初级前端模块笔记

html/css 模块

1.html 基本结构

(1) html 标签是由<>包围的关键词。
(2) html 标签通常成对出现，分为标签开头和标签结尾。
(3) 有部分标签是没有结束标签的，为单标签，单标签必须使用 / 结尾。
(4) 页面所有的内容，都在 html 标签中。
(5) html 标签分为三部分：标签名称，标签内容，标签属性。
(6) html 标签具有语义化，可通过标签名能够判断出该标签的内容，语义化的作用是网页
(7) 结构层次更清晰，更容易被搜索引擎收录，更容易让屏幕阅读器读出网页内容。
(8) 标签的内容是在一对标签内部的内容。
(9) 标签的内容可以是其他标签。

2.标签属性

(1) class 属性：用于定义元素的类名
(2) id 属性：用于指定元素的唯一 id，该属性的值在整个 html 文档中具有唯一性
(3) style 属性：用于指定元素的行内样式，使用该属性后将会覆盖任何全局的样式设定
(4) title 属性：用于指定元素的额外信息
(5) accesskey 属性：用于指定激活元素的快捷键
(6) tabindex 属性：用于指定元素在 tab 键下的次序
(7) dir 属性：用于指定元素中内容的文本方向，属性只有`ltr`或`rtl`两种

(8) lang 属性：用于指定元素内容的语言

3.事件属性

(1) **window 窗口事件**，onload，在网页加载结束之后触发，onunload，在用户从网页离开时发生（点击跳转，页面重载，关闭浏览器窗口等）

(2) **form 表单事件**，onblur，当元素失去焦点时触发，onchange，在元素的值被改变时触发，onfocus，当元素获得焦点时触发，onreset，当表单中的重置按钮被点击时触发，onselect，在元素中文本被选中后触发，onsubmit，在提交表单时触发

(3) **keyboard 键盘事件**，onkeydown，在用户按下按键时触发，onkeypress，在用户按下按键后，按着按键时触发。该属性不会对所有按键生效，不生效的有，alt,ctrl,shift,esc

(4) **mouse 鼠标事件**，onclick，当在元素上发生鼠标点击时触发，onblclick，当在元素上发生鼠标双击时触发，onmousedown，当元素上按下鼠标按钮时触发，onmousemove，当鼠标指针移动到元素上时触发，onmouseout，当元素指针移出元素时触发，onmouseup，当元素上释放鼠标按钮时触发。Media 媒体事件，onabort，当退出时触发，onwaiting，当媒体已停止播放但打算继续播放时触发。

4.文本标签

(1) 段落标签 ，段落标签用来描述一段文字

(2) 标题标签<hx></hx>，标题标签用来描述一个标题，标题标签总共有六个级别，<h1></h1>标签在每个页面中通常只出现一次

(3) 强调语句标签，，用于强调某些文字的重要性

(4) 更加强调标签，和标签一样，用于强调文本，但它强调的程度更强一些

(5) 无语义标签，标签是没有语义的

(6) 短文本引用标签<q></q>，简短文字的引用

(7) 长文本引用标签<blockquote></blockquote>，定义长的文本引用

(8) 换行标签

5.多媒体标签

(1) 链接标签，<a>

(2) 图片标签，<img/>

(3) 视频标签，<video></video>

(4) 音频标签，<audio></audio>

6.列表

(1) 无序列表标签，ul,li，
列表定义一个无序列表，
代表无需列表中的每一个元素

(2) 有序列表，ol,li

(3) 定义列表，<dl></dl>，定义列表通常和<dt></dt>和<dd></dd>标签一起使用

7.表格

(1) 表格标签<table></table>

(2) 表格的一行<tr></tr>

(3) 表格的表头<th></th>

(4) 单元格<td></td>

(5) 表格合并，同一行内，合并几列 colspan="2"，同一列内，合并几行 rowspan="3"

8.表单标签

（1）表单标签<form>

<form></form>表单是可以把浏览者输入的数据传送到服务器端，这样服务器端程序就可以处理表单传过来的数据。

<form method="传送方式" action="服务器文件">

action，浏览者输入的数据被传送到的地方

method，数据传送的方式

（2）输入标签<input/>

name：为文本框命名，用于提交表单，后台接收数据用。

value：为文本输入框设置默认值。

type：通过定义不同的 type 类型，input 的功能有所不同。

- text　单行文本输入框
- password　密码输入框（密码显示为***)
- radio　单选框 （checked 属性用于显示选中状态）
- checkbox　复选框（checked 属性用于显示选中状态）
- file　上传文件
- button 普通按钮
- reset　重置按钮（点击按钮，会触发 form 表单的 reset 事件）
- submit 提交按钮（点击按钮，会吃饭 form 表单的 submit 事件）
- email　专门用于输入 e-mail
- url 专门用于输入 url
- number 专门用于 number
- range　显示为滑动条，用于输入一定范围内的值
- date　选取日期和时间（还包含：month、week、time、datetime、datetime-local）
- color　选取颜色

button 按钮，下拉选择框<select></select>

- <option value="提交值">选项</option>是下拉选择框里面的每一个选项

（3）文本域：<textarea></textarea>，当用户想输入大量文字的时候，使用文本域。cols，多行输入域的列数，rows，多行输入域的行数。

9.其他语义化标签

(1) 盒子

(2) 网页头部<header></header>，html5 新增语义化标签，定义网页的头部，主要用于布局，分割页面的结构

(3) 底部信息<footer></footer>，html5 新增语义化标签，定义网页的底部，主要用于布局，分割页面的结构

(4) 导航<nav></nav>，html5 新增语义化标签，定义一个导航，主要用于布局，分割页面的结构

(5) 文章<article></article>，html5 新增语义化标签，定义一篇文章，主要用于布局，分割页面的结构

(6) 侧边栏<aside></aside>，语义化标签，定义主题内容外的信息，主要用于布局，分割页面的结构。

(7) 时间标签<time></time>，语义化标签，定义一个时间

10.网页结构

(1) <!DOCTYPE html>定义文档类型，告知浏览器用哪一种标准解释 HTML

(2) <html></html>可告知浏览器其自身是一个 HTML 文档

(3) <body></body>标签之间的内容是网页的主要内容

(4) <head></head>标签用于定义文档的头部，它是所有头部元素的容器

(5) <title></title>元素可定义文档的标题

(6) <link>标签将 css 样式文件链接到 HTML 文件内

(7) <meta>定义文档的元数据

11.模块划分

(1) 常见的企业网站，多由头部区，展示图片区域，主题区域，底部信息区域组成
(2) 网页拆分原则： - 由上到下 - 由内到外

12.CSS 代码语法

(1) CSS 全称为层叠样式表(Cascading Style Sheets)，它主要是用于定义 HTML 内容在浏览器内的显示样式，如文字大小、颜色、字体加粗等。
(2) css 代码通常存放在<style></style>标签内
(3) css 样式由选择符和声明组成，而声明又由属性和值组成
(4) 选择符{属性:值}
(5) 选择符：又称选择器，指明网页中要应用样式规则的元素

13.CSS 放置位置

(1) 行内样式，不建议使用
(2) 内联式样式表
(3) 外联样式表

14.CSS 的继承

(1) CSS 的某些样式是具有继承性的，继承是一种规则，它允许样式不仅应用于某个特定 html 标签元素，而且应用于其后代。
(2) 不可继承样式：display、margin、border、padding、background、height、min-height、max-height、width、min-width、max-width、overflow、position、left、right、top、bottom、z-index、float、clear
(3) 可以继承的样式： letter-spacing、word-spacing、white-space、

line-height、color、font、font-family、font-size、font-style、font-variant、font-weight、text-decoration、text-transform、direction、visibility、cursor

15.选择器的种类

(1) 标签选择器:通过标签的名字，修改 css 样式
(2) 通配符选择器:选择页面中所有的元素
(3) 属性选择器
(4) 后代选择器:选择某个父元素下面所有的元素
(5) 一级子元素选则器:选择某个父元素的直接子元素，后代选择器是选择父元素的所有子孙元素，一级子元素原则器只选择第一级子元素，不会再向下查找元素
(6) id 选择器：通过 id 查找页面中唯一的标签
(7) class 选择器:通过特定的 class（类）来查找页面中对应的标签，以 .class 名称
(8) 伪类选择器:

- :hover 鼠标移入某个元素;
- :before 在某个元素的前面插入内容;
- :after 在某个元素的后面插入内容

(9) 群组选择器:可以对多个不同的选择器设置相同的样式

16.选择器的优先级

（1）当有不同的选择器对同一个对象进行样式指定时，并且两个选择器有相同的属性被赋予不同的值时。
（2）通过测算那个选择器的权重值最高，应用哪一个选择器的样式
（3）权重计算方式:

标签选择器：1

class 选择器：10

id 选择器：100

行内样式：1000

!important 最高级别，提高样式权重，拥有最高级别

17.背景样式

（1）背景颜色 background-color

（2）背景图片 background-image

```
background-image:url(bg01.jpg);
```

（3）背景图片位置`background-position`

```
background-position:10px 100px;
// 代表坐标 x，y 轴
```

（4）背景图片重复`background-repeat`

```
background-repeat:no-repeat
// no-repeat 设置图像不重复，常用
// round 自动缩放直到适应并填充满整个容器
// space 以相同的间距平铺且填充满整个容器
```

（5）背景图片定位`background-attachment`

```
background-attachment:fixed
// 背景图像是否固定或者随着页面的其余部分滚动
// background-attachment 的值可以是 scroll（跟随滚动），fixed（固定）
```

（6）background 缩写

background:#ff0000 url(bg01.jpg) no-repeat fixed center

18.字体样式

（1）字体族 font-family

font-family:"微软雅黑","黑体";

（2）字体大小 font-size

font-size:12px;

网页默认字体大小是 16px

（3）字体粗细 font-weight

font-weight:400;

- normal（默认）
- bold（加粗）
- bolder（相当于和<b>标签）
- lighter （常规）
- 100 ~ 900 整百（400=normal，700=bold）

（4）字体颜色 color

颜色的英文单词 color:red;

十六进制色：color: #FFFF00;

RGB(红绿蓝)color:rgb(255,255,0)

RGBA（红绿蓝透明度）A 是透明度在 0~1 之间取值。color:rgba(255,255,0,0.5)

（5）字体斜体 font-style

font-style:italic

19.文本属性

（1）行高 line-height

line-height:50px;

可以将父元素的高度撑起来

（2）文本水平对齐方式 text-align

- left 左对齐
- center 文字居中
- right 右对齐

（3）文本所在行高的垂直对齐方式 vertical-align

- baseline 默认
- sub 垂直对齐文本的下标，和<sub>标签一样的效果
- super 垂直对齐文本的上标，和<sup>标签一样的效果
- top 对象的顶端与所在容器的顶端对齐
- text-top 对象的顶端与所在行文字顶端对齐
- middle 元素对象基于基线垂直对齐
- bottom 对象的底端与所在行的文字底部对齐
- text-bottom 对象的底端与所在行文字的底端对齐

（4）文本缩进 text-indent

text-indent:2em;

通常用在段落开始位置的首行缩进

（5）字母之间的间距 letter-spacing

（6）单词之间间距 word-spacing

（7）文本的大小写 text-transform

- capitalize 文本中的每个单词以大写字母开头
- uppercase 定义仅有大写字母
- lowercase 定义仅有小写字母

（8）文本的装饰 text-decoration

- none 默认
- underline 下划线
- overline 上划线
- line-through 中线

（9）自动换行 word-wrap

word-wrap: break-word;

20.基本样式

（1）宽度 width

width:200px;

