当条件 不满足时，执行另外一些语句

-if ... else 结构

3、if结构

1、语法

if(条件){

//语句块(满足条件时要做的操作)

}

条件：尽量放 boolean 的变量 或 表达式

判断条件

如果条件 为真的话，则执行语句块中的代码

如果条件 为假，则跳过语句块，执行其他语句

2、注意

1、条件如果不是boolean的话，则会自动转换，以下操作会自动将条件转换为 false，其他都是true

if(0){}

if(0.0){}

if(undefined){}

if(NaN){}

if(""){}

if(null){}

ex:

if(35.0){} //true

if("Hello World"){}//true

if(0){} //false

2、if 后的 {} 可以省略不写，如果省略不写的话，那么if只控制它下面的第一条语句

4、if ... else 结构

1、作用

当满足条件时，执行某些操作

当不满足条件时，执行另外一些操作

2、语法

if(条件){

语句块1;

}else{

语句块2;

}

判断条件的结果

如果条件为真，则执行语句块1

否则(条件为假),则执行语句块2

5、else ... if 结构

1、判断

输入 一个 考试成绩

成绩 >= 100 分 ，奖励一台 兰博基尼

或者 成绩 >= 90分，奖励一台 法拉利

或者 成绩 >= 80分，奖励一台 奥迪

或者 成绩 >= 70分，奖励一台 奥拓

或者 成绩 >= 60分，奖励一台 自行车

否则 奖励俩 大嘴巴子!

2、语法

if(条件1){

语句块1;

}else if(条件2){

语句块2;

}...else if(条件n){

语句块n;

}else{

语句块n+1;

}

流程：

从上向下判断条件，满足哪个条件则执行对应的语句块，然后跳出整个结构

所有条件都不满足时，执行else结构

注意：

最后的 else 可以省略不写，如果省略的话，以上条件都不满足，则什么都不执行

1、分支结构-2

2、循环结构-1

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

1、分支结构-2

1、"等值" 判断的分支结构 - switch

1、语法

switch(变量/表达式){

case 值1:

语句块1;

break; -- 结束switch的运行,可选的

case 值2:

语句块2;

break; -- 结束switch的运行，可选的

case ...:

语句块....;

break;

default:

语句块n;

break;

}

流程：

1、计算 switch后的 变量 或 表达式的值

2、判断值与哪个case块后面的值能匹配上，匹配上哪个case块则执行哪个语句块

3、碰到 break ， 则结束switch结构的运行，如果没有break，则继续执行后续的case块，直到碰到break，或执行结束为止

4、如果任何case都没有匹配上的话，并且具备default模块的话，则执行default里面的内容。否则什么都不执行

注意：

1、各个case块与default之间的位置可以随便放

2、JS 默认是采用 === 的方式做等值判断的

1 和 "1" 是不相等的

2、特殊用法 - 直落

直落：两个 或 多个case块要执行相同的内容，可以采用直落的形式

switch(变量/表达式){

case 值1:

case 值2:

case 值3:

语句块;

break;

}

1、计算 变量 或 表达式的值

2、如果计算出来的值是 值1或值2或值3，都执行相同的语句块。

month = Number(prompt());

switch(month-1){

case 11:

totalDays += 30;

case 10:

totalDays += 31;

... ...

}

2、循环结构-1

1、问题

1、控制台中输出一句Hello World

2、控制台中输出10句Hello World

3、控制台中输出1000句Hello World

4、将 1000 句 Hello World 改成 "你好世界!"

5、在 1000 句你好世界基础上 增加

第 1 句 你好世界!

...

第1000句 你好世界!

2、什么是循环

循环：一遍又一遍的重复的执行相同或者相似的代码

生活中的循环：

1、上学

2、运动员跑圈

3、活着

4、地球公转和自转

循环的特点：

1、循环条件

指定循环的次数，或者继续执行循环的条件

2、循环体：也成为循环操作，重复做的事情是什么

3、while 循环

语义：当 条件 满足的时候，去做某些事情

1、语法

while(条件){

循环体(循环操作)

}

执行流程：

1、判断条件,如果条件为真，则执行循环体中的内容，如果条件为假，则跳出循环结构

2、执行完循环体，再一次判断条件，

2、练习

1、打印输出 1 - 100 之间所有的数字

2、打印输出 1 - 100之间所有的偶数

3、计算1 - 100 之间所有的奇数和

3、循环中的流程控制

1、continue

终止本次循环的执行，继续开始下次循环

ex:

打印 1 - 100之间所有非3的倍数的数字

2、break

结束整个循环的运行

可以用在：不确定循环次数的时候去结束循环

ex:

要求用户 从弹框中 录入数据，录入完直接打印在控制台上，直到输入 exit 位置

4、do...while 循环

1、特点

先执行循环操作，再判断循环条件

2、语法

do{

循环操作

}while(条件);

3、while 与 do...while 之间的区别

1、while

先判断条件，再执行循环体

最少执行 0 次循环

2、do...while

先执行循环体，再判断循环条件

最少执行 1 次循环

环结构-2

1、for 循环

1、while

打印 1 - 10 之间所有的数字

var i = 1; //1、声明循环条件变量

while(i <= 10){ //2、循环条件判断

console.log(i);//3、循环操作

i++;//4、更新循环变量

}

2、for 循环

目的：为了简化 while 循环的开发

for(表达式1;表达式2;表达式3){

循环操作

}

表达式1：声明循环条件变量

表达式2：循环条件判断

表达式3：更新循环变量

流程：

1、计算表达式1的值 即声明循环条件(只执行1次)

2、计算表达式2的值 即计算循环条件时否满足，如果条件为真，则继续向下执行，否则退出循环

3、条件为真时，执行循环体

4、计算表达式3的值 即更新循环变量

5、计算表达式2的值 即继续判断循环条件，回到第2步

3、for 循环表达式的特殊用法

1、表达式1 允许为空

可以省略表达式1，但;不能省

for(;条件;自增){

}

ex:

var i=1;

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

console.log(i);

}

2、表达式2 允许为空

可以省略表达式2，但;不能省

如果省略表达式2，那么就是 "死循环"

3、表达式3 允许为空

死循环

4、三个表达式都可以省略，但;不能省

for(;;){

if(条件)

break;

i++;

}

5、表达式1 和 表达式3 的多样化

表达式1：允许声明多个变量，用,隔开

表达式3：允许更新多个变量，用,隔开

4、循环嵌套

for,while,do...while都可以实现嵌套操作

ex:

for(var i=1;i<=9;i++){ //外层循环

for(var j=1;j<=9;j++){ //内层循环

}

}

1. \*\*\*数组:

什么是: 内存中连续存储多个数据的存储空间，再起一个名字

vs 变量: 只能存一个数据，数组可以存多个数据

为什么: 便于查找和操作一组相关的多个数据

程序=数据结构+算法

好的数据结构，可极大提高程序的执行效率

何时: 只要存储一组连续的多个数据都要用数组

如何:

创建: 3种情况:

1. 创建空数组:

var arr=[];

var arr=new Array();

何时: 只要创建数组时，暂时不知道数组内容

2. 创建数组同时，初始化数组内容:

var arr=[值1, 值2, ...];

var arr=new Array(值1, 值2, ...)

何时: 只要创建数组时，已经知道数组内容

3. 创建n个空元素的数组:

var arr=new Array(n);

何时: 只要创建数组时，只知道元素个数，不知道内容。

访问数组元素:

元素: 数组中每个数据就是一个元素

下标: index 数组中唯一标识一个元素的序号

默认从0开始，依次递增1

获取: arr[i]

设置: arr[i]=值

总结: 数组中每个元素得用法和普通变量完全一样

数组: 一组变量的集合，再起一个统一的变量名

3个不限制:

1. 不限制元素的数据类型:

2. 不限制元素个数: 可随时向数组中添加新元素

3. 不限制下标越界: 无论取值还是赋值时，下标越界都不报错！

赋值: 自动添加一个新元素保存新数据

取值: 不报错，仅返回undefined

稀疏数组: 下标不连续的数组

length属性: 记录数组中理论上的元素个数

length属性永远是最大数字下标+1

最大数字下标永远比length小1

固定套路:

1. 获取数组最后一个元素: arr[arr.length-1]

获取数组倒数第n个元素:arr[arr.length-n]

2. 向数组末尾追加一个新元素:

arr[arr.length]=值;

3. 删除数组末尾一个元素: arr.length-=1;

删除数组末尾n个元素:arr.length-=n;

存储: 数组是引用类型的对象

按值传递: 两变量间赋值或将变量作为参数传入函数时，都是将原变量中的值复制一个副本给对方

原始类型的值: 修改新变量，不影响原变量！

引用类型: 通过新变量修改原数组，原变量同样受影响。

1. \*\*\*数组:

遍历, 垃圾回收, \*\*\*\*关联数组

数组API

1. 遍历:

什么是: 对数组中每个元素执行相同操作

何时: 只要对数组中每个元素执行相同操作时

如何: for(var i=0;i<arr.length;i++){

//操作当前元素arr[i]

}

2. 垃圾回收:

什么是: js引擎会自动释放不再被任何变量引用的存储空间

垃圾回收器: 专门释放不再被使用的存储空间的小程序

引擎自带的，自动伴随主程序执行

原理:

每个存储空间都有一个属性count记录被引用的次数

每多一个变量引用，count就+1

每当一个变量释放引用(赋值为null)时，count就-1

当count减为0时，垃圾回收器就释放该存储空间

习惯: 当一块存储空间不再使用时，要主动将变量赋值为null

3. 关联数组:

什么是: 可自定义下标名称的数组

vs索引数组: 下标都是数字的数组

为什么: 索引数组中的下标都是无意义的数字,不便于查找

何时: 只要希望在数组中按照意义快速定位一个元素时

如何:

创建: 2步:

1. 创建空数组: var ym=[];

