###课程整体目标?

培养掌握前端核心技术并且了解后端,具有全栈思维的程序员-全栈工程师

> 1. 课程以前端为主,占到学习比例的70%左右,后端在25%左右,就业指导在5%左右

>

> 2. 课程面向对象为零基础学员,所以整个课程是一个由浅入深循序渐进的过程

>

> 3. 前端是每一个学员必须要掌握的,后端是每一个学员要了解,但对于有能力的学员,可以深入学习,我们会提供相应的指导

>

> 4. 全栈工程师符合二八定律(帕累托法则),就是能够了解100的技能，但精通其中的20%，并用这常用的20%技能，去实现80%的特效。

###全栈工程师主要学什么?

####软件(辅助开发的工具)

\* 浏览器(运行前端代码的环境)

>1.常见浏览有:Chrome/IE/Safari/Opera/FireFox

>

>2.浏览器统计网站: [国外浏览器统计](http://gs.statcounter.com/) [国内浏览器统计网站](https://tongji.baidu.com/data/browser)

>

>3.推荐安装使用Chrome.

>

>4.浏览器的内核

>> 1.不同的浏览器有不同的浏览器内核,内核包含了浏览器的渲染引擎,而渲染引擎决定了浏览器是如何显示网页的内容，从而导致了浏览器兼容性问题出现

>>

>> 2.主要内核

>>> 1. IE内核 Trident

>>> 2. 谷歌内核 WebKit / Blink

>>> 3. 火狐内核 Gecko

>>> 4. Safarri内核 WebKit

>>> 5. 欧朋内核 Presto

\* 编辑器(写代码的工具)

>notepad/DW/HB/VIM/Sublime....

\* Git(分布式版本控制工具)

\* PhotoShop(测量,切图,修图)

\* 常用电脑操作

> 1.英文字符和符号的输入

>

> 2.软件的下载和安装

>

> 3.了解扩展名(扩展名是用来标示文件格式的一种标志,作用是让系统决定当用户想打开这个文件的时候用哪种软件运行)

>

> 4.查看文件的扩展名(Windows用户:文件夹-->查看-->将文件扩展名的选项勾选上)

>

> 5.了解什么是纯文本(能够被记事本打开, 并且能够正常显示的文件都是纯文本文件)

>

> 6.了解电脑上的一个文件是可以同时被多个软件打开的,不同的软件打开可能会有不同的效果

####语言(和计算机沟通的工具)

\* HTML(Hyper Text Markup Language)超文本标记语言-结构

>1.用html语言开发出来的代码文件的扩展名是.html,.html的文件可以被记事本打开, 并且能够正常显示,所以.html文件是一个纯文本文件

>

>2.解释

>>语言:和浏览器沟通的工具

>>

>>标记:标签,浏览器识别的符号

>>

>>文本:网页中包含的文字和标签当中包含的文字

>>

>>超:页面上除了文本以外的内容,例如图片,音频,视频。。。。

\* CSS(Cascading Style Sheet)层叠样式表-样式

>样式是一层一层叠加起来的

\* JavaScript 脚本语言-行为

> 1.前端是运行在浏览器中

>

> 2.后端是运行在NodeJs环境中

# 什么是Git

Git是目前世界上最先进的分布式版本控制系统,Git的作者就是Liunx的作者-Linus

# Git安装

## Windows用户

windows用户[下载](https://git-for-windows.github.io)后安装,安装后通过Git bash打开终端

## Mac用户

```

$ brew install git

```

## Linux用户

```

sudo apt-get install git

```

# 学点常用的命令

## 查询目录内容命令 ls(list)

```

ls //查看当前目录下的所有目录

```

## 改变目录 cd(change directory)

```

cd test //切换到test目录下

```

> 注意:

>

> 1. .代表当前目录

>

> 2. ..代表上一层目录

>

> 3. /代表根目录或者目录分割符

>

> 4. ~代表用户的家目录,直接输入cd可以进入这个目录

>

## 新建目录 mkdir(make directories)

```

mkdir test //在当前所在的目录下面新建一个目录(test)

```

## 查询当前目录 pwd(print working directory)

```

pwd

```

# 自报家门

```

git config --global user.name tom //你是谁

git config --global user.email 76876201@qq.com //怎么联系到你

git config user.name //查看我是谁

git config user.email //查看我的联系方式

```

# 创建版本库

```

mkdir test

cd test

git init

```

<img src="img/03-1.png" alt="">

> 注意:

>

> 1.新建的目录必须是英文

>

> 2.git为隐藏目录,不能随便更改

# 查看状态

```

git status

```

> \* 如果查看时中文显示乱码,可以添加如下配置

```

git config --global core.quotepath false

```

# 添加文件到版本库中

\* 第一步，用命令\*\*git add\*\*告诉Git，把文件添加到仓库

```

git add test.txt

```

\* 第二步，用命令\*\*git commit\*\*告诉Git，把文件提交到仓库

```

git commit -m 'add text file'

```

> 注意:

>

> 1.Git只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等

>

> 2.不要使用Windows自带的记事本编辑任何文本文件

>

> 3.文件一定要放在.git所在目录或者所在目录的子目录中

>

> 4.可以使用\*\*git add . \*\*把所有的改动添加到仓库

>

> 5.\*\*git add\*\*可以多次执行

# 工作区,版本库,暂存区

## 工作区（Working Directory）

电脑里能看到的目录

## 版本库（Repository）

隐藏目录.git

## 暂存区（Stage）

在版本库中的一片存储区域

> 注意:

>

> 1.git add命令实际上就是把要提交的修改放到暂存区（Stage）

>

> 2.git commit命令实际上就是一次性把暂存区的所有修改提交到仓库

>

> 3.git commit只负责提交暂存区的修改,在工作区修改了但是没有添加到暂存区的修改不会被提交

# 删除文件

## 方法一(推荐),通过\*\*git rm\*\*删除,然后git commit

\* 第一步,通过\*\*git rm\*\*删除

```

git rm text.txt

```

\* 第二步,通过git commit把修改提交到仓库

```

git commit -m 'delete file'

```

## 方法二,在工作区删除,然后git add 和 git commit

\* 第一步,在工作区把文件删除

\* 第二步,通过git add把修改提交到暂存区

```

git add text.txt

```

\* 第三步,通过git commit把修改提交到仓库

```

git commit -m 'delete file'

```

> 注意:git rm的方式删除其实就是把工作区的文件删除掉然后提交到暂存区

# 重命名文件

\* 第一步,通过\*\*git mv\*\*重命名

```

git mv test.txt test2.txt //把test.txt重命名为test2.txt

```

\* 第二步,通过\*\*git commit\*\*把修改提交到仓库

```

git commit -m 'rename file'

```

> 注意:在工作区手动更改文件名会被git认为是删除了一个文件然后再新建一个文件

# 撤销修改

## \*\*git checkout -- file\*\* 丢弃工作区的修改

```

git checkout -- test.txt //丢弃工作区对text.txt的修改

```

> 注意:

>

> 1. git checkout -- file 是让这个文件回到最近一次git commit或git add时的状态

>

> 2. git checkout -- file 命令中的--不能省略,省略就变成了“切换到另一个分支”

## \*\*git reset HEAD file\*\* 丢弃暂存区的修改

```

git reset HEAD test.txt

```

> 注意:

>

> 1. git reset HEAD file只是把暂存区的修改撤销掉,工作区的修改还在,可以继续通过git checkout -- file来撤销工作区的修改

>

> 2. HEAD表示当前版本

## \*\*git reset --hard HEAD或者commit\_id\*\* 版本回退

```

git reset --hard HEAD //回退到当前版本,即工作区的内容和最后一次commit的内容一致

git reset --hard 34cd34343 //回退到commit id为34cd34343的版本

```

> 注意:

>

> 1.Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是改变HEAD指针的指向

>

> 2.用HEAD表示当前版本,上一个版本就是HEAD^，上上一个版本就是HEAD^^

>

> 3.回退前,用\*\*git log\*\*可以查看提交历史，以便确定要回退到哪个版本

>

> 4.如果找不到回退前的commit id,可以用\*\*git reflog\*\*查看命令历史，以便确定哪个版本

>

> 5.--hard中间不能有空格

# 远程仓库GitHub

## 什么是GitHub

GitHub是一个基于git的免费的远程仓库

## 申请GitHub帐号

## 创建一个新的空仓库

# 同步远程仓库

## 本地有仓库,同步到GitHub上

\* 第一步,在Github上创建一个空仓库

\* 第二步,在本地仓库所在目录添加一个远程仓库

```

git remote add origin https://github.com/kuazhu/test.git

```

> 注意:

>

> 1.origin远程库的名字，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库

>

> 2.https://github.com/kuazhu/test.git 是远程库的地址,github地址分为https协议和ssh协议两种,如果没有添加ssh的公钥到github上可以使用https协议的地址

>

> 3.如果第一次使用,https的地址在推送内容到远程库上时需要输入github的用户名和密码

>

> 4.可以通过 git remote remove origin来删除远程库

>

> 5.可以通过 git remote -v 来查看远程库

\* 第三步,本地库的内容推送到远程库

```

git push -u origin master

```

> 注意:

>

> 1.用git push命令，实际上是把当前master分支推送到远程

>

> 2.由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来,下次推送时用\*\*git push\*\*就可以了

>

> 3.origin就代表了远程仓库的地址,这里也可以直接用远程仓库的地址,如:

>

> ```

> git push -u https://github.com/kuazhu/test.git master

> ```

>

## GitHub上有仓库,同步到本地

\* 第一步,在github上创建一个空仓库

\* 第二步,把github上的仓库克隆到本地

```

git clone https://github.com/kuazhu/test2.git

```

> 注意:

>

> 1.和在本地添加远程仓库一样,github地址分为https协议和ssh协议两种,如果没有添加ssh的公钥到github上可以使用https协议的地址

>

> 2.如果第一次使用,https的地址在推送内容到远程库上时需要输入github的用户名和密码

>

> 3.git clone的仓库在本地可以直接用\*\*git push\*\*推送修改到远程仓库

>

> 4.推荐使用这种方式来新建并管理自己的项目

# 使用ssh协议的仓库地址

\* 第一步,生成ssh key(如果没有的话)

```

ssh-keygen -t rsa -C '76876291@qq.com' //邮件写配置git时的邮件地址

```

> 注意:

>

> 1.会生成一个私钥文件(id\_rsa)和公钥文件(id\_rsa.pub)

>

> 2.保管好私钥文件不被泄漏

\* 第二步,把公钥文件的内容原封不动的拷贝到github上

\* 第三步,使用ssh协议的远程地址

> 注意:

>

> 1.可以在添加远程库或者从github上克隆时使用

>

> 2.使用ssh除了不需要输入密码外,由于ssh支持原生的git协议,所以速度最快

>

> 3.ssh是一种网络协议，主要用于计算机之间的加密登录

>

> 4.可以用\*\*ssh -T git@github.com\*\*测试

# 集中式vs分布式简介

# 团队协作开发

## 基本配置

\* 管理员在远程库创建一个项目仓库

\* 在项目仓库中添加开发人员的ssh公钥,以github为例,项目仓库的主页中的设置->Deploy keys->Add deploy key

\* 开发人员使用git clone把项目克隆到本地,注意此时的克隆地址需要用ssh协议,否则没有权限提交

## 分支管理

### 分支的基本概念

\* 分支就是一条提交的时间线,HEAD指针指向当前提交,准确的来说应该是HEAD指针指向当前的分支指针,而当前的分支指针才正真指向当前的提交

\* master分支是主分支,在git初始化仓库的时候会自动创建,一开始master指向最新的提交,HEAD指向master

\* 每次提交，master分支会向前移动一步,HEAD指针也随之移动

### 分支操作

#### 创建分支

```

git branch dev

```

> 该命令相当于创建了一个名字叫dev的指针,该指针指向当前的提交

#### 切换分支

```

git checkout dev

```

> 该命令相当于把HEAD指针指向叫dev的分支指针

#### 创建并切换分支

```

git checkout -b dev

```

> 该命令相当于git branch dev和git checkout dev的简写

#### 查看当前分支

```

git branch

```

> 该命令列出所有分支,\*号表示当前分支,注意此时的提交是在当前分支上进行的,并不会影响其它分支

#### 合并分支

```

git merge dev

```

> \* 该命令把指定分支合并到当前分支,执行该操作前经常会切换到master分支,意思是把当前的分支的提交合并到master分支上

>

> \* 通常在合并时git会用Fast-forward(快进模式)合并,也就是直接把master指向dev的当前提交,但是并不是每次合并都能Fast-forward

#### 删除分支

```

git branch -d dev

```

> 该命令相当于把名字叫dev的指针删除掉

#### 解决冲突

> \* 在合并分支时,如果指定分支和当前分支都有新的提交,并且同时修改了同一个文件,这时合并就可能会有冲突,Git会标记出不同分支的内容,我们必须人为的分析后手动解决冲突,然后再提交

> \* 实际项目中,master分支仅用来发布新版本,是最稳定的,平时不在上面开发测试,该分支需要时刻与远程同步

> \* 在仓库中建一个dev分支,dev分支用来开发和测试,到合适的时候再把dev分支合并到master分支上,该分支需要与远程同步

#### 查看分支合并图

```

git log --graph

```

### 团队协作中的分支

#### 抓取分支

```

git checkout -b dev origin/dev

```

> \* 从远程库clone时,默认情况下,只能看到本地的master分支,如果要在dev分支上开发,就必须创建远程origin的dev分支到本地

> \* 抓取分支前在远程库里必须有该分支

#### 推送主分支

```

git push origin master

```

> 当你从远程仓库克隆时,实际上Git自动把本地的master分支和远程的master分支对应起来了,并且,远程仓库的默认名称是origin。以后使用git push就相当于使用git push origin master

#### 推送其它分支

```

git push origin dev

```

> 该命令会本地的dev分支推送到远程库,注意如果只写git push,会推送所有关联的分支

#### 解决冲突

\* 在推送时,如果其他人已经向分支上推送了提交,而你也对同样的文件作了修改,此时就会发生冲突

\* 解决冲突的办法是先用git pull把最新的提交从远程分支上抓下来,然后,在本地合并,解决冲突,解决冲突的办法和合并分支一样,再推送

\* 如果git pull提示“no tracking information”,则说明本地分支和远程分支的链接关系没有创建,用以下命令建立链接后再用git pull

```

git branch --set-upstream-to=origin/dev dev

```

或者

```

git branch --set-upstream dev origin/dev

```

## 1.为什么用sublime?

1. 简单好用

2. 多点编辑

3. 强大的包管理

4. 深度可定制

5. 快速切换文件 Cmd+p

6. 好用的命令执行面板 Shift+Cmd+p

7. 社区活跃

## 2.安装

- 参考 <https://www.sublimetext.com/>

- Windows用户注意操作系统类型(64位还是32位)

## 3.常用操作

1. sublime的打开

- 直接双击图标

- 通过Spotlight呼出

- 通过命令行调出

2. 查看快捷键

- 菜单栏查看

- 在命令面板(Shift+Cmd+p)中模糊输入命令查看

- 在配置文件中查看

3. 文件相关操作

- Cmd+o 打开文件

- Cmd+n 新建文件

- Cmd+s 保存文件

- Cmd+p 模糊输入文件名打开文件,支持以下几种形式

- 文件名:行号

- 文件名@标志(函数名之类的)

- 文件名#搜索关键字

- 目录名 文件名

- Ctrl+- 跳转到打开文件前所在的文件

- Shift+Ctrl+- 文件跳转过了回跳

- Ctrl+Tab 打开的多个文件之间跳转

4. 文件搜索

- Cmd+f 搜索当前文件 Enter查找下一个 Shift+Enter查找上一个

- Alt+Cmd+f 搜索替换

- Shift+Cmd+f 文件夹中查找替换

- 鼠标悬停会在 类/函数/方法的上会出现其定义的链接

5. 多点选择编辑

- 多点选择,按住 Cmd 键,点击鼠标左键

- 列选择，按住 Alt 键，用鼠标左键拖动

- Cmd+d 选中当前单词，继续敲可以选中多个

6. 移动选择

- 方向键,一次移动一个字符

- Alt+方向键,一次移动一个单词

- Cmd+方向键,一次移动一行

- Shift+移动,移动的同时进行选择

7. 编辑

- Cmd+/ 注释掉/取消注释

- Ctrl+ ]/[ 缩进

- Ctrl+l 选择当前行

- Shift+Cmd+d 复制当前行

- Ctrl+z 撤销

## 4.用户数据目录

1. 用户的数据都放在一个固定的目录,操作系统不同,目录的路径也不同,一般情况下参考以下目录

- OS X: ~/Library/Application Support/Sublime Text 3

- Windows: %APPDATA%\Roaming\Sublime Text 3

2. 以我自己的Mac为例,所有的用户数据都存放在/Users/air/Library/Application Support/Sublime Text 3/目录中,以下简称用户数据目录

3. 用户自定义的相关文件存放在 用户数据目录/Packages/User/ 目录下,以下简称用户自定义目录

4. 自定义的内容一般分为Default和User

- Default是编辑器自带的的默认配置,一般情况下不允许更改

- User是用户自定义的文件,优先级高于Default,用户可以根据爱好配置

5. 如果想把自己的sublime恢复到初始状态,可以删除整个用户数据目录,建议删除前备份该目录

## 5.自定义快捷键

1. 查找精确的命令名

- Ctrl + \`(反引号)打开控制台

- 在控制台中输入 sublime.log\_commands(True)

- 再次执行操作,在控制台中就可以看到精确的命令名

2. 打开Key Bindings中的User配置(对应 用户自定义目录 下的Default (OSX).sublime-keymap 文件)

3. 复制一条配置,更改键和对应的命令名,如果是首次打开,可以从Default配置中拷贝一条配置然后修改

> 默认浏览器打开设置

```

方式一：

{ "keys": ["ctrl+b"], "command": "open\_in\_browser" },

方式二：

{ "keys": ["command+b"], "command": "side\_bar\_files\_open\_with",

"args": {

"paths": [],

"application": "/Applications/Google Chrome.app",

"extensions":".\*"

}

},

```

## 6.自定义配置

1. 打开Settings中的User配置(对应 用户自定义目录 下的Preferences.sublime-settings 文件)

2. 添加或者修改相关的配置,比如,Sublime默认是不显示行号的,如果想显示行号,可以添加

"highlight\_line": true,

## 7.包管理

1. 在Mac系统中,Sublime自带默认的包安装在 /Applications/Sublime Text.app/Contents/MacOS/Packages 中

2. 可以通过安装PackageControl来安装和管理包,安装方法参考 <https://packagecontrol.io/>

3. 安装完成后,可以在命令面板(Shift+Cmd+p)中输入Install Package来打开包安装面板

4. 在包安装面板中直接输入需要安装的包名就可以安装对应的包了

5. 安装完成后

- 会在 用户自定义目录 下生成一个PackageControl.sublime-settings文件,这个文件纪录了所有用户自己安装的包信息

- 会在 用户数据目录/Installed Packages/ 目录下生成这个包的文件

- 有的包还会在 用户数据目录/Packages/ 下或者 用户自定义目录 下生成相关的文件

6. 如果想删除某个包,在命令面板中输入Remove Package打开卸载面板直接输入需要删除的包名就可以了

## 8.常用包介绍

1. emmet html/css代码补全

- 参考 <https://docs.emmet.io/cheat-sheet/>

2. SyncedSideBar 在侧边栏中同步展开当前打开的文件

3. Markdown Preview

> 快捷键设置:{ "keys": ["alt+m"], "command": "markdown\_preview", "args": {"target": "browser", "parser":"markdown"} },

## 9.自定义代码段 Snippets

1. Tools->Developer->New Snippet 新建一个Snippet

2. 修改snippet 内容(content)/触发词(tabTrigger)/作用范围(scope)

3. 打开相关的文件,用快捷键 Alt+Cmd+p 可以查看该文件所在的作用范围

4. 保存文件到 用户自定义目录 下, 以 .sublime-snippet结尾

## 10.管理自己的配置

1. 将 用户自定义目录 下的配置文件文件推送到github上

2. 如果新安装Sublime,可以从github上把文件clone下来放到 用户自定义目录 下

3. 安装PackageControl,这样所有的包会自动安装上,所有的配置也会生效

###1. 数据请求过程

![](./05-1.png)

1. 打开浏览器

2. 输入网址

3. 回车执行

4. 浏览器发送请求

5. 接受请求并处理

6. 返回处理后的数据

7. 发送处理后的数据

8. 浏览器接收处理返回的数据并渲染

>注意:

>1. 一个HTTP请求报文由请求行（request line）、请求头部（header）、空行和请求数据4个部分

>

>2.一个HTTP响应报文由响应行、响应头、响应体组成

###2. 浏览器处理页面原理

1. 当浏览器收到请求数据后，会把网络传输过来的文件缓存到本地，然后再渲染出来给用户看

> 对于OS Chrome来说,缓存文件在:/Users/用户名/Library/Caches/Google/Chrome/Default/Cache

2. 因为本地缓存的原因，上过的网站访问速度会比较快

3. 缓存的文件根据网站的大小，所缓存的文件多少也不一样

##HTML的基本结构

```html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

</body>

</html>

```

\* <!DOCTYPE html>是DTD文档声明,作用是告诉浏览器, 这个网页是用哪一个版本的HTML规范来编写的. 浏览器就能够按照这个规范来正确的编译/解析/渲染我们的网页

> \* 声明必须是 HTML 文档的第一行

> \* 声明不是 HTML 标签,所以没有结束标签

> \* 浏览器并不是完全依赖于这个声明, 浏览器有一套自己的默认的处理机制,不写也能运行,但要求大家都写

> \* HTML5之前有2大种规范(HTML和XHTML), 每种规范中又有3种小规范(严格的,过渡的,带有框架的),HTML5之前的DTD文档声明只做了解,例如:

```html

HTML4.01:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

XHTML 1.0

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```

\* html标签的作用是告诉浏览器我是一个HTML文档

> \* 所有的标签都必须写在html标签里面

\* head标签的作用是给网站添加一些配置信息,如:指定网站的标题,引用外部的css/js文件

> \* head标签内部的内容都不会显示给用户查看

\* meta标签,元数据标签,主要用来添加网站的SEO相关的信息,浏览器适配相关的内容

> \* meta标签内容都不会显示给用户查看

> \* meta标签中指定的字符集可以理解为浏览器中的的一个翻译,负责把页面上的字符正确的显示出来

>> \* 和现实中的翻译一样,有的通晓各国语言,有的就差一点,所以字符集也有所不同的

>> \* GBK(GB2312) 仅仅会翻译汉字和一些常用外文,但他体积比较下,翻译速度快

>>

>> \* UTF-8 可以翻译世界上所有的文字,但他体积比较大,翻译速度慢

>>

>> \* 如果网站仅仅包含中文, 那么推荐使用GB2312,如果除了中文以外, 还包含了一些其它国家的语言 , 那么推荐使用UTF-8,如果懒得记就直接用UTF-8

>>

>> \* 在指定字符集时,HTML文件中指定的字符集必须和保存这个文件的字符集一致,否则会出现乱码

\* title标签,指定网站的标题

> \* title标签必须写在head标签里面

\* body标签,定义显示给用户查看的内容(文字/图片/音频/视频)

> \* body标签就相当于浏览器窗口

> \* 虽然说有时候你可能将内容写到了别的地方在网页中也能看到, 但是千万不要这么干, 一定要将需要显示的内容写在body中

> \* 一对html标签中只能有一对body标签

##HTML的语法总结:

> \* 所有标签都在<\>中

> \* 大多数标签有起始标签<\>和结束标签</\>,我们称为双标签,也有只有起始标签没有结束标签的,我们称为单标签

> \* 起始标签可以包括属性,属性分为名称和值,名称和值用等号连接,值一般包含在引号中,多个属性用空格

##HTML标签的分类

> \* 兄弟关系(并列/平级)