定义元素的宽度

（2）高度 height

height:300px

- 元素默认没有高度
- 需要设置高度
- 可以不定义高度，让元素的内容将元素撑高

（3）鼠标样式 cursor

定义鼠标的样式 cursor:pointer

- default 默认

- pointer 小手形状
- move 移动形状

（4）透明度 opacity

opacity:0.3

- 透明度的值 0~1 之间的数字，0 代表透明，1 代表完全不透明
- 透明的元素，只是看不到了，但是还占据着文档流

（5）可见性 visibility

visibility:hidden;

visible 元素可见

hidden 元素不可见

collapse 当在表格元素中使用时，此值可删除一行或一列，不会影响表格的布局。

（6）溢出隐藏 overflow

设置当对象的内容超过其指定高度及宽度时如何显示内容

- visible 默认值，内容不会被修剪，会呈现在元素框之外
- hidden 内容会被修剪，并且其余内容是不可见的
- scroll 内容会被修剪，但是浏览器会显示滚动条以便查看其余的内容
- auto 如果内容被修剪，则浏览器会显示滚动条以便查看其余的内容

（7）边框颜色 outline

input 文本输入框自带边框，且样式丑陋，我们可以通过 outline 修改边框

- outline:1px solid #ccc;
- outline:none 清除边框

21.样式重置

早期的网页没有 css 样式，为了界面美观，很多元素自带 margin、padding 等样式，但这些样式在不同浏览器解析的值都不一样，需要将 css 样式重置，保证在不同浏览器显示一致。

- 清除元素的 margin 和 padding
- 去掉自带的列表符
- 去掉自带的下划线

22.盒模型样式

（1）块状元素、内联元素和内联块状元素。

块级元素:

- 每个块级元素都从新的一行开始，并且其后的元素也另起一行。
- 元素的高度、宽度、行高以及顶和底边距都可设置。
- 元素宽度在不设置的情况下，是它本身父容器的 100%，除非设定一个宽度。

行内元素:

- 和其他元素都在一行上
- 元素的高度、宽度、行高及顶部和底部边距不可设置
- 元素的宽度就是它包含的文字或图片的宽度，不可改变。

行内块状元素:

- - 和其他元素都在一行上
- - 元素的高度、宽度、行高以及顶和底边距都可设置。

（2）元素分类转换 display

- block：将元素转换为块级元素
- inline：将元素装换为行级元素
- inline-block：将元素转换为内联块元素
- none：将元素隐藏

（3）描边 border

border:2px solid #f00;

线条的样式：

dashed（虚线）| dotted（点线）| solid（实线）。

css 样式中允许只为一个方向的边框设置样式：

- 下描边 border-bottom:1px solid red;
- 上描边 border-top:1px solid red;
- 右描边 border-right:1px solid red;
- 左描边 border-left:1px solid red;

（4）间距 margin

div{margin:20px 10px 15px 30px;}

（5）内填充 padding

padding:10px

23.浮动 float

（1）浮动原理

- 浮动使元素脱离文档普通流，漂浮在普通流之上的。
- 浮动元素依然按照其在普通流的位置上出现，然后尽可能的根据设置的浮动方向向左或者向右浮动，直到浮动元素的外边缘遇到包含框或者另一个浮动元素为止，且允许文本和内联元素环绕它。
- 浮动会产生块级框（相当于设置了 display:block），而不管该元素本身是什么。

（2）清除浮动带来的影响

clear 清除浮动：

- none : 不清除（默认值）。
- left : 不允许左边有浮动对象
- right : 不允许右边有浮动对象
- both : 不允许两边有浮动对象

（3）利用伪类实现清除浮动

```
.clearFix {
    content="";
    display:block;
    width:0;
    height:0;
    clear:both;
}
```

24.定位 position

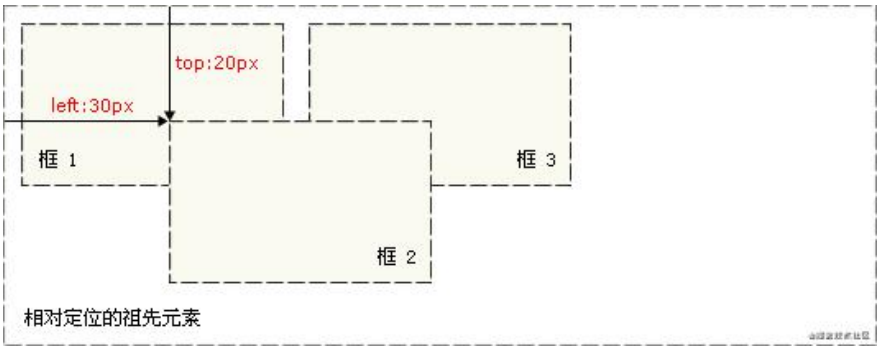
（1）定位功能可以让布局变的更加自由。

（2）层模型--绝对定位（相对于父类）

绝对定位使元素的位置与文档流无关，因此不占据空间。这一点与相对定位不同，相对定位实际上被看作普通流定位模型的一部分，因为元素的位置相对于它在普通流中的位置。

```
#box_relative {
  position: absolute;
  left: 30px;
  top: 20px;
}
```

如下图所示：



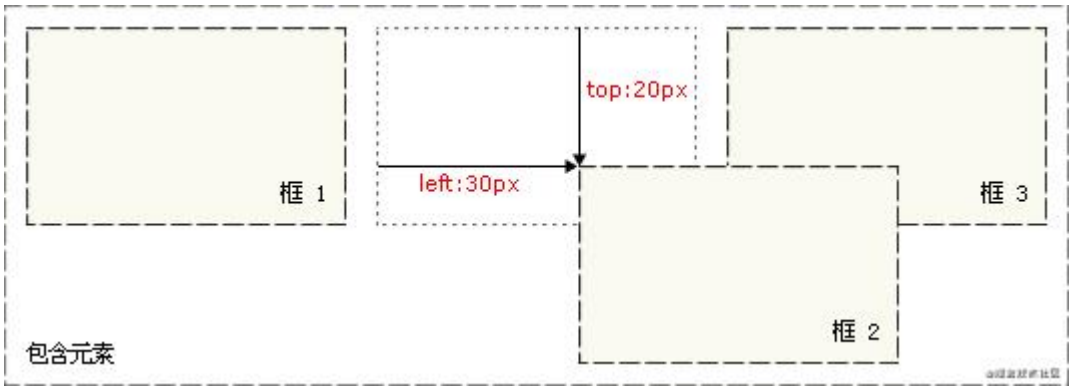
如果想为元素设置层模型中的绝对定位，需要设置`position:absolute`(绝对定位)，这条语句的作用将元素从文档流中拖出来，然后使用`left、right、top、bottom`属性相对于其最接近的一个具有定位属性的父包含块进行绝对定位。如果不存在这样的包含块，则相对于`body`元素，即相对于浏览器窗口。

（3）层模型--相对定位（相对于原位置）

相对定位是一个非常容易掌握的概念。如果对一个元素进行相对定位，它将出现在它所在的位置上。然后，可以通过设置垂直或水平位置，让这个元素“相对于”它的起点进行移动。

```
#box_relative {
  position: relative;
  left: 30px;
  top: 20px;
}
```

如下图所示：



如果想为元素设置层模型中的相对定位，需要设置`position:relative`（相对定位），它通过`left、right、top、bottom`属性确定元素在正常文档流中的偏移位置。相对定位完成的过程是首先按`static(float)`方式生成一个元素(并且

元素像层一样浮动了起来),然后相对于原位置移动,移动的方向和幅度由`left、right、top、bottom`属性确定，偏移前的位置保留不动。

（4）层模型--固定定位（相对于网页窗口）

position:fixed

与 absolute 定位类型类似，但它的相对移动的坐标是视图（屏幕内的网页窗口）本身。由于视图本身是固定的，它不会随浏览器窗口的滚动条滚动而变化，除非你在屏幕中移动浏览器窗口的屏幕位置，或改变浏览器窗口的显示大小，因此固定定位的元素会始终位于浏览器窗口内视图的某个位置，不会受文档流动影响，这与 background-attachment:fixed 属性功能相同。

25.浏览器默认样式

（1）页边距

IE 默认为 10px，通过 body 的 margin 属性设置

要清除页边距一定要清除这两个属性值

```
body { margin:0; padding:0;}
```

（2）段间距

IE 默认为 19px，通过 p 的 margin-top 属性设置

p 默认为块状显示，要清除段间距，一般可以设置

```
p { margin-top:0; margin-bottom:0;}
```

html5

1.HTML5 的优势

解决跨浏览器，跨平台问题

增强了 web 的应用程序

2.HTML5 废弃元素

frame frameset noframes

acronym applet dir

basefont big center font strike tt

3.HTML5 新增元素

<header> 用于定义文档或节的页眉

<footer> 用于定义文档或节的页脚

<article> 用于定义文档内的文章

<section> 用于定义文档中的一个区域（或节）

<aside> 用于定义与当前页面或当前文章的内容几乎无关的附属信息

<figure> 用于定义一段独立的引用，经常与说明(caption)<figcaption>配合使用，通常用在主文中的图片，代码，表格等。

<figcaption> 用于表示是与其相关联的引用的说明/标题，用于描述其父节点<figure>元素里的其他数据。

<hgroup> 用于对多个<h1>~<h6>元素进行组合

<nav> 用于定义页面上的导航链接部分

<mark> 用于定义高亮文本

<time> 用于显示被标注的内容是日期或时间（24 小时制

<meter> 用于表示一个已知最大值和最小值的计数器

<progress>用于表示一个进度条

<audio> 定义声音，比如音乐或其他音频流

<video> 定义视频，比如电影片段或其他视频流

4.HTML5 表单相关元素和属性

input 新增 type 类型

- color 用来创建一个允 许用户使用颜色选择器，或输入兼容 CSS 语法的颜色代码的区域
- time 生成一个时间选择器
- datetime 生成一个 UTC 的日期时间选择器
- datetime-local 生成一个本地化的日期时间选择器
- date 显示一个日期输入区域，可同时使用日期选择器，结果值包括年、月、日，不包括时间。
- month 生成一个月份选择器，它结果值包括年份和月份， 但不包括日期
- week 生成一个选择的几周的选择器
- email 生成一个 E-mail 输入框
- number 生成一个只能输入数字的输入框
- range 生成一个拖动条，通过拖动条，使得用户只能输入指定范围，指定步长的值
- search 生成一个专门用于输入搜索关键字的文本框
- tel 生成一个只能输入电话号码的文本框
- url 生成一个 URL 输入框

HTML5 input 新增属性

- placeholder 主要用在文本框，规定可描述输入字段预期值的简短的提示信息
- autocomplete 为了完成表单的快速输入，一般浏览器提供了自动补全的功能选择
- autofocus 当为某个表单控件增加该属性后，当浏览器打开这个页面， 这个表单控件会自动获得焦点
- list 为文本框指定一个可用的选项列表，当用户在文本框中输 入信息时，会根据输入的字符，自动显示下拉列表提示，供用户从中选择
- pattern 用于验证表单输入的内容，通常 HTML5 的 type 属性，比如

email、tel、 number、url 等，已经自带了简单的数据格式验证功能了，加上 pattern 属性后， 验证会更加高效

- novalidate 当提交表单时不对其进行验证
- required 必需在提交之前填写输入字段
- spellcheck 拼写检查，为<input>、<textarea>等元素新增属性
- formenctype 规定在发送到服务器之前应该如何对表单数据进行编码
- formtarget 带有两个提交按钮的表单，会提交到不同的目标窗口
- multiple 一次上传多个文件
- maxlength wrap <textarea>新增
maxlength：用于规定文本区域最大字符数。
wrap：是否包含换号符（soft/ hard）

css3

1.CSS3 新增选择器

1）兄弟选择器

元素 1~ 元素 2 第 1 个元素之后，所有的元素 2 都会被选择，且这些元素和第一个元素拥有同一个父元素（两个元素之间不一定要相邻）。

2）属性选择器

- E[attribute^=value]用于选取带有以指定值开头的属性值的元素
- E[attribute$=value]用于选取属性值以指定值结尾的元素
- E[attribute*=value]用于选取属性值中包含指定值的元素，位置不限，也不限制整个单词