2. 向空数组中添加新元素，使用自定义的下标名称

ym["name"]="杨幂";

ym["math"]=81;

ym["chs"]=53;

ym["eng"]=89;

访问: ym["下标名"]——用法同普通变量

特点: length属性失效，永远为0

遍历: for( var key in ym){

//in依次取出ym数组中的下标名保存到key中

//key仅获得下标名

//ym[key]才能获得下标名对应的元素值

//强调: key不加""，因为key是变量，不断变化

}

原理:

索引数组的问题: 查找只能靠遍历

结果: 查找速度受存储位置和元素总数的影响极大

关联数组解决:

hash算法: 根据一个字符串，计算出一个尽量不重复的序号

添加元素: 将自定义的下标名交给hash算法计算出不重复的序号，将元素保存到序号位置

取值时: 将自定义的下标名较为hash算法，计算出和存入时相同的序号，直接去序号位置获取元素值

优: 无需遍历, 查找速度和存储位置以及元素总数无关

4. 数组API:

API: 别人写好的，我们用现成的函数

学一项新东西: 2个方面: 1. 存储； 2. API

1. 转字符串: 2种:

1. var str=String(arr), 将arr中每个元素都转为字符串，用逗号分隔——给数组内容拍照

2. var str=arr.join("自定义连接符"), 将arr中每个元素都转为字符串，用自定义的连接符连接

固定套路:

1. 无缝拼接: 错误: arr.join() =>等效于String()

正确: arr.join("")

2. 判断空数组: arr.join("")===""

3. 动态生成页面元素:

"<ANY>"+arr.join("</ANY><ANY>")+"</ANY>"

2. 拼接和选取: 都无权修改原数组，只能返回新数组，必须用变量接住新的返回值

拼接: 将其他数组或单个元素拼接到当前数组末尾

var newArr=arr.concat(值1, 值2, arr2, ...)

强调: concat可打散数组参数，逐个元素拼接

选取: 复制原数组中指定开始位置到结束位置之间的元素组成新数组返回。

var subArr=arr.slice(starti,endi+1);

复制starti位置开始到endi位置的元素，组成新数组返回

强调: 凡是两个参数都是下标的函数，都含头不含尾

简写: 1. 省略第二个参数表示从starti一直到结尾

2. 支持负数参数,表示倒数第n位

选择最后n个元素

arr.slice(-n) => 相当于 arr.slice(arr.length-n)

3. 如果省略两个参数，表示复制整个数组

3. 修改数组: splice 删除，插入，替换数组中的元素

强调: 直接修改原数组

删除:

arr.splice(starti,n) 删除starti位置开始的n个元素

强调: 不用考虑含头不含尾，因为只有第一个参数是下标。n不用+1。

简写: 1. 省略n，会一直删到结尾

2. 支持负数参数, 等效于length-n

其实: splice有返回值:

var deletes=arr.splice(starti,n);

deletes是被删除的元素组成的临时数组

插入: arr.splice(starti,0,值1,值2,....)

在starti位置插入值1,值2,... 原位置的值向后挤压

强调: splice不支持打散数组参数，如果传入子数组，会将子数组整体保存在一个下标内。

替换: arr.splice(starti,n,值1,值2,...)

先删除starti开始的n个元素，再在starti位置插入新值1,值2,...

强调: 删除的元素个数和插入的新元素个数不必一致

4. 翻转: arr.reverse();

强调: 不负责排序，只是原样颠倒

5. 排序:

什么是: 将数组的元素按照从小到大或从大到小的顺序重新排列。

何时: 今后所有数据都要先排好序，再展示给客户

如何: 2种:

1. 自定义排序算法: 冒泡，快速，插入

1. \*\*\*数组:

\*\*\*\*排序

栈和队列

二维数组

2. \*\*\*String

1. \*\*\*\*排序:

arr.sort() 默认将arr中每个元素转为字符串，再按升序排列

问题: 只能对字符串类型，升序排列

解决: 自定义一个比较器函数

什么是比较器函数: 专门比较任意两数大小的函数

何时: 只要sort方法的默认规则不是想要的，就要用自定义的比较器函数代替.

如何定义: 要求: 1. 2个参数: a,b

2. 返回值: 如果a>b，就返回正数

如果a<b，就返回负数

否则, 就返回0

最简单的数字升序比较器:

function cmp(a,b){ return a-b; };

如何使用: 将比较器函数作为参数传入sort方法即可

arr.sort(cmp)

强调: cmp后不要加()，sort会在调用cmp函数时，自动向cmp中传入要比较的两个元素。

问题: 如何改升序为降序:

解决: 颠倒比较器结果的正负号,

最简单的数字降序比较器:

function cmp(a,b){ return b-a; };

2. 栈和队列:

js中没有专门的栈和队列类型，都是用数组+API模拟的

栈(stack): 一端封闭，只能从另一端进出的数组

何时: 希望始终操作最后进入数组的最新元素时

特点: FILO

如何:

1. 开头封闭，从结尾出入栈:

结尾入栈: arr.push(值) => arr[arr.length]=值

结尾出栈: var last=arr.pop()

2. 结尾封闭，从开头出入栈:

开头入栈: arr.unshift(值)

开头出栈: var first=arr.shift()

开头入栈和结尾入栈的存储顺序是相反的

队列(queue): 只能从一端进入，从另一端出的数组

何时: 只要希望按照先来后到的顺序使用数组元素时

特点: FIFO

如何:

结尾入队列: arr.push(值)

开头出队列: var first=arr.shift()

3. 二维数组:

什么是: 数组中的元素又是另一个子数组

何时: 2种:

1. 展现横行竖列的二维数据

2. 对一个大的数组中的元素，进一步细分分类时

如何:

创建: 2种:

1. 先创建空数组，再添加子数组:

var data=[];

data[0]=[0,0,0,0];

data[1]=[0,0,0,0];

2. 创建数组时初始化子数组:

var data=[

[0,0,0,0],

[0,0,0,0],

...

];

访问: data[r][c]

遍历: 外层循环控制行，内层循环控制列

for(var r=0;r<data.length;r++){

for(var c=0;c<data[r].length;c++){

data[r][c] //当前元素

}

}

\*\*\*String

什么是: 由多个字符组成的一个只读字符数组

vs 数组:

相同:

1. 下标, 访问每个字符

2. length, 记录字符的个数

3. concat和slice，拼接或选取数组内容

不同: 字符串无法使用除concat和slice之外的数组API

内置对象 包装类型

String API

2. \*\*\*\*\*正则表达式

1. 内置对象:

什么是: ES标准中规定的，浏览器厂商已经实现的现成的API

包括: 11个:

String Number Boolean ——包装类型

Array Date Math RegExp

Error

Function Object

Global(在浏览器中被window代替)

2. 包装类型:

什么是: 专门封装一个原始类型的值，并提供操作值的API的类型。

为什么: 原始类型的值本身，什么功能都没有。

何时: 当试图用原始类型的值调用API时，引擎自动创建包装类型的对象封装原始类型的值，并调用包装类型的API

其实: var n=345.678;

n.toFixed(2) => new Number(n).toFixed(2)

=> 345.68

var str="张";

str.charCodeAt()=>new String(str).charCodeAt()

=>24352

3. String API:

所有StringAPI都无权修改原字符串，只返回新字符串

大小写转换: 将str中所有字母统一转为大写/小写

str.toUpperCase(); str.toLowerCase();

何时: 只要不区分大小写时，都要转为统一的大小写，再判断和比较

获取指定位置的字符:

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

获取指定位置的字符的unicode号

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

将unicode号反向转为正文:

String.fromCharCode(unicode);

获取子字符串:

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

强调: substring不支持负数参数！

解决: length-n 表示倒数第n个

str.substr(starti,n) 获取str中starti开始的n个字符

相当于: str.substring(starti,starti+n)

强调: 不考虑含头不含尾

查找关键词: 4种:

1. 查找一个固定的关键词出现的位置:

var i=str.indexOf("关键词",fromi);

在str中从fromi位置开始，找下一个"关键词"出现的位置

如果没找到，返回-1

强调: 每次只能找下一个关键词的位置

如果找所有，必须用循环反复调用

查找最有一个关键词的位置:

var i=str.lastIndexOf("关键词")

在str中查找最有一个"关键词"的位置

问题: 只能查找一个固定的关键词

解决: 正则表达式模糊匹配

2. 使用正则判断是否包含符合规则的关键词:

var i=str.search(/正则/i)

在str中查找符合正则要求的关键词，返回其位置

如果找不到，返回-1

问题1: 所有正则，默认都区分大小写

解决: 加后缀i 表示忽略大小写

问题: 1. 只能获得位置，无法获得内容

2. 只能找第一个。

解决: match方法

3. 查找所有关键词的内容

var kwords=str.match(/正则/ig)

查找str中所有符合正则要求的关键词，保存在结果数组kwords中

\*\*\*如果找不到，返回null!

强调: 只要API的返回可能是null，就必须先验证不是null，再使用！

问题: 所有正则，默认仅匹配第一个关键词

解决: 加后缀g 表示查找全部

问题: 只能返回所有关键词的内容，无法返回每个关键词的位置。

4. 即查找每个关键词的内容，又查找每个关键词的位置

?