```html

<meta charset="UTF-8">

<title>Document</title>

```

> \* 父子关系(嵌套/上下级)

```html

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

```

##HTML5的特性

###1.兼容性

> 对于过去技术的过渡和兼容,一旦浏览器不支持H5的某项功能,备选就会悄悄的运行

###2.合理性

> 新增的标签都是从实际中提炼出来的经典,如header/footer...

###3.效率

> 用户即上帝

###4.分离

> 结构行为样式的分离,老版本中HTML的样式表现部分渐渐不被支持

###5.简化

>简化DOCTYPE,简化字符集....

###6.无插件

>canvas/video/audio....

###1. 注释

\* 语法：

```

<!--被注释的内容-->

```

\* 作用：注解代码，方便程序员之间的交流和自己今后阅读

\* 注意：注释不会在浏览器中显示，注释不能嵌套

###2. 标题标签 h1-h6(Heading)

\* 作用：指定标题

\* 注意：标题标签只有h1到h6,h1的语义最强，h6的最弱，一般一个页面只有一个h1标签,标签是用来给文字添加标题语义的,而不是改变文字的样式的

###3. 段落标签 p(Paragraph)

\* 作用：指定段落

\* 注意：会独占一行

###4. 水平分割线签 hr(horizontal rules)

\* 作用：画一条水平分割线

\* 注意：会独占一行，是单标签,另外，hr标签是用来修改样式的,所以不推荐使用.今后开发中添加水平线一般都使用CSS盒子来做

###5. 换行标签 br(break)

\* 作用：内容换行

\* 注意：不另起一个段落进行换行, 而网页中99.99%需要换行时都是因为另起了一个段落,所以应该用p来做

###b(bold)

>\* 加粗文本

###u(underline)

>\* 给文本添加下划线

###i(italic)

>\* 将文本倾斜

###s(strikethourgh)

>\* 给文本添加删除线

注意,以上标签没有任何语义,已经被HTML5废弃掉,如果一定要使用, 一般情况下都是用来作为CSS的钩子来使用

HTML新增了以下标签来代替

---

###strong

>\* 定义重要性强调的文字

###ins(inseted)

>\* 定义插入的文字

###em(emphasized)

>\* 定义强调的文字

###del(deleted)

>\* 定义被删除的文字

###新旧标签对应关系

```

strong == b

ins == u

em == i

del == s

```

###sub(subscript)

>\* 定义下标文本

###sup(superscript)

>\* 定义上标文本

###1. 无序列表 ul(unordered list) li(list item)

\* 作用：给内容添加无序列表语义，主要用在导航条，商品列表，新闻列表等场景

\* 注意：

> 1.列表中的条目不分先后顺序

>

> 2.列表不是用来添加前面的小圆点的

>

> 3.无序列表是一个标签组，由ul标签和li标签组成， ul中不能包含其他标签,li标签不能单独存在，但是li标签可以包含其他标签

###2. 定义列表 dl(definition list) , dt(definition title),dd(definition description)

\* 作用：给内容添加定义列表语义, 通过dt罗列出列表的条目,然后再通过dd给每个条目进行相应的描述，主要用在网站底部相关信息,图文混排

\* 注意：

> 1.由于dl和dt、dd是一个整体, 所以dl里面不推荐包裹其它标签

>

> 2.dd和dt和li标签一样是容器标签, 里面可以添加任意标签

###3. 有序列表 ol(ordered list) li(list item)

\* 作用：给内容添加有序列表语义，主要用排行榜

\* 注意：

> 1.列表中的条目有先后顺序之分，其他注意点和无序列表一样

>

> 2.ol的应用场景并不多，因为能用ol做的用ul都能做

###1.表格标签 table tr(table row) td(table defines)

\* 作用：给内容添加表格语义，以表格的形式来展现数据

\* 注意：

> 1.默认情况下表格的边框为0，可以通过border属性来改变

>

> 2.表格标签和列表标签一样, 它是一个组合标签, 所以table/tr/td要组合出现

>

> 3.表格曾经是网页的主角,例如:[2004年新浪网](http://2004.sina.com.cn/)

###2.表格标签的属性

\* 宽度(width)和高度(height)

> 设置对象：table td

>

> 属性值:数字+长度单位(px)

>

> 默认值:宽度是由内容自动计算出来的

>

> 如果给td标签设置widht/height属性, 会修改当前单元格的宽度和高度, 不会影响整个表格的宽度和高度

>

\* 水平对齐(align）

> 设置对象：table tr td

>

> 属性值:left/center/right

>

\* 垂直对齐(valign）

> 设置对象：tr td

>

> 属性值:top/center/bottom

>

\* 外边距(cellspacing)

> 设置对象: table

>

> 属性值:数字+长度单位(px)

>

> 外边距是单元格和单元格之间的距离,默认有2个像素的距离

\* 内边距(cellpadding)

> 设置对象: table

>

> 属性值:数字+长度单位(px)

>

> 内边距是单元格的边框和文字之间的距离,默认有1个像素的距离

>

\* 背景颜色bgcolor(background color)

> 设置对象: table/tr/td

注意：以上属性只做了解，样式用CSS来控制

###3.细线表表格

\* 通过指定外边距为0来实现细线表格是不靠谱的, 其实它是将2条线合并为了一条线

\* 细线表格的制作方式:

> 1.给table标签设置bgcolor

>

> 2.给tr标签设置bgcolor

>

> 3.给table标签设置cellspacing = "1px"

>

```html

<table bgcolor="black" cellspacing="1px">

<tr bgcolor="white">

<td bgcolor="red">1.1</td>

<td>1.2</td>

</tr>

<tr bgcolor="white">

<td>2.1</td>

<td>2.2</td>

</tr>

</table>

```

###4.完整的表格结构

\* 表头:caption

> 给整个表格设置标题,一定要嵌套在talbe标签内部才有效

\* 标题:thead tr th

> th标签会给每一列设置标题,内容会自动加粗，居中

\* 主体:tbody tr td

> tbody用来存放页面中的主体数据, 如果不写会自动加上

\* 表尾:tfoot tr td

> tfoot用来存放表格的页脚

###5.单元格合并

\* 水平方向上的单元格合并：可以给td标签添加一个colspan属性, 来指定把某一个单元格当做多个单元格来看待

\* 垂直方向上的单元格合并：可以给td标签设置一个rowspan属性, 来指定把某一个单元格当做多个单元格来看待

\* 单元格合并只能向后或者是向下合并

###6.完成案例

\* 汽车排行榜

\* 个人信息表

###1.表单的作用

收集用户信息

> 浏览器中所有的表单标签都有特殊的外观和默认的功能

###2.常用表单标签

####1.form标签

> \* 作用:创建表单用于向服务器传输数据

>

> \* 注意点:

>> \* 所有的表单内容，都要写在form标签里面

>> \* 表单的action属性用来指定提交地址,method属性用来指定提交方法,这些只有在向服务器发送数据时会用到

####2.input标签

> \* 作用:提供输入框让用户输入

>

> \* 注意点:

>> \* name属性用来指定向服务器传输数据时的名字,服务器端的程序可以根据这个名字取到用户输入的数据

>> \* type属性有很多类型的取值, 取值的不同就决定了input标签的功能和外观不同

>>> \* type="text":明文输入框,value属性用来指定默认值,value属性的值是传给服务器的值

>>> \* type="password":暗文输入框,value属性用来指定默认值,value属性的值是传给服务器的值

>>> \* type="radio":单选框

>>>> \* 单选按钮，天生是不互斥的，如果想互斥，必须要有相同的name属性

>>>> \* 可以给input标签添加一个checked属性,值也是checked,让单选框默认选中某一个

>>>> \* 如果属性的取值和属性的名称一样, 可以只写属性名称

>>>> \* value属性的值是传给服务器的值

>>> \* type='checkbox':多选框

>>>> \* 多选框可以一次选择多个值

>>>> \* 在同一个表单中用相同的name属性可以把多选框的值合并发给服务器

>>>> \* 默认值的设置和单选框一样

>>>> \* value属性的值是传给服务器的值

>>> \* type="button":普通按钮,value属性用来指定按钮的内容,配合JS完成一些操作

>>> \* type="image":定义图片按钮,src属性用来指定图片的位置,配合JS完成一些操作

>>> \* type="reset":清空表单的数据,value属性用来指定按钮的内容

>>> \* type="submit":提交表单的数据到服务器

>>>> 数据提交到远程服务器的条件

>>>>

>>>> \* form表单有action属性

>>>> \* 表单元素有name属性

>>> \* type="hidden":隐藏数据,配合提交按钮将一些数据偷偷的提交到服务器

>>> \* type="email":自动校验邮箱的合法性

>>> \* type="url":自动校验url的合法性

>>> \* type="number":只能输入数字

>>> \* type="date":通过日历来选择时间

>>> \* type="color":通过取色板来选择颜色

####3.label标签

> \* 作用:把文字和输入框进行绑定,点击文字时让对应的输入框得到焦点

>

> \* 绑定步骤:

>> \* 将文字用label标签包裹起来

>> \* 给输入框添加一个id属性

>> \* 在label标签中把for属性的值设置成输入框中的id的值就可以进行绑定了

####4.datalist标签

> \* 作用:把待选项和输入框进行绑定,在输入框中输入文字时如果匹配到待选项的内容会有提示

>

> \* 绑定步骤:

>> \* 新建datalist列表并且添加一个id

>>

>> \* 给输入框添加一个list属性,将datalist的id的值赋值给list属性即可

```html

<input type="text" list="cars">

<datalist id='cars'>

<option>待选项内容1</option>

<option>待选项内容2</option>

</datalist>

```

####5.select

> \* 作用:定义下拉列表

>

> \* 格式:

```html

<select>

<optgroup label="分组名称">

<option value='1'>列表数据1</option>

<option value='2' selected>列表数据2</option>

</optgroup>

</select>

```

> \* 注意点:

>> \* 下拉列表的内容只能选择不能输入

>> \* 给option标签添加一个selected属性来指定列表的默认值

>> \* 可以用optgroup标签把option标签包裹起来就行分组,给optgroup添加label属性来指定分组的名称

>> \* option中的value属性的值是传给服务器的值

####6.textarea标签

> \* 作用:定义一个多行输入框

> \* 注意点:

>> \* 默认情况下输入框可以无限换行

>> \* 默认情况下输入框有自己的宽度和高度

>> \* 可以通过cols和rows来指定输入框的宽度和高度, 但是虽然指定了列数和行数, 但是还是可以无限往下输入

>> \* 默认情况下输入框是可以手动拉伸的,如果要禁止手动拉伸,需要设置css为: resize: none;

####7.fieldset和legend标签

> \* fieldset标签表单添加一个边框

> \* legend标签给边框指定一个标题

> \* fieldset标签在form标签中,把需要加边框的元素包裹在其中

> \* legend标签在fieldset标签中

```html

<form action="https://www.kuazhu.com">

<fieldset>

<legend>注册界面</legend>

<p>

用户名: <input type="text" name="username">

</p>

</fieldset>

</form>

```

###图像标签img(image)

> \* 作用:告诉浏览器显示一张图片

> \* 注意点:

>> \* img标签中的src(source)属性是指定图片的地址的

>> \* img标签不会独占一行

>> \* 可以手动指定img标签的宽度(width)属性和高度(height)属性,但有可能会变形

>> \* 想指定宽度和高度,又不想让图片变形,可以只指定宽度和高度其中的一个值,系统会自动计算出另外一个的值, 并且都是等比拉伸的

>> \* 如果img标签没有指定宽高, 那么系统会按照图片默认的宽高来显示

>> \* title属性是当鼠标悬停在图片上时, 显示的提示内容

>> \* alt(alternate)属性是用于告诉浏览器当图片找不到时显示什么内容

###视频标签video

> \* 作用:播放视频

> \* 格式1:

```html

<video src="">

</video>

```

> \* 格式2:

```html

<video>

<source src="" type=""></source>

<source src="" type=""></source>

</video>

```

> \* video标签的属性介绍:

>> \* src: 需要播放的视频地址

>> \* autoplay: 是否需要自动播放视频

>> \* controls: 是否需要显示控制条

>> \* poster: 视频没有播放之前显示的占位图片

>> \* loop: 一般用于做广告视频, 视频播放完毕之后是否需要循环播放

>> \* preload: 预加载视频, 但是需要注意preload和autoplay相冲, 如果设置了autoplay属性, 那么preload属性就会失效

>> \* muted:静音

>> \* width/height: 和img标签中的一模一样

> \* 格式2说明

>> \* 格式2是为了解决浏览器适配问题,因为没有一种视频格式是所有浏览器都支持的

>> \* video支持三种视频格式, 我们可以把这三种格式都通过source标签指定给video标签, 那么以后当浏览器播放视频时它就会从这三种中选择一种自己支持的格式来播放

>> \* source标签的type属性来指定视频格式,取值有:video/webm,video/mp4,video/ogg

>> \* source标签的src属性来指定视频某一个视频的地址

>> \* 不管是格式1还是格式2都需要浏览器支持HTML5,如果不支持,需要引入相应的框架,如:[html5media](https://html5media.info/)

> \* 视频格式和浏览器支持情况

![](./008-01.png)

>> \* Ogg = 带有 Theora 视频编码和 Vorbis 音频编码的 Ogg 文件

>> \* MPEG4 = 带有 H.264 视频编码和 AAC 音频编码的 MPEG 4 文件

>> \* WebM = 带有 VP8 视频编码和 Vorbis 音频编码的 WebM 文件

###音频标签audio

> \* 作用: 播放音频

> \* 格式1:

```html

<audio src="">

</audio>

```

> \* 格式2:

```html

<audio>

<source src="" type="">

</audio>

```

> \* 注意点:

>> \* audio标签的使用和video标签的使用基本一样,video中能够使用的属性在audio标签中大部分都能够使用, 并且功能都一样,只不过有3个属性不能用, height/width/poster

>> \* 格式2中的source标签的type属性来指定视频格式,取值有:audio/ogg,audio/mpeg

> \* 音频格式的浏览器支持情况

![](./008-02.png)

###路径问题

> \* 相对路径:从执行文件(.html)所在的文件夹开始查找

>> \* 同级：资源文件和执行文件在同一个文件夹下

>>> 用法：直接写资源文件的名字 或者 ./资源文件的名字,./表示当前文件夹

>> \* 下级：资源文件存储在执行文件所在文件夹的子文件夹中

>>> 用法：从执行文件所在文件夹开始一层一层的写

>> \* 上级：资源文件存储在执行文件所在文件夹的父文件夹中

>>> 用法：../代表找到当前文件夹的上一级文件夹

> \* 绝对路径:windows从指定的盘符开始查找，linux从根目录开始查找

> \* 注意点:

>> \* 路径中不要出现中文, 否则可能出现未知问题

>> \* 如果是通过"相对路径"来指定资源文件, 不能跨盘符

>> \* 尽量使用相对路径,这样易于移植项目

> \* 路径练习,在test.html中显示所有的照片

```

└── 007-pathTest

├── 5.jpg

├── a

│   └── a1

│   └── 1.jpeg

└── b

├── 2.jpeg

└── b1

├── 3.jpg

├── b11

│   └── 4.jpg

└── test.html

```

###a标签(anchor)

> \* 链接到另一个页面

> \* a标签之间一定要有内容, 如果没有, 那么在页面上找不到这个标签,内容可以是文字也可以图片

> \* a标签必须有一个href(hypertext reference)属性,用来指定需要跳转的目标地址

> \* a标签的target属性来指定打开的方式,有\_blank(新页面打开), \_self(当前页面打开),默认就是\_self

> \* a标签的title属性来指定鼠标移动到超链接上时显示的信息

###假链接

> \* 什么是假链接?

>> 点击之后不会跳转的链接

> \* 为什么会有假链接?

>> 在项目开发前期, 其它页面都没有写出来, 那么我们就不知道应该跳转到什么地方, 所以就只能使用假链接来代替. 当项目后期其页面都已经完成时再将假链接体会为真链接

> \* 假链接的格式:

```html

<a href="#">假链接1</a>

<a href="javascript:">假链接2</a>

```

> \* 两者之间的区别

>> \#的假链接会自动回到网页的顶部, 而javascript:的假链接不会自动回到网页顶部

###a标签跳转到指定的位置

> \* a标签跳转到指定的位置的步骤

>> \* 第一步,给目标位置的标签添加一个id属性, 然后指定一个唯一值

>> \* 第二步,把a标签的href属性设置为 '#目标标签的id值'

```

<h2 id="top">我是顶部</h2>

<a href="#top">回到顶部</a>

```

> \* a标签跳转到指定的位置, 是没有过度动画的

> \* a标签还可以跳转到其他页面的指定位置,做法就是把href属性设置为'其他页面.html#其他页面中目标标签的id值'

```html

<a href="其他页面.html#bottom" target="\_blank">跳转到其他页面的底部</a>

//其他页面.html中

<h2 id="bottom">其他页面的底部</h2>

```

###base标签

> \* 为页面上的所有链接规定默认地址或默认打开的方式

> \* base标签必须写在head标签中

> \* base标签没有结束标签

> \* 如果既在base中指定了target又在a标签中指定了target,那么浏览器会按照a标签中指定的来执行

> \* 如果在base中指定了默认地址,a标签的地址和img标签的地址都会以base标签的地址为相对地址

###div(division)

>\* 无实际语义,主要用来布局的

###span

>\* 无实际语义,主要用来布局的

---

以下标签是H5中新增的标签

###header

>\* 代表页面头部内容

###nav(navigator)

>\* 页面的导航部分

###section

>\* 代表一个区域,里面可以有其他的标签,如h标签...

###article

>\* 页面当中的核心内容文章

###aside

>\* 页面文章的辅助信息,有时也代表页面的侧边栏

###footer

>\* 代表页面底部内容

\*\*\*\*

###meter

>\* 特定范围的数值

>\* 注意点

>> \* max属性,定义最大值。默认值是 1。

>> \* min属性,定义最小值。默认值是 0。

>> \* value属性,当前取值

###time

>\* 添加时间语义

>\* 注意点

>> \* datetime属性,规定日期 / 时间。否则，由元素的内容给定日期 / 时间。

>> \* pubdate属性,指定时间为发布日期

###progress

>\* 表示进度条

>\* 注意点

>> \* max属性,总共完成的值

>> \* value属性,当前值

注意:以上标签主要用来网页的布局,需要配合css的使用

###什么是字符实体?

在HTML中有的字符是被HTML保留的, 有的HTML字符在HTML中是有特殊含义的, 是不能在浏览器中直接显示出来的, 那么这些东西要想显示出来就必须通过字符实体

> \* 常见字符实体

```

&nbsp;//non-breaking space,不间断空格,有多少个&nbsp;就有多少个空格

&lt;//less than,小于符号 <

&gt;//greater than,大于符号 >

&copy; 版权符号

```

> \* 在HTML中对空格/回车/tab不敏感, 会把多个空格/回车/tab当做一个空格来处理,如果想显示多个空格,就可以写多个&nbsp;

> \* [字符实体参考](<http://www.w3school.com.cn/html/html_entities.asp>)

## 元信息标签meta(metadata)

> \* 作用: 主要用来添加网站的SEO相关的信息,浏览器适配相关的内容和添加服务器发送到浏览器的http头部内容

> \* 注意点

>> \* 标签永远位于 head 元素内部。

>> \* 数据总是以 名称/值 的形式成对出现的。

> \* 属性说明

>> \* name属性,提供 名称/值中的名称,HTML没有指定任何预先定义的名称。通常情况下，开发者可以自由使用有意义的名称,而大多数情况下这个名称有浏览器的开发者指定,常见的名称有:Keywords,Descriiption,Author,Render等

>> \* http-equiv属性,提供 名称/值中的名称。用来告诉服务器在发送实际的文档之前先发送给浏览器包含名称/值的文档头部。通常情况下,服务器会发送很多这种名称/值,但是只有浏览器可以接受并能以适当的方式使用它们时，这些字段才有意义。常见的有Content-Type

>> \* content属性,提供 名称/值中的值,在有http-equiv或name属性的时候，一定要有content属性。该值可以是任何有效的字符串。

## 常见元信息设置举例

###keywords

>\* 作用: 告诉搜索引擎当前网页的关键词, 在SEO中非常重要, 能够提高搜索命中率,让别人更容易找到你,搜索引擎会用这些关键字对文档进行分类。

>\* 格式:

```html

<meta name="keywords" content="技术学习" />

```

###descriiption

> \* 作用: 告诉搜索引擎当前网页的主要内容,在SEO中非常重要,当别人在搜索引擎中搜索到你的网站时会自动作为对你网站的描述信息展示给用户, 能够提高搜索命中率, 让别人更容易找到你

>\* 格式:

```html

<meta name="description" content="这是一个技术学习网站" />

```

###author

> \* 作用: 告诉搜索引擎当前网页的作者

> \* 格式:

```html

<meta name="author" content="Tom chen" />

```

###renderer

> \* 作用: 告诉浏览器，用什么内核进行解析，当然前提是浏览器有相应的内核才可以,一般需要参考浏览器的开发文档,例如[360浏览器内核说明](http://se.360.cn/v6/help/meta.html)

> \* 格式:

```html

<meta name="renderer" content="webkit">

```

###Content-Type

>\* 作用:告诉浏览器准备接受一个什么类型的文档,常见的有text/html表示html文本文档

>\* 格式:

```html

<meta http-equiv="Content-Type" content="text/html">

```

###Refresh

>\* 作用:用http头部内容控制页面重定向

>\* 格式:

```html

<meta http-equiv="Refresh" content="5;url=https://www.kuazhu.com" />

```

###1.什么是属性？

其是属性生活中无处不在，就是通过属性来描述物体的。HTML当中的属性就是用来描述HTML标签的。

###2.属性的使用格式？

```html

<img src="" alt="">

```

> \* 写在HTML标签的起始标签里面

> \* 属性是用key＝value的形式，其中key就是属性名称，＝号是赋值的意思，value就是需要赋的值

> \* value可以用双引号，单引号，或者什么都不用，但是强烈推荐使用引号

> \* 一组属性是用空格来隔开的

> \* 通常value的值是有规定的，有的是规定了值，有的是规定了写法

###3.属性的分类

> \* 全局属性,所有的HTML标签都具备的属性，常见的有

>> class, id, style, title,contenteditable

> \* 局部属性,某个HTML标签自己所具备的属性,局部属性主要是根据标签的作用而不同，比如a标签的href属性

> \* 事件属性,浏览器中触发动作的能力，比如当用户点击元素时执行javascript代码

### 1. 行内样式

> \* 用法: 把样式写在标签的style属性里面

> \* 缺点: 重复代码多,维护困难

### 2. 内部样式

> \* 把样式写在style标签里面

> \* 缺点: 只适用于当前页面

> \* 注意:

>> \* style标签必须写在head标签里面

>> \* style标签中的type属性可以不用写, 默认就是type="text/css"

>> \* style标签中的样式设置有固定格式：

![](./002-01.gif)

> \* CSS的语法

>> \* CSS有选择器和声明(一条或者多条)两部分组成

>> \* 花括号前面的是选择器,花括号里面的是声明

>> \* 每条声明由一个属性和一个值组成,属性和值用冒号分开

>> \* 每条声明用分号分开

>> \* 属性不同,值的设置方式也会有所不同

### 3. 外部样式

> \* 把css写在一个外部文件里面，这个外部文件用.css作为扩展名，在需要使用的页面通过link标签引用即可。

> \* 注意:

>> \* link标签要写在head标签内部

>> \* css文件中不用写style标签

>> \* 这种方式也是项目中使用最多的一种方式

### 4. 导入样式

> \* 通过@import将外部样式文件导入进来