3）伪类选择器

- :root 选择文档的根元素，HTML 里，永远是`<html>`元素
- :last-child 向元素添加样式，且该元素是它的父元素的最后一个子元素

- :nth-child(n) 向元素添加样式，且该元素是它的父元素的第 n 个子元素
- :nth-last-child(n) 向元素添加样式，且该元素是它的父元素的倒数第 n 个子 元素
- :only-child 向元素添加样式，且该元素是它的父元素的唯一子元素
- :first-of-type 向元素添加样式，且该元素是同级同类型元素中第一个元素
- :last-of-type 向元素添加样式，且该元素是同级同类型元素中最后一个元素
- :nth-of-type(n) 向元素添加样式，且该元素是同级同类型元素中第 n 个元 素
- :nth-last-of-type(n) 向元素添加样式，且该元素是同级同类型元素中倒数第 n 个元素
- :only-of-type 向元素添加样式，且该元素是同级同类型元素中唯一的元素
- :empty 向没有子元素（包括文本内容）的元素添加样式

4）伪元素选择器

- :enabled 向当前处于可用状态的元素添加样式，通常用于定义表单的样式或者超链接的样式
- :disabled 向当前处于不可用状态的元素添加样式，通常用于定义表单的样式或者超链接的样式
- :checked 向当前处于选中状态的元素添加样式
- :not(selector) 向不是 selector 元素的元素添加样式
- :target 向正在访问的锚点目标元素添加样式
- ::selection 向用户当前选取内容所在的元素添加样式

2.CSS3 新增属性

1）新增背景属性

- background-cli 设置背景覆盖范围 border-box/paddingbox/content-box
- background-origin 设置背景覆盖的起点 border-box/paddingbox/content-box
- background-size 设置背景的大小 cover/contain/长度/百分比

2）新增的字体文本相关属性

- text-overflow 设置当文本溢出元素框时处理方式 clip/ellipsis
- word-wrap 规定单词的换行方式 normal/break-word
- word-break 规定自动换行的方式 normal/break-all/keep-all

3）新增盒模型属性

- box-shadow 阴影 h-shadow v-shadow
blur spread color inset
- resize 调整尺寸 none/both/horizontal
- outline-offset 轮廓的偏移量 length/inherit

3.新增变形动画属性

1）transform

- translate(x,y)
- rotate(angle)
- scale(x,y)
- skew(angleX ,angleY)

2）transform-origin

表示元素旋转的中心点，默认值为 50% 50%。

- 第一个值表示元素旋转中心点的水平位置，它还可以赋值 left、right、center、长度、百分比。
- 第二个值表示元素旋转中心点的垂直位置，它还可以赋值 top、bottom、center、长度、百分比。

4.3D 变形属性

1）transform 3D 函数
transform 增加了三个变形函数：

- rotateX：表示元素沿着 x 轴旋转
- rotateY：表示元素沿着 y 轴旋转
- rotateZ：表示元素沿着 z 轴旋转

2）transform-style 用来设置嵌套的子元素在 3D 空间中显示效果。
3）perspective 设置成透视效果，透视效果为近大远小。
4）perspective-origin 设置 3D 元素所基于的 x 轴和 y 轴，改变 3D 元素的底部位置，该属性取值同 transform-origin，默认值为 50% 50%。
5）backface-visibility 用来设置当元素背面面向屏幕时是否可见，通常用于设置 不希望用户看到旋转元素的背面。

它的属性值有 visible（背面可见，默认值）、 hidden（背面不可见）两个。

5.CSS3 的过渡属性

- transition-delay 设置过渡的延迟时间
- transition-duration 设置过渡的过渡时间
- transition-timing-function 设置过渡的时间曲线
- transition-property 设置哪条 CSS 使用过渡
- transition 一条声明设置 所有过渡属性

6.CSS3 的动画属性

1）animation

- @keyframes 定义动画选择器
- animation-name 使用@keyframes 定义的动画

- animation-delay 设置动画的持续动画时间
- animation-timing-function 设置动画的时间曲线
- animation-iteration-count 设置动画播放次数
- animation-direction 设置动画反向播放
- animation-play-state 设置动画播放状态
- transition 一条声明设置所有动画属性

7.CSS3 新增多列属性

- column-count 设置元素应该被分隔的列数
- column-width 设置列的宽度
- columns 一条声明设置列宽和列数 `column`
- column-gap 设置列之间的间隔
- column-span 设置元素应该横跨的列数
- column-rule-style 设置列之间间隔的样式
- column-rule-color 设置列之间间隔的颜色
- column-rule-width 设置列之间间隔的宽度
- column-rule 一条声明设置列之间间 隔所有属性

8.CSS3 新增单位

px、em、rem、vh、 vw 和% 移动端长度单位

使用 CSS 单位 px、em、rem、vh、 vw 等实现页面布局。

- px：绝对单位，页面按精确像素展示
- em：相对单位，基准点为父节点字体的大小，如果自身定义了 font-size 按自身来计算（浏览器默认字体是 16px），整个页面内 1em 不是一个固定的值。

em 会根据父级元素的大小而变化，但是如果嵌套了多个元素，要计算它的大小，

是很容易出错的，这样就引申出了 rem。

- rem：相对单位，可理解为”root em”，相对根节点 html 的字体大小来计算，CSS3 新加属性。
- %：% 百分比，相对长度单位，相对于父元素的百分比值
- vw、vh、vmin、vmax 主要用于页面视口大小布局

vw：viewpoint width，视窗宽度，1vw 等于视窗宽度的 1%。

vh：viewpoint height`，视窗高度，1vh 等于视窗高度的 1%。

- vmin：vw 和 vh 中较小的那个
- vmax：vw 和 vh 中较大的那个

9.弹性盒模型

弹性盒模型的语法基础概念

任何一个容器都可以指定弹性布局

JavaScript

1.JavaScript 基础

1）外部引入 js 文件：通过<script src="main.js"></script>

2）关键词

break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

3）变量名大小写敏感

4）命名规范

2.JavaScript 数据类型

1）字符串(String)

2）数字（Number）

3）布尔值（Boolean）

4）未定义（Undefined）

```
//undefined 有两种结果
//1、真的没定义
alert(typeof dada); //undefined

//2、定义了，但是没有放东西进去
var dada;
alert(dada); //undefined
```

undefined，表示未定义或只声明未给值的变量

5）对象（Object）

js 中内置了如下的对象：

- Object 是所有 JS 对象的超类(基类），JS 中的所有对象都是继承自 Object 对象的
- Array 数组对象 定义数组属性和方法
- Number 数字对象
- Boolean 布尔对象 布尔值相关
- Error 错误对象 处理程序错误
- Function` 函数对象 定义函数属性和方法

- Math 数学对象
- Date 日期对象
- RegExp 对象正则表达式对象 定义文本匹配与筛选规则
- String 字符串对象 定义字符串属性和方法

3.算术运算

var y = 3;

运算符	描述	例子	结果
+	加	x=y+2	x=5
-	减	x=y-2	x=1
*	乘	x=y*2	x=6
/	除	x=y/2	x=1.5
%	取余（模运算）	x=y%2	x=1
++	递增（累加）	x=++y ; x=y++	x=4 ; x=3
--	递减	x=--y ; x=y--	x=2 ; x=3

4.强制转换

1）字符串转数字`parseInt() parseFloat() isNaN()`

2）数字转为字符串`toString()`

5.赋值运算

复合的赋值运算符 `+= -= *= /= %=`

6.关系运算

1）关系运算：`> < <= >= != == === ==和=== !=和!==`

“=”、“==”、“===”有什么区别？

1）`=` 是赋值符号

2）`==` 忽略数据类型的判断 是否相等

3）`===` 数值和数据类型都要相等才判断为相等

7.逻辑运算

- 逻辑与`&&`
- 逻辑或 `||`
- 逻辑非 `!`
- 复合逻辑表达式

8.三元运算

条件运算符？：

三元运算符：(比较表达式)？结果 1:结果 2

9.分支循环

程序运行的三大结构：顺序结构、选择结构、循环结构

- 单分支选择：`if`语句
- 双分支选择：`if-else`语句
- 多分支语句：`if-else if-else` 语句

10.switch

语法格式：

```
switch(num){ //表达式
  case 1:
```

```
    //执行代码块 1
    break;  //中断执行，跳出
    ...
  default:  //默认，其他都不是的情况下执行
    //执行代码块
    break;
}
//强调：break 非常重要，如果不加 break 的话，程序会一直继续往下执行；
```

11.while

while 循环的特点：不知道具体执行的次数时，使用最合适

语法格式:

```
while(条件表达式){
     //要重复执行的代码段 - 循环体
}
```

12.do-while

语法格式：

```
do{
  //循环体
}while(循环条件判断);
```

- do-while 是先执行循环体，再检测循环条件。
- do-while 能保证循环体至少执行一次。
- 其他循环无法保证循环至少执行一次。

13.for

```
for(
1 循环变量初始化;
2 循环条件判断;
4 循环变量的修改
){
  3 循环体
}
```

14.break 和 continue

1）break 退出循环

2）continue 跳过本次循环，继续下一次循环

15.数组

1）数组定义

```
var arr = new Array();
var arr = [];
```

2）字面量方式定义

```
var arr = ["1","2"];
```

3）向数组赋值

```
arr[0] = "1";
arr[1] = "2";
alert(arr[0]+","+arr[1]);
```

4）数组索引

arr[0]+","+arr[1]

5）数组长度

//语法

arr.length

//最后一个元素的索引

arr.length-1

16.数组方法

1）indexOf

数组可以通过 indexOf(`来搜索一个指定的元素的位置，如未找到返回 -1

2）concat

concat()方法把当前的 数组 和 另一个 数组连接起来，并返回一个新的 数组

var newArr = arr1.concat(arr2,"dada");

3）push 和 pop

push()向数组的末尾添加若干元素，pop()则把 数组的最后一个元素删除掉

arr.push("a","b");

console.log(arr);

arr.pop();

console.log(arr);

//空数组继续 pop 不会报错，而是返回 undefined

4）unshift 和 shift

unshift() 向数组前面添加若干元素，shift() 则把数组的第一个元素删除掉

```
arr.unshift("a","b");
arr.shift();
```

5) Slice

slice() 截取数组的部分元素，然后返回一个新的数组

```
console.log(arr.slice(0,3)); //从索引 0 开始，到索引 3 结束，但不包括 3
console.log(arr.slice(3));  //从索引 3 开始到结束
```

如果不给`slice()`传递任何参数，就会从头到尾截取所有元素。利用这一点，可以很容易的复制一份新的数组

6) sort

sort() 可以对当前数组排序

```
var arr = ["b","c","a"];
arr.sort();
arr;//["a","b","c"]
```

7) reverse

reverse() 把整个数组的元素给掉个个

8) join

join() 方法把数组的每个元素用指定的字符串连接起来

```
var arr = ["a","b","c"];
arr.join("-"); //"a-b-c"
```

9）splice

可以从指定的索引开始删除若干元素，然后再从该位置添加若干元素

17.二维数组

```
var arr = [[1,2,3],["a","b","c"],"dadaqianduan"];
var x = arr[1][1]; //b
```

18.字符串

1）字符串属性 length-字符串的长度属性
2）slice()

slice(start[,end])，start--开始索引 end--结束索引

3）substr()

substr(start[,length])，start：开始，取 length 个字符

4）split()

split([separator[,limit]])，按条件分割字符串，返回数组

5）indexOf()

在父串中首次出现的位置，从 0 开始！没有返回-1

6）lastIndexOf()

倒序查找

7）charAt(index)

charAt(index) 指定索引的字符

8）toLowerCase()

转小写

9）toUpperCase()