替换: 将字符串中找到的敏感词替换为新内容

简单: 将所有敏感词替换为一个统一的新值

str=str.replace(/正则/ig,"新值");

高级: 根据不同的敏感词，动态选择要替换的新值

str=str.replace(/正则/ig,function(kword){

return 根据kword动态选择不同的替换值返回

})

其中: replace会在每个找到的关键词上执行function

每次执行，都会将本次找到的敏感词自动赋值给kword参数变量。

\*\*\*\*\*正则表达式:

什么是: 规定一个字符串中字符出现规律的表达式

何时: 2种:

1. 按照规则模糊匹配多种关键词

2. 按照规则验证字符串格式时

如何:

1. 关键词的原文就是最简单的正则表达式

2. 字符集:

什么是: 规定一位字符可选字符列表的集合

何时: 如果一位字符，有多种备选字时

如何: [备选字符列表]

强调: 一个字符集只能匹配1位

简写: 如果备选字符列表中字符是连续的，可用-省略中间的字符

比如: 1位数字: [0-9]

1位小写字母: [a-z]

1位大写字母: [A-Z]

1位字母:[A-Za-z]

1 位汉字: [\u4e00-\u9fa5]

特殊: 除了XXX [^字符列表]

3. 预定义字符集:

什么是: 对常用字符集的简化写法

包括: 一位数字: \d [0-9]

一位字母,数字或下划线: \w [0-9A-Za-z\_]

一位空字符: \s 空格，制表符等空字符

通配符: . 任意字符

强调: 只有在规则完全匹配时才能使用预定义字符集

问题: 无法匹配字符出现的个数

解决: 量词

4. 量词:

什么是: 规定一位字符集出现次数的规则

何时: 只要规定一位字符集出现的次数

如何:

强调: 总是跟在字符集之后

1. 有明确范围的:

6~8位 {6,8}

6位以上 {6,}

必须6位 {6}

2. 没有明确范围:

? 可有可无，最多一个

\* 可有可无，多了不限

+ 最少一个，多了不限

5. 选择和分组:

1. 选择: 在多个条件中选其一匹配

规则1|规则2 其中|读作或

2. 分组: 将多个规则编为一组

为什么: 默认一个量词只能修饰一个字符集

何时: 只要希望一个量词可修饰多个字符集时

都要先将多个字符集分为一组

如何: (多个字符集)

身份证号: 15位数字 2位数字 1位数字或X或x

后三位，可有可无，最多一次

\d{15} (\d\d [0-9Xx])?

手机号:

+86或0086 可有可无，最多一次

空字符 可有可无，多了不限

1

3,4,5,7,8

9位数字

(\+86|0086)?\s\*1[34578]\d{9}

微信 微 信 weixin wei xin wx w x

(微|w(ei)?)\s\*(信|x(in)?)

6. 指定匹配位置:

^ 匹配字符串开头

$ 匹配字符串结尾

比如: 仅匹配开头的空字符: ^\s+

仅匹配结尾的空字符: \s+$

同时匹配开头或结尾的空字符: ^\s+|\s+$

\b 匹配单词边界: 开头 结尾 标点 空字符

1. \*\*\*String

替换: 删除，格式化

切割:

2. \*\*\*RegExp

验证 查找

3. Math

4. Date

\*\*\*String

1. 替换: 衍生

删除关键词: 将关键词替换为空

格式化: 将原字符串分组后，再按新的格式拼接

2步: 1. 用正则将原字符串分组

每个分组会自动获得一个分组序号

从1开始

2. 在替换值中使用$n代替每个分组的子内容

2. 切割: 将字符串按指定的关键词分隔为多段子字符串

简单: 分隔符是固定的

var subStrs=str.split("分隔符");

返回切割后的子字符串组成的数组

强调: 切割后的数组中不包含分隔符

高级: 分隔符不固定时

var subStrs=str.split(/正则/)

固定套路:

1. 将字符串打散为字符数组: str.split("")

2. 将页面元素的内容转为数组:

var arr=

str.replace(/^\s\*<ANY>|<\/ANY>\s\*$/)

.split(/<\/ANY>\s\*<ANY>/)

\*\*\*RegExp:

什么是: 专门封装一条正则表达式，并提供使用正则表达式执行验证和查询的API

何时: 2种:

1. 验证字符串格式

2. 即查找关键词内容，又找位置时

如何:

创建: 2种:

1. 如果正则表达式是固定的: 用正则直接量

var reg=/正则/ig;

强调: 如果正则中包含/，要转为\/

2. 如果正则表达式需要动态生成: 用new

var reg=new RegExp("正则","ig");

API:

1. 验证: var bool=reg.test(str)

验证str是否符合reg的规则要求

问题: 正则表达式默认只要部分匹配就验证通过

解决: 今后凡是验证必须前加^后加$

2. 查找关键词: 即找关键词内容，又找位置:

var arr=reg.exec(str)

在str中查找下一个符合reg要求的关键词的位置

返回值: arr: [

0: 本次找到的关键词

index: 关键词的位置

]

如果找不到，返回null

强调: 1. 每次只能找一个

2. 反复调用,可自动找下一个

reg.lastIndex属性记录下次开始的位置

默认从0开始

每找到一个关键词,就将lastIndex修改为:

arr["index"]+arr[0].length

Math

什么是: 专门封装数学计算所用的常量和函数的对象

何时: 只要进行数学计算

如何: Math不用创建(不能new)，就可直接使用

API:

1. 取整:

上取整: 只要小数部分超过，就取下一个整数

Math.ceil(num)

下取整: 舍弃小数部分

Math.floor(num)

vs parseInt 相同: 都是舍弃小数部分

不同: parseInt可去掉数字后非数字字符

Math.floor只能转换纯数字

四舍五入取整: Math.round(num)

vs n.toFixed(d)

1. Math.round()只能取整，不能指定小数位数

n.toFixed(d) 可按任意小数位数四舍五入

2. 返回值: Math.round()返回number，可直接计算

n.toFixed(d)返回字符串

2. 乘方和开平方:

乘方: Math.pow(底数,幂)

比如: 10的2次方: Math.pow(10,2) =>100

开平方: Math.sqrt(num)

3. 最大值和最小值:

Math.max(值1, 值2, ...)

Math.min(值1, 值2, ...)

问题: 不支持获得数组中的最大/小值

解决: Math.max.apply(null,arr)

4. 随机数:

Math.random() 0<=r<1

min~max之间取随机:

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

简写: 0~max

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

Date:

什么是: 专门封装一个时间，并提供操作事件的API

何时: 今后只要保存和操作时间

如何:

创建: 4种:

1. 创建日期对象，同时获得当前系统时间:

var now=new Date();

强调: 获取的是客户端本地时间

2. 创建日期对象，封装自定义时间

var date=new Date("yyyy/MM/dd hh:mm:ss")

new Date(yyyy,MM-1,dd,hh,mm,ss)

3. 使用毫秒数创建:

var date=new Date(ms); //ms转换当地时区时间

何时: 今后服务器上存储时间都是用毫秒数

因为不用考虑时区问题

客户端就需要用new Date将毫秒数转为当地时区对应的时间显示。

4. 复制一个日期:

var date1=new Date(xxx);

var date2=new Date(date1);

何时: 因为日期的计算都是直接修改原日期对象

所以旧时间无法保留

如果希望同时保留计算前后的两个时间时

就必须先将原日期复制一个副本，再用副本计算

原理: 其实日期对象中保存的是一个毫秒数

从1970年1月1日0点至今

API:

单位: FullYear Month Date Day

Hours Minutes Seconds Milliseconds

1. 每个单位都有一对儿getXXX/setXXX方法

其中: getXXX() 负责获取指定单位的值

setXXX(n) 负责设置指定单位的新值

setXXX(n)可自动根据n调整进制

比如: getDate() setDate(n)

特例: Day没有setDay()方法，星期不允许修改

2. 取值范围: 月中的日(date) 从1~31结束，

其余都是从0开始到进制-1结束：

Month: 0~11 比现实少1

Date: 1~31

Day: 0~6

hours: 0~23

minutes/seconds: 0~59

计算: 2种:

1. 两日期对象可直接相减，结果是毫秒差

何时: 计算倒计时

2. 对一个日期的任意单位做加减:

3步: 1.取值, 2.计算, 3.放回去

其实可简化为: date.setXXX(date.getXXX()+n)

日期的格式化:

date.toString() 将日期转为当地时间的完整格式

date.toLocaleString() 将日期转为当地时间的简化格式

date.toLocaleDateString() 仅保留日期部分

date.toLocaleTimeString() 仅保留时间部分

date.toGMTString(); 将当地时间转为0时区标准时间

1. Error

2. \*\*\*Function

创建 重载 匿名

\*\*\*\*作用域/作用域链 \*\*\*\*\*闭包

1. Error

什么是错误(bug): 程序执行过程中导致程序无法继续执行的情况

一旦出现错误，程序都会强行退出。

什么是错误处理: 即使程序出错也保证不会强行退出的机制

如何:

try{

可能出错的代码

}catch(err){//只在发生错误时才执行

错误处理代码: 1. 提示用户, 2. 记录日志

}

其中: err 错误对象: 在发生错误时自动创建的保存错误信息的对象。

错误的类型: 6种:

SyntaxError : 语法错误

ReferenceError: 引用错误, 找不到要使用的变量

RangeError: 范围错误, 参数超范围

TypeError: 类型错误, 错误的调用了类型的函数

EvalError, URIError

效率: 频繁使用try catch可能降低程序的效率

解决: 如果可提前预知错误原因，可用if else代替try catch

主动抛出错误:

throw new Error("自定义错误提示")

2. \*\*\*Function

什么是: 封装一项任务的步骤清单的代码段，再起一个名字

何时: 只要一项任务可能被反复使用，都要先封装为一个函数，再反复调用函数。

如何:

创建: 3种:

1. 声明: function 函数名(参数列表){

函数体;

return 返回值;

}

参数: 只要函数必须某些数据才能正常执行时

返回值: 如果调用者需要获得函数的执行结果时

强调: 声明提前hoist

什么是: 在程序开始执行前，都会先将var声明的变量和function声明的函数提前到当前作用域的顶部集中创建。但是, 赋值留在原地！

2. 直接量: var函数名=function(参数列表){...}

揭示了函数的本质:

函数名其实就是一个普通的变量

函数是封装代码段的引用类型的对象

函数名通过地址引用函数

强调: 直接量方式不会被声明提前

何时: 只要不希望函数的定义被声明提前时

3. 用new:

var 函数名=

new Function("参数1","参数2",...,"函数体");

强调: 所有参数都必须写在""中

以下创建函数正确的是：

function cmp(a,b){return a-b}

var cmp=function(a,b){return a-b};

var cmp=new Function(a,b,"return a-b"); X

var cmp=new Function("a","b","return a-b")

重载: overload

什么是: 相同函数名，不同参数列表的多个函数，在调用时可根据传入参数的不同，自动选择对应的函数执行

为什么: 减少API的个数

何时: 只要同一件事, 根据不同的参数，执行不同的逻辑

如何:

问题: js语法默认不支持重载

解决: arguments

什么是: 函数内自动接收传入函数的所有参数值的类数组对象。

何时: 不确定参数个数时,就可用arguments接收所有参数值

如何:

1. 每个函数内都自动创建了arguments对象

2. 类数组对象: 长的像数组的对象:

相同: 1. 下标，访问每个元素

2. length属性，记录元素个数

不同: 类型不同:

数组: Array 可使用Array中的所有API

类数组对象: Object 无法使用Array API

arguments vs 参数

参数的作用: 1. 提示调用者如何正确使用函数

2. 在函数内用最简单的方式表示一个有意义的值

总结: 绝大多数函数，必须使用参数接收数据

只有参数个数不确定时，才被迫使用arguments

匿名函数:

什么是: 定义函数时不指定函数名的函数

为什么: 只用一次，节约内存

何时: 只要一个函数，只调用一次，就立刻释放时

如何:

1. 回调: 将一个函数传入另一个函数，被另一个函数执行

比如: arr.sort(function(a,b){return a-b})

2. 自调: 定义函数后，立刻调用自己，调用后立刻释放

为什么: 划分临时函数作用域，避免产生全局变量

何时: 只要不希望产生全局变量时

如何: +function(){...}()

(function(){...})()

(function(){...}())

总结: 只要能在匿名函数中指定的代码，都首选匿名函数

除非一个函数确实要被反复调用，采用有名函数

\*\*\*\*作用域和作用域链:

什么是作用域(scope): 变量的可用范围

包括: 2种:

1. 全局作用域: window

全局变量: 优: 随处可用，可反复使用

缺: 极容易被全局污染——尽量不要使用全局变量

2. 函数作用域: 活动对象AO

局部变量: 优: 仅函数内可用,不会被污染

缺: 无法反复使用

函数生命周期:

0. 程序开始执行时:

创建执行环境栈: 执行过程中，保存正在调用的函数的执行环境

首先压入全局执行环境元素

其次创建全局作用域window对象

ECS中的全局执行环境元素引用window对象

1. 函数定义:

创建函数对象，保存函数的代码段

函数名其实是一个引用函数对象的变量

2. 函数调用开始时:

在执行环境栈中压入当前函数的执行环境元素

为本次函数调用创建活动对象AO

什么是AO: 专门保存本次函数调用时使用的局部变量的对象.

执行环境元素引用着活动对象

3. 函数调用过程中:

所有变量的使用顺序: 先局部，再全局

4. 函数调用后:

本次函数调用的执行环境元素出栈

导致函数作用域对象一同释放

导致其中的局部变量也释放

什么是作用域链: 由多级作用域对象组成的链式结构

保存着所有变量, 控制着变量的使用顺序

\*\*\*\*\*闭包:

什么是: 即重用变量，又保护变量不被污染的机制

为什么: 全局变量: 优: 重用; 缺: 污染

局部变量: 优: 不会被污染; 缺: 不可重用

何时: 希望一个变量即反复使用，又不被污染

如何: 3步:

1. 创建外层函数包裹受保护的变量和使用变量的内层函数

2. 外层函数要返回内层函数对象

3. 调用外层函数，获得内层函数的对象

优: 即重用变量，又避免污染

缺: 闭包比一般函数占用更多内存空间

解决: 释放闭包: 将引用内层函数的变量赋值为null

\*\*\*\*\*OOP

创建

直接量, new Object, 构造函数

原型和继承

多态

\*\*\*\*\*OOP

什么是: 程序中都是用对象来描述现实中的事物

为什么: 便于维护

何时: 几乎所有程序，都要使用面向对象的思想编写

如何: 三大特点: 封装，继承，多态

1. 封装: 将现实中一个事物的属性和功能集中定义在一个对象中。

现实中事物的属性会成为程序中对象的属性

功能 方法

属性: 对象中，描述对象的一个属性值的变量

方法: 对象中，定义对象的一个功能的函数

对象: 集中存储一个事物的属性和方法的存储空间，再起一个名字。

属性和方法统称为对象的成员

比如: iPhone7

内存: 32

尺寸: 4.7

摄像头:1200

指纹识别功能: function(){实现指纹识别}

WIFI通信功能: function(){实现WIFI通信}

如何: 3种:

1. 对象直接量:

var obj={//创建一个对象的意思

属性名: 属性值 ,

... : ...

功能名(){//ES6

...

}

}

2. 用new: 2步:

1. 创建空对象: var obj=new Object(); //{}

2. 为空对象添加新属性: obj.属性名=值

何时: 只要创建对象时还不知道对象的成员

\*\*\*js中所有对象底层都是关联数组

所以可随时向对象中添加新属性

3. 构造函数contructor:

什么是: 描述一类对象相同结构的函数

为什么: 用直接量反复创建对象，会产生大量重复代码，不便于维护。

何时: 只要反复创建多个相同结构的对象时

作用: 1. 描述结构, 2. 创建对象，并存储属性值

如何: 2步:

1. 定义构造函数

function 类型名(属性参数列表){

this.属性名=属性参数;

... = ... ;

this.方法名=function(){

... this. 属性名 ...

}

}

2. 调用构造函数:

var obj=new 类型名(属性值列表);

new: 4件事:

1. 创建一个新的空对象

2. 让新对象继承构造函数的原型对象

3. 用空对象调用构造函数: 调用构造函数，并将this自动指向新对象

4. 返回新对象的地址

访问对象的成员:

对象.属性名 -> 用法和普通的变量完全一样

对象.功能名() -> 用法和普通的函数完全一样

问题: 对象的方法中不能写死属性值

解决: 对象的方法如何访问自己的属性

自己的 => this.

\*\*\*\*\*this: 函数作用域中自动定义的

引用调用函数时.前的对象

的关键词

总结: 今后，对象自己的方法，要使用自己的属性，必须加this.

2. 继承:

封装的问题: 将方法定义在构造函数中，会为每个对象都创建相同方法的副本。浪费内存。

解决: 继承

什么是: 父对象的成员，子对象无需创建即可直接使用

为什么: 代码重用！节约内存 !

何时: 只要同一类型的多个子对象，有相同的成员时

如何: js中的继承都是通过原型对象实现的

什么是原型对象: 集中存储同一类型子对象共有成员的父对象。

何时: 只要同一类型的多个子对象，有相同的成员时，都要将相同的成员集中保存在原型对象中一份即可。所有子对象自动共享。

如何:

创建: 买一赠一: 定义构造函数时，已经自动赠送了一个原型对象:

向原型对象中添加共有成员:

类型名.prototype=成员

内置对象的原型对象:

其实每种内置类型都是一个构造函数:

比如: Array => function Array(){...}

String => function String(){...}

每种内置类型的构造函数都有对应的原型对象

比如: Array.prototype

String.prototype

内置类型的原型对象中保存着该类型所有子对象共用的API

比如: Array.prototype { sort() push() slice() ....}

String.prototype { indexOf() search() ....}

解决浏览器兼容性问题: 旧浏览器无法使用新版的API

原型链: 由多级父元素连续继承形成的链式结构

保存着所有的属性和方法，

控制着成员的使用顺序: 延原型链自底向上依次查找使用

vs 作用域链: 保存着所有的变量，控制着变量的使用顺序

3. 多态:

什么是: 同一个事物在不同情况下表现出不同的状态

js中多态只支持重写

重写override:

什么是: 如果子对象觉得父对象的成员不好用, 可在子对象定义和父对象成员同名的自有成员。

结果: 再调用该成员时，优先使用自有成员代替父对象中的共有成员

为什么: 体现父子对象间的差异！

何时: 只要子对象觉得父对象的成员不好用。

总结: 面向对象三大特点: 封装, 继承, 多态

\*\*\*\*\*自定义继承:

1. 只修改一个对象的父对象:

何时: 只要一个对象，想使用另一个对象中的成员，就需要继承另一个对象为父对象

如何: child.\_\_proto\_\_=father

问题: \_\_proto\_\_是内部属性, 一般不允许随意访问

解决: Object.setPrototypeOf(child,father)

设置child的父对象为father

2. 修改多个对象的继承关系

如何: 修改构造函数的原型对象

时机: 在创建第一个子对象之前，就要更换

类型名.prototype=father;

3. 两种类型间的继承:

何时: 只要发现两种类型间存在相同的属性结构或功能时

如何: 抽象出一个公共父类型:

1. 将共同的属性结构定义在父类型的构造函数中

2. 将共有的方法定义在父类型的原型对象中

3. 设置子类型的原型对象继承父类型的原型对象

4. 在子类型构造函数中借用父类型构造函数

1. 自定义继承:

\*\*\*call和apply

2. \*\*\*ES5

对对象的保护

保护属性:

数据属性:

\*\*\*\*\*访问器属性:

防篡改

Object.create

数组API

\*\*\*\*\*bind

1. 自定义继承:

子类型构造函数借用父类型构造:

问题: 直接调用父类型构造, 父类型中的属性会被添加到全局！

原因: this->如果一个函数，不用任何对象就直接调用，this默认都指window

解决: 只要调用函数时，this不是想要的，就可用call强行替换函数中的this为想要的另一个对象。

如何: fun(参数) fun中的this->window

fun.call(obj,参数) fun中的this就被替换成obj

在执行的一瞬间，相当于obj.fun(参数)

问题: 参数重复:

解决: apply: 用途和call完全一样！

唯一差别: apply要求参数必须以数组或集合方式集中传入.