```

@import "style.css";

```

> \* 导入样式是css2.1增加的内容，存在兼容性问题，实际项目中并不常用。

###1.文字的样式

> \* font-style: italic;

> \* 取值：italic(倾斜的)/normal(正常的),默认是normal

###2.文字的加粗

> \* font-weight: bold;

> \* 取值：bold(加粗)／bolder(比加粗还要粗)/lighter(细线)/100-900之间整百的数字,默认是lighter

###3.文字的大小

> \* font-size: 30px;

> \* 取值：值＋单位(px)

###4.设置字体

> \* font-family: STSong;

> \* 取值：字体的名字,如:STSong,Arial,STHeiti

> \* 注意:

>> \* 字体可以设置备选,写法font-family: 字体1,字体2,字体3.....

>> \* 设置备选字体时,如果前面的字体找不到,就用后面的字体,如果都找不到就用默认字体

>> \* 设置的字体必须是用户电脑里面已经安装的字体,我们也可以使用自定义字体,自定义字体的方法单独介绍

###5.文字属性的缩写

> \* font:样式 加粗 大小 字体;

> \* 注意点：

>> \* 样式和加粗可以互换位置，并且可以省略其中一个或者都省略

>> \* 大小和字体必须写在一起，并且只能放在最后

### 颜色的取值

#### 1.用英文单词

> \* 英文单词表示颜色主要用来调试,实际开发中用的不多

> \* 17 种标准色是 aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, yellow。

#### 2.用rgb表示

> \* 什么是rgb?

>> rgb就是r(red)红 g(green)绿 b(blue)蓝的缩写，也就是咱们通常说的三原色。可以通过设置三原色的值来确定颜色。

> \* 注意点：

>> \* R,G,B的取值范围0-255,0代表不发光，255代表该颜色最亮

>> \* 三原色的颜色设置一样的情况下可以调出灰色,值越大越灰

#### 3.用rgba表示

> rgb和上述的rgb是一样的，a(alpha)就是透明度，取值从0到1，0表示全透明，1表示不透明

#### 4.十六进制表示方法

> \* 什么是十六进制？逢16位向前进一位,取值从0-9 a-f

> \* 十六进制转十进制

>> 十进制 = 十六进制的第一位\*16 + 十六进制的第二位

>>

>> 如: FF = F\\*16 + F = 15\\*16 + 15 = 240 + 15 = 255

> \* 十进制转十六进制

>> 除16取余，直到商为0为,余数倒着

> \* 格式:#RRGGBB

>> \* 两位表示一个十六进制数，RR就是红色，GG就是绿色，BB就是蓝色

>> \* 当十六进制的每一个颜色的两位是相同的情况下可以简写,例如：

>>

>> \#FF0000 ＝ \#F00;

>>

>> 注意，简写的条件是每一个颜色的两位表示的值要一样

### 1.装饰属性

> \* text-decoration: underline|overline|line-through|none;

> \* 常见的用途就是用于去掉超链接的下划线

### 2.水平对齐方式

> \* text-align: left(默认)|center|right;

### 3.首行缩进

> \* text-indent: 数字＋单位(px|em);

> \* 这个单位可以是px(像素),也可以是em(一个文字的宽度),用的最多的还是em

### 4.文本阴影

> \* text-shadow: 水平偏移距离 垂直偏移距离 模糊距离 阴影颜色;

> \* 取值：水平偏移距离 垂直偏移距离 模糊距离的取值是值加单位，例如10px,

阴影颜色就是的颜色取值

### 5.空白符处理

> \* white-space:normal|nowrap

> \* 取值说明:

>> \* normal,默认值,文字超出容器宽度会自动换行,容器会变高

>> \* nowrap,文字超出容器宽度不会换行，文本会在在同一行上溢出。

>> \* 注意浏览器会丢掉多余的空白符。换行字符（回车）会转换为空格，一行中多个空格的序列也会转换为一个空格。

### 6.文本溢出包含元素时的处理方式

> \* text-overflow:clip(剪贴)|ellipsis(显示省略号...);

> \* 注意:该属性一般需要配合其他属性一起使用,例如

```css

p{

width: 100px;

white-space:nowrap;

overflow: hidden;

text-overflow: clip;

}

```

### 7.长单词换行到下一行

> \* word-wrap: normal|break-word(单词换行);

## 什么是选择器？

选择器其实就是一套规则，规定了选择页面上元素的规则。就上{}前面的部分。

### 1.标签选择器

通过标签名称找到当前页面上所有的元素，然后给他们设置样式

\* 格式：

```

标签名称{

属性: 值;

属性: 值;

...

}

```

\* 注意点：

> \* 不管标签藏的有多深，都能找到

> \* 标签选择器不能单独选中某一个标签

> \* 所有的html标签都可以作为标签选器 p/h/table/tr/td/div/span/ul/ol/li.....

### 2.类选择器

通过给标签添加class属性，然后通过这个属性找到该元素，为它设置样式

\* 格式：

```

.类的名称{

属性: 值;

属性: 值;

...

}

```

\* 注意点：

> \* 必须以英文.开头，中间不能添加空格

> \* 类名是可以重复的，也就是说同一个类名可以添加到不同的标签上

> \* 类名的命名规范

>> \* 只能包括数字(0-9)，下划线(\_)，英文字符(a-z)

>> \* 只能以下划线和英文字符开头，不能以数字开头,一般都是以英文字符开头

>> \* 类名不能是HTML标签的名字

> \* 类(class)是所有标签都有的属性，这个属性是专门用来添加样式的

> \* 一个标签可以添加多个类,每个类中间用空格分开,格式：class="class1 class2 ...."

### 3.ID选择器

根据页面元素的ID找到该元素然后设置样式

\* 格式：

```

#ID的名称{

属性: 值;

属性: 值;

...

}

```

\* 注意点：

> \* 必须以#号开头，中间不能添加空格

> \* ID是唯一的，也就是说当前的页面上所有的ID不能重复

> \* ID的命名规范

>> \* 只能包括数字(0-9)，下划线(\_)，英文字符(a-z)

>> \* 只能以下划线和英文字符开头，不能以数字开头,一般都是以英文字符开头

>> \* ID不能是HTML标签的名字

> \* ID是所有标签都有的属性

> \* ID不是用来作为选择器添加样式的，它的主要作用是给JS使用

\* ID选择器和类选择器的区别

> \* ID是唯一的，一次只能选中一个标签，类不唯一，一次可以选中一组标签

> \* ID是以#号开头的,类是以.号开头的

> \* ID是给JS使用的,类是给CSS使用的，除非特殊情况,不要使用id去设置样式

> \* 一般情况下可以把共用的样式抽取到一个类选择器中, 然后在需要使用这个共用样式的标签上添加这个类

### 4.后代选择器

找到指定选择器的所有后代，然后给他们设置样式

\* 格式：

```

选择器1 选择器2{

属性: 值;

属性: 值;

...

}

```

> 意思就是先找到选择器1选中的元素，然后再找选择器1选中元素里面所有的选择器2选中的元素

\* 注意点

>> \* 选择器和选择器之间至少有一个空格，一般就打一个空格

>> \* 除了标签之外，还可以写其他的选择器，比如类，ID

>> \* 后代选择器可以一直写下去

>> \* 后代选择器不仅仅可以找到它的直接后代(儿子)，同时还可以找到它的孙子，重孙子。。。。。

### 5.子选择器

找到指定选择器的儿子元素，然后给他们设置样式

\* 格式：

```

选择器1>选择器2{

属性: 值;

属性: 值;

...

}

```

> 意思就是先找到选择器1选中的元素，然后再找选择器1选中的元素里面所有的儿子中被选中器2选中的元素

\* 注意点

> \* 选择器和选择器之间用>分隔

> \* 除了标签之外，还可以写其他的选择器，比如类，ID

> \* 子选择器可以一直写下去

> \* 子选择器只能找到他的儿子

\* 后代选择器和子选择器的区别

> \* 后代选择器是用空格来分隔,子选择器是用>号来分隔

> \* 后代选择器选中特定选择器的所有的后代，包括儿子，孙子...,子选择器只能选中特定选择器的儿子

\* 后代选择器和子选择器的共同点

> \* 都可以使用标签，类，id选择器

> \* 都可以根据自己的分隔符一直延续下去

\* 如何选择后代选择器和子选择器

根据应用场景，如果想选择特定选择器下面所有的后代的话就用后代选择器,如果只是想选中特定选择器下面的儿子元素的话就用子选择器

### 6.交集选择器

选中所有满足条件的集合，然后设置样式

\* 格式：

```

选择器1选择器2...{

属性1：值1；

属性2：值2；

...

}

```

\* 注意点：

> \* 每个选择器中间不能添加任何字符

> \* 选择器可以是标签，ID，类，但是标签必须放在第一个

### 7.并集选择器

找到每个选择器选中的元素，然后相加，再设置样式

\* 格式：

```

选择器1,选择器2...{

属性1：值1；

属性2：值2；

...

}

```

\* 注意点：

> \* 各个选择器之间用,号分隔

### 8.兄弟选择器

选择兄弟关系中的元素

\* 相邻兄弟选择器

```

选择器1+选择器2{

属性1：值1；

属性2：值2；

...

}

```

>\* 相邻兄弟选择器注意点

>> \* 只能选择紧邻的那个元素

>> \* 如果中间有隔开元素，就不会被选中

>> \* 各个选择器用＋号隔开

>> \* 选择器可以是标签，ID，类

\* 通用兄弟选择器

```

选择器1~选择器2{

属性1：值1；

属性2：值2；

...

}

```

>\* 通用兄弟选择器注意点

>> \* 可以选中所有的满足条件元素

>> \* 不管中间有没有隔开元素，都会被选中

>> \* 各个选择器用～号隔开

>> \* 选择器可以是标签，ID，类

### 9.序选择器

就是根据元素所在的顺序位置来选择，然后给他们添加样式

\* 同级别不区分类型

> \* 选择器:first-child 找到同级别的第一个元素，不区分类型

> \* 选择器:last-child 找到同级别的最后一个元素，不区分类型

> \* 选择器:nth-child(n) 找到同级别的第n个元素，不区分类型

>> 括号里的n除了可以写具体的数字之外，还可以写：

>>> odd 奇数

>>>

>>> even 偶数

>>>

>>> xn+y x,y是用户可以自定义，n的取值是从0到最大元素个数递增，每次增一,例如: p:nth-child(3n+0)可以选中3的倍数

> \* 选择器:nth-last-child(2) 找到同级别的倒数第n个元素，不区分类型

> \* 选择器:only-child 找到父元素的唯一子元素，不区分类型

\* 同级别同类型

> \* 选择器:first-of-type 找到同级别的第一个元素，区分类型

> \* 选择器:last-of-type 找到同级别的最后一个元素，区分类型

> \* 选择器:nth-of-type(n) 找到同级别的第n个元素，区分类型

>> 括号里的n除了可以写具体的数字之外，还可以写：

>>> odd 奇数

>>>

>>> even 偶数

>>>

>>> xn+y x,y是用户可以自定义，n的取值是从0到最大元素个数递增，每次增一,例如: p:nth-child(3n+0)可以选中3的倍数

> \* 选择器:nth-last-of-type(2) 找到同级别的倒数第n个元素，区分类型

> \* 选择器:only-of-type 找到父元素的唯一子元素，区分类型

\* 注意点

> \* 选择器可以是标签也可以是类

> \* 选择器可以是id,但是写id没有意义,因为id本身就是唯一的

> \* 选择器和冒号之间不能有空格,可以理解为这是一个交集选择器的写法

> \* 不区分类型是指所有的兄弟元素都算数,区分类型是指只有被选择器选中的兄弟元素才算数

### 10.属性选择器

根据标签设置的属性来找到需要的标签，给他们设置样式

\* 直接通过属性名称

```

[属性名称]{

属性1：值1；

属性2：值2；

...

}

```

> 找到所有具有属性名称的元素

\* 属性＝值的方式

```

[属性名称=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 找到所有具有属性名称并且值为属性值的元素,如果属性的值有.的话，需要添加引号

\* 属性的值以什么开始

```

[属性名称|=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 只能选中用“-”分隔的和独立的,是CSS2中的选择器

```

[属性名称^=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 所有以属性值开始的都可以选中,是CSS3中的选择器

\* 属性的值以什么结束

```

[属性名称$=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 所有以属性值结尾的都可以选中,是CSS3中的选择器

\* 属性的值包括什么

```

[属性名称~=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 只能选中包括用空格分隔的和独立的,是CSS2中的选择器

```

[属性名称\*=属性值]{

属性1：值1；

属性2：值2；

...

}

```

> 所有包括该属性值的都可以选中,是CSS3中的选择器

###11.通用选择器

可以选择页面上所有的元素

\* 格式：

```

\*{

属性1：值1；

属性2：值2；

...

}

```

\* 注意点：

> \* 通用选择器只做为了解，真实项目中用的不多，因为它要选择页面上所有的元素，效率比较低

### 12.伪类选择器

针对标签的状态进行选择，然后添加样式

\* a链接的4种状态

> \* 没有任何操作，默认状态

> \* 鼠标悬停在a链接上

> \* 鼠标点击时

> \* 被访问过

\* 设置方式

> \* a:link //默认状态

> \* a:visited //被访问过

> \* a:hover //鼠标悬停时

> \* a:active //鼠标点击时

\* 注意点

> \* 四种状态可以同时都写，也可以单独写

> \* 如果全写的情况下，必须遵循一定的顺序，特别是active要写到hover后面,否则就会被覆盖,因为点击的同时鼠标肯定是悬停在内容上的

> \* 一般情况下遵循爱恨原则love hate＝ lvha

> \* 默认的样式可以简写，a = a:link

> \* hover对其他元素也起作用

\* 伪类选择器补充

> \* :focus //获取焦点时添加的样式

> \* :after //指定元素的内容后面添加内容

> \* :before //指定元素的内容前面添加内容

### 1. 层叠性

当多个选择器选中同一个元素，并且设置了相同的属性时，CSS解决这种冲突的能力就叫做CSS的层叠性

### 2. 继承性

子元素具备父级元素的样式

\* 注意点

> \* 不是所有的属性都可以被继承，常见的可以被继承的属性有 color/font-/text-/line-

> \* a标签不能继承来自父元素的颜色和下划线属性

> \* h标签不能继承来自父元素的的字体大小和加粗属性,但是字体的大小会随着父级元素大小的改变而改变

> \* 一般情况下可以通过给body元素设置一些通用的样式来保持页面风格的统一

### 3. 优先级

当多个选择器同时选中某个元素时，并且设置了相同的样式，CSS会根据优先级来判断那个样式有效

\* 通过继承间接选中

> 优先级根据就近原则，就是说谁离被选中的元素近听谁的

\* 直接选中

> 当多个选择器同时选中同一个元素，并且设置相同样式时，选择器越精确优先级越高,具体根据以下优先级来判断。

>

> ID选择器>类选择器>标签选择器>通用选择器>继承间接选中>浏览器默认的样式

\* 改变优先级 !important

> \* !important必须写在需要改变优先级属性的分号;前面

> \* !不能省略

> \* !important只能提高某个属性的优先级，不能提高该选择器中其他属性的优先级

> \* 由于!important会改变选择器的优先规则,所以在实际项目中谨慎使用

\* 选择器混合使用时直接选中的优先级判断

> \* 看谁的ID选择器多，就听谁的

> \* 如果ID选择器相同，看类选器，谁的类选择器多听谁的

> \* 如果ID选择器和类选择器一样多，看标签选择器，谁的标签选择器多听谁的

> \* 如果ID选择器，类选择器和标签选择器一样多的情况下，就不继续往下找了，谁的样式写在后面听谁(就近原则)

### 盒模型介绍

所有的HMTL标签都是一个盒子，他和咱们现实中的盒子是一样的,盒模型其实是现实生活中盒子的一个比喻

<img src="008-box.png" width="400px">

\* 以戒指盒子为列,通过观察,我们发现戒指盒子有以下特性

> \* 戒指盒子和戒指盒子之间有一定的距离

> \* 戒指盒子自身有一个边框

> \* 戒指和边框之间有一定的距离

> \* 戒指自身有自己高度和宽度

#### 边框

围绕在元素四周的线条,边框有三要素,即宽度,样式,颜色

\* 边框设置

> \* 复合样式1(设置所有)

>> \* border:宽度 样式 颜色

>>> \* 颜色可以省略，默认的是黑色

>>> \* 常见样式有dotted(点状),dashed(虚线),solid(实线)

>>> \* 宽度可以省略，默认是1px

> \* 复合样式2(按属性)

>> \* border-width:上 右 下 左;

>> \* border-style:上 右 下 左;

>> \* border-color:上 右 下 左;

> \* 按照顺时针方向设置。如果省略了一个边，就取对边的值，如果只设置了一个边，那么所有的边都一样

> \* 复合样式3(按方向)

>> \* border-top: 宽度 样式 颜色;

>> \* border-right: 宽度 样式 颜色;

>> \* border-bottom: 宽度 样式 颜色;

>> \* border-left: 宽度 样式 颜色;

> \* 单一样式(一个方向上设置一个属性)

>> \* border-top-width: 宽度;

>> \* border-top-style: 样式;

>> \* border-top-color: 颜色;

>> \* border-right-width: 宽度;

>> \* border-right-style: 样式;

>> \* border-right-color: 颜色;

>> \* border-bottom-width: 宽度;

>> \* border-bottom-style: 样式;

>> \* border-bottom-color: 颜色;

>> \* border-left-width: 宽度;

>> \* border-left-style: 样式;

>> \* border-left-color: 颜色;

\* 利用边框绘制倒三角举例

```css

div{

width: 0px;

border-top: 50px solid black;

border-right: 50px solid white;

border-bottom: 50px solid white;

border-left:50px solid white;

}

```

#### 内边距

内容和边框之间的距离

\* 内边距的设置

> \* 单一样式

>> \* padding-top

>> \* padding-right

>> \* padding-bottom

>> \* padding-left

> \* 复合样式

>> \* padding:上 右 下 左;

>> \* 按照顺时针方向设置。如果省略了一个边，就取对边的值，如果只设置了一个边，那么所有的边都一样

\* 注意点

> \* 设置了padding后，元素的高度和宽度会发生变化

> \* 内边距是有背景色的，他的背景色和内容的背景色一样

#### 外边距

标签和标签之间的距离

\* 外边距的设置

> \* 单一样式

>> \* margin-top:

>> \* margin-right:

>> \* margin-bottom:

>> \* margin-left:

> \* 复合样式

>> \* margin:上 右 下 左;

>> \* 按照顺时针方向设置。如果省略了一个边，就取对边的值，如果只设置了一个边，那么所有的边都一样

\* 注意点

> \* 水平方向和垂直方向上的margin值可以叠加

> \* 普通元素垂直方向上的margin值发生重叠现象，重叠后取较大的值

> \* 普通的父子元素，给子元素设置margin top时候，这个值会传递到父元素,可以通过给父元素添加边框或者是内边距来阻止margin值的传递,还可以给父元素添加overflow:hidden来阻止margin的值传递

> \* 外边距没有背景色

> \* 外边距不会影响元素的宽和高，但是会影响整个元素空间的宽和高

> \* padding主要是用来设置父元素和子元素之间的距离，margin主要是用来设置元素和元素之间的距离

> \* margin可以取负值,表示相反的方向

#### 内容

可以真正使用的部分，由width/height来设置

\* 内容的设置

> width:

>

> height:

\* 注意点

> \* 元素的宽和高决定了真正可以填充内容的空间

> \* 取值可以是具体的像素也可以是百分比,百分比是相对于自己的父元素来说的

#### 盒子模型的计算

<img src="008-boxModel.jpeg" alt="">

\* 内容的宽和高

> \* 内容的宽:通过width属性设置的值

> \* 内容的高:通过height属性设置的值

\* 元素的宽和高

> \* 元素的宽：左边框＋左内边距＋内容的宽度＋右内边距＋右边框

> \* 元素的高：上边框＋上内边距＋内容的高度＋下内边距＋下边框

\* 元素空间的宽和高

> \* 元素空间的宽：左外边距＋左边框＋左内边距＋内容的宽度＋右内边距＋右边框＋右外边距

> \* 元素空间的高：上外边距＋上边框＋上内边距＋内容的高度＋下内边距＋下边框＋下外边距

#### 盒子模型的注意点

\* 盒子新增padding和border之后,如何保证盒子元素的宽度和高度不变

> \* 方法一,通过计算并修改元素的内容的宽和高来实现

> \* 方法二,添加box-sizing:border-box

>> \* box-sizing有两个取值,content-box(默认值),不会自动计算宽度和高度,border-box,自动计算宽度和高度

>> \* 使用border-box属性,元素的宽高 == width/height的宽高

\* 居中

居中分为水平居中和垂直居中，水平居中就是元素的左右边框离父元素的左右边框距离一样，垂直居中同理。

> \* 元素居中的方法

>> \* 方法一,设置父元素的padding值

>> \* 方法二,设置子元素的margin值

>> \* 方法三,通过设置左右边距为auto的方法让元素在父元素水平居中

>>>> \* 使用auto水平居中的前提是元素需要设置宽度

>>>> \* 垂直居中不可以通过auto的方法

> \* text-align:center和margin:0 auto的区别

>> \* text-align:center只能让内容(文本／图片)居中

>> \* margin:0 auto 是让元素自身相对于父元素水平居中

\* 样式重置

> \* 为什么要清空默认的内外边距

>> 因为HTML标签自身有默认的内外边距，这样不利于对盒模型空间的计算，为了解决这个问题，在实际的开发中需要清空默认的内外边距

> \* 样式重置的方法

>> \* 方法一,用通用选择器(不推荐使用)

>>

>> ```

>> \*{

>> margin: 0;

>> padding: 0;

>> }

>> ```

>>

>> \* 方法二,用成熟的框架,如[雅虎CSS重置代码](http://yui.yahooapis.com/3.5.0/build/cssreset/cssreset-min.css)

```css

html{color:#000;background:#FFF}body,div,dl,dt,dd,ul,ol,li,h1,h2,h3,h4,h5,h6,pre,code,form,fieldset,legend,input,textarea,p,blockquote,th,td{margin:0;padding:0}table{border-collapse:collapse;border-spacing:0}fieldset,img{border:0}address,caption,cite,code,dfn,em,strong,th,var{font-style:normal;font-weight:normal}ol,ul{list-style:none}caption,th{text-align:left}h1,h2,h3,h4,h5,h6{font-size:100%;font-weight:normal}q:before,q:after{content:''}abbr,acronym{border:0;font-variant:normal}sup{vertical-align:text-top}sub{vertical-align:text-bottom}input,textarea,select{font-family:inherit;font-size:inherit;font-weight:inherit}input,textarea,select{\*font-size:100%}legend{color:#000}#yui3-css-stamp.cssreset{display:none}

```

> \* 注意样式重置不仅仅是清空默认的边距,还有其他的默认样式

\* 行高

行高指的是每行内容的高度,可以通过设置line-height属性来改变行高

> 注意点:

> \* 文字默认的是在自身的行内垂直居中的，所以可以利用这个属性，将行高和盒子的高度设置成一样来让一行的文字垂直居中

> \* 如果盒子中有多行文字, 就不能使用设置行高等于盒子高来实现让文字垂直居中, 只能通过设置padding来让文字居中

> \* 行高和盒子高不是同一个概念,盒子高指的是元素的高度

\* 盒子阴影

给盒子添加阴影

> box-shadow: 水平距离 垂直距离 模糊距离 阴影大小 阴影颜色 内外阴影;

>

> 注意点:

> \* 背景阴影默认的颜色是盒子内容的颜色一样

> \* 快速添加阴影只需写前三个参数即可

> \* 前两个参数允许取负值

> \* 默认是外阴影(outset),可以把最后一个参数改为inset来设置为内阴影

\* 圆角边框

给边框添加圆角

\* 圆角的设置

> \* 单一样式

>> \* border-top-left-radius:取值1 [取值2];

>> \* border-top-right-radius:取值1 [取值2];

>> \* border-bottom-right-radius:取值1 [取值2];

>> \* border-bottom-left-radius:取值1 [取值2];

> \* 复合样式

>> \* border-radius:取值1(左上 右上 右下 左下)/[取值2(左上 右上 右下 左下)];

>> \* 按照顺时针方向设置。如果省略了一个点，就取对点的值，如果只设置了一个点，那么所有的点都一样

>

> \* 取值说明

>> \* 取值可以是具体的数字加单位也可以是百分比,百分比是相对于元素的宽和高的,即包含边框和内边距

>> \* 取值1代表水平方向半径,取值2代表竖直方向半径

>> \* 取值2省略的情况下，水平方向和竖直方向的半径相等

>> \* 大值特性,当值很大的时候，只会使用能够渲染的圆角大小渲染。

>> \* 圆角具有外半径和内半径之分,当radius半径值小于border的值时，没有内半径，所以内角是直角，当radius半径值大于border的值时，才有内半径，也才有内圆角。内半径的值等于外半径的值减去border的值。

>> \* 圆心是从边框的外沿垂直向内由水平和垂直的半径相交的点

>> \* 只要水平和垂直的半径是相等的，产生的就是四分之一圆。

>> \* 半径为50%,如果宽高一样,就是一个正圆,如果宽高不一样时就是一个椭圆

>> \* 圆角是设置给边框的

> \* 绘制月牙举例

```css

.moon{

width:100px;

height:100px;

border-right:20px solid #F52;

border-top-right-radius: 50%;

border-bottom-right-radius: 50%;

}

```

#### 盒子模型案例-百度新闻

<img src="008-box-demo.png" alt="">

### 设置背景颜色

\* 格式:background-color: 颜色的英文单词|rgb|rgba|十六进制

###设置背景图片

\* 格式:background-image:url(图片地址)

\* 注意点:

> \* 图片默认是在水平和垂直方向上平铺的

> \* 图片的地址可以是本地资源，也可以是网络资源

###背景图片平铺的设置

\* 格式:background-repeat: repeat-x（只在水平方向上平铺）| repeat-y（只在垂直方向上平铺）| no-repeat 水平和垂直方向上都不平铺;

###背景图片固定

\* 格式:background-attachment: scroll(背景图片会随着滚动条的滚动而滚动)|fixed(背景图片固定在某个位置,不随滚动条的滚动而滚动);

###背景图片的位置定位

\* 格式:background-position: 水平移动距离 垂直移动距离;

\* 取值:具体的数值+px|方向单词

> \* 方向单词：水平:right center left 垂直:top center bottom

> \* 移动的参考点是容器的左上点

###背景的复合样式

\* 格式:background:背景颜色 背景图片 平铺设置 背景图片固定 图片定位

\* 注意点:复合样式中的属性可以都写，也可以省略，也可以交换位置，但是图片定位的两个属性要写在一起不能拆开

###CSS精灵图的使用

由于每显示一张图片，浏览器都会请求一次服务器，所以我们可以把多张图片合成一张

，这种合成图片的技术就叫做精灵图

\* 使用CSS精灵图的方法

> \* 给元素设置一个宽高，宽高就是需要显示区域的宽和高

> \* 给元素添加背景图片

> \* 在图片中找到需要显示元素的坐标，然后根据坐标设置图片背景的位置

\* 图片标签和背景图片的区别

> \* 图片标签占用空间，背景图片不占用

> \* 图片标签的语义强，背景图片只是起到装饰作用

###设置背景图片的大小

\* 格式:background-size:宽度 高度 | 单词

\* 注意点:

> \* 宽度和高度的取值有:

>> \* 具体的像素

>> \* 百分比(相对于容器)

>> \* auto(谁设置为auto把谁等比拉伸)

> \* 单词的取值有:

>> \* cover,图片等比拉伸到宽度和高度都填满元素

>> \* contain,图片等比拉伸到宽度或高度都填满元素

###指定背景图片开始显示区域

\* 格式:background-origin:padding-box(从内边距开始显示)|border-box(从边框开始显示)|content-box(从内容区域开始显示)

\* 注意点:该属性用来指定从哪个区域开始显示背景图片,默认是padding-box

###指定背景颜色开始显示区域

\* 格式:background-clip:padding-box(从内边距开始显示)|border-box(从边框开始显示)|content-box(从内容区域开始显示)

\* 注意点:该属性用来指定从哪个区域开始显示背景颜色,默认是border-box

###背景图片练习

<img src="009-bj-demo.png" alt="">



### HTML中标签的分类

\* 容器级

能嵌套其它所有标签的标签

> div header nav section article footer h ul ol dl li dt dd a ...

\* 文本级

只能嵌套文字/图片/视频/音频/超链接的标签

> p span b u i s stong em ins del ...

### CSS中标签的分类

<img src="010-CSS显示模式.png" alt="">

\* 常见的块级元素

> p div header nav section article footer h ul ol dl li dt dd ...

\* 常见的行内元素

> span b u i s stong em ins del ...

\* 常见的行内块级元素

> img input textarea ...

### CSS元素显示模式的转换

\* 给元素添加display属性可以手动的改变元素的显示模式,常见取值有:

> \* block 块级

> \* inline 行内

> \* inline-block 行内块级

### 行内块状元素垂直对齐

\* 什么是基线

其实网页中的文字可以想象为显示在四线各种,而四线格中底部倒数第二条线就是基线

<img src="010-base\_line.jpg" alt="">

> 注意点:如果给元素设置了行高,四线格的高度还是文字的高度

\* 如何设置垂直对齐

> \* 格式:vertical-align:baseline|top|middle|bottom|text-bottom

> \* 注意点

>> \* 只有行内块状元素设置vertical-align属性才有效

>> \* 默认的对齐方式是baseline(基线对齐)

>> \* top是相对于父元素的顶部对齐

>> \* middle是相对于父元素的中部对齐

>> \* bottom是相对于是相对于父元素的底部对齐

>> \* text-bottom是相对于父元素的文字底部对齐

### 1.什么是网页布局？

对网页中各个元素进行排版

### 2.标准流的网页布局

标准流的网页布局是浏览器默认的排版方式

> \* 行内元素、行内块状元素从左到右

> \* 块状元素从上向下

> \* 流可以理解为流式排列

### 浮动流布局

给元素添加float属性,元素就属性浮动流了

> \* float的取值只有左(left)对齐和右(right)对齐，没有中间对齐,对齐是相对于父元素的左右边框的

> \* 浮动流主要是用来水平排版的

> \* 虽然把元素设置为行内块状也可以实现水平排版,但有不足之处,例如左右水平排版,如果父元素宽度不固定的情况下会发生错位,所以在项目中一般都用浮动流来实现水平排版

### 浮动元素脱标

可以想象为在我们看到标准流浏览器的上面还浮动了一层透明的浮动流浏览器,浮动元素脱标就是当给这个元素添加浮动属性后,他就会从标准流浏览器从删除,然后跑到了那层透明的浮动流浏览器中

> \* 浮动元素脱标后对标准流的影响

>> \* 由于脱离标准后,标准流中位置空闲了出来,因此后面标准流中的元素会去顶替脱离标准流的那个元素的位置

> \* 浮动元素脱标后的在浮动流中的特点

>> \* 因为脱离标准流,所以浮动的元素不区分块状，行内，行内块状，他的属性和行内块状一样

>> \* 浮动流的元素不可以用margin: 0 auto;来水平居中

> \* 浮动元素脱标后在浮动流中的排列规则

>> \* 同方向上,先浮动的显示在前面，后浮动的显示在后面,注意这个先后指的结构上的先后而不是css的书写先后

>> \* 不同方向上,左浮动找左浮动元素，右浮动找右浮动元素

>> \* 浮动后元素的位置由浮动前元素在标准流中的位置决定,这个位置主要指的是浮动前在标准流中是第几行,浮动后还是显示在第几行

>> \* 由于浮动元素的位置和浮动前在标准流的位置有光,所以严格意识上说浮动元素是非完全脱标

### 浮动元素贴靠现象

当父元素的宽度不足以撑起所有浮动的子元素的时候，会发生贴靠，即后面的元素会找他前面元素的上一个元素贴靠，如果还不够，再向上找，如果都不够的话，会贴靠父元素

### 浮动元素围绕现象

浮动的元素不能盖住没有浮动元素的文字

### 浮动布局规律

垂直方向上用标准流块状元素,水平方向上用浮动元素

### 浮动的元素无法撑起父级块状元素的高度

标准流中的块状元素的默认高度是由内容撑起来的,但是由于浮动元素脱离了标准流，所以浮动的元素是撑不起来父级块状元素的高度的

### 为什么要清除浮动

由于浮动元素撑不起盒子的高度，会让后面盒子里浮动的元素找前面盒子里浮动的元素

### 清除浮动的方式

> \* 方式一,给前面的盒子添加高度

>> \* 项目中一般高度是有内容撑起来的,尽量不设置高度

> \* 方式二,通过给第二个盒子添加clear属性来清除前面元素浮动带来的影响

>> \* clear属性的取值有:left|right|both|none(默认)

>> \* clear属性的意思是不让自己的子元素找前面浮动的元素

>> \* 通过clear属性清除浮动会导致第一个盒子的margin-bottom和第二个盒子的margin-top失效,第二个盒子的margin-top失效是因为margin的值传递

> \* 方式三,给第一个盒子和第二个盒子之间添加一个空盒子，并且给这个空盒子设置clear:both属性,俗称外墙法

>> \* 第一个盒子的margin-bottom依然不可以用,但是第二个盒子的margin-top可以用

>> \* 可以通过设置空盒子的高度来达到设置margin-bottom/top的效果

> \* 方法四,给第一个盒子所有元素的后面添加一个空盒子，并且给这个空盒子设置clear:both属性,俗称内墙法

>> \* 第一个盒子的margin-bottom可以用,第二个盒子的margin-top也可以用

>> \* 也可以通过设置空盒子的高度来达到设置margin-bottom/top的效果

>> \* 内墙法可以撑起盒子的高度，而外墙法不可以

> \* 方法五,通过伪元素选择器(:after)实现内墙法

>> \* 作用和内墙法一样，但是不用额外添加空元素，可以实现结构和样式分离，在实际项目开发中推荐使用

```css

.clearfix:after{

content: "";

clear: both;

display: block;

height: 0px;

visibility: hidden;

}

.clearfix{

\*zoom:1;

}

```

> \* 方式六,通过给第一个盒子添加overflow: hidden来清除浮动

### 布局练习

\* 布局练习1

<img src="012-float-demo.png" width='980px' alt="">

\* 布局练习2

<img src="012-float-demo2.png" width='980px' alt="">

### 相对定位

相对于自己在标准流当中的位置发生的移动

\* 注意点

> \* 只写position: relative;元素是不会发生任何移动的,需要配合设置四条边的移动距离来一起使用

> \* 设置四条边的移动距离分别用 top|right|bottom|left

> \* 四条边的移动距可以写一个,也可以写两个,写两个的时候要注意在同一个方向的只能设置一个边

> \* 相对定位元素是不脱离标准流的,所以相对定位元素在标准流当中的空间不会释放出来

> \* 相对定位元素是不脱离标准流的,所以相对定位元素区分块状，行内，行内块状

> \* 如果元素有margin/border/padding值,相对定位是在margin/border/padding值生效后起作用

> \* 相对定位元素主要用来对对元素进行微调和设置绝对定位的参考元素

### 绝对定位

相对于父元素发生位置的移动

\* 注意点

> \* 只写position: absolute;元素是不会发生任何移动的,需要配合设置四条边的移动距离来一起使用

> \* 设置四条边的移动距离分别用 top|right|bottom|left

> \* 四条边的移动距可以写一个,也可以写两个,写两个的时候要注意在同一个方向的只能设置一个边

> \* 绝对定位元素是脱离标准流的,所以绝对定位元素在标准流中的空间会被释放出来

> \* 绝对定位元素是脱离标准流的,所以绝对定位元素不区分块状,行内,行内块状

\* 绝对定位元素的参考点

> \* 绝对定位元素会参考离他最近的具有定位属性的祖先元素,如果祖先中没有定位元素就参考body元素,换句话说如果一个元素设置为绝对定位,首先去找他的父元素,看他的父元素有没有position属性,如果没有就找他的父亲的父亲,直到找到第一个具有position属性的祖先,然后相对于该祖先发生位置移动,如果所有祖先都没有position属性,就相对于body发生位置的移动

> \* 具有定位属性指的是相对定位,绝对定位,固定定位,但不包括静态定位

> \* 父元素的margin/border的值对绝对定位的子元素的位置没有影响,绝对定位的子元素是从父元素的padding区域开始参考的,但如果相对于body,始终相当于相对于浏览器的首屏的可视区域

\* 相对定位和绝对定位配合使用的方法

> \* 子元素用绝对定位

> \* 需要参考的父级元素用相对定位

\* 绝对定位的元素水平居中

> left:50%

> margin-left: - 元素宽度的一半

### 固定定位

参照body发生位置的移动,固定定位的元素不会随着滚动条的移动而移动

\* 注意点

>\* 只写position: fixed;元素是不会发生任何移动的,需要配合设置四条边的移动距离来一起使用

>\* 设置四条边的移动距离分别用 top|right|bottom|left

>\* 固定定位元素是脱离标准流的,所以固定定位元素在标准流中的空间会被释放出来

>\* 固定定位元素是脱离标准流的,所以固定定位元素不区分块状,行内,行内块状

>\* 固定定位元素不会随着滚动条的移动而移动

>\* 固定定位元素始终参考的都是body

### 静态定位

>\* 默认情况下是静态定位

### 定位流中元素显示的顺序

\* 默认情况下,定位流中的元素会盖住标准流中的元素

\* 默认情况下,定位流中后面的元素会覆盖住前面的元素

\* 可以通过z-index来更改元素的覆盖关系,值越大,显示越靠前,默认值是0

\* 父级元素的z-index属性会影响到子元素的z-index,子元素的z-index属性最终的值是所有父级元素中最顶层父级元素的z-index值

### 案例练习

<img src="013-demo.png" alt="">

### 过渡

让元素的状态逐渐的发生变化

\* 添加过渡的必要条件

> \* 给需要过渡的元素添加transition-property属性来指定那个属性需要过渡,如果有多个属性同时变化,将多个属性用逗号隔开即可

> \* 给需要过渡的元素添加transition-duration属性来指定过渡的执行时间,如果有多个属性同时变化,将过渡的执行时间用逗号隔开即可

> \* 给需要过渡的元素添加一个触发事件(通常是hover),在事件中指定需要过渡的属性的变化值

\* 过渡的其他属性

> \* 延迟过渡的执行, 例如: transition-delay: 2s;

> \* 过渡的执行速度, 例如: transition-timing-function: linear|ease 默认的|ease-in|ease-out|ease-in-out|cubic-bezier(x1,y1,x2,y2)

>> \* 预设的速度都可以用三次贝塞尔曲线函数cubic-bezier(x1,y1,x2,y2)来实现

>> <img src="./014-cubic.png" alt="">

>> \* 分别用A,B,C,D表示这四个点，其中起始点固定值为A(0,0),终止点固定为D(1,1)剩下的中间点B(x1,y1),C(x2,y2)也就是所要动态操控的两个点了,对应cubic-bezier (x1,y1,x2,y2)中的四个参数,通过改变B,C两点的坐标值来动态生成一条贝塞尔曲线表示动画中的速度变化。

\* 过渡复合样式的写法

> \* 一组属性的每个属性用空格隔开

>> transition:需要过渡的属性 执行过渡的时间 执行过渡的速度 延迟时间;

```css

transition:width 3s linear 2s;

```

> \* 多组属性之间用逗号隔开

```css

transition:width 3s linear 2s,background-color 3s linear 2s;

```

> \* 如果有多个属性发生同样的过渡效果的话可以把多个属性简写为all

```css

transition:all 3s;

```

\* 添加过渡的步骤

> \* 布局页面

> \* 触发修改属性

> \* 给修改属性添加过渡效果

### 过渡案例

手风琴效果

### 2D转换

\* 旋转

> \* 格式: transform: rotate(ndeg);

>

> \* 注意点:

>> \* n是一个具体的数字,deg代表度数的单位

\* 平移

> \* 格式: transform: translate(水平方向移动距离,垂直方向移动距离);

>

> \* 注意点:

>> \* 移动距离可以是负数,负数代表向反反向移动

\* 缩放

> \* 格式: transform: scale(水平方向缩放比例,垂直方向缩放比例);

>

> \* 注意点:

>> \* 缩放比例如果是1代表不变,大于1代表放大,小于1代表缩小

>> \* 如果水平和垂直方向上缩放比例一样,可以简写一个

\* 复合样式

> \* 格式: transform: rotate(ndeg) translate(水平方向移动距离,垂直方向移动距离) scale(水平方向缩放比例,垂直方向缩放比例);

>

> \* 注意点:

>> \* 多个转换之间用空格隔开

>> \* 旋转会修改坐标系,因此旋转后再平移(即旋转写在平移前面)是在旋转修改后的坐标系下平移

\* 修改旋转基点

旋转基点就是以哪个点为参考点进行旋转

> \* 格式: transform-origin: 水平方向移动距离 垂直方向移动距离;

>

> \* 注意点:

>> \* 默认情况下所有的元素都是以自己的中心点作为参考来旋转的

>> \* 移动距离可以是具体的像素,百分比,和关键字,关键字在水平方向上有left center right,在垂直方向上有top center bottom

>> \* 移动基点的参考点是元素的左上角顶点

\* 修改旋转轴向

在rotate后面加上轴向可以改变旋转的轴向,即rotateZ是沿着Z轴旋转,rotateX是沿着X轴旋转,rotateY是沿着Y轴旋转

> \* 默认情况下元素是沿着Z轴旋转的,即rotateZ和rotate是等价的

> \* 为了让元素呈现近大远小效果,需要给元素的父元素上面添加透视距离属性perspective

### 案例

\* 倒地照片

\* 相片墙

\* 翻转菜单

### 动画的基本使用

\* 创建动画的步骤

> \* 给需要执行动画的元素指定动画的名称

> \* 创建指定名称的动画

> \* 给需要执行动画的元素指定动画的执行时间

\* 过渡和动画的区别

> 过渡需要人为的触发,而动画不需要

### 动画的其他属性

\* 延迟执行

```css

animation-delay: 2s;

```

\* 执行速度

```css

animation-timing-function: linear|ease 默认的|ease-in|ease-out|ease-in-out|cubic-bezier(x1,y1,x2,y2)

```

> 注意：动画执行的速度和过渡的一样

\* 执行次数

```css

animation-iteration-count: 具体的数字(默认是1)|infinite(无限循环);

```

\* 是否执行往返动画

```css

animation-direction: normal(默认状态不执行往返动画)|alternate(执行往返动画);

```

> 注意如果是执行往返动画,返回执行也算一次动画

\* 改变动画的执行状态

```css

animation-play-state: running(默认状态动画执行)|paused(动画暂停)

```

\* 规定动画等待时和结束时的显示状态

```css

animation-fill-mode:none(不做改变)|backwards(动画等待时显示动画的第一帧)|forwards(动画执行结束保持显示动画的最后一帧)|both(动画执行前显示动画的第一帧并且动画执行结束显示动画的最后一帧)

```

### 动画的复合样式写法

```css

animation:动画的名称 动画的执行时间 动画的速度 延迟时间 动画的执行次数 是否执行往返动画;

```

> 注意：动画名称和执行时间不可以省略

### 创建动画的方式二

用百分比创建动画

> 注意点:

>

> \* 0%代表动画开始,100%代表动画结束

>

> \* 0%和100%推荐都写

### 案例

滚动焦点图

### 添加3D效果

给需要实现3D效果的元素的父元素添加transform-style: preserve-3d;属性

\* 注意点:

> \* 元素默认是2D展示的

> \* 3D是有厚度的,是立体的,相当于长方体,而2D没有厚度,是平面的,相当于长方形

### 案例

\* 正方体

\* 长方体

\* 3D轮播图

### 常见字体文件格式

\* EOF(Embedded Open Type) 嵌入式字体，是微软开发的技术。

\* WOFF(Web Open Font Format) Web开发字体格式,一种专门为了Web而设计的字体格式标准

\* TTF(TrueType Font) Windows和Mac系统最常用的字体格式

\* SVG(Scalable Vector Graphics) 由W3C制定的开放标准的图形格式

### 字体浏览器的支持情况

<img src="img/018-1.png" alt="">

### 自定义字体的方法

> \* 准备自定义字体文件,通常可以从网上下载

> \* 使用@font-face 定义一个新字体

```css

@font-face{

font-family: 自定义字体名称;

src:url(字体文件1),

url(字体文件2);

[font-weight: bold;]

[font-style: italic;]

}

```

> \* 在CSS中使用自定义的字体

```css

font-family: 自定义字体名称;

```

> \* 注意点

>> \* src中的url可以是本地字体文件也可以是网络上的字体文件,如果是网络上的字体文件注意跨域的问题

>> \* 自定义字体默认的font-weight和font-style都是normal

>> \* font-weight和font-style是可选项

>> \* 加粗和倾斜本质上也是对应有一个字体文件

>> \* font-weight需要单独指定一个包含了粗体的字体文件,如果不指定,使用自定义字体时,浏览器会使用默认的加粗效果

>> \* font-style需要单独指定一个包含了斜体的字体文件,如果不指定,使用自定义字体时,浏览器会使用默认的倾斜效果

### 自定义字体实现图标

把图标做成字体文件,在页面中以字体的形式使用这些图标

\* [fontawesome图标库](https://fontawesome.com/)

\* 使用方法

> \* 方法一: CSS+字体

>

```css

<i class="fas fa-camera-retro"></i>

```

>

> \* 方法二: HMTL+字体

```css

<i class="fas">&#xf083;</i>

```

> 注意点:

>> \* 无论是方法一还是方法二,本质都是在网页中插入一个四位的16进制数,然后用特殊的字体把他显示出来

>> \* 网页中插入16进制数,需要在前面面添加&#x

>> \* 无论是方法一还是方法二,都需要在使用前引入相应的css文件

>> \* 需要注意引入的css文件中使用的字体文件的路径

###浏览器的历史

\* Tim Berners-Lee发明万维网的同时发布了第一款浏览器 Nexus,也叫www浏览器

\* 由于Nexus只能显示文字, 马克安德森在伊利诺伊大学开发了第一款可以浏览图片网页的浏览器 Mosaic

\* 1994年4月,马克安德森和吉姆克拉克成立了Mosaic Communication Corporation

\* 但是由于Mosaic的版权归伊利诺伊大学所有,并且伊利诺伊大学将技术转让给了Spy Glass公司,所以在1994年11月Mosaic公司改名为Netscape Communication Corporation,浏览器改名为Netscape Navigator

\* 1996年微软买下Spy Glass公司,以Mosaic为基础开发出Internet Explorer

\* 1996年,挪威的Telenor公司开发了Opera浏览器

\* 2003年,Netscape公司倒闭,将Netscape Navigator浏览器源代码开放,Mozilla公司在Netscape Navigator浏览器的基础上开放出了FireFox

\* 2003年,苹果公司开发了safari浏览器

\* 2008年,Google发布Chrome浏览器

###JavaScript历史

\* 1995年,兰登艾奇(Brendan Eich)开发出LiveScript,并首次运行在Netscape Navigator 2上,用来完成简单的表单验证

\* 因为市场宣传需求,LiveScript与Sun公司合作,发布前将LiveScript改名为JavaScript

\* 随后在Netscape Navigator 3中发布了JavaScript1.1,同时微软在IE3中加入JavaScript的实现JScript,ScriptEase中加入JavaScript的实现CEnvi,至此浏览器上有三种不同版本的脚本语言