转大写

正则表达式

1.创建正则表达式

```
var reg = new RegExp("a","i");
// 将匹配字母 a，第二个参数 i,表示匹配时不分大小写
```

2，元字符

a-z	英文小写字母
A-Z	英文大写字母
0-9	数字
\r \n \t	非打印字符
\d	数字，相当于 0-9

\D	\d 取反
\w	字母 数字 下划线
\W	\w 取反
\s	空白字符
\S	非空白字符
[ ]	任意匹配[ ]中单个字符
.	匹配任意字符（换行符除外）
{n}	匹配 n 次
{n,}	匹配至少 n 次
{n,m}	至少 n 次，最多 m 次
*	匹配 0 个或多个，相当于{0,}
+	匹配 1 个或多个，相当于{1,}
?	匹配 0 个或 1 个，相当于{0,1}
^	1、匹配正则开头 2、放在[^]，内容取反
$	匹配正则结尾
|	匹配 | 两侧任选其一
()	1.分组 2.子存储

3.模式修饰符

符号	说明
i	忽略大小写
g	全文查找
m	多行查找（必须与 g 一起实用,并且,当使用^和$模式时才会起作用）

4.正则方法

1）test 方法

检索字符串中指定的值。

2）exec 方法

该方法用于检索字符串中的正则表达式的匹配，该函数返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为 null。

5.支持正则的 String 方法

方法	说明
search()	第一个与正则相匹配的字符串的索引（不执行全局匹配，它将忽略标志 g）
match()	找到一个或多个正则表达式的匹配，没有找到任何匹配的文本，返回 null，否则，返回一个数组
replace()	替换与正则表达式匹配的子串
split()	把字符串分割为字符串数组

js 对象

1.定义对象

```
//使用 new 运算符
var obj = new Object();

//字面量
var obj={
    name:"dadaqianduan",
    age:12,
    sex:"男"
}
```

2.对象的数据访问

```
//用.语法
obj.name
//用[]语法
obj["name"]
```

3.JSON

json(JavaScript Object Notation)，是一种轻量级的数据交换格式。

```
var man = {
    "name":"dadaqianduan",
    "age":12,
    "sex":"男"
};
```

4.内置对象

- Object 是所有 JS 对象的超类(基类），JS 中的所有对象都是继承自 Object

对象的

- Array 数组对象
- Number 数字对象
- Boolean 布尔对象
- Error 错误对象
- Function 函数对象
- Math 数学对象
- Date 日期对象
- RegExp 对象正则表达式对象
- String 字符串对象

5.Math 方法

- abs() 绝对值 （去除正负）
- random()随机数，`0-1`之间的随机数，`1`不会出现
- round() 四舍五入
- floor(x) 下舍入(向下取整)
- ceil(x) 上舍入(向上取整)
- max(x,y) x 和 y 中的最大值
- min(x,y) x 和 y 中的最小值
- cos(x) x 的余弦
- sin(x) x 的正弦
- pow(3,4) 返回 3 的 4 次方

6.Date 方法

- getFullYear() 返回 年（4 位）
- getMouth()返回 月（0--11）
- getDate() 返回 日期

- getDay() 返回 星期 （0-6）
- getHours()返回 小时
- getMinutes() 返回 分钟
- getSeconds() 返回秒
- getTime() 返回1970年1月1日午夜到指定日期（字符串）的毫秒数
- setFullYear() 设置 年份
- setMouth()设置 月
- setDate() 设置 天
- setHours()设置小时
- setMinutes() 设置 分钟
- setSeconds() 设置 秒
- setTime() 使用毫秒的形式设置时间对象

```
//判断闰年
function runYear(year){
    if(year%4==0 && year%100!=0 || year%400==0){
        return true;
    }
};
```

面向对象是一种编程思想

1）类是一个抽象的概念

2）对象：具体的事物

3）类是对象的抽象，对象是类的具体实例

4）类不占用内存，对象占用内存空间

5）对象的访问 声明对象

6）遍历对象 - for in 循环

1.定义对象

1）字面量创建

2）工厂模式

```
// 工厂模式中的函数，首字母大写
function Cat(n,c){
    return {
        name:n,
        color:c,
        say:function(){
          alert("dadaqianduan")
        }
    }
}
```

3）构造函数

Javascript 提供了一个构造函数（Constructor）模式。

所谓"构造函数"，其实就是一个普通函数，但是内部使用了 this 变量。

对构造函数使用 new 运算符，就能生成实例，并且 this 变量会绑定在实例对象上。

构造函数首字母大写

构造函数中的 this，指向的 实例化的对象

```
function Cat(n,c){
  this.name=n;
  this.color=c;
}
```

生成实例对象

```
var cat1 = new Cat("dadaqianduan","黄色")
// 自动含有一个 constructor 属性，指向它们的构造函数
```

实例：自动含有一个`constructor`属性，指向它们的构造函数

```
alert(cat1.constructor == Cat); //true
```

4）Javascript 还提供了一个 instanceof 运算符

验证原型对象与实例对象之间的关系。

```
var txt = 'dadaqianduan';
alert(txt instanceof String); //false

var age = 123123;
alert(age instanceof Number); //false

var res = /\d/;
alert(res instanceof RegExp); //true

var arr = [];
alert(arr instanceof Array); //true
```

2.原型和原型链

构造函数都有一个 prototype 属性，指向 另一个对象 。这个对象的所有属性和方法，都会被构造函数的实例继承。

所有的函数都是 Function 的实例。

在构造函数上都有一个 原型 属性 prototype，prototype 也是一个对象；这个对象上有一个 constructor 属性，该属性指向的就是构造函数。

实例对象上有一个_proto_属性，该属性也指向原型对象，该属性不是标准属性，

不可以用在编程中，该属性用于浏览器内部使用。

Constructor

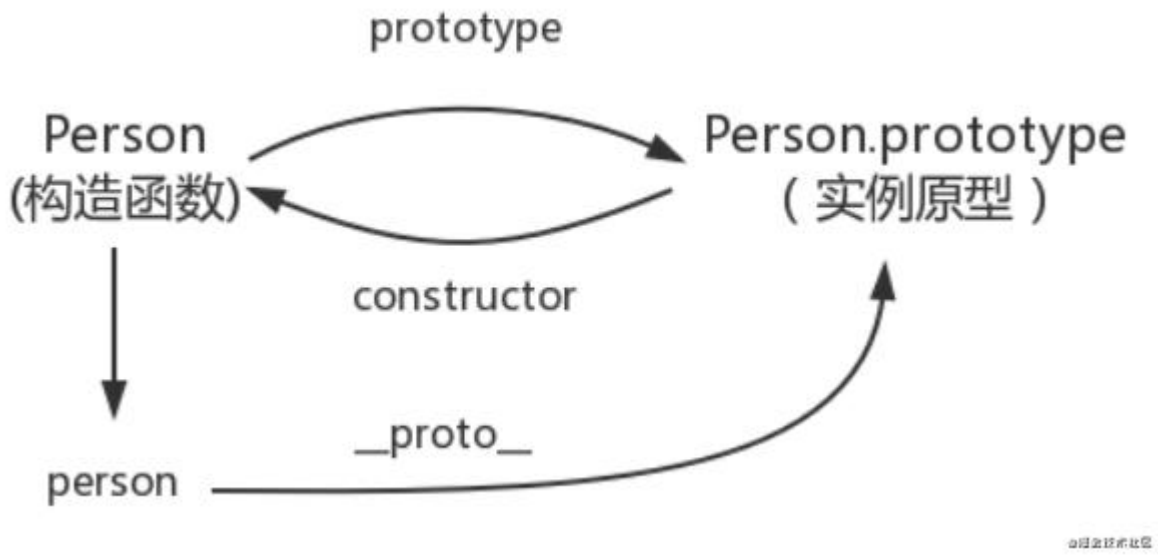
1）constructor 是构造函数 创建的实例的属性，该属性的作用是 指向 创建当前对象的 构造函数。

```
son.constructor == parent; // true
```

每个原型都有一个`constructor`属性，指向该关联的构造函数。

```
function Person() {
}
console.log(Person===Person.prototype.constructor)  //true
```

关系图：



区分一下普通对象和函数对象

```
function f1(){};
var f2 = function(){};
var f3 = new function(){};

var o1 = {};
var o2 = new Object();
var o3 = new f1();

console.log(typeof Object); //function
console.log(typeof Function);//function
console.log(typeof f1) //function
console.log(typeof f2) // function
console.log(typeof f3) //function
console.log(typeof o1) //object
console.log(typeof o2) //object
console.log(typeof o3)// object
```

1）在 JavaScript 中，原型是一个对象，原型的作用是 实现对象的继承。
2）在 JavaScript 中的所有函数对象中，都存在一个属性，prototype，该属性对应当前对象的原型。
3）所有的 JavaScript 对象，都存在一个_proto_属性，_proto_属性指向实例对象的构造函数的原型。

```
var p = new Person(); // 实例对象
console.log(p._proto_ === Person.prototype); // true
```

p 是实例对象，Person 是 p 的构造函数。p 的_proto_属性指向构造函数 Person 的原型。

js 是如何通过原型进行继承的：

```
var parent = function(name) {
 this.name = name;
}
parent.prototype.getName = function() {
 return this.name;
}
var son = new parent("dadaqianduan");
console.log(son.getName()); // dadaqianduan
```

son 继承了 parent 的原型中的函数属性 getName

3.原型链

除了 Object 的 prototype 的原型是 null 外,所有的对象和原型都有自己的原型，对象的原型指向原型对象。

在层级多的关系中，多个原型层层相连则构成了原型链。

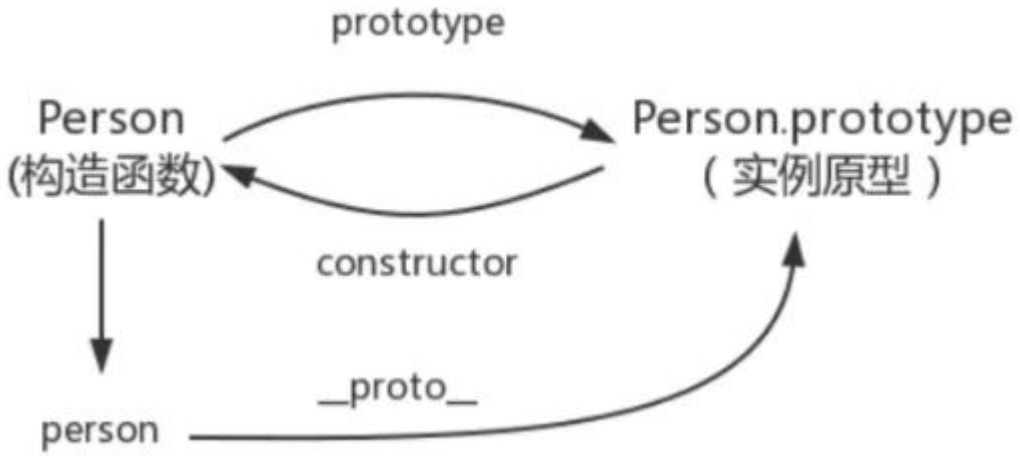
在查找一个对象的属性时，如当前对象找不到该属性，就会沿着原型链一直往上查找，直到找到为止，如果到了原型链顶端，没找到，则返回 undefined

4.原型

1）所有引用类型都有一个__proto__属性
2）所有函数都有一个 prototype 属性
3）所有引用类型的__proto__属性指向它构造函数的 prototype