原因: apply可打散传入的参数数组，再单个数据传递给函数。

\*\*\*ES5

对对象的保护:

问题: 普通的对象，可对属性赋任意值，可随时添加删除属性。

解决: 为对象添加自我保护

1. 保护属性

属性:

命名属性: 可随时用.访问到的属性

数据属性: 实际存储属性值的属性

4种保护: 通过属性内部的四大特性实现的

包括: value: 实际存储属性值

writable: 是否可修改

enumerable: 是否可遍历

强调: 仅禁止for in遍历，依然允许用.直接访问

configurable: 是否可删除

是否可修改其它特性

如何:

获取一个属性的四大特性:

var attrs=

Object.getOwnPropertyDescriptor(obj,"属性名")

修改属性的特性:

一次仅修改一个属性的特性:

Object.defineProperty(obj,"属性名",{

特性: 值, 特性: 值, ...

})

强调: 如果要修改的属性不存在，则自动添加

问题: js中存在静默失败: 不允许操作，但也不报错

解决: 严格模式: 比传统js更严格的执行模式

将静默失败升级为错误！

问题: 特性的值可能被反向修改

解决: configurable:false 不可逆！

总结:

今后只要修改特性，都要设置configurable:false

同时修改多个属性的特性:

Object.defineProperties(obj,{

属性名:{四大特性},

属性名:{四大特性},

})

问题: 使用自定义规则保护属性

解决: 访问器属性

访问器属性: 不实际存储属性值，而是提供对其它属性/数据的保护。

何时: 只要使用自定义规则保护属性

如何: 四大特性:

get:function(){return 属性值;}

//只要试图获取访问器属性的值时，自动调用get

set:function(val){ 验证val -> 将val保存到实际属性中}

//只要试图给访问器属性赋值时，自动调用set，同时将要赋的新值传递给val参数.

enumerable, configurable

总结: 访问器属性的用法和普通数据属性完全一样。只不过会自动调用访问器属性中的get和set方法

问题: 访问器属性的值应该保存在哪儿?

错误: 保存在另一个数据属性中

问题: 可绕过访问器属性，直接修改数据属性的值

正确: 使用匿名函数自调，封装一个受保护的局部变量,形成闭包。在get/set函数中，给闭包中的局部变量赋值/取值。

内部属性: 无法用.访问的属性, 比如: \_\_proto\_\_

2. 防篡改

什么是: 防止对对象的结构再调整

三个级别:

1. 防扩展: 禁止添加新属性:

Object.preventExtensions(obj)

2. 密封: 在防扩展基础上进一步禁止删除旧属性

在防扩展基础上，将每个属性的configurable自动都改为false

Object.seal(obj)

3. 冻结: 禁止对对象的一切修改(属性和属性值)

在密封基础上，自动修改所有属性的writable都为false

Object.freeze(obj)

Object.create(): 创建一个子对象，继承现有父对象，并为子对象扩展新属性.

何时: 如果只有父对象，没有构造函数时，也向创建子对象。

如何: var child=Object.create(father);

创建一个子对象child，继承father

如何同时为子对象扩展新属性:

var child=Object.create(father,{//defineProperties

新属性:{四大特性},

... : ...

});

数组API:

1. 判断: 判断数组中的元素是否符合要求

1. 判断所有元素是否都符合要求:

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

//val，自动获得当前元素值

//i, 自动获得当前位置

//arr, 表示当前数组对象本身

return 判断条件;

});

判断arr中每个元素是否都符合function的要求

2. 判断是否包含符合要求的元素:

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

//同every中的内容

})

2. 遍历: 对每个元素执行相同的操作

1. 对原数组中每个元素执行相同的操作:

arr.forEach(function(val,i,arr){

直接对arr[i]执行操作

})

2. 复制原数组中每个元素，执行相同操作后，放入新数组中

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

return 新值;

})

3. 过滤和汇总:

1. 过滤: 筛选出原数组中符合条件的元素，组成新数组

原数组保持不变

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

return 判断条件;

});

2. 汇总: 将数组中所有值统计出一个计算结果

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

//prev指目前为止的临时汇总值

return prev+val;

},start);

DOM

1. 概述:

什么是DOM: Document Object Model

专门操作网页内容的API

为什么: 只有DOM API才能访问网页内容

何时: 只要程序要访问(增，删，改，查)网页的内容

如何: 查找元素->修改(内容,样式)

->添加/删除

W3C DOM: 几乎所有浏览器100%兼容

核心DOM: 操作一切结构化文档: HTML, XML

特点: 万能, 但繁琐

HTML DOM: 对核心DOM部分常用功能，专门针对HTML特点进行了简化

特点: 简洁, 但不是万能

总结: 不必区分。优先选择简洁的API。如果实现不了，再选择核心DOM API补充。

2. \*\*\*DOM 树

什么是: 网页中一切内容在内存中都以树形结构存储

为什么树形结构: 树形结构最适合存储上下级包含关系

包括: 浏览器读到HTML内容时:

1. 创建根节点对象: document

2. 依次将所有网页内容，创建为document的子节点，形成树型结构

总结: 网页中一切内容都是树上的一个节点对象Node

节点包括四大类: document, 元素, 属性, 文本

DOM操作: 查找->修改

->添加/删除

3. \*\*\*查找: 4种:

无论任何查找，获得都是节点对象: Node

Node: 三大属性:

nodeType: 获得节点类型——一个数字

何时: 判断节点类型

包括: document 9

element 1

attribute 2

text 3

nodeName: 获得节点名称

何时: 进一步区分标签名

包括: document #document

element 元素的标签名(全大写)

attribute 属性名

text #text

nodeValue: 获得节点的值——了解

包括: document null

element null

attribute 属性值

text 文字内容

1. 节点间关系查找:

何时: 只要已经获得一个节点，仅找周围节点时

节点树: 包括一切网页内容的树结构

包含2大类:

父子: elem.parentNode

elem.childNodes ——集合(类数组对象)

强调: 仅返回直接子节点

第i个子元素: elem.childNodes[i]

elem.firstChild

elem.lastChild

兄弟: elem.previousSibling 前一个兄弟

elem.nextSibling 后一个兄弟

优: 完整！

问题: 节点树中，看不见的空字符，也是文本节点，可能对查找产生干扰

解决: 元素树

元素树: 仅包含元素节点的树结构

包含2大类:

父子: elem.parentElement 父元素

elem.children 直接子元素

elem.firstElementChild 第一个子元素

elem.lastElementChild 最后一个子元素

兄弟: elem.previousElementSibling 前一个元素

elem.nextElementSibling 后一个元素

特点: 优: 仅包含元素节点，查找时不受空字符干扰

缺: 不包含除元素外的一切节点，不完整

元素树不是一棵新的树，只是节点树的子集。

总结: 如果仅获得元素时，首选元素树API

如果希望获得所有节点时，就用节点树API

2. 递归遍历所有节点: 利用节点间关系API

什么是: 获得指定父元素下的所有后代节点

如何: 2步:

1. 定义函数仅遍历指定父元素的直接子节点

2. 对每个子节点执行和父节点完全相同的操作

算法: 深度优先: 当同时有子节点和兄弟节点时，优先遍历子节点。子节点遍历完了，才返回遍历兄弟。

问题1: 遍历childNodes和children

childNodes和children返回的是动态集合:

什么是: 不实际存储数据，每次访问集合，都重新查找DOM树

解决: for(var i=0,len=childNodes.length;i<len;i++)

问题2: 递归调用的效率极低

解决: 绝大多数递归都可以转换为循环

利用遍历API: NodeIterator

什么是: 每次只遍历一个节点对象的API

为什么: 用循环代替递归, 提高效率

何时: 只要遍历父节点下所有子代节点时

如何: 2步:

1. 用父节点创建一个NodeIterator

var iterator=document.createNodeIterator(

parent, NodeFilter.SHOW\_ALL, null, false

.SHOW\_ELEMENT

)

2. 循环反复调用iterator的nextNode()方法

var currNode=iterator.nextNode();

如果返回null，说明到头了

强调: iterator内置了深度优先算法

总结: 开发中很少手写遍历算法。

但几乎所有查找API底层都使用的是遍历方式

所以，查找方法应该尽量少用。

3. 按HTML查找: 4种:

1. 按id查找:

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

返回值: 一个元素, 找不到，返回null

强调: 只能用document调用

2. 按标签名查找:

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

返回值: 元素的集合——动态集合

找不到，返回空集合

强调: 1. 可在任意父元素上调用

2. 不但查找直接子元素，且查找所有后代元素

3. 按name属性查找:

var elems=document.getElementsByName("name")

返回值: 元素的集合——动态集合

找不到，返回空集合

强调: 只能在document上调用

4. 按class查找:

var elems=parent.getElementsByClassName("class")

返回值: 元素的集合——动态集合

找不到，返回空集合

强调: 1. 可在任意父元素上调用

2. 不但找直接子元素，而是找所有子代元素

3. 只要class中包含指定过的类名就选择，不要求完全一样

4. 用选择器查找: 2个:

1. 仅找一个元素:

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

2. 找多个元素:

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

返回值: 非动态集合: 实际存储所有数据，反复访问，也不会重复查找DOM树

强调: 任意父元素都可调用

总结: getXXX vs querySeletor

1. 返回值: getXXX返回动态集合

querySelector 返回非动态集合

2. 效率: 首次查找效率:

getXXX 不返回所有内容, 所以效率高

querySelector 返回所有内容，所以效率低

3. 易用: getXXX 繁琐

querySelector 简洁

结论: 如果只要一个条件就能查找出结果: getXXX

如果查找条件复杂: querySelector

.innerHTML: 获取或修改元素开始标签和结束标签之间的html代码片段

.textContent: 获取或修改元素开始标签和结束标签之间去掉HTML标记后的纯文本内容

textContent: 去掉内部的所有标签，只保留内容文字

将转义字符转为正文

兼容性: IE8不兼容, .innerText

表单元素的内容: .value

练习: option的selected属性表示选项是否被选中

遍历过程中，可能删除数组元素，则都要反向遍历

属性:

HTML标准属性: 2种办法:

1. 核心DOM: 4个:

elem.getAttribute("属性名")

elem.setAttribute("属性名","值")

elem.hasAttribute("属性名")

elem.removeAttribute("属性名")

其实,属性也是节点对象，都保存在元素的attributes中

2. HTML DOM: elem.属性名

特例: html中的class属性与js中的class关键词冲突，所以DOM中都用className代替class属性。

问题: 三大状态: disabled checked selected

不能用核心DOM操作，应该用HTML DOM

自定义扩展属性:

HTML: <ANY data-属性名="值"

为什么: id只能代表一个元素

class 专门定义样式，且可能频繁修改

如何即表示多个元素，又不容易修改

何时: 代替id，元素，class，作为选择器，选择多个元素

查找: 使用属性选择器，查找元素:

.querySelectorAll("[data-属性名='值']")

访问: 2种:

1. 核心DOM:

问题: HTML DOM中无法用.访问自定义属性

2. H5: elem.dataset.属性名

样式:

内联样式:

elem.style.css属性名 获取或设置css属性名

其中css属性名都要去横线变驼峰

强调: 修改时，只能修改内联样式

好处: 修改不会影响其他元素，且优先级最高

所以，今后只要修改样式，都用.style.css属性

获取时，只能获得内联样式

问题: 获得的样式不完整！

解决: 获得计算后的样式: 最终应用到元素上的完整样式的列表。且将所有属性值计算为最终结果。

var style=getComputedStyle(elem)

style.css属性名——只读

所以，今后只要获取样式，都用getComputedStyle，获得计算后的完整样式

问题: 一条js语句只能修改一个css属性

如果一次修改多个css属性，就需要多句话

解决: 其实，开发中，都是先将一套css属性定义在一个css class中。通过设置class，来批量修改元素的样式。

内部/外部样式表:

获取: getComputedStyle(elem)

修改: 禁止修改！

1. 添加和删除

2. \*\*\*HTML DOM常用对象:

Image Select/Option Table/.... From/Element

1. 添加和删除:

添加: 3步:

1. 创建空元素:

var a=document.createElement("a");

相当于: <a></a>

2. 设置关键属性

a.href="http://tmooc.cn";

a.innerHTML="go to tmooc";

相当于:

<a href="http://tmooc.cn"> go to tmooc </a>

仅在内存中创建了一个单独的元素，还未加到DOM树

3. 将新元素添加到DOM树

末尾追加: parent.appendChild(a)

中间插入: parent.insertBefore(a,child)

在child之前插入a

替换: parent.replaceChild(a,child)

用a替换现有child

\*\*\*优化: 尽量少的操作DOM树

原因: 页面加载过程:

html->DOM Tree

↓

render Tree-->\*\*\*layout\*\*\*-->paint

↑ 计算和布局

css->CSS Model

频繁操作DOM树，会导致频繁layout

如何: 2种:

1. 如果同时添加父元素和子元素，就要现在内存中将子元素添加到父元素，再一次性将父元素添加到DOM树

2. 如果同时添加多个平级子元素，就要用文档片段

文档片段: 内存中临时保存子元素的虚拟父元素

为什么: 减少操作DOM树的次数

何时: 同时添加多个平级子元素到DOM树

如何: 3步:

1. 创建文档片段

var frag=document.createDocumentFragment()

2. 将子元素添加到文档片段:

frag用法和普通父元素一样

frag.appendChild(child)

3. 将文档片段一次性添加到DOM树

parent.appendChild(frag);

将子元素添加到DOM树后，frag自动释放，不会成为实际的节点对象。

删除: parent.removeChild(a)

节约变量: a.parentNode.removeChild(a);

2. \*\*\*HTML DOM常用对象

Image: 代表页面上一个img元素

创建: var img=new Image();

Select: 代表页面上一个select元素

属性:

.selectedIndex: 获得当前选中项的下标

.options: 获得select下所有option元素

.options.length: 获得select下option的个数

.length: 等效于.options.length

.length=0 可清空select下所有option

.value: 获得当前select选择项的值或内容

如果选中项有value,则返回选中项的 value

如果选中项没有value，则返回选中项的内容

方法:

.add(option) 向select下追加一个option

.remove(i) 删除select下i位置的option

事件:

.onchange: 当选中项发生改变时自动触发

Option: 代表一个option元素

创建: var opt=new Option(text,value);

创建option同时设置text和vlaue

简化: 向select中添加一个新option

select.add(new Option(text,value));

属性: text, value, index

Table

获取:

.tHead

.tBodies

.tBody

.tFoot

创建: var thead=.createTHead()

tbody .createTBody()

tfoot .createTFoot()

删除: .deleteTHead() .deleteTFoot()

没有deleteTBody

行分组: tHead tBody tFoot

获取: 行分组.rows 获得当前行分组下的所有tr

添加: var tr=行分组.insertRow(i)

在行分组内i位置插入一个新行

简写: 省略参数i，就是末尾追加一个新行

删除: 行分组.deleteRow(i)

强调: i不能省略

行: tr

获取: tr.cells 获得当前行中所有格

添加: tr.insertCell(i) 在tr中i位置插入一个新格

简写: 省略参数i, 就是末尾追加一个新格

强调: insertCell只能创建td，不能创建th

删除: tr.deleteCell(i)

删除行: 2种:

1. 行分组.deleteRow(i) i指当前行在行分组内的下标

2. table.deleteRow(i) i指当前行在整个表中的下标

tr.rowIndex: 记录当前行相对于整个表的下标

总结: 今后删除行: table.deleteRow(tr.rowIndex)

Form: 代表网页上的一个表单元素

获取: var form=document.forms[i/id/name]

属性:

.elements 获得表单中所有表单元素

包括: input select textarea button

.elements.length 所有表单元素的个数

获得表单元素: .elements[i/id/name]

简写: 只有包含name属性的元素:

form.name

.length 等效于.elements.length

方法: .submit(); //手动提交表单

事件: .onsubmit: 最终提交表单前自动触发

表单中的元素:

方法: .focus() 让当前表单元素获得焦点

.blur() 让当前表单元素失去焦点

BOM

1. 什么是

2. 打开和关闭窗口

3. 窗口大小和窗口定位

4. \*\*\*定时器

1. 什么是: Browser Object Model

专门操作浏览器窗口的API

问题: 没有标准 —— 严重的兼容性问题

包括:

window: 2个角色

1. 代替global，作为全局作用域对象

2. 封装所有ES,DOM,BOM的API

history: 当前窗口打开后成功访问过的历史记录栈

location: 当前窗口正在打开的url地址

navigator: 保存着浏览器软件的配置信息

document: 封装DOM API的对象

screen: 保存显示设备的信息, 区分设备的种类

event: 封装事件信息的对象

2. 打开和关闭窗口:

打开窗口： window.open("url","name")

1. 打开超链接的方式: 4种:

1. 在当前窗口打开，可后退

html: <a href="url" target="\_self">

js: window.open("url","\_self")

强调: \_self不能省略

2. 在当前窗口打开，禁止后退

js: location.replace("新url")

用新url替换history中当前url

3. 在新窗口打开，可打开多个

html: <a href="url" target="\_blank">

js: .open("url","\_ blank ")

4. 在新窗口打开，只能打开一个

html: <a href="url" target="自定义name">

js: open("url","自定义name")

name属性: 内存中唯一标识一个窗口的名称

浏览器规定: 相同name属性的窗口只能打开一个

后打开的同名窗口会覆盖下打开的

name属性可自定义: 2种:

<a href="url" target="name"

open("url","name")

预定义的name:

\_self: 自动获取当前窗口name

\_blank: 不指定name，让浏览器随机生成

关闭: window.close();

3. 窗口大小和窗口定位:

窗口大小:

获取: 2种:

1. 完整大小:window.outerWidth/outerHeight

2. 文档显示区大小: 浏览器中用于显示网页的部分

window.innerWidth/innerHeight

设置: 2种:

1. 在打开窗口时，就设置新窗口的大小: 2步:

1. 定义配置字符串:

var config="left=值, top=值, width=值, height=值";

2. 在打开窗口时，将config作为参数

open("url","name",config)

2. 修改一个已经打开的窗口大小:

window.resizeTo(width,height)

window.resizeBy(width的增量,height的增量)

强调: 只能修改打开时使用config改过大小的窗口

窗口位置: 相对于屏幕左上角

获取: x/left: window.screenX/screenLeft

y/top: window.screenY/screenTop

设置: 2种:

1. 在打开窗口时: 依然使用open的config参数

2. 修改已经打开的窗口位置:

window.moveTo(left,top)

window.moveBy(left的增量,top的增量)

4. \*\*\*定时器: 2种:

1. 周期性定时器:

让程序按照指定的时间间隔, 反复执行相同任务

何时: 只要让程序按照指定时间间隔反复执行任务

动画

如何: 3件事:

1. \*\*\*任务函数: 定时器反复执行的函数

function task(){...}

2. 启动定时器: timer=setInterval(task,间隔ms);

强调: 参数task后不要加()，因为不是立刻执行，而是交给定时器，代为执行——回调

3. 停止定时器: clearInterval(timer)

timer: 唯一标识一个定时器的序号

只能在启动定时器时，迅速获得

停止: 2种:

1. 手动调用clearInterval()