\* 1997年,为了统一,欧洲计算机制造商协会ECMA(European Computer Manufacturers Association）的39号技术委员会(TC39 Technical Committee#39)以JavaScript1.1为蓝本,完成了ECMA-262号标准的制定,即定义了一种名为ECMAScript的新的脚本语言标准,届时ES1诞生

\* 1998年,ES2发布,但是ES2没有任何改动,该版本主要是为了和ISO/IEC-16262保持一致

\* 1999年,ES3发布,ES3新增了正则,异常处理,控制语句等,自此ES成了一门正真的语言

\* 2000年,ES4开始准备,但是这个版本最后没有通过,但是它的大部分内容被ES6继承了

\* 2007年,ES4草案发布,ES4做了一次全面的修改,包括强类型变量,类和继承,但是由于改动太大,ECMA和各大浏览器厂商发生了严重分歧,最后ECMA开会决定,中止ES4的开发

\* 2009年12月3号,ES5发布,新增JSON对象,严格模式等,到了2011年,ES5成为了ISO国际标准

\* 2015年6月,ES6发布

\* 2016年,ES7发布

\* 2017年6月,ES8发布

###浏览器组成

\* Shell部分

\* 内核

> \* 内核的组成

>> \* 渲染引擎

>> \* JS引擎

>> \* 其他模块

> \* 主流内核介绍

>> \* Trident(IE内核)

>> \* Gecko(FireFox内核)

>> \* Webkit(Safari内核)

>> \* Blink(Chrome和Opera内核)

### JS引擎

\* 各个浏览器厂商的JS引擎去实现ES标准

\* 初期JS引擎由渲染引擎来负责,执行效率非常慢

\* 2001年,IE6将JS引擎从渲染引擎中分离出来并做了优化

\* 2008年,Google发布Chrome浏览器,并使用新的JS引擎V8,V8可以直接把JS代码转换为机器码,所以执行速度大大提高

### JS的组成

\* ECMAScript

\* DOM(Document Object Model)

\* BOM(Browser Object Model)

### JS运行在哪里

JS是一门解释性语言,运行在浏览器中,由浏览器的JS引擎负责解释运行JS代码。

### JS学什么

\* 学所有程序共同的

> 常量／变量／表达式／控制结构／数据类型／算法

\* JS这门语言独有的

> JS的内置的对象／函数

\* 前端开发相关

> 操作 DOM／BOM

### 1.写在元素标签的起始标签中

> \* 用法:把JS代码写在元素起始标签的事件属性中

> \* 缺点:重复代码多,维护困难,使用场景有限

#### 案例1:点击改颜色/免登陆提示框

\* 通过案例总结：

> JS可以修改元素的属性

> 修改元素属性的步骤

>> \* HTML页面布局

>> \* 确定修改哪些元素

>> \* 确定触发修改元素的事件，并将该事件添加在触发元素上

>> \* 编写JS代码来修改属性

> 注意点

>> \* 任何属性都可以修改

>> \* 任何标签都可以有ID属性,可以通过ID属性获取该元素

>> \* 修改属性时，HTML当中怎么写，JS当中就怎么写,但class是特例,用className

>> \* 修改样式的格式：obj.style.属性＝值

>> \* JS用style修改属性时，如果CSS的属性有"-"的话，通常把"-"去掉，然后把"-"后面的单词首字母大写

### 2.写在script标签中

> \* 把样js代码写在在script标签里面

> \* 缺点: 只适用于当前页面

> \* 注意:

>> \* 传统的做法是把script标签写在head标签里面

>> \* script标签中的type属性可以不用写, 默认就是type="text/javascript"

#### 案例2:修改多个样式

\* 通过案例总结：

> \* 可以把多行代码写在一起,然后给他起个名字,这就是函数

> \* 函数分为定义和执行,二者缺一不可

> \* 函数可以多次执行

> \* 变量相当于值的一个别名,变量出现的地方就相当于值本身

#### 案例3:网页换肤

\* 通过案例总结：

> \* 任何标签都可以加ID

> \* 任何标签的任何属性都可以修改

### 3.写在外部的单独文件中,用script标签引入

> \* 把JS写在一个外部文件里面,这个外部文件用.js作为扩展名,在需要使用的页面通过script标签的src属性引用即可

> \* 注意:

>> \* 传统的做法是把script标签写在head标签里面

>> \* script标签中的type属性可以不用写, 默认就是type="text/javascript"

>> \* 延迟属性defer,添加该属性后,脚本文件会在整个页面都解析完毕后执行

>> \* 异步属性async,添加该属性后,脚本会立即下载执行,但不妨碍页面中的其他操作

### 4. noscript标签

指定在不支持脚本的浏览器中显示替代内容

> \* 代替内容只在以下情况出现

>> \* 浏览器自身不支持脚本

>> \* 浏览器持脚本,但是被禁用了

### 语法基础

\* 区分大小写

\* 标示符(变量,函数,属性,参数的名字)的规则

> \* 第一个字符必须是字母,下划线(\_)或者美元符号($)

> \* 其他字符可以是字母,下划线(\_)或者美元符号($)或者数字

> \* 命名一般采用驼峰大小写格式,即第一个字母小写,剩下的每个单词首字母大写

> \* 不能用关键字,保留字做为标示符

\* 注释

> \* 单行注释

```javascript

// 我是单行注释

```

> \* 多行注释

```javascript

/\*

\*

\* 我是多行注释

\*

\*/

```

\* 严格模式(strict mode)

> \* 作用:告诉JS引擎切换到严格模式下解析代码,这种模式可以消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为,这是在ES5中新增的一种运行模式

> \* 使用:

>> \* 针对整个文件,将"use strict"放在脚本文件的第一行,则整个脚本都将以"严格模式"运行,如果这行语句不在第一行,则无效

>> \* 针对单个函数,将"use strict"放在函数体的第一行,则整个函数以"严格模式"运行。

> \* 注意点:

>> \* 同样的代码,在"严格模式"中,可能会有不一样的运行结果;一些在"正常模式"下可以运行的语句,在"严格模式"下将不能运行。 例如,以下代码在两种运行模式下的返回值是不同的

```javascript

function sayHi(){

alert(!this);

}

sayHi();

```

\* 每条语句以分号结尾

> 注意点:语句结尾的分号不是必须的,但建议都加上

### 程序的流程控制

#### 1. 顺序结构

> 按照代码书写的先后顺序来执行,这是程序的默认执行方式

#### 2. 分支结构

\* 语法1

```javascript

if(条件){

语句

}else{

语句

}

```

\* 注意点

> 条件是一个布尔值,布尔值只有真或者假

案例:弹出菜单

\* 语法2

```javascript

switch(变量){

case 值1:

语句1;

break;

case 值2:

语句2;

break;

....

default:

语句n;

}

```

\* 语法3 三元运算

条件 ? 语句1 : 语句2

> 条件成立执行语句1,否则执行语句2

#### 3. 循环结构

\* 语法1-while循环

```

初始值;

while(循环退出条件){

循环体(语句)

循环退出条件的改变

}

```

\* 语法2-for循环

```

for(初始值;循环退出条件;循环退出条件的改变){

循环体(语句)

}

```

\* 语法3-for in 循环

```

for(下标 in 对象){

循环体(语句)

}

```

\* break和continue

> \* break 中止所有循环

> \* continue 中止本次循环

### 数组

数据的有序列表

\* 通过[]定义一个数组

\* 注意点:

> \* 可以通过下标来访问数组中的元素

> \* 数组的下标是从0开始的

> \* 可以通过数组的length属性获取到数组的长度

> \* getElementsByTagName方法获取的数据返回的就是一个数组

### Json

JavaScript Object Notation,JavaScript的对象表示法,是一种数据格式

\* 注意点

> \* Json的遍历需要用for in循环

### 补充知识

> \* innerHTML 的使用

> \* 字符串拼接

> \* 字符串拼接的优先级

> \* document.write输出内容到文档中

案例1:设置多个元素的样式

案例2:全选/全不选/反选

案例3:乘法表

案例4:倒三角

案例5:选项卡

案例6:月历

### 1. 变量的类型

\* Undefined 未定义

> \* 未定义类型只有一个值undefined,这个值的意思是变量没有定义

> \* 未定义有两种情况,一个是真的没有定义,一个是定义了但是没有赋值

\* Null 空对象

> \* 空对象类型只有一个值null,代表一个空对象指针

\* Boolean 布尔类型

> \* 布尔类型有两个值:true和false

\* Number 数字类型

> \* 用来表示数字

> \* 可以表示整数和小数(也叫浮点数或者双精度数)

\* String 字符串类型

> \* 用单引号或者双引号括起来的字符序列

> \* 单引号和双引号表示的字符串完全一样

\* Object 对象类型

> \* 对象是一组数据和功能的集合

> \* null代表了一个空对象指针,所以用 typeof 检测时会返回 'object'

> \* 函数在ES中是对象,但函数也有一些特殊的属性,所以用 typeof 检测时会返回 'function'

> \* 数组在ES中是对象

\* 注意点:JS变量的类型由值决定,并且随着值的变化而变化

### 2. 转数值

\* 显式转换parseInt 和 parseFloat

> \* parseInt用来转换整数,parseFloat可以转换整数和小数

> \* 转换规则为

>> \* 忽略字符串前面的空格

>> \* 如果第一个字符不是数字或者负号加数字,会返回NaN(Not a Number)

>> \* 从左到右解析直到第一个非数字为止

\* isNaN

> 由于NaN和NaN并不相等,所以用isNaN函数来判断转换后的值是不是数字

\* 隐式转换

> 减法(-) 乘法(\*) 除法(/) 比较(==)

\* 案例:加法计算器

### 3. 转字符串

\* toString()

\* +""

### 4. 布尔值

> \* 真:true,非零数字,非空字符,非空对象,

> \* 假:false,数字零,空字符,null,undefined

### 5. 变量的作用域

\* 全局变量

> 定义在所有函数外面,所有的函数都可以使用的变量

\* 局部变量

> 定义在函数的内部,指定在函数内部或者函数的子函数中使用

### 6. 闭包

子函数可以使用父函数中的变量的一种现象

### 7. 变量的类型

\* 基本类型

> \* 基本类型指定的是简单的数据,包括,Undefined,Null,Boolean,Number,String

> \* 基本类型访问的变量的实际值

> \* 基本类型不能动态的添加属性

> \* 基本类型变量复制时会新建一个变量并分配空间,将变量的值分配在新的空间上

\* 引用类型

> \* 引用类型主要指保存在内存中的对象,包括对象,数组,函数

> \* 引用类型访问的是引用,引用是指向内存的一个指针

> \* 引用类型可以动态的添加属性

> \* 引用类型复制时不会新建空间,只是复制了一个新的引用

\* 引用类型

### 8. 变量命名

\* 变量名 = 类型 + 对象描述

> \* 类型指变量的数据类型,JS命名类型有

>> array 数组a

>>

>> boolean 布尔值b

>>

>> float 浮点数l

>>

>> function 函数fn

>>

>> int 整型i

>>

>> object 对象o

>>

>> regular 正则r

>>

>> string 字符串s

> \* 对象描述指对象名字全称或名字的一部分，要求有明确含义，命名要容易记忆容易理解

### 1. 算数

> 加(+), 减(-), 乘(\*), 除(/), 模(%),++,- -

>

> 案例1:隔行变色

> 案例2:秒变时间

### 2. 赋值

> =, +=, -=, \*=, /=, %=

### 3. 关系

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

### 4. 逻辑

> 与(&&),或(||),否(!)

### 5. 优先级

> 括号()

### 函数参数

\* 函数的参数,参数其实就是一个占位符号,只有在调用的时候才来赋值,经常变的部分用参数

\* 匿名函数,没有名字的函数就是匿名函数

\* 注意点:

> \* 在使用属性时,用'[]'可以代替所有的'.',但是'.'不一定能代替'[]'

>\* 案例1:改变元素的样式函数

### 函数的返回值

在函数体内通过return把函数的执行结果返回

> \* 一个函数只能有一个返回值

> \* 没有返回值或者只有return的函数返回值是undefined

> \* 函数的返回值是在调用的时候获取的

### 变量的作用域

\* 全局变量

> 定义在所有函数外面,所有的函数都可以使用的变量

\* 局部变量

> 定义在函数的内部,指定在函数内部或者函数的子函数中使用

\* 作用域的决定

> 函数执行时依赖于变量的作用域,这个作用域时在定义的时候决定的,而不时在调用的时候决定

### 闭包

子函数可以使用父函数中的变量

### 函数作用域

\* 块级作用域:在代码块(花括号中的代码)中声明的变量在代码块外不可用

\* 函数作用域:在函数内部声明的变量在函数内部以及子函数中都可以使用

\* JS没有块级作用域,而是函数作用域

\* JS中因为是函数作用域,所以有声明提升的现象,即JS函数中声明的所有变量(不涉及赋值)都被提前到函数的顶部

### 递归调用

自己调用自己的函数

> \* 案例:阶乘的计算

### 函数的不定参数

> \* 每一个函数都有一个参数数组,数组的名字是arguments

> \* 参数数组中的值由调用时传递,按照传递参数的顺序存储在数组中

> \* 案例1:求和函数

> \* 案例2:获取和设置样式函数

### 获取样式函数

\* 获取行间样式 obj.style[attr]

\* 获取非行间样式 getComputedStyle(obj,false)[attr]

### 数组的定义

\* 方法1 通过Array构造函数

```

var arr = new Array(1,2,3);

```

\* 方法2 通过数组字面量[]

```

var arr = [1,2,3]

```

\* 注意点

> \* 方法1和方法2的作用时一样的,方法2效率上更高一些

> \* 可以改变数组的length属性,改变后的值时数组的实际长度,可以通过这种方法清空数组

### 数组的操作

push(元素) 在数组末尾添加

pop() 删除数组的末尾元素

shift() 删除数组的第一个元素

unshift(元素) 把元素添加到数组的头部

splice(起始下标,删除个数,插入值1,插入值2...)

join(字符) 用指定的字符连接数组

数组1.concat(数组2) 把数组1和数组2合并生成一个新的数组

slice(起始下标,结束下标) 基于当前数组创建一个新数组

indexOf(查找项,起始下标) 从数组头部开始向后查找,返回查找到的数组的下标,如果没有找到返回-1

lastIndexOf(查找项,起始下标) 从数组未部开始向前查找,返回查找到的数组的下标,如果没有找到返回-1

### 数组的排序

\* 按字符排序

\* 指定排序函数

### 去除数组的重复元素

\* 数组是引用类型

### DOM基础

\* DOM节点介绍

\* 子节点

> \* 节点的分类

>> 文本节点

>>

>> 元素节点

> \* 获取子节点的方法

>> \* oParentNode.childNodes,返回一个数组,包含父节点中的所有文本节点和元素节点

>> \* 通过nodeType属性来判断节点的类型,1为元素节点,3为文本节点

>> \* oParentNode.children,返回一个数组, 包含父节点中的所有元素节点

\* 父节点

> \* oNode.parentNode,子节点的父节点对象

> \* oNode.offsetParent,子节点绝对定位时,获取参考的父节点对象

> \* 隐藏父元素案例

\* 首尾子节点

> \* firstChild, firstElementChild

> \* lastChild, lastElementChild

\* 兄弟节点

> \* nextSibling,nextElementSibling

> \* previousSibling,previousElementSibling

\* 操作属性

> \* 获取属性的值:getAttribute(名称),有属性获取值,没有属性返回null

> \* 设置属性:setAttribute(名称,值)

> \* 删除属性:removeAttribute(名称)

\* 通过类名获取元素集合

> \* 封装函数

\* 创建元素

> document.createElement('标签名')

\* 追加元素

> parentNode.appendChild(node)

\* 插入元素

> parentNode.insertBefore(node1,node2),在node2之前插入node1

\* 删除元素

> parentNode.removeChild(node)

> \* 删除父元素案例

### DOM 应用

\* 获取表格元素

> \* 隔行变色案例

> \* 添加删除行

> \* 搜索内容

>> \* toLowerCase()

>> \* search

>> \* split

\* 按内容排序

### 定时器

\* setInterval(fn,时间) 开启循环型定时器

\* setTimeout(fn,时间) 开启延迟型定时器

\* clearInterval(定时器) 关闭循环型定时器

\* clearTimeout(定时器) 关闭延迟型定时器

\* 注意点:

> \* 循环型定时器在定时器关闭前会一直按照设定的间隔时间执行,延迟型定时器只执行一次

> \* 定时器中指定的时间指的是该时间后把代码加入到执行队列中,如果队列空闲就立即执行

\* 案例1:数码时钟

> \* 系统时间对象 Date,方法 getHours、getMinutes、getSeconds,getFullYear(),getMonth(),getDate(),getDay()

> \* 字符串的charAt方法

\* 案例2:分类菜单

\* 案例3:焦点图1

\* 案例4:焦点图2

### 匀速动画

\* 匀速动画函数的要点

```javascript

function aninamtion(obj,attr,iTarget){

//1.一开始关闭定时器是为了防止重复触发函数时有多个定时器工作加快动画执行

//2.定时器做为对象的参数是为了多个物体动画时不相互干扰

clearInterval(obj.timer);

var iSpeed = 0;

obj.timer = setInterval(function(){

//由于getComputedStyle的返回值有单位,所以需要类型转换,由于透明度可能有小数,所以用parseFloat

//offsetWidht/offsetHeight指的是元素的宽和高,而不是内容的宽和高,所以用getComputedStyle

var curr = parseFloat(getComputedStyle(obj,false)[attr]);

//透明度的处理以百分比的形式计算,所以乘以100

if(attr == 'opacity'){

curr = curr \* 100;

}

//根据当前值和目标值来决定速度的正负

//如果当前值小于目标值,速度为正

//如果当前值大于目标值,速度为负

if(curr <= iTarget){

iSpeed = 9;

}else{

iSpeed = -9;

}

//动画结束的条件,

//1.当前值和目标值的差不够一次动画时,即当目值和目标值的差小于速度时,动画结束并且物体直接到达目标

//2.由于速度有正负,所以用绝对值

if(Math.abs(iTarget - curr) <= Math.abs(iSpeed)){

//结束动画

clearInterval(obj.timer);

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = iTarget/100;

//其他有px单位的属性处理

}else{

obj.style[attr] = iTarget + 'px';

}

//当前值和目标值的差大于一次动画时,执行动画

}else{

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = (curr + iSpeed)/100;

//其他有px单位的属性处理

}else{

obj.style[attr] = curr + iSpeed + 'px';

}

}

},30)

}

```

\* 案例分享到

\* 模拟鼠标hover淡入淡出

\* 扩充知识点:

> Math.abs(num) 取num的绝对值

\* 案例侧边栏固定(底部和中部)

\* 案例顶部滚出导航

> \* 扩充知识点:

>> \* window.innerWidth 窗口可视宽度

>> \* window.innerHeight 窗口可视高度

>> \* document.body.scrollTop 滚动距离,safari用

>> \* document.documentElement.scrollTop 滚动距离,chrome/ff...用

>> \* window.onresize事件,窗口大小发生变化时触发

>> \* window.onscroll事件,滚动条变化时触发

### 减速动画

\* 基本减速动画

> \* 扩充知识点:

>> Math.ceil(num) 向上取整

>> Math.floor(num) 向下取整

\* 多物体减速动画

\* 任意值减速动画

>\* 对透明度单独处理

>\* 扩充知识点:

>> Math.round(num) 四舍五入

\* 综合动画-匀速动画函数和减速动画函数合并

\* 链式动画

> \* 参数接受一个函数

\* 多值动画

> \* 参数接受一个对象,对象里指定属性和值

\* 综合动画函数要点

```javascript

function animation(obj,opation,isLinear,fnEnd){

//1.一开始关闭定时器是为了防止重复触发函数时有多个定时器工作加快动画执行

//2.定时器做为对象的参数是为了多个物体动画时不相互干扰

clearInterval(obj.timer);

//初始化速度

var iSpeed = 0;

obj.timer = setInterval(function(){

//1.isStopAll用来代表改对象上的所有属性的动画是否执行完毕

//2.每次定时器函数执行时把isStopAll改为true,在属性循环执行动画中如果有一个属性没有结束动画,就把该值该为false

var isStopAll = true;

//循环执行每个属性的动画

for(attr in opation){

//1.由于getComputedStyle的返回值有单位,所以需要类型转换,由于透明度可能有小数,所以用parseFloat

//2.offsetWidht/offsetHeight指的是元素的宽和高,而不是内容的宽和高,所以用getComputedStyle

var curr = parseFloat(getComputedStyle(obj,false)[attr]);

//单次属性的动画是否结束,默认时false,表明不结束,只有符合结束条件时改为true

var isStop = false;

//透明度的处理以百分比的形式计算,所以乘以100

if(attr == 'opacity'){

curr = Math.round(curr\*100)

}

//匀速动画

if(isLinear){

//1.根据当前值和目标值来决定速度的正负

//2.如果当前值小于目标值,速度为正

//3.如果当前值大于目标值,速度为负

if(curr > opation[attr]){

iSpeed = -10;

}else{

iSpeed = 10;

}

//匀速动画动画结束的条件:

//1.当前值和目标值的差不够一次动画时,即当目值和目标值的差小于速度时,动画结束并且物体直接到达目标

//2.由于速度有正负,所以用绝对值

if(Math.abs(opation[attr] - curr) <= Math.abs(iSpeed)){

isStop = true;

}else{

//如果当前动画没有结束,不能结束该对象的动画

isStopAll = false;

}

//减速动画

}else{

//减速动画的目标离当前值越近速度越小

iSpeed = (opation[attr] - curr)/10;

//由于减速动画的计算公式会产生小数,所以用取整函数

iSpeed = iSpeed > 0 ? Math.ceil(iSpeed) : Math.floor(iSpeed);

//减速动画的结束条件是目标值和当前值相等,即速度是0

if(!iSpeed){

isStop = true;

}else{

//如果当前动画没有结束,不能结束该对象的动画

isStopAll = false;

}

}

//单个属性动画结束

if(isStop){

//匀速动画结束时的处理

if(isLinear){

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = opation[attr] / 100;

//其他有px单位的属性处理

}else{

obj.style[attr] = opation[attr] + 'px';

}

}

//单个属性动画执行

}else{

//透明度的处理

if(attr == 'opacity'){

obj.style[attr] = (curr + iSpeed)/100 ;

//其他有px单位的属性处理

}else{

obj.style[attr] = curr + iSpeed + 'px';

}

}

}

//如果循环中没有把isStopAll改为false,说明所有动画都执行完毕了

if(isStopAll){

//结束改对象上的动画

clearInterval(obj.timer);

//如果有后续函数的话,调用后续函数

if(fnEnd){

fnEnd();

}

}

},30)

}

```

### window对象

\* window代表一个浏览器对象

\* window对象是一个全局对象,因此在全局作用域中声明的变量函数会变成window对象的属性和方法

### window.open()

> 打开一个新窗口并返回新打开的窗口对象

\* window.open("about:blank","\_blank");

\* window.open("about:blank");

\* window.open("about:blank","\_self");

\* window.open("http://www.kuazhu.com");

### window.close()

\* 不能关闭用户打开的窗口

### window.navigator.userAgent

\* 当前浏览器的信息

### window.location

\* 当前页面的相关信息对象,可以读取和赋值

\* document.loaction和window.location是同一个对象

\* window.location.href 属性代表当前页面的完整url

\* window.location.assign(url),assign方法会打开url,如果把location.href或者location设置为一个url值,也会调用assign方法