构造函数和实例原型之间的关系：

Person（构造函数）的 prototype 指向 Person.prototype

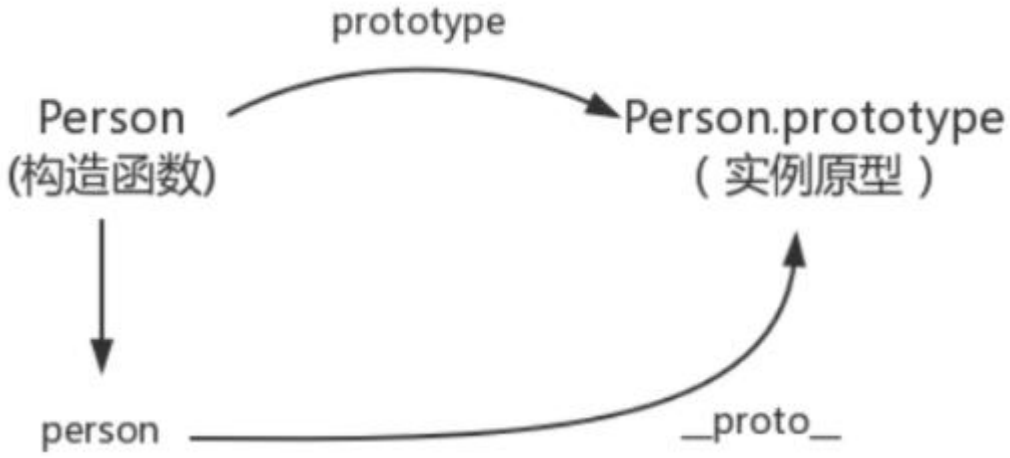


4）__proto__

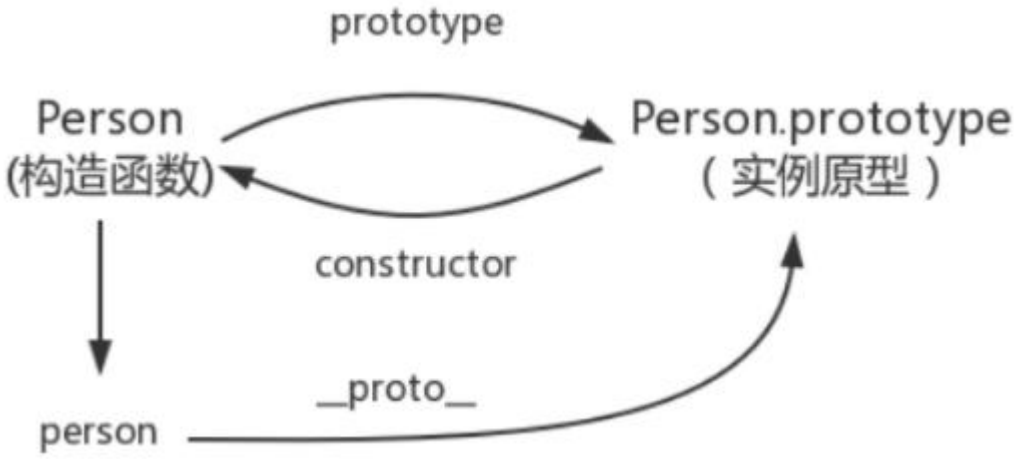
每个对象，除 null 外，都有的属性叫__proto__，这个属性会指向该对象的原型。

```
function Person() {
}
var person = new Person();
console.log(person.__proto__ === Person.prototype); // true
```

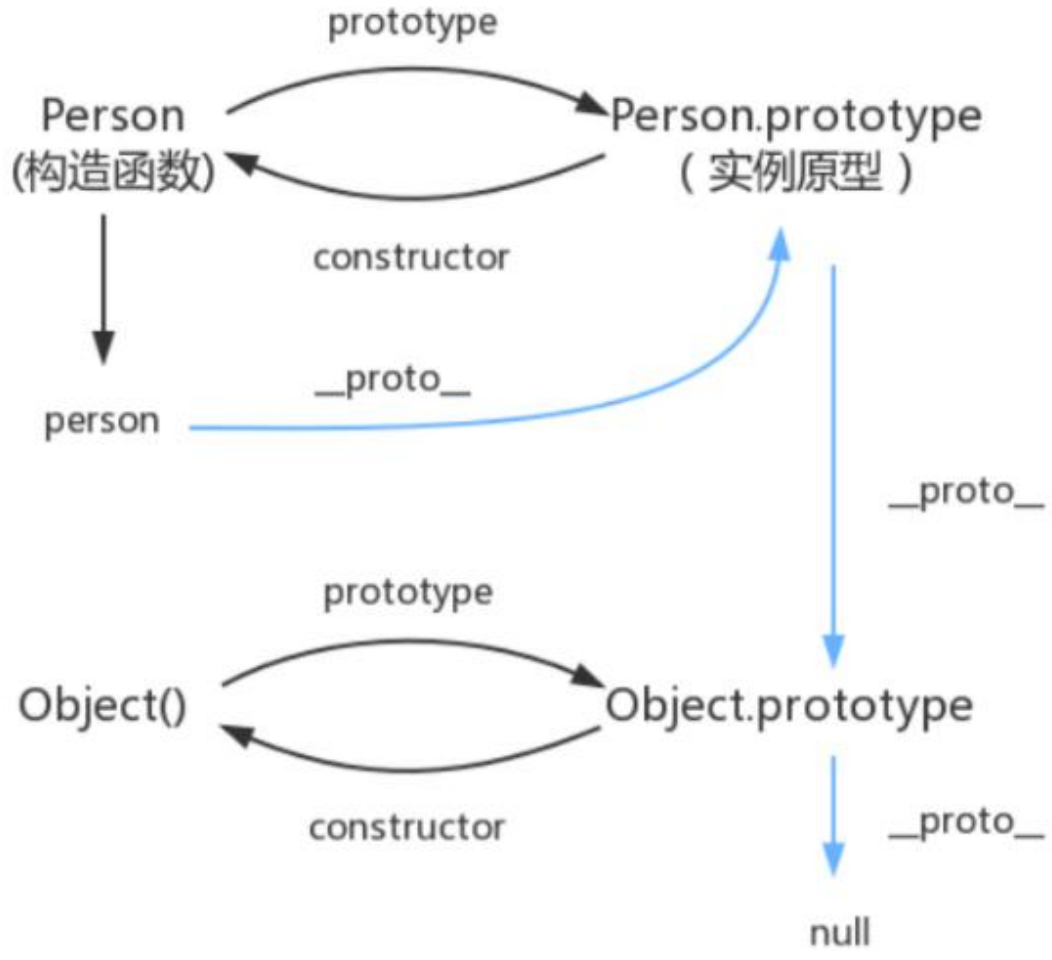
关系图：



关系图：



关系图：



>梳理：

写一个构造函数`Person`，一般构造函数区别与普通函数要求首字母大写：

```
function Person(){}
```

> prototype 原型

原型是一个对象，在原型 prototype 上定义的属性，通过“继承”，实现实例 也有这个属性。

继承是在 new 操作符内部实现的。

> 构造函数内部有个 prototype 的属性，通过这个属性就能访问到原型。

Person 是构造函数，Person.prototype 是原型。

5）实例

有构造函数，可以在原型上创建可继承的属性，通过`new`操作符创建实例：

```
function Person(){}
person = new Person()
da = person instanceof Person // 检查 person 是否是 Person 的实例
da // true
// 继承
function Person() {}
Person.prototype.name = 'dadaqianduan.cn'
person = new Person()
da = person.name // 实例继承的属性
da // 'dadaqianduan.cn'
```

6）proto

实例通过_proto_访问到原型。

```
function Person() {}
Person.prototype.name = 'dadaqianduan.cn'
```

```
person = new Person()
da = person.__proto__ === Person.prototype
da // true
```

7）constructor 构造函数

> 原型也可以通过`constructor`访问到构造函数

```
function Person() {}
Person.prototype.name = 'dadaqianduan.cn'
person = new Person
da = Person.prototype.constructor === Person
da // true
```

> 小结

（1）所有引用类型（函数，数组，对象）都拥有__proto__属性。

（2）所有函数拥有 prototype 属性。

（3）每个实例对象(Object)都有一个私有属性，为__proto__指向它的构造函数的原型对象(prototype)。该原型对象也有一个自己的原型对象__proto__，层层向上直到一个对象的原型对象为 null，null 没有原型，并作为这个原型链中的最后一个环节。

常用的 JavaScript 设计模式

百度百科：

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。

使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。 毫无疑问，设计模式于己于他人于系统都是多赢的；设计模式使代码编制

真正工程化；设计模式是软件工程的基石脉络，如同大厦的结构一样。

1.单体模式

> 单体是一个用来划分 命名空间并将一批相关的属性和方法组织在一起的对象，如果它可以被实例化，那么它只能被实例化一次。

特点：

（1）可以来划分命名空间，从而清除全局变量所带来的危险。

（2）利用分支技术来来封装浏览器之间的差异。

（3）可以把代码组织的更为一体，便于阅读和维护。

2.工厂模式

工厂模式的定义：

> 提供创建对象的接口，意思就是根据领导（调用者）的指示（参数），生产相应的产品（对象）。

1）创建一个对象常常需要复杂的过程，所以不适合在一个复杂的对象中。

2）创建对象可能会导致大量的重复代码，也可能提供不了足够级别的抽象。

工厂就是把成员对象的创建工作转交给一个外部对象，好处在于消除对象之间的耦合(也就是相互影响)。

> 分类：

简单工厂模式：使用一个类，通常为单体，来生成实例。

复杂工厂模式定义：将其成员对象的实列化推到子类中，子类可以重写父类接口方法以便创建的时候指定自己的对象类型。

父类只对创建过程中的一般性问题进行处理，这些处理会被子类继承，子类之间

是相互独立的，具体的业务逻辑会放在子类中进行编写。

> 应用场景：

以下几种情景下工厂模式特别有用：

（1）对象的构建十分复杂；

（2）需要依赖具体环境创建不同实例；

（3）处理大量具有相同属性的小对象。

3.例模式

单例模式定义了一个对象的创建过程，此对象只有一个单独的实例，并提供一个访问它的全局访问点。也可以说单例就是保证一个类只有一个实例，实现的方法一般是先判断实例存在与否，如果存在直接返回，如果不存在就创建了再返回，这就确保了一个类只有一个实例对象。

使用闭包方式来实现单例:

```
var single = (function(){
    var unique;
    function getInstance(){
        // 如果该实例存在，则直接返回，否则就对其实例化
        if( unique === undefined ){
            unique = new Construct();
        }
        return unique;
    }
    function Construct(){
        // ... 生成单例的构造函数的代码
    }
    return {
        getInstance : getInstance
    }
```

})();

> unique 是返回对象的引用，而 getInstance 是静态方法获得实例。Construct 是创建实例的构造函数。

可以通过 single.getInstance() 来获取到单例，并且每次调用均获取到同一个单例。这就是单例模式所实现的效果。

> 应用场景：
1）单例模式是一种常用的模式，有一些对象我们往往只需要一个，比如全局缓存、浏览器的 window 对象。
2）借助单例模式，可以把代码组织的更为一致，方便阅读与维护。

函数

1.函数的定义

```
// 使用关键字 function 定义函数
// 定义函数，吃饭
function dada(){
  console.log("点餐");
  console.log("拿筷子");
  console.log("吃东西");
}
```

2.局部变量和全局变量

1）变量的作用域

- ❖ 函数体内的变量：局部变量，仅在函数体内可以使用
- ❖ 函数体外的变量：全局变量，对全局可见

2）局部变量

```
function da(){
  var dadada = "dada"; //局部变量
  alert(dadada);
}
da(); //调用
alert(dadada); //报错访问不到，函数体外对 dadada 不可见
```

3）全局变量

```
var da = "我是全局变量";
function home(){
  var da = "我是局部变量";
  alert(da);
}
home();
----------------------
var a=3;  //全局
function da(){
  alert(a); //3
  var b=5;
  alert(b);//5
}
da();
console.log(b); //报错，访问不到
```

典型错误，不使用`var`声明的变量也是全局变量（不建议这样用）

```
function dada(){
    da = "123";  //全局变量
}
dada();
alert(da);
```