2. 任务函数中设定临界值，自动判断是否结束

2. 一次性定时器:

让程序先等待一段时间，再执行一次任务

何时: 只要一项任务，需要等待一段时间后，才自动执行

如何: 3步:

1. \*\*\*任务函数: function task(){...}

2. 启动定时器: 开始等待

timer=setTimeout(task,等待ms)

3. 停止定时器: 停止等待，不再执行

clearTimeout(timer)

\*\*\*定时器的原理: 定时器中的任务函数，只能在主程序中所有函数执行完，才能开始调用。

for(var i=0;i<3;i++){

setTimeout("console.log(i)",0)

}

3个3

定时器中回调函数的this默认都指window

问题: 动画叠加: 反复无节制的创建新定时器，作用在同一个元素上导致混乱

解决: 1. 判断只有在没有动画播放时，才能启动新定时器

if(timer==null) 才启动新动画

2. 在启动新定时器之前，停止当前动画

clearInterval(timer), 再启动新动画

1. BOM常用对象

history: 保存当前窗口打开后成功访问过的url的历史记录栈

前进: history.go(1) history.go(n)

后退: history.go(-1) history.go(-n)

刷新: history.go(0)

location: 封装当前窗口正在打开的url的对象

属性:

.href: 获取或设置完整url

在当前窗口打开新地址: location.href="新url"

.protocol: 协议

.host: 主机名+端口号

.hostname: 主机名

.port: 端口号

.pathname: 网页的相对路径

.hash: 获得#锚点地址

.search: ?查询字符串

方法:

在当前窗口打开，可后退：

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

简化: location="新url"

在当前窗口打开新url，禁止后退：

location.replace("新url")

重新加载:

location.reload(false/true);

前提: 服务器默认会设置静态资源缓存，让浏览器在客户端本地缓存资源，避免反复下载，浪费时间。

参数: force: 强迫

默认:false: 优先使用浏览器本地缓存的资源

除非服务器上的资源更新了，比本地缓存新，才从服务器下载新资源

改为:true: 绕过浏览器本地缓存，强制从服务器下载新的资源

navigator: 封装浏览器配置信息的对象

cookieEnabled: 判断是否启用了cookie

cookie: 在客户端本地持久保存用户私密数据的小文件。

为什么: 内存中存储的数据，一旦程序结束或关机，就不复存在。无法持久保存

何时: 只要希望在客户端持久保存数据时

判断:

plugins: 封装所有插件信息的集合

plugins["插件名"]

userAgent: 封装浏览器名称和版本号的字符串

2. \*\*\*event:

事件: 浏览器自动触发的或用户手动触发的页面元素状态的改变

事件处理函数: 当事件发生时，自动调用的函数

如何绑定: 3种:

1. HTML元素的开始标签中:

<ANY on事件名="js语句"

问题: 不符合内容与行为分离的元素，不便于维护

解决: 事件处理函数都是在js中集中绑定

2. js中: elem.on事件名=function(){

...this->elem...

}

问题: 一个事件只能绑定一个处理函数

解决: addEventListener

3. js中,添加事件监听:

elem.addEventListener("事件名",fn)

优: 一个事件可同时绑定多个处理函数

elem.removeEventListener("事件名",fn)

事件周期/模型:

DOM: 3个阶段:

1. 捕获: 由外向内，依次记录各级父元素上绑定的事件处理函数。到目标元素结束。

目标元素: 最初实际触发事件的元素

强调: 捕获阶段只记录函数，不调用！

2. 目标触发: 首先触发目标元素上绑定的事件处理函数

3. 冒泡: 按捕获顺序的反向，由内向外，依次执行捕获的处理函数

事件对象: 事件发生时，自动创建的封装事件信息的对象

何时: 只要希望获取事件的信息或操作事件

如何: 事件对象默认作为处理函数的第一个参数传入

function handler(e){ ...e->事件对象...}

取消冒泡: e.stopPropagation();

利用冒泡:

问题: 浏览器在触发事件时，是遍历所有事件监听

事件监听创建的越多，响应速度就越慢

解决: 尽量少的添加事件监听——优化

如何: 如果多个平级子元素，绑定相同的事件处理函数，则只需要在父元素绑定一份，所有子元素共用即可

难题1: 如何获得目标元素:

错误: this->父元素

正确: e.target

难题2: 如何判断目标元素是否想要的:

target.nodeName

取消事件/阻止默认行为:

e.preventDefault();

2大用途:

1. 当用a作为按钮时，阻止在地址栏添加#

2. 表单提交之前，如果验证没通过，可取消提交

form.onsubmit: 正式提交表单之前触发

无论以何种方式提交表单，都会触发onsubmit

通常: 对表单中所有元素执行验证

只要一项没有验证通过，就取消提交

3. H5做拖拽时，都要取消浏览器默认的拖拽行为

事件坐标:

相对屏幕左上角: e.screenX|screenY

相对于文档显示区左上角: e.clientX|x

e.clientY|y

相对于事件发生的元素左上角: e.offsetX|offset

页面滚动距离 window).scrollTop();

元素相对于页面顶部的距离 $(selector).offset().top

Jquery

1. jQuery

什么是: 第三方, 快速，简洁的javascript函数库

为什么:

1. DOM操作的终极简化:

学jQuery还是在学DOM，只不过API更简单

核心DOM: 万能，API太繁琐

HTML DOM: 简洁, 不是万能

jQuery简化了:

1. DOM四大操作: 查找, 修改, 删除, 添加

2. 事件处理

3. 动画操作:

4. Ajax操作:

2. 屏蔽浏览器兼容性问题:

凡是jQuery让使用，都没有兼容性问题

2. 如何:

第三方: 不是浏览器内置的，也不是开发人员自己编写

由其它第三方组织或个人编写的

下载: jquery.com

版本: 1.x 兼容旧浏览器

jquery-1.x.js 未压缩版本 可读性好, 体积大

——学习

jquery-1.x.min.js 压缩版本体积小,可读性差

——生产环境

3件事: 1. 去掉注释

2. 去掉空格/缩进

3. 变量名最简化

2.x 不再兼容旧浏览器

CDN网络: 内容分发网络

共享的资源在网络中多台服务器上都有副本

客户端请求资源时, CDN网络会智能选择离客户端最近，网络最优的服务器下载资源。

为什么: 就近快速获得资源

何时: 只要公共的/通用的库/框架到从CDN网络中

如何:

script src="http://CDN地址 "

使用jQuery之前，必须先引入jQuery.js文件: 3

3. jQuery对象:

什么是: 用jQuery工厂函数包装DOM对象后形成的新对象

为什么: 普通的DOM对象不包含jQuery简化版API

何时: 只要希望使用jQuery简化版API之前，都要将DOM对象包装为jQuery对象

如何:

1. 将现有DOM对象包装为jQuery对象

var $xxx=jQuery(DOM对象)

何时: 如果已经获得了一个DOM对象，想用简化版API。 比如: this, e.target,转为jQuery对象

var $this=jQuery(this);

var $target=jQuery(e.target);

2. 直接用jQuery工厂函数查找:

var $xxx=jQuery("选择器")

何时: 绝大多数情况，在未获得任何元素时，首选用jQuery工厂函数直接查找元素。

简写:

引入jQuery.js文件时: 3件事:

1. 向window中添加了一个jQuery工厂函数，用于创建jQuery类型的对象

2. 为jQuery工厂函数添加了一个原型对象

在jQuery工厂函数的原型对象中封装了所有简化版API。

每次使用jQuery工厂函数获得的jQuery结果对象多继承自jQuery原型对象，可直接使用原型对象中保存的简化版API

3. 向window中添加全局变量$，引用jQuery工厂函数。——所有jQuery都可用$代替。

问题: 如果选择器查找到多个元素,如何保存

其实: jQuery对象是一个类数组对象

找到的每个DOM对象都是类数组对象中的一个元素

且jQuery简化版API都自带forEach效果

只需调用一次，即可自动应用到每个DOM对象上

Chrome 控制台中也可使用$("选择器")查找元素:

Chrome 控制台也提供了$函数，类似于jQuery的$，但无需引入jQuery也可使用。

如何: 只找一个: $("选择器") 相当于querySelector

找多个: $$("选择器") 相当于querySelectorAll

学习jQuery，还是在学DOM:

查找->绑定事件->修改->添加/删除

4. \*\*\*查找:

1. 选择器: CSS，jQuery新增

基本: 5个 同CSS

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

层级: 4个 同CSS

parent child parent>son

prev+next prev~siblings

练习:

1. 事件绑定: $(...).事件名(function(){...})

2. 修改css样式: $(...).css("css属性名","值")

强调: 对$(...)调用一次API，相当于自动对$(...)找到的每个DOM元素分别调用API——自带forEach

1. 查找:

按选择器查找:

按节点间关系查找:

1. 查找:

按选择器查找: 过滤选择器

基本过滤: 位置过滤——jQuery独有，css没有

什么是: 根据元素在查找结果集合中的\*下标\*位置选择元素

强调: 1. 将所有元素先查找到集合中，再按下标过滤

2. 下标从0开始

3. 和元素在父元素中的相对位置无关

何时: 只要希望根据元素在结果集合中的位置查找元素

如何: 包括:

:first/last 获取结果集合中第一个/最后一个元素

:even/odd 获取结果集合中偶数/奇数位置的元素

:lt/gt/eq(i) 获取结果集合中小于/大于/等于i位置的元素

特殊: :animated 选择正在播放动画的元素

:not(selector) 否定伪类,同CSS

练习: 问题: 一个.css只能设置一个css属性

解决: $(...).css({

css属性:值,

... : ... ,

})

强调: 1. css属性必须去横线变驼峰

强烈建议，只要在js中使用css属性，都去横线变驼峰

2. 如果属性值为数字, 可不加引号，且可省略单位

子元素过滤: 同css

什么是: 根据元素在其\*父元素中\*的\*相对\*位置选择

强调: 1. 序号从1开始

2. 序号相对于父元素内部编号

何时: 只要根据元素在其父元素中的位置选择

如何: 包括:

:first/last-child 获取作为其父元素下第一个子元素的所有元素

:nth-child(n|2n|2n+1|even|odd)

:only-child 获取作为独生子女的子元素

内容过滤: 根据元素的内容查找

:contains(text) 查找内容中包含text的元素

比如: span:contains("购物车")

强调: 只能是文字中包含text的，如果是标签名中包含，则不选择。

:empty 查找内容为空的元素

:parent 查找内容不为空的元素 :not(:empty)

强调: 只要包含内容(子元素或文本都行)就选择

:has(selector) 选择包含符合selector要求的子元素的父元素

比如: li:has(.active) 选择包含class为active的子元素的父元素li

可见性过滤:

:hidden : 选择所有不可见的元素

让网页元素不可见: 4种:

display:none, visibility:hidden, opacity:0

type="hidden"

但是:

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

:visible : 选择所有可见的元素

属性过滤: 同CSS的属性选择器

什么是: 按照任意属性的任意值选择元素

何时: 只要按除id,元素,class之外的其他属性作为条件查找时，甚至模糊查找时

如何:

[属性名] 选择包含指定属性的元素

[属性名=值] 选择属性的值等于指定值的元素

[属性名^=值] 选择属性值以指定值开头的元素

[属性名$=值] 选择属性值以指定值结尾的元素

[属性名\*=值] 选择属性值包含指定值的元素

[属性名!=值] 选择属性值不等于指定值的元素

其实: :not([属性名!=值])

强调: 即选择包含title属性，但值不符合

也选择不包含title属性的

特殊: 且

[属性选择器1][属性选择器2]...

表单元素过滤:

什么是: 利用表单元素的标签名和type属性值选择表单中的元素

何时: 只要查找表单中的表单元素时

如何: 包括:

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

每种type都对应一个选择器:

:text :password :radio :checkbox :submit :reset

:file :button :hidden :image

状态过滤:

什么是: 选择处于三大状态之一的元素

何时: 只要根据元素的状态选择元素

如何: 包括: :disabled :checked :selected

练习: 问题1: .css只能修改css属性

解决: .attr("属性名",新值) 可修改标准属性

.attr("属性名") 获取属性值

总结: jQuery API第二大特定:

一个函数两用: 没提供新值，就读取属性现有值

提供了新值，就修改属性为新值

问题2: .attr无法访问三大状态属性

解决: .prop() 专门操作三大状态属性

1. 查找:

按节点间关系查找:

何时: 如果已经获得一个元素，找周围的元素时

包括:

1. 父子:

$(...).parent() -> elem.parentNode

$(...).children([selector]) -> elem.children

强调: 1. 仅查找直接子节点

2. 可用选择器仅选择感兴趣的子节点

但选择器可写可不写

$(...).find(selector)

查找所有后代元素中满足selector条件的子元素

强调: 1. 不但查找直接子元素，且查找所有后代

2. 选择器必须写

2. 兄弟:

1. $(...).prev() 选择和当前元素紧邻的前一个兄弟

强调: 必须紧邻的前一个

$(...).prev([selector])

要求: 必须紧邻的前一个且满足selector

$(...).prevAll([selector]) 选择当前元素之前的所有兄弟

2. $(...).next(...)/.nextAll(...) 同prev/prevAll

3. $(...).siblings([selector]) 除自己之外的所有前后的平级兄弟

总结: jQuery所有API通用特点: 3个  
 1. 子代forEach

2. 一个函数两用

3. 每个API都返回正在使用的jQuery对象，为了后续链式操作

2. 修改:

内容: $(...).html([html代码片段]) -> elem.innerHTML

$(...).text([文本]) -> elem.textContent

$(...).val([值]) -> elem.value

属性: $(...).attr("属性名"[,"值"]) -> elem.属性

移除属性: $(...).removeAttr("属性名")

问题: 不能访问三大状态: disabled checked selected

解决: $(...).prop(...)

问题: 一句attr只能修改一个属性

解决: $(...).attr({

属性名: 值,

... : ... ,

})

样式: $(...).css("属性名"[,"值"])

获取: 自动获取计算后的完整样式

修改: 自动仅修改内联样式

强调: 凡是从页面上获得的属性/样式的值都是字符串

如要计算，必须先转为浮点数，去单位。

问题: 一句css只能修改一个属性

解决: $(...).css({

css属性名: 值,

... : ... ,

})

强调: 1. 所有css属性名都要去横线变驼峰

2. 如果属性值为数值, 可省略引号和单位

问题: 要修改的css属性很多，则代码繁琐

解决: 今后批量修改css，都要将css集中定义在class中，通过应用class来批量修改css样式

如何:

1. 为元素添加class : $(...).addClass("类名",...)

2. 从元素上移除类名 : $(...).removeClass("类名")

3. 判断元素是否包含指定class :

$(...).hasClass("类名")

4. 切换class: $(...).toggleClass("类名")

相当于: if($(...).hasClass(...))

$(...). removeClass("in");

else

$(...).addClass("in");

3. 添加/删除/复制/替换

添加: 2步:

1. 用html代码片段创建新元素

var $elem=$("html代码片段")

强调: html代码片段中如果同时包含父元素和子元素，则一并创建。

2. 将新元素添加到页面

$(parent).append($elem) -> appendChild

.prepend($elem) 在开头插入

$(child).before($elem) -> insertBefore

.after($elem) 插入到child之后

删除: $(...).remove();

替换:

$(现有元素的选择器).replaceWith(新元素代码片段)

用右边的新元素，代替左边的现有元素

$(代码片段).replaceAll(现有元素的选择器)

用左边创建的新元素，代替右边的现有元素

复制:

var $elem\_clone=$(elem).clone()

1. 事件:

终极简化, 模拟触发, 页面加载后触发, hover

2. 动画:

3. 类数组对象操作:

1. 事件:

废弃: $(...).live(事件名, fn) $(...).die(事件名, fn)

极端: 将所有事件集中绑定在根节点(document)上

终极简化:

1. 代替bind: $(...).on("事件名",fn)

2. 代替delegate:

$(parent).on("事件名","selector",fn)

解绑: $(...).off("事件名",["selector",]fn)

模拟触发: 希望通过代码代替用户操作，触发元素的事件

$(...).trigger("事件名")

简写: $(...).事件名()

比如: $(form).submit(); 提交表单

总结: jQuery简化版事件处理函数两用:

$(...).事件名([fu]) 有fn，则绑定事件

没有fn，则模拟触发事件

页面加载后执行:

1. 整个页面加载完成: window.onload=function(){...}

HTML,CSS,JS,图片全部加载完成

2. DOM内容加载完成: DOMContentLoaded

仅HTML,JS加载并执行完成

有兼容性问题

DOM内容加载完成比整个页面加载完成提前触发

何时: 凡是对DOM的初始化操作，都要放在DOM内容加载完成时提前执行，不必等待整个页面加载完成

比如: 动态生成DOM内容和事件绑定

如何: jQuery

$(document).ready(function(){...})

简写: $().ready(function(){...})

$(function(){...})

终极简化: 只要将代码直接放在body结尾的script中

本身就是DOM内容加载后执行

特殊事件: hover:

什么是: 同时绑定两个事件: mouseover和mouseout

何时: 只要同时绑定mouseover和mouseout时

如何: $(...).hover(fun1, fun2)

其中: fun1绑定给mouseover

fun2 绑定给mouseout

更简化: 如果可将fun1和fun2统一为一个函数时，可简写为: $(...).hover(fn);

无论mouseover还是mouseout都执行fn

2. 动画:

简单动画: 3种:

1. 显示/隐藏: $(...).show/hide/toggle();

问题: 默认show/hide使用display瞬间显示隐藏

解决: 在参数中加入时间

问题: 所有效果属性变化，都写死在jQuery源代码内部，无法按需求更改。

解决: 1. 少用简单动画

2. class+transition

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

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

自定义动画:

什么是: 可对任意数值类型的CSS属性执行动画效果

何时: 只要希望对任意数值类型执行动画效果

如何: $(...).animate({

css属性: 目标值,

... : ... ,

},speed[,easing,fn])

其中: speed是动画持续的总毫秒数

easing 是动画变换的效果: linear swing

fn 是动画结束后的回调函数

强调: 只能对单个纯数字的属性值执行动画变换

不支持颜色动画

并发和排队:

并发: 多个css属性同时变化

1. 放在同一个animate内的多个css属性都是并发变化

2. 同时给多个元素添加animate效果，不同元素间的动画也是并发。

排队: 多个css属性，先后逐个变化

先后调用的不同animate内的多个css属性是排队变化——动画队列

停止动画: $(...).stop()

问题: 默认仅停止队列中正在执行的当前动画。后续动画，继续执行

解决: $(...).stop(true) true表示清空队列

推迟执行: $(...).delay(ms).animate(...)

查找一个正在播放动画的元素: :animated

3. 类数组对象:

.size() -> 代替.length获得类数组对象中的元素个数

.get(i) -> 获得数组对象中i位置的元素

.each(fn)-> 相当于数组的forEach，对类数组对象中每个元素执行相同操作

其中: fn :

function(i){ //i 当前元素的下标

this->当前DOM元素->$(this)->jQuery对象

}

.index() -> 在类数组对象中查找指定元素的下标位置。类似于indexOf函数

$("selector").index(DOM/$(...))

在满足selector的所有元素中找右侧DOM/JQ对象的位置——在左边的集合中找右边的对象

$("selector1").index("selector2")

在满足selector2的所有元素中找左侧selector1元素的位置——在右边的集合中找左边的对象

$("selector").index()

返回selector元素在其父元素下平辈子元素中的下标位置