\*window.location.reload()重新刷新该页面

\*window.history.back()返回上一次页面

\* location的其他属性

> host

> hostname

> pathname

> port

> search

### window.histroy

\* 用户上网的记录,从窗口被打开的那一刻算起

history.go(-1) == history.back()

history.go(1) == history.forward()

### 交互

var res = prompt("请输入你的姓名？");

var res = confirm("你确定删除吗?");

### 事件对象 event

\* 事件对象兼容处理

\* 事件流

> 事件冒泡

> 事件冒泡指具体元素接受事件,然后逐级向上传播事件

> 阻止事件冒泡 event.stopPropagation()

\* 案例: 下拉菜单

### 鼠标事件

\* 鼠标移动 onmousemove

\* 获取坐标

> event.clientX 和 event.clientY是鼠标的可视区坐标

> 通常情况需要加上滚动距离

\* 案例1: 随着鼠标移动的物体

\* 案例2: 随着鼠标移动的多个物体

### 键盘事件

\* 键盘按下 obj.onkeydown

\* 键盘抬起 obj.onkeyup

\* 键盘代码 event.keyCode

\* 辅助键 event.ctrlKey event.shiftKey event.altKey

\* 案例1:左右键移动物体

\* 案例2:ctrl+回车提交

\*

### 默认行为

> 默认行为是浏览器的自带行为

\* document.oncontextmenu 右键菜单

\* 阻止默认行为

> \* 方法一 return false

> \* 方法二 event.preventDefault()

\* 案例1:右键菜单

\* 案例2:只能输入数字的输入框

### 拖拽

> 原理:鼠标和物体的距离始终不变

obj.onmousedown

obj.onmouseup

obj.onmousewheel

event.wheelDelta

> \* 基础拖拽

> \* 限制在浏览器中

> \* 限制在父容器中

> \* 磁性吸附

> \* 带框的拖拽

> \* 自定义滚动条(控制大小,控制透明度,控制文字)

> \* 碰撞检测

> \* 愤怒的小鸟

> \* 放大镜效果

### 事件

\* 事件是用户或者浏览器执行的某中动作,如click/mouseover

\* 事件处理程序是响应事件的函数,事件处理程序以on开头,如onclick/onmouseover

\* 事件绑定

> obj.addEventListener(事件名,函数,false)

>

> 事件名没有'on'

>

> false事件冒泡,默认是false

>

> true事件捕获

\* 事件移除

> obj.removeEventListener(事件名,函数,false)

> 移除事件监听只能是移除有函数名的，不能移除匿名函数的

\* 事件委托

> 事件委托是利用了事件冒泡原理

### event对象常见的属性和方法总结

\* event.type 现在的事件类型是什么

\* event.target 事件源对象

\* event.currentTarget 绑定事件的对象

\* event.clientY/event.clientX 可视区(浏览器)的内沿到鼠标的距离

\* event.pageY/event.pageX 可视区(浏览器)的内沿加滚动距离到鼠标的距离

\* event.screenY/event.screenX 显示器屏幕内沿到鼠标的距离

\* event.preventDefault() 阻止默认行为

\* event.stopPropagation() 阻止冒泡

\* event.keyCode 键盘编码

\* event.ctrlKey event.shiftKey event.altKey 辅助键

> \* 蒙板效果

### 常见事件类型总结

onload

onscroll

onresize

onclick

onmouseover/onmouseenter

> \* 鼠标穿过事件源对象或其子元素时触发onmouseover,对应的有onmouseout

> \* 鼠标穿过事件源对象时触发onmouseenter,对应的有onmouseleave

onmouseout/onmouseleave

onmousemove

onmousedown

onmouseup

onkeydown

onkeyup

onfocus 表单获取焦点触发

onblur 表单失去焦点触发

onchange 页面内容有改动,失去焦点执行

onkeypress 键按下去出发,键不太起会一直触发

onkeyup 键抬起触发,包括功能键

oninput 输入后触发,不包括功能键

onsubmit form表单提交事件

onreset form表单重置事件

> \* 简单提交验证和输入提示

### 对象的基本概念

\* 什么是对象

> 对象就是一个可以提供特定功能的黑盒子,使用者不用关心功能的具体实现,只要知道怎么用就可以了

> 举例:电视就是一个对象,通过遥控器或者电视面板上的按钮操作电视

\* 什么是面向对象的程序设计

> 开发具有特定功能的黑盒子

\* 对象学什么

> \* 使用对象 如:Date/Array/Math...

> \* 开发自己的对象

\* 对象的特点

> \* 封装 把具体的功能实现包起来,对外不公开

> \* 抽象 根据具体的业务需求把需要的属性和方法抽取出来

> \* 继承 子类具有父类的方法和属性,然后子类再开发自己特有的方法和属性

> \* 多态 把子类赋给父类,赋值后的父类具有子类的特性

\* 对象的组成

> \* 属性(变量)

> \* 方法(函数)

### 创建对象

\* 字面量创建

> \* 单体对象,适合整个系统只有一个对象的场景

> \* 对象属性的特征:无序唯一

>> \* 对象属性特征的应用,去除重复数组

>>> \* 把数组里面的值转换为对象的属性,由于对象的属性不能重复，所以此时对象的属性就把重复的值过滤掉

>>> \* 把对象的属性转换为数组的值

> \* js对象和json的区别

>> \* json是一种数据传输格式,js对象是类的实例,不能直接传输

>> \* json的键必须添加双引号,值不能是函数,js对象键不用加双引号,值可以是任意类型,如果js对象的属性不是一个合法的变量名时需要加双引号

>> \* json可以通过JSON.parse(jsonstr)转换为js对象,js对象可以用JSON.stringify(jsobj)转换为json

\* 构造函数创建

> \* 用系统自带的对象创建对象

>> \* 缺点:会有把系统自带的方法和属性覆盖掉的风险

> \* 用Object对象来创建对象

>> \* Object是所有对象的父类

>> \* 创建的对象方法和属性不能共用

> \* 用工厂模式的函数创建对象

>> \* 不能用new来创建对象

>> \* 方法不能被实例共享,每个对象实例都生成一个相同的方法

> \* 用new关键字创建对象 － 构造函数

>> \* new 的作用,函数会在一开始创建一个对象，并且这个对象就是this,最后会把这个this对象返回出去

>> \* 直接用new后可以把用工厂模式函数中创建对象和返回对象省略,但是方法还是不能被实例共享

>> \* 用new后,把工厂模式函数可以看做是类,类名一般首字母大写

> \* 构造函数方法共享－将构造函数里面的方法指向全局函数

>> \* 缺点1.全局函数会在其他地方调用，这时会有风险

>> \* 缺点2.如果方法太多，就需要创建大量的全局函数

>> \* 缺点3.违背面向对象的封装特性

> \* 构造原型模式

>> \* 原型 prototype 是存在于构造函数上的一个对象,所有的实例对象都可以访问原型对象

>> \* 可以在原型对象prototype添加共用的属性或者方法

>> \* 变化的属性和方法写在构造器中

> \* 动态原型模式

>> \* 可以把原型对象prototype上的定义写在构造函数中,定义之前需要唯一执行判断

### 对象中的名词解释

构造函数和类

> 用来创建对象的函数叫构造函数,也叫类

对象实例

> 由构造函数创建的一个具体的对象

静态方法和实例方法

> 添加在构造函数上的方法是静态方法,静态方法的调用方法是 构造函数.方法名()

> 添加在构造函数的原型对象上的方法是实例方法,静态方法的调用方法是 对象的实例.方法名()

### 内存分析

<img src="img/014-001.png">

### 原型链和继承

\* 原型链

> \* 原型就是一个对象，所有的实例共享对象上的方法和属性

> \* 一但有构造函数，系统会自动的给构造函数添加一个原型对象prototype

> \* 系统会给原型对象上添加一个constructor对象(prototype.constructor)，就是构造器，整个构造器指向构造函数本身

> \* 系统会给原型对象上添加另外一个\_\_proto\_\_的对象(prototype.\_\_proto\_\_)，该对象指向Object的原型对象(Object.prototype)

> \* Object的原型对象(Object.prototype) 里面也有constructor对象和\_\_proto\_\_对象,constructor对象指向Object自己，\_\_proto\_\_对象指向null

> \* Object的原型对象(Object.prototype) 里面除了constructor对象和\_\_proto\_\_对象外，还有一些系统自带的方法，比如toString(),valueOf()等等

> \* 可以在原型对象上添加自己的方法和属性，这些方法和属性也会被所有的实例共享

> \* 对象的实例有自己的属性和方法,同时系统会给对象实例添加\_\_proto\_\_对象，\_\_proto\_\_对象对象就是构造函数的原型对象

>\* 实例对象的\_\_proto\_\_对象-->构造函数的原型对象，构造函数的原型对象的\_\_proto\_\_-->Object的原型对象，Object的原型对象的\_\_proto\_\_ -->null

\* 原型链上的属性

> 属性的添加

>> 在实例上添加一个实例上的同名属性会覆盖原来的

>> 如果在实例对象上添加了一个构造函数原型对象上的同名属性，访问时会找到实例对象上的该属性

> 属性的删除

>> 通过delete 删除属性

>> 删除实质是判断有没有该属性，如果有删除返回真，若没有，直接返回真

>> 如果删除了原型上的属性，所有实例都访问不到该属性

> 属性的判断

>> 判断对象上是否有自己实例上指定的属性,如obj1.hasOwnProperty('name')

>> 判断对象的原型链上是否有指定的属性,如"LEGS" in obj1;

>> 判断属性只在原型上有

```javascript

function hasPrototypeProperty(obj,name){

return !obj.hasOwnProperty(name) && (name in obj);

}

```

> 属性的定义

>> 定义单个属性

```javascript

Object.defineProperty(对象,属性名,

{value:属性值,

enumerable:true,//是否可以枚举,默认是false

writable:true,//是否可写(赋值),默认是false

configurable:true//是否可删除,是否可以修改属性的权限，默认是false

}

);

```

>> 定义多个属性

```javascript

Object.defineProperties(对象,{

属性名1:{value:属性值,writable:true},

属性名2:{value:属性值,enumerable:true},

属性名3:{value:属性值}

});

```

>> 查看属性的权限

```javascript

Object.getOwnPropertyDescriptor(对象,属性名)

```

\* 继承

> 改变函数内部的this指向

>> 调用函数时,函数名.call(参数1,参数2....),call 第一个参数是被调用函数里面的this,后面是对应的参数列表

>> 调用函数时,函数名.apply(参数1,[]),apply 第一个参数是被调用函数里面的this,第二个是参数接受一个数组做为参数列表

>> 函数名.bind(参数),bind会返回一个函数,函数内部的this就是bind的参数

> 继承1

```javascript

function 子类(属性1,属性2,属性3){

父类.call(this,属性1,属性2);//属性的继承

this.属性3 = 属性3; //子类的属性

}

子类.prototype = 父类.prototype; //方法的继承

子类.prototype.方法 = function(){} //子类的方法

```

> 缺点:

> 由于方法的继承是引用赋值,所以父类也会有子类的方法

> 继承2

```javascript

function 子类(属性1,属性2,属性3){

父类.call(this,属性1,属性2);//属性的继承

this.属性3 = 属性3; //子类的属性

}

//方法的继承

for(i in 父类.prototype){

子类.prototype[i] = 父类.prototype[i];

}

子类.prototype.方法 = function(){} //子类的方法

```

> 缺点:

> 如果父类的方法是不可枚举的,子类就继承不到

> 继承3

```javascript

function 子类(属性1,属性2,属性3){

父类.call(this,属性1,属性2);//属性的继承

this.属性3 = 属性3; //子类的属性

}

//方法的继承

子类.prototype = new 父类();//子类的prototype指向父类的一个实例

子类.prototype.constructor = 子类;

子类.prototype.方法 = function(){} //子类的方法

```

> 缺点:

> 调用两次父类的构造函数,同时在子类的原型上产生了不必要的父类的属性

> 继承4

```javascript

function 子类(属性1,属性2,属性3){

父类.call(this,属性1,属性2);//属性的继承

this.属性3 = 属性3; //子类的属性

}

//方法的继承

子类.prototype = Object.create(父类.prototype);

子类.prototype.constructor = 子类;

子类.prototype.方法 = function(){} //子类的方法

```

> Object.create(参数)

> Object.create(参数)会创建一个新对象,新创建对象的\_\_proto\_\_对象就是参数

### 面向对象案例:

\* 选项卡

\* 拖拽

\* 轮播图

### 正则体验

\* 从字符串中找出数字

```javascript

str.match(/\d+/g);

```

\* 替换字符

```javascript

var reg = /is/g;

str.replace(reg,"\*\*");

var reg = /\bis\b/g;

str.replace(reg,"\*\*");

```

\* 过滤关键词

```javascript

var reg = /淘宝|京东/g;

str.replace(reg,"\*\*");

```

\* 替换html标签

```javascript

var reg = /<[^<>]+>/g;

str.replace(reg,"\*\*");

```

\* 邮箱校验

```javascript

var reg = /^\w+@[0-9a-z-]+\.[a-z]+$/i;

reg.test(value)

```

\* 是否有汉字

```javascript

var reg = /[\u4e00-\u9fa5]+/g;

reg.test(value)

```

\* 计算汉字的长度

```javascript

function getLength(str){

return str.replace(/[\u4e00-\u9fa5]/g,"aa").length;

}

```

\* URL解析

```javascript

var reg = /(\w+):\/\/([\w\.]+)\/(\w\*)/;

str.match(reg)

```

\*\*\*\*

### 什么是正则

匹配字符串的规则

### 创建正则对象

\* 方法一(构造函数): new RegExp(规则,修正模式) 如:new RegExp('a','g')

\* 方法二(字面量): /规则/修正模式 如:var reg = /a/

### 原子(正则中的最小匹配单位)

\* 可见原子-键盘输出后可以看得见的字符,如符号,英文,数字等

> 注意,以下字符具有特殊含义,如果要使用原本的字符意思,需要在前面添加\来转义

> > . \* + ? $ ^ | \ () {} []

\* 不可见原子-键盘输出后看不见的字符,如换行(\n),回车(\r),制表符(\t),垂直制表符(\v) 换页符(\f)

### 原子筛选

\* | 竖线两边任意匹配

\* [] 括号中任意一个原子,可以写区间(闭区间),如[a-z0-9]

\* [^] 括号中任意一个原子之外的原子

\* 注意:一般情况下,正则中的一个字符对应字符串中的一个字符, 例如:/ab\t/表示字符串中连续出现字符 ab制表符

### 原子集合

\* . 除了回车和换行外的任意字符 等价[^\r\n]

\* \d 任意数字,等价[0-9], \D 任意非数字 等价[^0-9]

\* \w 任意数字,字母,下划线 等价[0-9a-zA-Z\_],\W 任意非数字,字母,下划线 等价[^0-9a-zA-Z\_]

\* \s 空白符 等价于[\t\n\f\r],\S 非空白符 等价于[^\t\n\f\r]

### 量词(限定前面原子出现的次数)

\* {n} 前面原子恰好出现n次

\* {n,} 前面原子最少出现n次

\* {n,m} 前面原子最少出现n次,最多出现m次

\* \+ 前面原子最少出现1次 等价{1,}

\* ? 前面原子最少出现0次,最多出现一次 等价{0,1}

\* \\* 前面原子最少出现0次 {0,}

### 匹配模式

\* 贪婪模式-尽可能多的匹配(默认)

```javascript

'12345'.replace(/\d{2,4}/,'X');

"X5"

```

\* 非贪婪模式-尽可能少的匹配(在量词后添加?)

```javascript

'12345'.replace(/\d{2,4}?/,'X');

"X345"

```

### 边界控制

\* ^ 匹配字符串开始的位置,限制必须以其后面的原子开头，前面不能有任何东西

\* $ 匹配字符串结尾的位置，限制必须以其前面的原子结束，后面不能有任何东西

\* \b 单词边界

\* \B 非单词边界

\* () 把括号里面的正常表达匹配的结果当作一个原子来看,起到分组的效果,分组后可以通过反向引用来取得分组后的值,反向引用从$1开始

```javascript

//匹配四个一个小写字母加一个数字的字符串

'a1b2c3d4aaaa'.replace(/([a-z]\d){4}/,"\*");

"\*aaaa"

//将年月日替换成月日年

'2018-05-09'.replace(/(\d{4})-(\d{2})-(\d{2})/,'$2/$3/$1');

"05/09/2018"

```

### 修正模式

\* i(ignoreCase) 忽略大小写,默认区分大小写,可以通过正则对象上的ignoreCase属性查看

\* g(global) 匹配所有的,默认只找到第一个,可以通过正则对象上的global属性查看

### 前瞻

\* 正则从文本头部向文本尾部解析,文本尾部成为"前",前瞻就是向前(文本尾部)检查是否符合断言

\* 正向前瞻 exp(?=assert) 匹配到exp的内容后再看前面的内容符合不符合assert,符合的话就匹配exp的内容

```javascript

//字符后面是数字的字符

'a2\*3b&'.replace(/\w(?=\d)/g,'X')

X2\*3b&

```

\* 负向前瞻 exp(?!assert) 匹配到exp的内容后再看前面的内容符合不符合assert,不符合的话就匹配exp的内容

```javascript

//字符后面不是数字的字符

'a2\*3b&'.replace(/\w(?!\d)/g,'X')

"aX\*XX&"

```

### 正则对象的方法

\* 正则对象属性

> lastIndex 当前正则匹配内容的最后一个字符的下一个位置

> source 正则的文本字符串

\* RegExp.prototype.test(str);

> test方法测试字符串是否有匹配到正则的字符串,有返回true,没有返回false

> 如果全局匹配时,每一次匹配后正则对象上的lastIndex属性会更改,改为匹配内容的最后一个字符的下一个位置,所以结果会不可靠,一般使用test方法时建议不要全局匹配

> 如果非全局匹配时,lastIndex始终是0,也就是说始终从第0个元素开始查找

```javascript

var reg = /\w/g;

console.log("0::",reg.lastIndex);//0

console.log("1::",reg.test('ab'));//true

console.log("2::",reg.lastIndex);//1

console.log("3::",reg.test('ab'));//true

console.log("4::",reg.lastIndex);//2

console.log("5::",reg.test('ab'));//false

console.log("6::",reg.lastIndex);//0

```

\* RegExp.prototype.exec(str);

> exec方法对字符串执行搜索,并更新正则对象上的lastIndex属性,如果没有匹配的返回null,有的话返回一个数组:

>> \* 数组的第0个元素是匹配的内容,如果有分组,从下标是1开始是对应分组的内

>> \* index 是匹配内容第一个字符的位置

>> \* input 是被搜索的字符串

> \* 如果非全局匹配时,lastIndex始终是0,也就是说始终从第0个元素开始查找

```javascript

var reg = /(\d{4})-(\d{2})-(\d{2})/g;

var str = 'a2018-05-10b';

console.log(reg.lastIndex);//0

console.log(reg.exec(str));//["2018-05-10", "2018", "05", "10", index: 1, input: "a2018-05-10b", groups: undefined]

console.log(reg.lastIndex);//11

console.log(reg.exec(str));//null

console.log(reg.lastIndex);//0

console.log(reg.exec(str));//["2018-05-10", "2018", "05", "10", index: 1, input: "a2018-05-10b", groups: undefined]

```

### 字符串方法中使用正则

\* String.prototype.search(reg)

> \* 从字符中检索于正则相匹配的字符串,返回第一个匹配结果的index,查不到返回-1

> \* search方法不执行全局匹配,它会忽略g,并且总是从字符串开始检索

\* String.prototype.math(reg)

> \* 非全局匹配的结果和exec一样

> \* 全局匹配,会找到字符串中的所有匹配字符串,如果没有匹配的话返回null,有的话返回一个数组:

>> \* 数组中的每一项就是一个匹配结果,数组中没有index和input等属性

>> \* 全局匹配不需要分组

\* String.prototype.split(reg)

> 根据正则匹配的内容把字符串分隔成数组

```javascript

'a1b2c3d'.split(/\d/);

["a", "b", "c", "d"]

```

\* String.prototype.replace(reg,str2)

> 把字符串中匹配到正则的子串用str2进行替换

\* String.prototype.replace(reg,function(匹配结果,index,原值))

> 把字符串中匹配到正则的子串传递到回调函数进一步处理,用函数的返回值来替换匹配项

```javascript

var str = 'a1b2c3d4'.replace(/\d/g,function(match,index,origin){

return parseInt(match)+1;

})

```

### String 对象方法

\* 根据位置返回字符 str.charAt(index)

> \* 位置在0和str.length-1之间,如果不在返回空字符

> \* 一个汉子是一个字节

\* 根据位置返回字符的编码 str.charCodeAt(index)

> \* 位置在0和str.length-1之间,如果不在返回空字符

> \* 一个汉子是一个字节

> \* charCodeAt返回的是字符的Unicode码

> \* Unicode码是统一的编码,Unicode码包含了ASCII码(0-127)

\* 根据字符返回位置 str.indexOf(str1) / str.lastIndexOf(str1)

> \* indexOf 从前向后找,找不到返回-1

> \* lastIndexOf 从后向前找,找不到返回-1

\* URI 编码 encodeURIComponent(str)

\* URI 解码 decodeURIComponent(str)

\* 连接字符串 str1.concat(str2)

\* 截取字符串 slice(起始下标,结束下标),下标可以是负数,负数表示从尾部开始计算

\* 截取字符串 substr(起始下标,长度),下标可以是负数,负数表示从尾部开始计算

\* 截取字符串 substring(起始下标,结束下标),下标不可以是负数

\* 把字符串转换为小写 toLowerCase()

\* 把字符串转换为大写 toUpperCase()

### 获取表单元素

\* document.forms 代表页面所有的表单集合

\* document.forms[0].elements 代表某一个表单中的所有表单元素集合

\* 获取一个表单中的元素的方法

> 表单元素集合[下标]

>

> 表单元素集合["元素名"]

>

> 表单元素集合.元素名

>

> 表单.元素名

>

> 同名的表单元素返回一个节点集合

\* 表单元素.form 代表表单元素所在的form

### 表单元素的属性

\* 禁用表单元素

> \* 表单元素.disabled = true;

>> 针对所有表单元素

>>

>> 值不会被提交

> \* 表单元素.readOnly = true;

>> 只针对input(text/password)/textarea

>> 值会被提交

\* 复选框选中属性 checked

> \* 设置选中 aCheckBox[0].checked = true

\* 单选框选中属性 checked

> \* 设置选中 aRadio[0].checked = true

\* 下拉框选中属性 selected

> \* 设置某个选项选中 oSelect.options[i].selected = true

> \* 通过选中属性获取所有选中项

```

function getSelectedElements(oSelect){

var result = [];

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

if(oSelect.options[i].selected){

result.push(oSelect.options[i]);

}

}

return result;

}

```

### 表单元素方法

\* 设置焦点 表单元素.focus();

\* 移除焦点 表单元素.blur();

### 表单的提交和重置

\* HTML按钮默认提交

> input type="submit"

> input type="image"

> button type="submit"

\* 表单提交事件 表单.onsubmit

> \* 表单被提交,数据没有发送前触发

> \* 可以在该事件中通过阻止默认行为来阻止表单的提交

\* 给普通按钮绑定事件,调用 form的submit()方法提交

> form的submit()方法不会触发form.onsubmit事件

\* HTML按钮默认提交重置

> input type="reset"

> button type="reset"

\* 表单重置事件 表单.onrest

> \* 事件在表单重置前触发

> \* 可以通过阻止默认行为来阻止重置

\* 给普通按钮绑定事件,调用 form的reset()方法重置

> form的reset()方法不会触发form.onrest事件

### 语法错误(SyntaxError)

> 不符合JS语法的错误,控制台会出现错误并且有行号提示(行号不一定准备)