3.返回值

return 的含义：

```
//理解返回值
function getNum(){
 return 2; //return 的作用，将函数的结果返回给当前函数名
}
var result = getNum(); //如果希望返回值保存，就放在变量中；
console.log(result);  //2
```

return 使用方法：

- return 只能返回一个数据
- 如果函数中没有 return，则返回 undefined

return 可以用来结束一个函数

```
function Fun(){
  console.log("helloweb");
  return;
  console.log("我还会执行吗？");
```

```
}
Fun();

function fn(){
  for(var i=0;i<10;i++){
    if(i == 3){ //循环 3 次就 return
       break;
    }
    console.log("谁最帅！"); //打印 3 次
  }
  return "看 return 会不会执行我"; //return 不执行，break 执行
}
```

4.匿名函数

```
//使用函数表达式定义函数
//say 本质上是一个指向函数对象的变量，我们称为函数变量
var say = function(){
   console.log("hello");
};
say();
var oBtn = document.getElementById("box");
oBth.onclick = function(){
  alert("你点击我啦");
}
```

5.自运行函数

```
function fn(){
  //这里是代码
}
fn(); //运行 fn 函数
```

```
----------------------
var fn = function(){
  //这里是代码
}
fn(); //运行 fn 函数
--------------------
//(fn)() 等价于 fn()
(function(){
    //这里是代码
})()
```

6.闭包

闭包 (closure)有权访问另外一个函数作用域中的变量的函数。

创建闭包的常见方式有：

- 在一个函数内部创建另外一个函数，并且把这个函数 return 出去。
- 用函数为元素绑定事件，当事件发生时，还可以操作该函数中的变量。

> 特性

1）可以读取其它函数内部的变量

2）让这些变量的值始终保持在内存中

示例：

```
//方式 1----函数内部 return 一个函数
function run(){
    var a=10;
    return function(){
        a++;
        console.log(a);
    };
};

var b=run();
//alert(b);   //b 是一个函数
```

```
b();//可以访问另外一个作用域中变量的函数

//方式2--函数内部为绑定事件
function addClick(){
    var txt="abcd";
    document.getElementById('box').onclick=function(){
        alert(txt);
    }
};
addClick();

//方式3--函数内部将变量作为回调函数的参数
function play(num,fn){
    if(num>10){
        return fn && fn(num);
    };
};
var ss=play(20,function(n){
    return n+1;
});
console.log(ss);
```

BOM

1.BOM概述

BOM（browser object model）浏览器对象模型

BOM提供了独立于内容而与浏览器窗口进行交互的对象、载入页面的相关信息，其核心对象是window对象

BOM没有相关标准，每个浏览器都定义了自己的属性，并且，都有其自己对BOM的实现方式

W3C虽然没有为BOM统一制定标准,但是其中的窗口对象、导航对象等,因功能趋同,实际上已经成为默认的标准

2.window 方法

- alert()　提示框
- confirm() 带有确认 取消 按钮的提示框
- prompt()　带有可输入内容的文本域的提示框
- setInterval() 间隔定时器，可按照指定的周期（以毫秒计）来调用函数或计算表达式
- setTimeout()　超时定时器，在指定的毫秒数后调用函数
- clearInterval()　清除间隔定时器
- clearTimeout()　清除超时定时器
- requestAnimationFrame　帧定时器

3.frames [ ] 框架集

把浏览器窗口分成几个窗框，每个窗框同时取得多个`URL`地址，显示不同网页内容。

4.history 历史记录

- window.history.go(1) 前进（跳转）
- window.history.go(-1) 后退（跳转）
- window.history.forward() 前进
- window.history.back() 后退

5.location 定位

- window.location.href='http://www.baidu.com/' 页面跳转
- window.location.reload() 页面重载

6.navigator 导航

window.navigator.userAgent 浏览器类型、版本、操作系统类型、浏览器引擎类型等信息

7.screen 屏幕

- window.screen.width 返回当前屏幕宽度(分辨率值)
- window.screen.height 返回当前屏幕高度(分辨率值)

8.document 文档

window 的 document 的属性，代表所有 html 的元素，这部分是 js 主要操作的部分，因此这部分必须规范，才能进行统一开发。因此，W3C 将这部分进行了规范---DOM 标准。

DOM

DOM（document object model ）文档对象模型，定义了表示和修改文档所需的对象、行为和属性，以及这些对象之间的关系。

1.DOM 对象方法

- getElementById(id）通过 id 获取 DOM 对象（对象）
- getElementsByTagName(tag)　通过标签获取 DOM 对象（“类似数组”对象）
- getElementsByName(name) 通过 name 获取 DOM 对象（“类似数组”对象）
- getElementsByClassName(class)　通过 class 获取 DOM 对象（IE8 以下不支持）

2.操作 DOM 间的关系

- createElement(tag) 创建元素
- removeChild(对象) 删除元素
- appendChild(对象) 插入元素
- replaceChild(替换对象，被替换对象) 替换元素
- insertBefore(对象，目标对象) 前部插入

> appendChild replaceChild insertBefore 都具有移动对象的功能

节点属性:

父节点 parentNode
第一级所以子节点 childNodes
第一个子节点 firstChild
最后一个子节点 lastChild
前一个兄弟节点 previousSbiling
后一个兄弟节点 nextSibling

克隆节点:

cloneNode(布尔值)
true：复制本节点以及所有子节点
false：只复制节点本身

3.DOM 节点属性

- setAttribute("属性名",属性值) 设置属性

- getAttribute(属性名) 获取属性
- removeAttribute(属性名) 删除属性:
- hasAttributes(属性名) 判断属性是否存在（返回`ture/false`）

事件

```
var oDiv=document.getElementById('box');
oDiv.onclick=function(){
    alert('hello world');
};
```

1）addEventListener( ) 增加事件监听
2）removeEventListener( ) 删除事件监听

1.事件分类

1）window 事件
- onload 加载（某个页面或图像被完成）
- onunload 用户退出页面
- onresize 窗口或框架被调整尺寸
- onscroll 滚动条事件

2）鼠标事件
- onclick 鼠标点击
- ondblclick 鼠标双击
- onmousedown 鼠标按键按下
- onmouseup 鼠标按键被松开
- onmouseout 鼠标从某元素移开
- onmouseover 鼠标被移到某元素之上

- onmouseenter 鼠标进入某元素
- onmouseleave 鼠标离开某元素
- onmousemove 鼠标移动
- oncontextmenu 右键菜单

3）input 事件

- onblur 元素失去焦点
- onfocus 元素获得焦点。input 输入框
- onchange 内容改变时触发

4）键盘事件

- onkeydown 按键按下
- onkeypress 按键按下并释放
- onkeyup 按键释放

5）form 事件

- onreset 表单重置（重置按钮）
- onsubmit 表单提交(form 内有 text 被聚焦，直接回车可触发 onsubmit)

2.事件对象

获取事件数据，不同的事件会有不同数据

```
oDiv.onclick=function(ev){
 //ev 就是事件对象
}
```

3.事件流

事件流方向：捕获 → 事件目标→冒泡

1）事件捕获

2）事件目标

3）事件冒泡 与 阻止事件冒泡

4.事件目标

ev.targrt

ev.target.nodeName

ev.target.tagName

5.事件委派(delegate)

> 原理: 将事件绑定在父级上，利用事件冒泡原理，通过判断事件的“目标元素”来触发父级上绑定的事件

作用：

- ❖ 不用分别为子元素绑定事件
- ❖ 为未知元素绑定事件

6.事件监听

> 可以为一个元素，同时绑定多个事件

```
obj.addEventListener(事件，回调函数，冒泡/捕获）；
btn.addEventListener("click",function(ev){ //ev 事件对象
 alert('dadaqianduan');
},false)   //false 冒泡阶段
```

jQuery

一款轻量级的 js 库

丰富的 DOM 选择器

简单的事件操作

重新封装方法，让操作 DOM 属性更简单

链式操作

丰富的动画效果

Ajax 操作支持

浏览器兼容

插件扩展开发，可扩展性强

不能向后兼容

插件兼容性

多个插件冲突

1.jQuery 选择器

- id 选择器

```
$('#box')
```

- class 选择器

```
$('.box')
```

- 标记选择器

```
$('p')
```

- * 代表所有标签

2.属性选择器

- [attribute] 匹配包含给定属性的元素
- [attribute=value] 匹配给定的属性是某个特定值的元素
- [attribute!=value] 匹配给定的属性不是某个特定值的元素
- [attribute^=value] 匹配给定的属性是以某些值开始的元素
- [attribute$=value] 配给定的属性是以某些值结尾的元素
- [attribute*=value] 匹配给定的属性是以包含某些值的元素

3.位置选择器

- :first 匹配第一个元素
- :last 获取最后一个元素
- :not 去除所有与给定选择器匹配的元素
- :even 匹配所有索引值为偶数的元素，从 0 开始计数
- :odd 匹配所有索引值为奇数的元素，从 0 开始计数
- :eq 匹配一个给定索引值的元素
- :gt 匹配所有大于给定索引值的元素
- :lt 匹配所有小于给定索引值的元素

4.后代选择器

$("选择器 1 选择器 2 ……")

5.子代选择器

$("选择器 1>选择器 2>……")

6.选择器对象

- $("选择器").each(function(index){this}) 选择器对象的遍历
- $("选择器").find() 找前面选择器匹配到的元素的子元素
- $("选择器").not() 在前面选择器匹配到的元素中去除某个或某多个
- $("选择器").add() 在前面选择器中在追加节点

7.子元素

- :first-child 匹配第一个子元素
- :last-child 匹配最后一个子元素
- :first-of-type 选择所有相同的元素名称的第一个兄弟元素
- :last-of-type 选择所有相同的元素名称的最后一个兄弟元素
- :nth-child 匹配其父元素下的第 N 个子或奇偶元素
- :nth-last-child() 选择所有他们父元素的第 n 个子元素。计数从最后一个元素开始到第一个
- :nth-last-of-type() 选择的所有他们的父级元素的第 n 个子元素，计数从最后一个元素到第一个
- :nth-of-type() 选择同属于一个父元素之下，并且标签名相同的子元素中的第 n 个
- :only-child 如果某个元素是父元素中唯一的子元素，那将会被匹配
- :only-of-type 选择所有没有兄弟元素，且具有相同的元素名称的元素。

表单

- :input 匹配所有 input, textarea, select 和 button 元素
- :text 匹配所有的单行文本框
- :password 匹配所有密码框
- :radio 匹配所有单选按钮
- :checkbox 匹配所有复选框
- :submit 匹配所有提交按钮
- :image 匹配所有图像域

- :reset 匹配所有重置按钮
- :button 匹配所有按钮
- :file 匹配所有文件域
- :hidden 匹配所有隐藏域

表单对象属性

- :enabled 匹配所有可用元素
- :disabled 匹配所有禁用元素
- :checked 匹配所有选中的被选中元素
- :selected 匹配所有选中的 option 元素

8.DOM 操作

1）查找获取

- text() - 设置或返回所选元素的文本内容
- html() - 设置或返回所选元素的内容
- val() - 设置或返回表单字段的值

2）内部插入

- append() 向每个匹配的元素内部追加内容
- appendTo() 把所有匹配的元素追加到另一个指定的元素集合中
- prepend() 向每个匹配的元素内部前置内容
- prependTo() 把所有匹配的元素前置到另一个、指定的元素集合中

3）外部插入

- after() 在每个匹配的元素之后插入内容
- before() 在每个匹配的元素之前插入内容
- insertAfter() 把所有匹配的元素插入到另一个、指定的元素集合的后面
- insertBefore() 把所有匹配的元素插入到另一个、指定的元素集合的前面

4）包裹

- wrap() 把所有匹配的元素用其他元素的结构化标记包裹起来
- unwrap() 这个方法将移出元素的父元素。
- wrapAll() 将所有匹配的元素用单个元素包裹起来
- wrapInner() 将每一个匹配的元素的子内容(包括文本节点)用一个 HTML 结构包裹起来

5）替换

- replaceWith() 将所有匹配的元素替换成指定的 HTML 或 DOM 元素
- replaceAll() 用匹配的元素替换掉所有 selector 匹配到的元素

6）删除

- empty() 删除匹配的元素集合中所有的子节点
- remove() 从 DOM 中删除所有匹配的元素

7）克隆

- clone() 克隆匹配的 DOM 元素并且选中这些克隆的副本

9.JQuery 事件

页面载入

```
$(document).ready(function(){
})
//简写方式
$(function(){
})
```

事件绑定

```
$("#box").on("click",function(){
```

```
    /**/
})
$("#box").off("click,mousemove");
```

10.容器适应

获取元素的宽高有以下几种方法：

```
$(选择器).width() | innerWidth() | outerWidth()
$(选择器).height() | innerHeight() | outerHeight()
```

innerWidth()和 innerHeight()是指元素里面内容的宽高加上内边距的宽高；
outerWidth()和 outerHeight()是指元素里面内容的宽高加上内边距的宽高和边框；

> 获取窗口的宽高的方法如下：

```
$(window).width()
$(window).height()
```

11.标签样式操作

$(选择器).css (样式属性名[,值]) 方法设置或返回被选元素的一个或多个样式 属性
$(选择器).addClass(类别名) 增加类别样式
$(选择器).removeClass(类别名) 去除类别样式
$(选择器).toggleClass(类别名) 交替使用类别样式：有这个类别样式就去除，没有就追加；

12.滑动

show() 显示元素

hide() 隐藏元素

slidDown() 向下滑动显示

slideUp() 向上滑动收起隐藏

slideToggle() 交替滑动状态

fadeIn() 淡入

fadeOut() 淡出

fadeTo() 动画到指定透明度

fadeToggle() 交替淡出、淡入状态

13.自定义动画

animate() 自定义动画

stop() 停止所有在指定元素上正在运行的动画

delay() 设置一个延时来推迟执行队列中之后的项目

finish() 停止当前正在运行的动画，删除所有排队的动画，并完成匹配元素所有的动画

AJAX

$.ajax()

$.get() 通过远程 HTTP GET 请求载入信息

$.post() 通过远程 HTTP POST 请求载入信息

$.getJSON() 通过 HTTP GET 请求载入 JSON 数据

1.工作原理

> AJAX = 异步 JavaScript 和 XML

> 在浏览器中输入`url`地址请求服务器时，是通过`Ajax`发送`http`请求给服务器，服务的响应结果也是先返回给`Ajax`，先`Ajax`处理之后在返回给浏览器显示在页面。

2.XMLHttpRequest 对象

```
//第一步：
xhr = new XMLHttpRequest();

//第二步
xhr.open("post","test.php");

//第三步：
xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");

//第四步：
xhr.send("a=1&b=2");

//第五步：
xhr.onreadystatechange=function(){
    if(xhr.status==200 && xhr.readyState==4){
         var result=xhr.responseText;//获取到结果
   alert(result);
   }
```

}

3.XML 和 HTML 的区别

XML	HTML
所有的标记必须成对出现	不是所有的都需要成对出现
区分大小写	不区分大小写

4.get() 和 post()

$.get(url,data,callback,dataType)

$.post(url,data,callback,dataType)

> AJAX 工作原理

Ajax 的基本格式如下：

```
$.ajax({
    url:'请求的资源',
    type:'请求方式 get|post',
    data:'发送数据 名=值&名=值',
    dataType:'回传值的类型',
    success:function(res){ res 接收返回值
    }
})
```

HTTP

HTTP（超文本传输协议）是一种通信协议，它允许将超文本标记语言(`HTML`)文档从 Web 服务器传送到客户端的浏览器。

1.HTTP 消息结构

- request line

请求行：第一行必须是请求行，用来说明请求类型、要访问的资源以及使用的 HTTP 版本。

- header

请求头：用来说明服务器要使用的附加信息。

- blank line

空白行：请求头部与请求体之间必须有一个空白行，必不可少

- body

请求体：也叫请求正文，可以添加任意的其他数据

状态行：

Host　接受请求的服务器地址，可以是：IP：端口 或 域名

User-Agent 发送请求的应用程序名称（浏览器信息）

Connection 指定与连接相关的属性，如：Connection:Keep-Alive

Accept-Charset　客户端可以接受的编码格式

Accept-Encoding　客户端可以接受的数据压缩格式

Accept-Language　客户端可以接受的语言

referer　当前请求来自哪个链接（防盗连）

content-type　请求的文件类型

cookie 该网站相关的会话信息

2.url 请求过程

1）首先客户端与服务器需要建立连接。

2）建立连接后，客户端发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，客户端信息和可能的内容。
3）服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码（status Code 状态码），后边服务器信息、实体信息和可能的内容。
4）客户端接收完，服务器所返回的信息后，与服务器断开连接。

> 如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端。对于用户来说，这些过程是由`HTTP`自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

预加载

预加载：将所有所需的资源提前请求加载到本地，后面在需要使用就直接从缓存中存取资源

1）使用 image 对象

```
<img src="" style="display:none"/>
```

2）使用 image 对象

```
var image = new Image();
image.src="";
```

3）使用 XMLHttpRequest 对象

```
xmlhttprequest.onreadystatechange=callback;
xmlhttprequest.onprogress=progressCallback;
xmlhttprequest.open("GET",http://xx.jpg,true);
xmlhttprequest.send();
```

```
function callback(){
    if(xmlhttprequest.readyState=4 && xmlhttprequest.status==200){
        var responseText=xmlhttprequest.responseText;
    }else{
      console.log("Request was unsuccessful" + xmlhttprequest.status);
    }
}
function progressCallback(e){
    c=e||event;
    if(e.lengthComputable){
        console.log("Received"+e.loaded+"of"+e.total+"bytes");
    }
}
```

懒加载

首屏加载，技术上显示要用的技术就是图片懒加载，即到可视区域再加载。

性能优化

1.JavaScript 代码优化

1）代码与结构分离

2）样式与结构的分离

3）数据与代码分离

AMD：Asynchronous Module Definition，即异步模块加载机制。

CMD:Common Module Definition，即通用模块定义规范

导出（export）与导入（import）两个模块

2.提升文件加载速度

1）合并 JavaScript 代码，尽可能少的使用 script 标签。

2）无堵塞加载 JavaScript。

3）动态创建 script 标签来加载

webpack

webpack 是一个 module bundler（模块构建工具），由于 JavaScript 应用程序的复杂性不断增加，构建工具已成为 web 开发中不可或缺的一部分。它帮助我们去打包、编译和管理项目需要的众多资源文件和依赖库。

webpack 支持 CommonJS、AMD 和 ES6 模块系统，并且兼容多种 JS 书写规范，可以处理模块间的依赖关系，所以具有更强大的 JS 模块化的功能，它能压缩图片，对 CSS、js 文件进行语法检查、压缩、编译打包。

1.webpack 的特点

1）可以很好的用于单页应用

2）同时支持 require() 和 import 模块语法

3）允许非常前沿的 code splitting（代码分割） 特性

4）热加载可以让 React、Vue.js 和其它类似框架的本地开发更快

5）它是目前最受欢迎的构建工具

2.webpack 的缺点

1）不适合 web 开发的初学者

2）对于 CSS、图片和其它非 JS 资源文件时，需要先混淆处理

3）文档不够完善

4）变化很大，不同版本的使用方法存在较大差异

3.安装

全局安装：

```
//安装全局 webpack
npm install webpack -g

//安装全局 webpack-cli
npm install webpack-cli -g
```

4.webpack 基本应用

> SPA（single page web application） 单页应用程序，是 webpack 打包的典型应用

示例，主要的几个部分组成：

index.html 主文件

JS 文件 可能有多个 JS 文件，可通过 webpack 合并打包为一个文件

CSS 文件　可能有多个 CSS 文件，可通过 webpack 合并打包为一个文件

图片　可通过 webpack 压缩优化

示例：

```
//a.js
var run=function(){
    console.log("aaa");
};
//node CommonJS 模块
//module.exports.run=run;

//ES6 语法
export default {run};
```

```
//b.js
var play=function(arg){
    console.log(arg);
};
//node CommonJS 模块
//module.exports.play=play;

//ES6 语法
export default {play};

//index.js
//node CommonJS 引入 js 模块
//var a=require("./a.js");
//var b=require("./b.js");

//ES6 引入 js 模块
import a from "./a.js";
import b from "./b.js";

var txt = "hello world";
a.run();
b.play(txt);
```

> dist 文件夹（存放打包后的文件，可以先不创建，打包时可以自动创建）

-dis,dist,bulit

打包：

```
webpack --mode development
```

5.配置文件入门

默认的配置文件名为：webpack.config.js

> 核心概念

一个配置文件的基本结构如下：

```
//配置项
module.exports={
    //入口
    entry:".....",

    //输出配置
    output:{......},

    //模块
    module: {......},

    //解析
    resolve：{......},

    //插件（数组）
    plugins:[......],

    //开发服务器
    devServer:{......}
};
```

entry 入口 定义入口文件，默认入口文件：./src/index.js

output 输出 定义出口文件，默认出口文件：./dist/main.js

resolve 解析 路径映射、省略后缀名等

module 模块 定义不同 loader，让 webpack 能够处理非 JavaScript 模块

plugins 插件 扩展 webpack 功能

devServer 开发服务器 用于配置 webpack-dev-server 选项

设置配置文件自动完成：

```
// webpack 是基于 node 构建的，只支持 CommonJS 模块
module.exports={
```

```
    //入口配置
    entry:'./src/js/main.js',

    //出口配置
    output:{
        path:__dirname +'/dist', //输出目录  __dirname:本文件所在硬盘路径（node 全局变量）
        filename:'js/main.js' //文件名称（可以有子目录）
    }
};
```

修改 webpack.json 文件

在 webpack.json 中的"scripts"下增加：

```
"scripts": {
    "dev": "webpack --mode development",
    "build": "webpack --mode production"
  },
```

执行打包

```
npm run dev
```

entry 和 output

1）entry 入口配置 是指页面中的入口文件。也就是打包从哪个文件开始。默认入口文件：./src/index.js

2）output 出口配置 是指生成的文件输出到哪个地方去，默认出口文件：./dist/main.js，主要有以下属性：

- path　输出路径
- filename　输出文件名

示例：

```
//入口
entry: {
    index: './src/js/main.js',
},
//输出
output: {
    path: __dirname + "/dist",//打包后的文件存放的地方
    filename:"main.js"//打包后输出的文件名
},
```

1.module

webpack 只能打包 js 文件（只理解 JavaScript 语法），无法识别其他语法的文件，如果要让 webpack 打包其他文件，首先需要让 webpack 识别不同文件，这就需要特别的模块，这种模块统称为 loader。

loader 分类

- 转换编译　script-loader，babel-loader，ts-loader，coffee-loader
- 处理样式　style-loader， css-loader， less-loader， sass-loader，postcss-loader
- 处理文件　raw--loader，url-loader，file-loader
- 处理数据　csv-loader，xml-loader
- 处理模板语言　html-loader，pug-loader，jade-loader，markdown-loader
- 清理和测试 mocha-loader，eslint-loader

常用 loader

css-loader 解析 css 语句

style-loader 将 css-loader 解析后的文本，添加<style>标签

babel-loader 将 ES6+、JSX 语法转成 ES5 低版本语法

url-loader

url-loader 对未设置或者小于 limit byte 设置的图片以 base64 的格式进行转换

对于大于 limit byte 的图片用 file-loader 进行解析

file-loader

解析项目中的 url 引入（包括 img 的 src 和 background 的 url）

修改打包后文件引用路径，使之指向正确的文件

less-loader less 编译器

vue-loader

Vue 也推出了自己的 vue-loader，可以方便的打包 .vue 文件 的代码

在 vue-cli（快速构建单页应用的脚手架）中得到应用。

> css loader

```
//index.js
import a from "./a.js";
import b from "./b.js";
var txt = "hello world";
a.run();
b.play(txt);
```

//打包 css 文件
import "./style.css"; //注意：是相对路径

安装 loader（loader 也是依赖包，需要安装）

```
npm install css-loader --save-dev
npm install style-loader --save-dev
```