\* 常见语法错误

> \* 中文输入指令

> \* 少打,漏打,多打

> \* 变量名不合法

> \* 语句写错

\* 注意点

> \* 控制台中的错误提示中一个token就是指JS中的一个符号,例如分号,var关键字

### 运行时错误(Runtime Error)

> JS无语法错误,在运行时发生的错误

\* ReferenceError 变量引用异常触发的错误

\* TypeError 类型使用时错误

\* RangeError 递归爆栈时触发

### 逻辑错误

> 处理结果不符合预期

\* 借助控制太单步跟踪

> \* 在程序需要开始跟踪的地方打断点

>> \* 在程序中需要开始跟踪的地方添加debugger关键字

>> \* 在控制台的source面板中单击需要跟踪程序的行号

> \* Watch 需要跟踪的变量

>> \* 可以Watch一个表达式

>> \* 可以在控制台中更改变量的值

> \* Resume script execution 按钮,执行脚本到下一个断点的位置

> \* Step over next function call,跳过函数的内部执行,把函数当做一个语句来看

> \* Step into next function call,进入到函数的内部执行

> \* Step out current function ,跳出当前的到函数内部执行

> \* Deactivate breakpoints/Activate breakpoints,关闭和打开断点

> \* Pause on exceptions,异常处停止

>> \* 如果选上 Pause on caught exceptions的话会在try语句中停止

### 主动触发异常

在程序中如果有些不可避免的错误发生时,可以主动的抛出一个异常,然后用try-catch语句来捕获异常

\* 抛出异常

```javascript

throw new Error('error msg');

```

\* 捕获异常

```

try{

...//有可能出现异常的代码

}catch(e){

...//异常处理

}finally{

...//不管有没有异常都会处理,例如清理工作

}

### 闭包的使用

\* 使用函数内部的变量,将值保存在内存中

> 让函数返回一个引用了函数内部变量的函数

\* 封闭作用域,匿名函数自我调用解决变量作用域,不会污染全局变量

\* 封闭作用域传递参数,解决i值问题

> \* 子函数会使用作用域链中最近的变量,如果整个作用域中都没有定义,则报错

> \* 父函数不能访问子函数中定义的变量

\* 函数节流

> 函数节流举例

```javascript

function throttle(fn,delay){

var timer = null;

return function(){

clearTimeout(timer);

timer = setTimeout(fn,delay);

}

}

window.onresize = throttle(function(){

console.log(window.innerWidth);

},400)

```

\* 利用闭包和原型构建对象

### 什么是设计模式

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

### 观察者模式(发布订阅者模式)

\* 发布者维护一个数组,该数组用来存储所有关注者对象

\* 发布者提供一个订阅者可以加入数组的方法

\* 发布者发布消息时循环遍历订阅者数组，调用订阅者的监听方法

\* 订阅者在创建时调用发布者的加入数组方法

\* 订阅者需要提供监听方法供发布者调用传递消息

案例1:发布作业

案例2:汇率计算

### 单例模式

保证一个类仅有一个实例,并且提供一个访问它的全局访问点

案例1:皇帝上朝

案例2:惰性登录框

\* 遍历

```javascript

arr.forEach(function(value,index){

console.log(value);

})

```

> 遍历不会生成新的数组,没有返回值

\* 映射

```javascript

var newArr = arr.map(function(value,index){

return value\*10;

})

```

> 遍历数组,把函数的每一次返回值映射成新的数组并返回

\* 过滤

```javascript

var newArr = arr.filter(function(value,index){

return value > 10;

})

```

> 遍历数组,把函数的每一次返回值为真的原数组中的值组成新的数组并返回

\* 全真判断

```javascript

var bRes = arr.every(function(value,index){

return value > 10;

})

```

> 遍历数组,所有值为真的情况下返回真,否则返回假

\* 有真判断

```javascript

var bRes = arr.some(function(value,index){

return value > 10;

})

```

> 遍历数组,有一个值为真的情况下返回真,否则返回假

### 什么是HTTP

HTTP(HyperText Transfer Protocol)超文本传输协议

### 什么是协议

协议就是一套规则，用来规范/约束某一类事物

例如租房协议/买卖协议/离婚协议

HTTP协议就是用来规范客户端(浏览器)和服务端通信规则的

> 常见的网络协议还有很多，如

> \* FTP(File Transfer Protocol)文件传输协议

> \* POP3(Post Office Protocol Version 3)邮局协议-版本3

> \* SMTP(Simple Mail Transfer Protocol)简单邮件传送协议

> \* TCP/IP(Transmission Control Protocol/Internet Protocol)传输控制协议/Internet协议

### 什么是HTTPS

HTTPS（全称：Hyper Text Transfer Protocol over Secure Socket Layer）,安全套接字层超文本传输协议HTTPS。为了数据传输的安全，HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

### OSI(Open System Interconnection)开放式系统互联的七层架构

\* 1 物理层(设备之间的比特流传输,物理接口)

\* 2 数据链路层(成帧,用MAC地址访问媒介)

\* 3 网络层(写入IP地址(数据源和数据目标的地址),选路)

\* 4 传输层(确定是用TCP协议传输还是UDP协议传输)

> \* 传输控制协议TCP可靠但是慢

> \* 用户数据协议UDP不可靠但是快

\* 5 会话层(安全协议,目录访问协议)

\* 6 表示层(转码/解码,加密/解密)

\* 7 应用层(提供应用数据,各种应用协议,如HTTP,FTP,SMTP...)

> 发送从应用层开始向下传输,接受从物理层开始向上传输

### TCP/IP 4层模型

\* 网络接口层(对应7层中的物理层和数据链路层)

\* 网际互联层(对应7层中的网络层)

\* 传输层(对应7层中的传输层)

\* 应用层(对应7层中的应用层,表示层和会话层)

### 访问页面流程详解

\* 客户在浏览器输入网址或者刷新页面

\* 域名解析(以http://www.kuazhu.com/为例)

1 浏览器搜索自身的DNS缓存

查看浏览器缓存 chrome://net-internals/#dns

2 如果浏览器缓存没有或者过期，就去操作系统自身的DNS缓存查找

3 读取本地的HOST文件

windows：C:\Windows\System32\drivers\etc

OS:/etc/hosts

4 向宽带运营商服务器发起域名解析请求,宽带运营商服务器查看自身的缓存

5 运营商服务器发起一个迭代的DNS解析请求

5.1 找根域的服务器

5.2 根域服务器里面有com域的地址,根域让去找com域的服务器

5.3 com域服务器里面有kuazhu.com域的IP地址,com域服务器让去找kuazhu.com域的服务器

5.4 kuazhu.com域的服务器(一般是域名注册商)里面有对应的IP地址和域名的映射,找到后将IP地址发给运营商服务器

5.5 运营商服务器把结果返回给客户操作系统,同时操作系统将其缓存起来,

操作系统把结果返回给浏览器

\* 浏览器获取IP地址后,发起TCP/IP 三次握手

第一次握手：客户端发送syn包（syn=j）到服务器，并进入SYN\_SENT状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED（TCP连接成功）状态，完成三次握手。

完成三次握手，客户端与服务器开始传送数据

\* TCP/IP连接建立后，开始发送HTTP请求

浏览器发送请求行到服务器(请求行包括请求方法 uri 协议/版本等)

浏览器发送请求头到服务器,结束后发送一个空白行告诉服务器不再有请求头(请求头包含许多有关的客户端环境信息)

浏览器发送请求体到服务器(请求体主要是请求参数)

服务器应答处理并且返回响应行(响应行包括协议/版本 状态码 状态消息)

服务器返回响应头,结束后发送一个空白行告诉客户端不再有响应头(响应头包含许多有关的服务端环境信息)

服务器返回响应体(响应体主要是返回的数据)

关闭TCP连接

\* 浏览器渲染服务器返回的内容

\* HTTP一旦请求完成,连接就会断开,因此连接是没有记忆的,无状态的

### 在浏览器中看请求响应的信息-Timing

Queued at

添加到处理队列的时间

Started at

开始处理的时间

Stalled 浏览器要发出请求到这个请求可以发出的等待时间，一般是代理协商、以及等待可复用的TCP连接释放的时间，不包括DNS查询、建立TCP连接等时间等

DNS Lookup

执行DNS查找时间

Initial connection

建立TCP连接的时间，就相当于客户端从发请求开始到TCP握手结束这一段，包括DNS查询+Proxy时间+TCP握手时间。

Request sent:请求的第一个字节发出前到最后一个字节发出后的时间

Waiting：请求发出后到收到响应的第一个字节所耗费的时间，包括数据的传输和服务器对请求的处理时间

Content Download:收到响应的第一个字节开始到收到响应的最后一字节结束所耗费的时间

### HTTP的请求方法

GET方法

用于获取数据

请求是参数的大小有限制(400k)

参数以及参数的值会显示在地址栏中，安全性极低

POST方法

用于提交数据(表单的提交)

请求是参数的大小基本没有限制

参数不会显示在地址栏中，相对安全

### 什么是URL?

uniform resource locator 统一资源定位器

可以理解为地址栏中的网址

### URL的格式

完整格式：https://www.kuazhu.com:443/courseList

协议://域名:端口号/资源文件

### 常见状态码

1xx-信息提示

100 - 继续。

101 - 切换协议。

2xx - 成功

这类状态代码表明服务器成功地接受了客户端请求。

200 - 确定。客户端请求已成功。

201 - 已创建。

202 - 已接受。

203 - 非权威性信息。

204 - 无内容。

205 - 重置内容。

206 - 部分内容。

207 - 多状态 (WebDAV)。

3xx - 重定向

客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。

301 - 已永久移动

302 - 对象已移动。

304 - 未修改。

307 - 临时重定向。

4xx - 客户端错误

发生错误，客户端似乎有问题。例如，客户端请求不存在的页面，客户端未提供有效的身份验证信息。

400 - 错误的请求。

401 - 访问被拒绝。IIS 定义了几个不同的 401 错误，用于指示更为具体的错误原因。这些具体的错误代码在浏览器中显示，但不在 IIS 日志中显示：

401.1 - 登录失败。

401.2 - 服务器配置导致登录失败。

401.3 - 由于 ACL 对资源的限制而未获得授权。

401.4 - 筛选器授权失败。

401.5 - ISAPI/CGI 应用程序授权失败。

401.7 – 由 Web 服务器上的 URL 验证策略拒绝访问。这个错误代码为 IIS 6.0 所专用。

403 - 禁止访问：IIS 定义了几个不同的 403 错误，用于指示更为具体的错误原因：

403.1 - 执行访问被禁止。

403.2 - 读访问被禁止。

403.3 - 写访问被禁止。

403.4 - 要求 SSL。

403.5 - 要求 SSL 128。

403.6 - IP 地址被拒绝。

403.7 - 要求客户端证书。

403.8 - 站点访问被拒绝。

403.9 - 用户数过多。

403.10 - 配置无效。

403.11 - 密码更改。

403.12 - 拒绝访问映射表。

403.13 - 客户端证书被吊销。

403.14 - 拒绝目录列表。

403.15 - 超出客户端访问许可。

403.16 - 客户端证书不受信任或无效。

403.17 - 客户端证书已过期或尚未生效。

403.18 - 在当前的应用程序池中不能执行所请求的 URL。这个错误代码为 IIS 6.0 所专用。

403.19 - 不能为这个应用程序池中的客户端执行 CGI。这个错误代码为 IIS 6.0 所专用。

403.20 - Passport 登录失败。这个错误代码为 IIS 6.0 所专用。

404 - 未找到。

404.0 -（无） – 没有找到文件或目录。

404.1 - 无法通过请求的端口访问网站。

404.2 - Web 服务扩展锁定策略阻止本请求。

404.3 - MIME 映射策略阻止了此请求。

405 - 用来访问本页面的 HTTP 谓词不被允许（方法不被允许）

406 - 客户端浏览器不接受所请求页面的 MIME 类型。

407 - 要求进行代理身份验证。

412 - 前提条件失败。

413 – 请求实体太大。

414 - 请求 URI 太长。

415 – 不支持的媒体类型。

416 – 无法满足请求的范围。

417 – 执行失败。

423 – 锁定的错误。

5xx - 服务器错误

服务器由于遇到错误而不能完成该请求。

500 - 内部服务器错误。

500.12 - 应用程序正忙于在 Web 服务器上重新启动。

500.13 - Web 服务器太忙。

500.15 - 不允许直接请求 Global.asa。

500.16 – UNC 授权凭据不正确。这个错误代码为 IIS 6.0 所专用。

500.18 – 无法打开 URL 授权存储库。这个错误代码为 IIS 6.0 所专用。

500.19 - 此文件的数据在元数据库中配置不正确。

500.100 - 内部 ASP 错误。

501 - 页眉值指定了未实现的配置。

502 - Web 服务器用作网关或代理服务器时收到了无效响应。

502.1 - CGI 应用程序超时。

502.2 - CGI 应用程序出错。

503 - 服务不可用。这个错误代码为 IIS 6.0 所专用。

504 - 网关超时。

505 - HTTP 版本不受支持。

### 服务器

安装了Web服务器软件的电脑就可以提供管理网站的服务,我们称之为Web服务器。

常见的Web服务器软件有:Apache,IIS,Tomcat,Nginx,NodeJs等

### NodeJs 初识

\* 官网 https://nodejs.org/ 下载安装,注意要下载LTS版本的

\* NodeJs安装完成后,去终端用node -v 查看,如果有版本号说明安装成功

\* 如果node命令找不到,Windows的用户注意环境变量的配置

\* NodeJs安装的同时会安装npm(node package manager)

### ajax请求流程：

\* 客户端创建一个ajax对象

\* 用ajax对象和服务端建立连接

\* 用ajax对象发送请求到服务端

\* 用ajax对象接收服务端返回的数据

```javascript

function ajax(method,url,data,fnSucc,fnFaild){

// 1.创建一个ajax对象

var oAjax = new XMLHttpRequest();

//post方法处理

if(method.toLowerCase() == 'post'){

oAjax.open(method,url,true);

//设置请求头,该方法必须在open和send中间调用

oAjax.setRequestHeader("Content-type","application/x-www-form-urlencoded");

//post发送的数据

oAjax.send(data);

}else{

if(data){

url = url+"?"+data;

}

// 2.用ajax对象建立连接

// oAjax.open(请求方法，请求地址，是否异步)

oAjax.open(method,url,true);

// 3.用ajax对象发送请求

oAjax.send();

}

// 4.用ajax对象接收数据

oAjax.onreadystatechange = function(){

// console.log(oAjax.readyState);

// 0 还没有初始化

// 1 已经初始化，但是还没有调用open方法

// 2 send方法已经调用

// 3 返回了部分数据

// 4 数据完全返回

if(oAjax.readyState == 4){//请求完成，表明所有的数据已经返回，但并不代表成功

if(oAjax.status == 200){//成功

fnSucc(oAjax.responseText);

}else{//失败

if(fnFaild){

fnFaild(oAjax.status);

}

}

}

}

}

```

### 跨域方案

\* 请求端的协议,域名,端口和服务器的协议,域名,端口有一个不一致就会发生跨域

\* CORS(Cross-origin resource sharing)跨域方案

> 服务器端设置响应头

```javascript

res.setHeader("Access-Control-Allow-Origin","\*")

```

> 注意:

> \* Access-Control-Allow-Origin设置可以跨域的域名

> \* \\*代表所有

> \* 一次只能只能设置一个值

\* 简单请求同时满足条件

> \* 请求方法必须是 HEAD,GET,POST之一

> \* 请求头中的字段仅限于 Accept,Accept-Language,Content-Language,Last-Event-ID,Content-Type

> \* Content-Type 的值仅限于application/x-www-form-urlencoded,multipart/form-data,text/plain

\* 复杂请求

不满足简单请求的时候就是复杂请求,复杂请求会先发一次OPTIONS方法进行预检(preflight)请求

例如设置如下请求头后该请求就会变为复杂请求

oAjax.setRequestHeader('abc','abc');

需要在服务器端设置如下响应头才能实现请求

```javascript

res.setHeader("Access-Control-Allow-Headers","abc");

```

> Access-Control-Allow-Headers 表示服务端接受的跨域请求的字段

\* 服务器端响应头Access-Control-Expose-Headers

> Access-Control-Expose-Headers表示允许客户端通过getResponseHeader方法获取的字段

> CORS方式下默认只能获取6个基础字段,Cache-Control,Content-Language,Content-Type,Expires,Last-Modified,Pragma

### 什么是jQuery?

jQuery(js+Query)是一款优秀的JavaScript库,帮助开发人员用最少的代码做更多的事情,官网网站http://jquery.com/

### 为什么学习jQuery？

\* jQuery让元素选取,事件处理,动画,DOM操作,属性操作,Ajax变得简单,通过链式调用和隐形迭代让代码更加简洁,更重要的是他可以几乎兼容所有的浏览器

\* 几乎所有的巨头都在使用jQuery

\* 丰富的插件支持和强大的生态体系

\* 解读优秀框架,提高编程的视野

### jQuery学什么？

\* 熟悉jQuery的方法

\* 理解jQuery的原理

### 下载jQuery和版本选择

\* 下载地址:https://code.jquery.com/

\* 版本选择

> \* 1.x: 兼容ie678,但相对其它版本文件较大

> \* 2.x: 不兼容ie678,比1.x的文件小

> \* 3.x: 不兼容ie678,比1.x的文件小

> \* 由于兼容性的问题,国内大部分网站依旧选择1.x的版本,所以建议使用1.x

\* 压缩版和源码版

> \* 压缩版,将源代码进行了压缩,文件体积更小,项目上线时为了提高页面的加载速度使用

> \* 源码版,没有压缩,文件体积相对大,开发学习时为了理解jQuery的源码使用

### jQuery的引入

\* 方法一:将jQuery下载到本地,通过script标签引入到要使用的页面

```javascript

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

```

\* 方法二:通过script标签引入CDN的资源

```javascript

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

```

\* 由于CDN资源需要从网络上获取,所以在开发阶段使用本地文件,线上产品可以使用CDN资源

\* 在线上的项目为了确保jquery被引入,可以使用判断的方法引入

```javascript

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

<script>

window.jQuery || document.write('<script src="js/jquery.js"><\/script>');

</script>

```

### jQuery是使用

\* 方法一

```javascript

//当页面DOM结构加载完毕后执行回调函数

jQuery(document).ready(function(){

//jq代码1...

});

jQuery(document).ready(function(){

//jq代码2...

});

```

> 注意:

> > \* 上述jq代码1和jq代码2都会被执行,不会覆盖

> > \* 不会覆盖的原因是jQuery函数是一个闭包,传递进去的函数都有自己的作用域

\* jQuery(document).ready() 和 window.onload的区别

window.onload必须等到页面全部加载完成然后执行,并且只能执行一次,jQuery(document).ready()只需页面中的DOM结构加载完毕就执行,可以执行多次

\* 方法二

```javascript

//当页面DOM结构加载完毕后执行回调函数

$(document).ready(function(){

//jq代码1...

});

$(document).ready(function(){

//jq代码2...

});

```

> 注意:

>> \* $和jQuery是jQuery框架通过window对象暴露出来的全局变量,并且$=jQuery

>> \* jQuery内部实现:window.jQuery = window.$ = jQuery;

>> \* 用$代表jQuery是为了提高效率

\* 方法三

```javascript

//当页面DOM结构加载完毕后执行回调函数

$(function(){

//jq代码1...

});

$(function(){

//jq代码2...

});

```

> 注意:

>> \* 方法三是方法二的简写形式

>> \* 方法三中的$可以用jQuery代替

### $冲突解决

当其他框架也使用了$时就会和jQuery的$发生冲突,解决方法有

\* 方法一:在使用jQuery前释放jQuery的$使用权,用jQuery来代替$

```javascript

jQuery.noConflict();

jQuery(function(){

});

```

\* 方法二:在使用jQuery前释放jQuery的$使用权的同时自定义自己的符号,用自定义符号来代替$

```javascript

var kz = jQuery.noConflict();

kz(function(){

});

```

### jQuery的核心函数

\* 接收一个函数,返回一个包含了document的jQuery对象

\* 接收一个字符串

> \* 字符串选择器,返回一个包含了DOM元素的jQuery对象

> \* 字符串的HTML代码片段,返回一个包含了DOM元素的jQuery对象

\* 接收一个DOM元素,返回一个包含了DOM元素的jQuery对象

### jQuery对象

jQuery对象是一个伪数组

### jQuery对象和DOM对象转换

\* $(DOM对象)会得到包含了DOM元素的jQuery对象

> \* 原生的DOM元素不能使用jQuery的方法,如果需要使用jQuery对象,需要转换

\* jQuery对象.get(index) 或者 jQuery对象[index] 获取对应下标的DOM元素

### jQuery对象的静态方法

\* $.each(数组,回调) 遍历数组或者伪数组,返回被遍历的数组