在 module 中定义 ccs 模块相关的配置：

```
module: {
    rules: [
        {
            test: /.css$/, //正则
            //把 css 添加到 html 的 style 标签里(style-loader 要先加载)
            loader: ['style-loader','css-loader'], //loader 或者 use
            exclude: /node_modules/, //正则 排除 node_modules 目录
        }
    ]
},
```

> babel loader
babel 是一个 js 编译器，它通过语法转换器支持最新版本的 JavaScript（包括 JSX、TypeScript 等新语法）。 这些插件允许你立刻使用新语法，无需等待浏览器支持。

使用 Babel 首先要配置 .babelrc 文件，该文件用来设置转码规则和插件（json 格式），存放在项目的根目录下。

tips：在 linux 系统中，`rc`结尾的文件通常代表运行时自动加载的文件、配置等等。

在.babelrc 配置文件中，主要是对预设(presets) 和 插件(plugins) 进行配置。.babelrc 配置文件一般为如下：

```
{
   "presets": [
       ["env",{"modules":false}] //modules 是配置项
   ],
    "plugins": [
       [ "transform-runtime",{"polyfill":false}]  //polyfill 是配置项
    ]
}
```

> 预设 对 js 最新的语法糖进行编译，并不负责转译新增的 api 和全局对象。
> 插件 控制如何转换代码，babel 默认只转换新的 js 语法，而不转换新的 API

2.plugins

插件(plugins)可以扩展 webpack 的功能，loader 不能做的处理都能交给 plugin 来做。
如：HtmlWebpackPlugin 插件简化了 HTML 文件的创建，可以通过模板文件，生成一个 HTML 文件

3.resolve

resolve（译：解析）配置`webpack`如何寻找模块对应的文件。
alias （译：别名）通过别名将原来导入路径映射成一个新的导入路径

extensions （译：扩展）数组 导入模块时，可以省略的文件后缀名

```
resolve: {
    alias: {
        "@": path.join(__dirname,"./src") //将项目根目录下的 src 目录，映射为 @
    },
    extensions:[".js", ".json"]
}
```

其他配置项示例：

devtool　是否生成以及如何生成 sourcemap

devserver 开启一个本地开发服务器

watch 监听文件变化并自动打包

watchoption　用来定制 watch 模式的选项

performance　打包后命令行如何展示性能提示，如果超过某个大小是警告还是报错

4.webpack-dev-server

> webpack-dev-server 是一个小型的 web 服务器，可以自动监视项目文件的变化，自动刷新浏览器，其 HMR（Hot Module Replacement 热模块替换）方式只替换更新的部分，而不是重载页面，大大提高了刷新效率。

需要本地安装 webpack 和 webpack-cli

```
npm install webpack --save-dev
npm install webpack-cli --save-dev
npm install webpack-dev-server --save-dev
```

webpack.config.js 配置文件：

```
let path=require("path");
//HtmlWebpackPlugin 插件
let HtmlWebpackPlugin=require('html-webpack-plugin');
let htmlPlugin=new HtmlWebpackPlugin({
    filename:"index.html", //生成的新文件
    template:__dirname+"/src/index_temp.html",  //模板文件
    minify:{ //压缩
        removeComments:true,  //删除注释
        collapseWhitespace:true  //合并空格
    },
});
//配置项
module.exports = {
    //输入
    entry:'./src/js/main.js',//主入口文件
    //输出
    output: {
        path: __dirname + "/dist", //打包后的文件存放的地方
        filename:"main.js"          //打包后输出的文件名
    },
    //模块
    module: {
        rules: [
            {
                test: /.css$/, //正则 解析 css 文件
                //把 css 添加到 html 的 style 标签里(style-loader 要先加
载)
                use: ['style-loader','css-loader'],
                exclude: /node_modules/, //正则 必须要写 exclude！！
            },
            {
               test: /\.js$/,
               use: 'babel-loader',
               exclude: /node_modules/, //正则 必须要写 exclude！！
           },
```

```
        ]
    },
    //插件
    plugins:[
        htmlPlugin
    ],
    //解析
    resolve: {
        alias: {
            "@": path.join(__dirname,"./src") //将项目根目录下的 src
目录，映射为 "@"
        },
        extensions:['.js', '.json']
    },
    //开发服务器
    devServer: {
        inline:true, //支持 dev-server 自动刷新
        port:"8080", //端口
        open:true,       //自动打开默认浏览器
    },
}
```

webpack-dev-serve.cmd 是定义在.bin 目录中的

```
"scripts": {
    "dev": "webpack-dev-server --hot",
    "build": "webpack --mode production"
}
```

5.运行

```
npm start
```

vue

1.MVC 模式

MVC 模式是移动最广泛的软件架构之一，把应用程序强制性地划分为三部分：模型（Model）、视图（View）和控制器（Controller）。

2.MVVM 模式

MVVM 模式是把 MVC 模式的 Controller 改成 ViewModel。View 的变化会自动更新 ViewModel，ViewModel 的变化也会自动同步到 View 上显示。

3.基础语法

示例：

el　　把 Vue 实例挂载到 DOM 元素上，通过 id 绑定 html 元素

data　　数据对象，Vue 实例的数据（注意：数据不要与 methods 中的方法重名）

methods　　事件对象，包含事件所要触发的函数（注意：方法名不要与 data 中的数据重名）

computed　　计算属性

watch　　监听器

directives　　自定义指令

钩子（hook）函数（8 个）　　hook（钩子）函数，不同生命周期引发的动作

路由钩子函数（3 个）　　路由组件在不同状态时触发

components　　组件容器

template　　定义模板，可以是字符串，也可以是”#“选择器

props　　用于接收来自父组件的数据

router　　路由

store　　vuex 状态

4.实例属性/方法

vm.$el Vue 实例使用的根 DOM 元素

vm.$data　Vue 的 data 配置项

vm.$options　　用于当前 Vue 实例的初始化选项

vm.$props 当前组件接收到的 props 对象

vm.$parent 父实例（如果当前实例有的话）

vm.$root　当前组件树的根 Vue 实例

vm.$children　当前实例的直接子组件

vm.$refs　原生 DOM 元素或子组件注册引用信息

vm.$slots 用来访问被插槽分发的内容

vm.$router 全局路由（vue-router 插件）

vm.$store vuex 状态对象（vuex 插件）

方法

vm.$emit() 子组件可以使用 $emit 触发父组件的自定义事件

vm.$set() Vue.set 的别名

设置对象的属性， 这个方法主要用于避开 Vue 不能检测属性被添加的限制

vm.$watch 侦听数据变化

vm.$on()　监听当前实例上的自定义事件。事件可以由 vm.$emit 触发。回调函数会接收所有传入事件触发函数的额外参数。

vm.$mount　可以使用 vm.$mount() 手动挂载（Vue 实例化时没有 el 选项）

vm.$destroy　完全销毁一个实例。清理它与其它实例的连接，解绑它的全部指令及事件监听器。

触发 beforeDestroy 和 destroyed 的钩子。

属性绑定指令

v-bind 动态改变 dom 标签上的属性
v-bind :class="" 简写 :class=""

5.生命周期

4 个阶段：创建→挂载→更新→销毁

beforeCreate　实例创建前

created　实例创建后 初始化数据（Ajax 数据请求，获取 Vuex 状态、路由切换等）

beforeMount　载入前

mounted　载入后 需要操作 DOM 时（应用第三方插件等）

beforeUpdate　更新前

updated　更新后 通过事件修改数据、v-model 引发数据变化、AJAX 异步更新数据

beforeDestroy 实例销毁前

destroyed 实例销毁后 切换路由（组件将被销毁）

6.计算属性

> 多次访问计算属性会立即返回之前的计算结果，而不必再次执行函数。

> 计算属性具有缓存

7.数组的更新检查

push()、pop()、shift()、unshift()
splice()、sort()、reverse()

变异方法，可触发视图更新

filter(), concat() 和 slice()

非变异方法，不触发视图更新
它们可以返回新数组，用新数组替换旧数组，就可以刷新视图

8.事件对象

event.type 触发的事件类型
event.target 触发事件的 HTML 元素
event.preventDefault( ) 阻止事件的默认行为
event.stopPropagation( ) 阻止事件冒泡

9.Vue 组件

1）创建组件

```
let myTemp={
template:'#temp', //模板 id
data: function(){ //必须为函数(闭包)
    return { //必须有 return，返回值为对象{}
        title:"dadaqianduan"
    }
}
```

2）注册组件

```
//在 components 配置项中注册组件
let app=new Vue({
    el:"#box",
    components：{myTemp}
});
//全局注册组件，还可以使用 Vue.component 方法（仅限全局注册，不建议使用）
Vue.component('myTemp', MyTemp）；
```

3）使用组件

```
<!--在 Vue 实例中使用组件-->
<div id='box'>
    <!--组件名如果用驼峰定义，改为短横线命名-->
    <my-temp></my-temp>
</div>
```

4）事件通信

父子组件之间的数据操作，是通过 props 属性和$emit()方法来实现的

10.路由使用

> 定义路由包括路由路径(path)、路由名称(name)、路由组件对象(component)

```
routes: [
    {
      path: '/',         // 路由路径
      name: 'home',      // 路由名称
      component: Home    // 路由组件对象
    },
    {
      path: '/users',
      name: 'Users',
      component: UserComponent
    },
    {
      path: '/about',
      name: 'about',
      component: () => import( './views/About.vue')
    }
  ]
```

动态路由

```
routes: [
  {
    path: '/users/:username/post/:postid',
    name: 'Users',
    component: UserComponent,
  }
]

/user/:username
/user/tom
{username:'tom'}

/user/:username/post/:postId
/user/tom/post/3
{username:'tom',postId:'3'}
```

11.路由导航

1）路由导航守卫

什么是路由导航守卫可以简单理解为路由组件的生命周期回调函数。

```
// 路由导航守卫
// 作用：在第一次进入当前路由组件之前被调用
// 使用场景：获取 ajax 数据
beforeRouteEnter(to, from, next) {
    // to：表示要进入的路由组件
    // from：表示将要离开的路由组件
    // next：表示后续操作函数
    // 此时还未进入到组件中，故不能使用 this 获取当前组件的实例
    next(function(app) {
        // 进入到当前组件后，才执行的回调
        // 此时回调参数 app 表示当前组件的实例对象
        axios.get('/users/' + app.id).then(res => {
            app.user = res.data.data;
        });
    });
}
beforeRouteUpdate(to, from, next) {
    // 此时，可以使用 this 表示当前组件对象
    const app = this;
    // 发送 ajax 请求
    // this 表示切换前的状态
    // to 表示要切换到的路由对象 route
    axios.get('/users/' + to.params.id).then(res => {
        app.user = res.data.data;
    });
```

```
    // 执行后续
    next();
}
```

2）编程式路由导航

3）

```
methods: {
    login(){
        if(登陆成功){
          //实现页面跳转
          this.$router.push('/');
        }
    }
}
```

3）push()

> 跳转到指定的路由地址， 并把当前地址写入到 history 中，参数可以是字符串路径或描述地址信息的对象

字符串 router.push('home')

对象 router.push({path:'home'})

命名的路由 router.push({name:user,params:{userId:1}})

4）replace()：跳转到指定路由，它不会向 history 添加新记录，而是替换掉当前的 history 记录。

5）全局路由导航守卫

示例：

```
// 全局路由导航守卫
    router.beforeEach((to, from, next) => {
});
```

12.嵌套路由

```
children: [
    {
        path: "",
        component: 路由名
    },
    {
        path: "路径名",
        component: 路由名
    }
]
```

13.命名视图

> 使用<router-view> 可以使用 name 属性为其设置名称,即命名路由的视图简称命名视图。

示例：

```
<router-view/>
<router-view name="content"></router-view>
import About from './views/About.vue';
routes: [
{
  path: "/about",
  name: "about",
  components: {
    default: About,
    content: UserComponent
  }
}
]
```