```

$.each(arr,function(index,value){

})

```

\* $.map(数组,回调) 映射数组或者伪数组,返回映射后的数组

```

$.map(arr,function(value,index){

})

```

\* $.trim(str) 去除字符串两边的空格并返回

\* $.isWindow(arg) 判断参数对象是不是window对象

\* $.isArray(arg) 判断参数对象是不是真数组

\* $.isFunction(arg) 判断参数对象是不是函数

\* $.holdReady(布尔值) 参考为true时暂停jQuery入口函数的执行,参数为false时恢复jQuery入口函数的执行

### 选择器规则参考CSS中的选择器

\* $('CSS中的选择器');

### 过滤器

\* 基本过滤器

> $(':eq(index)') 选择索引对应的DOM元素

>> 可以用jQuery对象的index()方法获取当前jQuery对象的索引

\* 内容过滤器

> $('选择器:empty') 选择的元素中没有文本内容和子元素的

> $('选择器:parent') 选择的元素中有文本内容或者子元素的

> $('选择器:contains(文本内容)') 选择的元素中包含有指定内容的

> $('选择器:has(选择器)') 选择的元素中有指定的选择器的

\* 过滤方法

> jQuery对象.eq(index) 选择索引对应的DOM元素

> jQuery对象.siblings() jQuery对象中包含元素之外的兄弟元素

\* 注意:以上方法都返回jQuery对象

### 属性和属性节点

\* 属性就是添加在对象(自定义对象或者DOM对象)上的变量

\* 属性节点是HTML标签中指定的属性

> 属性节点可以watch

### 原生JS操作操作属性节点

\* 获取属性节点的值

> DOM对象.getAttribute("属性节点名称")

> DOM对象.属性节点名称,注意class用className

> DOM对象.["属性节点名称"]

> 注意:后面两种方法只适合DOM对象原有的属性

\* 设置属性节点的值

> DOM对象.setAttribute("属性节点名称", "值");

> DOM对象.属性节点名称 = 值

> DOM对象.["属性节点名称"] = 值

> 注意:后面两种方法只适合DOM对象原有的属性

\* 删除属性

> DOM对象.removeAttribute("属性节点名称")

### jQuery对象方法操作属性

\* 获取

> jQuery对象.attr("属性节点名称") 或者 jQuery对象.prop("属性节点名称")

> 注意:只能获取第一个元素的值

\* 设置

> jQuery对象.attr("属性节点名称","值1 值2...") 或者 jQuery对象.prop("属性节点名称","值1 值2")

\* 删除

> jQuery对象.removeAttr("属性节点名称") 或者 jQuery对象.removeProp("属性节点名称")

> 注意:删除所有元素的值

\* attr和prop的区别

> \* 属性节点的值具有true和false时用prop,如checked,selected,disabled,其它的属性节点用attr

> \* 属性节点的值具有true和false时,没有添加该属性时attr会返回undefined,prop返回true

### 操作class

\* 添加

> jQuery对象.addClass("class1 class2...")

\* 删除

> jQuery对象.removeClass("class1 class2...")

\* 切换(存在就删除不存在就添加)

> jQuery对象.toggleClass("class1 class2...")

### 操作代码/文本/值

\* jQuery对象.html([val])

> 设置或者获取元素中的HTML,相当于innerHTML

\* jQuery对象.text([val])

> 设置或者获取元素中的HTML,相当于innerText

\* innerHTML和innerText的区别

> \* 获取HTML内容包括HTML标签和文本，而获取文本内容只有标签里面的文本

> \* 设置带有标签的HTML内容,标签会起作用，而设置带有标签的文本内容，标签会被当做文本来显示

\* jQuery对象.val([val])

> 设置或获取元素value属性的值

### 操作css

\* 获取 jQuery对象.css("样式名称"）

\* 设置

> \* jQuery对象.css("样式名称", "值");

> \* jQuery对象.css("样式名称", "值").css("样式名称", "值");

> \* jQuery对象.css({"样式名称1":"值1","样式名称2":"值2"});

### 操作尺寸

\* jQuery对象.width([val]) jQuery对象.height([val])

> 设置或获取元素的宽高

\* jQuery对象.innerWidth([val]) jQuery对象.innerHeight([val])

> 设置或获取元素的宽高,包含内边距

\* jQuery对象.outerWidth([val]) jQuery对象.outerHeight([val])

> 设置或获取元素的宽高,包含边框和内边距

### 操作位置

\* jQuery对象.offset([对象])

> \* 设置或获取元素相对于body的偏移位

> \* offset返回不传参时返回一个包含偏移值对象

> \* offset传值需要传一个包含偏移值的对象

\* jQuery对象.position()

> \* 获取元素相对于父级定位元素的偏移位,返回一个包含偏移值对象

\* jQuery对象.scrollTop([val])

> 设置或获取元素垂直滚动距离

\* jQuery对象.scrollLeft([val])

> 设置或获取元素水平滚动距离

### 事件的绑定

\* 方法一: jQuery对象.事件名(function(){})

$('.btn').click(function(){})

\* 方法二: jQuery对象.on('事件名',function(){})

$('.btn').on('click',function(){})

\* 注意点:

> \* 方法一部分事件jQuery没有实现,方法二中可以写js中所有的事件

> \* 无论是方法一还是方法二,绑定的事件都会不会覆盖

### 事件的移除

\* jQuery对象.off() 移除所有的事件

\* jQuery对象.off('事件名') 移除所有事件名对应的事件

\* jQuery对象.off('事件名',函数名) 移除事件名对应的函数名的相关事件

### 阻止事件冒泡

\* 方法一 return false

\* 方法二 event.stopPropagation()

### 阻止默认行为

\* 方法一 return false

\* 方法二 event.preventDefault()

### 事件自动触发

\* jQuery对象.trigger('事件名') 不阻止事件冒泡和默认行为的事件触发

> 注意a标签用trigger不阻止事件冒泡,但是会阻止默认行为,如果需要发生默认行为需要触发a标签的子元素

\* jQuery对象.triggerHandler('事件名') 阻止事件冒泡和默认行为的事件触发

### 自定义事件

\* 自定义事件的方法

> \* 用jQuery对象.on('自定义事件名',function(){})绑定一个自定义事件

> \* 用jQuery对象.trigger('自定义事件名')来触发自定义的事件

### 事件命名空间

\* 事件命名空间的使用

> \* 用jQuery对象.on('事件名.名称空间',function(){})绑定一个自定义事件

> \* 用jQuery对象.trigger('事件名.名称空间')来触发自定义的事件

> \* 注意点:

>> \* 用trigger触发子元素有命名空间的事件时,父元素相同相同命名空间的相同事件会触发,没有命名 空间的事件不会触发

>> \* 用trigger触发子元素没有命名空间的事件时,子元素所有相同类型的事件和父元素所有相同类型的事件都会被触发

### 事件的委托

\* 事件委托的方式:$('父元素选择器').delegate('子元素选择器','事件名', 回调函数)

> 注意点：

>> \* 回调函数中的this是触发事件的元素,及具体的某个子元素

>> \* 页面中动态增加的元素需要通过事件委托的方式来添加事件

### 移入移出事件

\* mouseover/mouseout 移入移出子元素会触发

\* mouseenter/mouseleave 移入移出子元素不会触发

\* 用jQuery对象.hover(回调1,回调2)

> \* 移入执行回调1,移出执行回调2

> \* 如果只传一个回调函数,移入移出都执行该回调

> \* 移入移出子元素不会触发

### 显示和隐藏

\* jQuery对象.show(时长,回调)

\* jQuery对象.hide(时长,回调)

\* jQuery对象.toggle(时长,回调)

### 卷入卷出

\* jQuery对象.slideDown(时长,回调)

\* jQuery对象.slideUp(时长,回调)

\* jQuery对象.slideToggle(时长,回调)

### 淡入淡出

\* jQuery对象.fadeIn(时长,回调)

\* jQuery对象.fadeOut(时长,回调)

\* jQuery对象.fadeToggle(时长,回调)

\* jQuery对象.fadeTo(时长,淡入到值,回调)

### 自定义动画

\* jQuery对象.animate(需要动画的属性对象,时长,回调)

> \* 需要动画的属性对象可以使用累加累减动画

> \* 多个属性写在一起可以执行同步动画

### 列队动画

\* 多个动画链式调用,后面的动画会等待前面的动画执行完毕

\* 在动画方法的回调中再执行动画,回调中的动画会等待前面的动画执行完毕

\* 多个动画按调用的顺序执行

### 动画方法

\* 停止动画 jQuery对象.stop(是否清空未执行完的列队动画,是否跳到正在执行动画的最后状态);

> \* 如果不传参数,停止当前正在执行的动画,然后把后续的动画执行完毕

> \* 第一个参数默认是false,表示不清空,如果是true,后续动画不执行

> \* 第二个参数默认是false,表示停止在当前状态,如果是ture,则跳到正在执行动画的最后状态

\* 延迟动画 jQuery对象.delay(时长);

> 延迟动画只对后面的动画起作用

### 动画的速度

以上动画在回调前面可以指定速度,默认是'swing',可用参数"linear"

### 添加节点

\* 内部插入

> \* 将子节点插入到父节点内部后面

>> \* 父节点jQuery对象.append(子节点jQuery对象)

>> \* 子节点jQuery对象.appendTo(父节点jQuery对象)

> \* 将子节点插入到父节点内部前面

>> \* 父节点jQuery对象.prepend(子节点jQuery对象)

>> \* 子节点jQuery对象.prependTo(父节点jQuery对象)

\* 外部插入

> \* jQuery对象1.after(jQuery对象2) jQuery对象1中的节点后面插入jQuery对象2中的节点

> \* jQuery对象1.before(jQuery对象2) jQuery对象1中的节点前面插入jQuery对象2中的节点

### 删除节点

\* jQuery对象.remove() 删除jQuery对象中的节点同时删除事件

\* jQuery对象.empty() 清空jQuery对象总节点的内容

\* jQuery对象.detach() 删除jQuery对象中的节点但保留事件

### 替换节点

\* jQuery对象1.replaceWith(jQuery对象2|HTML代码) 用jQuery对象2中的节点替换jQuery对象1中的节点

\* jQuery对象1.replaceAll('选择器') 用jQuery对象1中节点替换掉所有选中的节点

### 复制节点

jQuery对象.clone(false|true) 复制一个节点,默认是false不会复制事件,true会复制事件

### ajax接口

```javascript

$.ajax({

url:'./data.json', //请求的地址

type:'GET',//指定请求的方法,默认是GET

data:{name:'Tom',age:18},//发送到服务器的数据，字符串键值对或者对象

dataType:'json',//返回数据的类型,可以是xml json script text html,默认是text

success:function(返回数据,状态信息,jqXHR对象){},//请求成功后的回调函数

error:function(jqXHR对象,状态信息){},//请求失败后的回调函数

complete:function(jqXHR对象,状态信息){},//请求完成后的回调函数

timeout:3000, //设置请求超时的时间,单位为毫秒

statusCode:{304:function(){},503:function(){}.....}//设置不同状态码的回调函数

headers:{test:'test'}//设置请求头

})

```

### 接口封装

\* $.get(url[,参数][,回调函数][,返回数据类型])//发送GET请求

\* $.post(url[,参数][,回调函数][,返回数据类型]) //发送POST请求

\* $.getScript(url[,回调函数]) //获取JS代码

\* $.getJSON(url[,回调函数]) //获取JSON数据

\* jQuery对象.load(url[,参数][,回调函数]) //将服务器返回的数据加载到jQuery对象的DOM元素中

> 所有的回调函数接收的参数依次为 返回数据,返回状态信息(成功为success 失败为error),jqXHR对象

### 全局配置

\* $.ajaxSetup(对象参数) //给所有的ajax请求设置默认的初始化参数

### 全局回调

\* $(document).ajaxStart(回调函数),请求开始,未完成其它请求时执行回调函数

\* $(document).ajaxStop(回调函数),请求结束,所有请求结束时执行回调函数

\* $(document).ajaxSuccess(回调函数),当有请求成功时执行回调函数

\* $(document).ajaxComplete(回调函数),当有请求完成时执行回调函数

\* $(document).ajaxError(回调函数),当有请求失败时执行回调函数

\* $(document).ajaxSend(回调函数),当有请求发送请求时执行回调函数

> 全局回调一般写在ajax请求前面,如果某个ajax不想触发全局事件,可以设置 global:false来取消

### jqXHR对象

\* jqXHR对象是浏览器原生XHR对象的一个超集

\* 可以通过调用jqXHR对象上的方法来实现连缀调用,常见的方法有:

> \* .done(回调函数) 请求成功时执行,回调函数的参数依次为 返回数据,状态信息,jqXHR对象

> \* .fail(回调函数) 请求失败时执行,回调函数的参数依次为 jqXHR对象,状态信息

> \* .always(回调函数) 无论成功还是失败都执行,如果成功,回调函数的参数和done的一样,如果失败，回调函数的参数和fail的一样

###扩展自己的插件

\* 类级别的

```javascript

$.myFunc = function(){

}

```

```javascript

$.extend({

myFunc:function(){

}

})

```

```javascript

var obj = {

myFunc:function(){

}

};

$.tom = obj;

\* 对象级别的,本质就是在jQuery对象的原型对象上增加方法

```javascript

$.fn.extend({

colorToRed:function(){

this.css({

color:'red'

});

return this;

}

})

````

```javascript

;(function($){

$.fn.tab = function(options){

//默认的配置属性

var defaults = {

}

//将默认属性和传入的属性合并

var options = $.extend(defaults,options);

this.each(function(){

//具体的逻辑实现

})

}

})(jQuery)

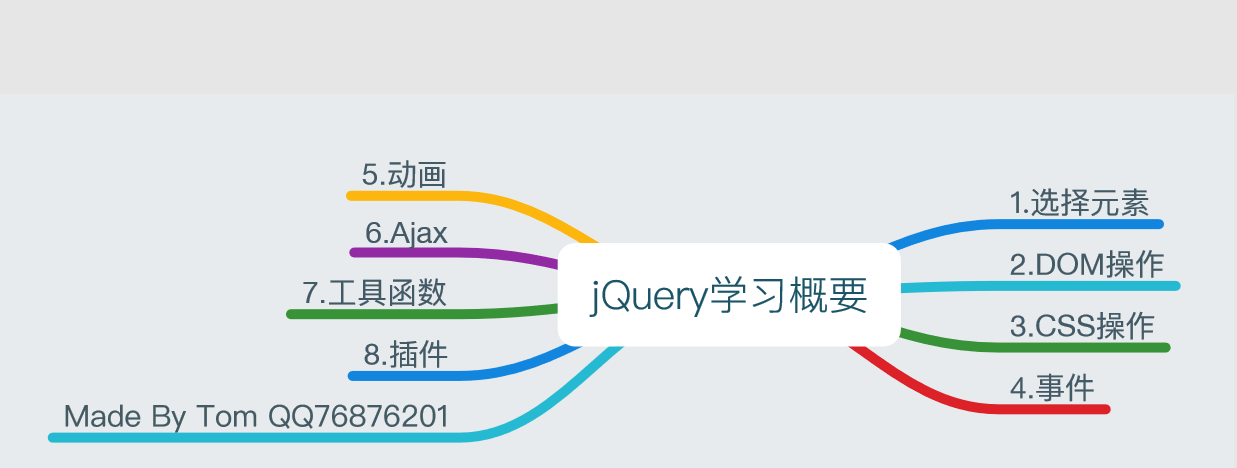
````

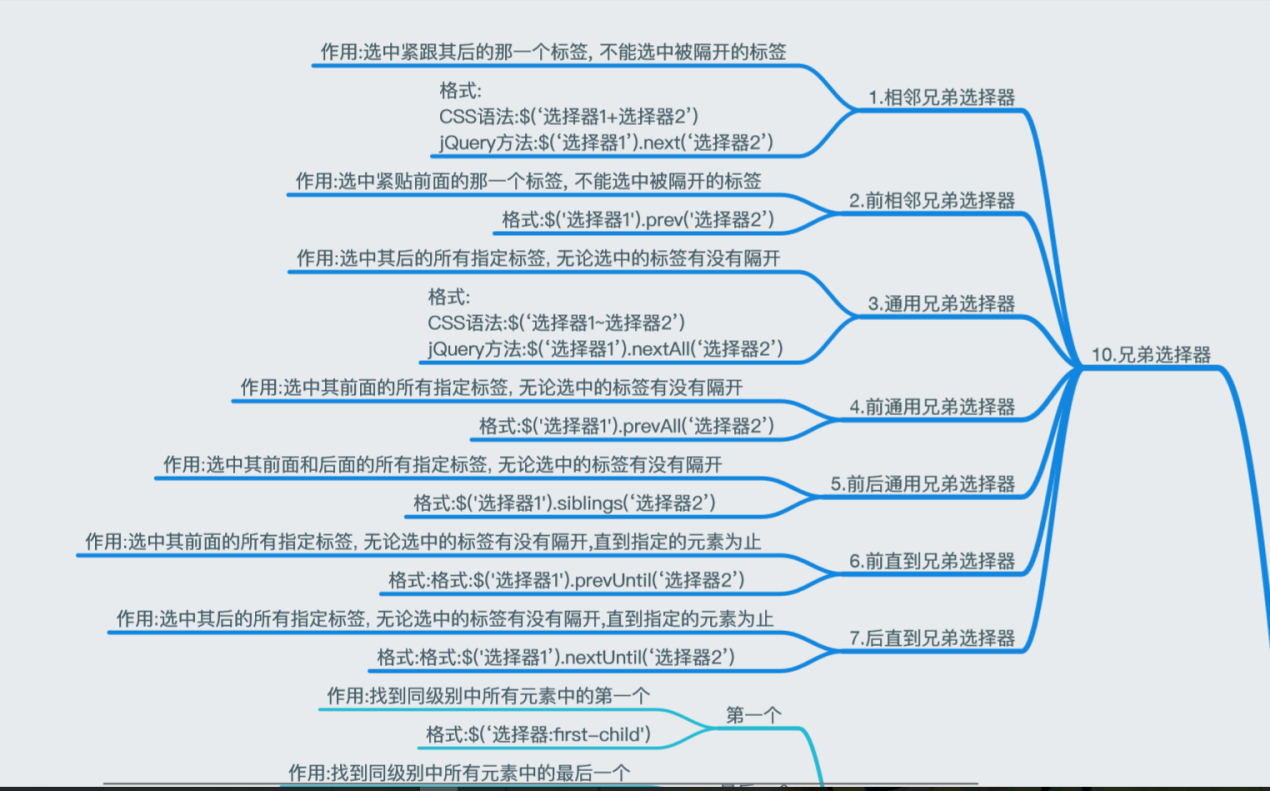
###使用已有的插件

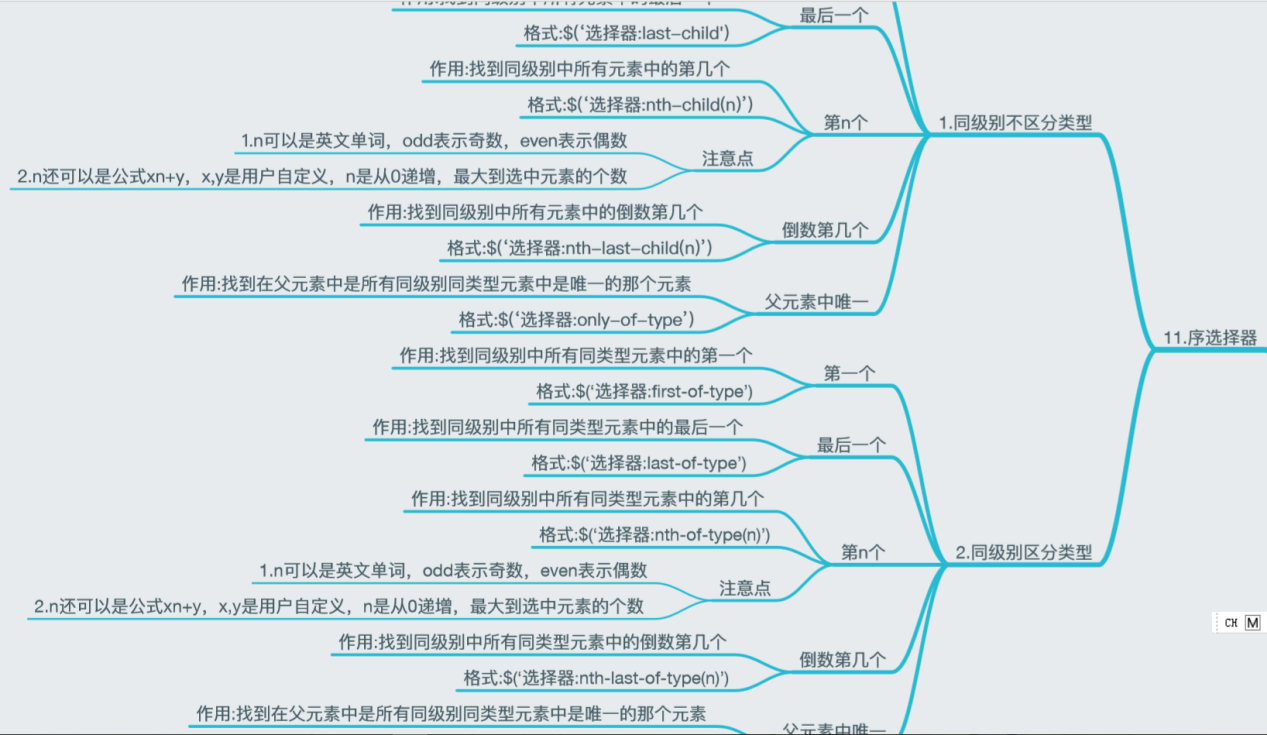
\* 电子书插件 turnjs(http://www.turnjs.com/)

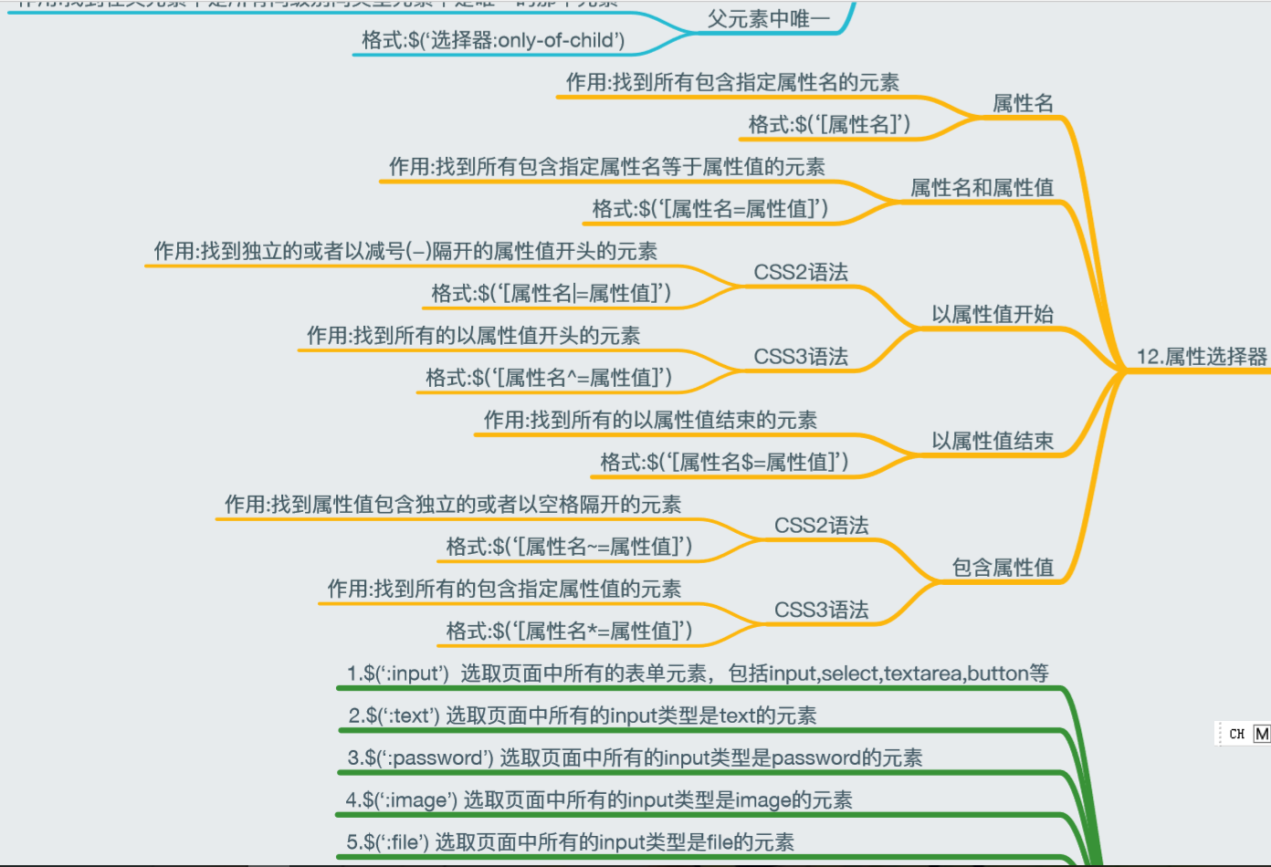
\* 表单验证插件 jQuery validation(https://jqueryvalidation.org/)

\* 全屏加载插件 FullPage (https://alvarotrigo.com/fullPage/)



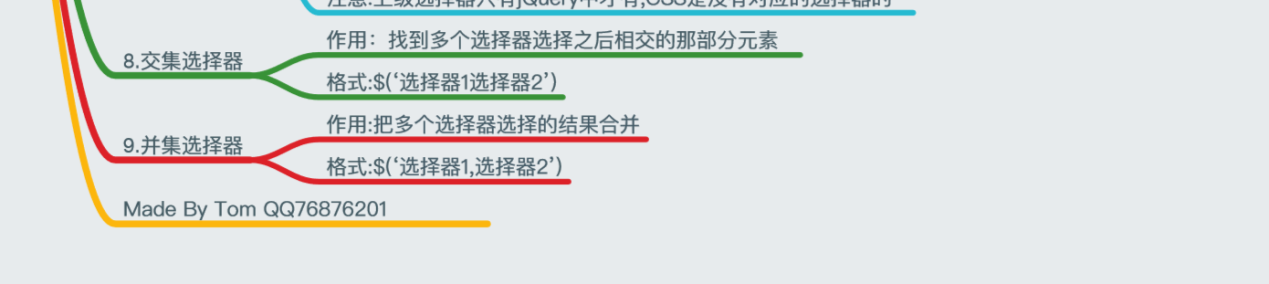


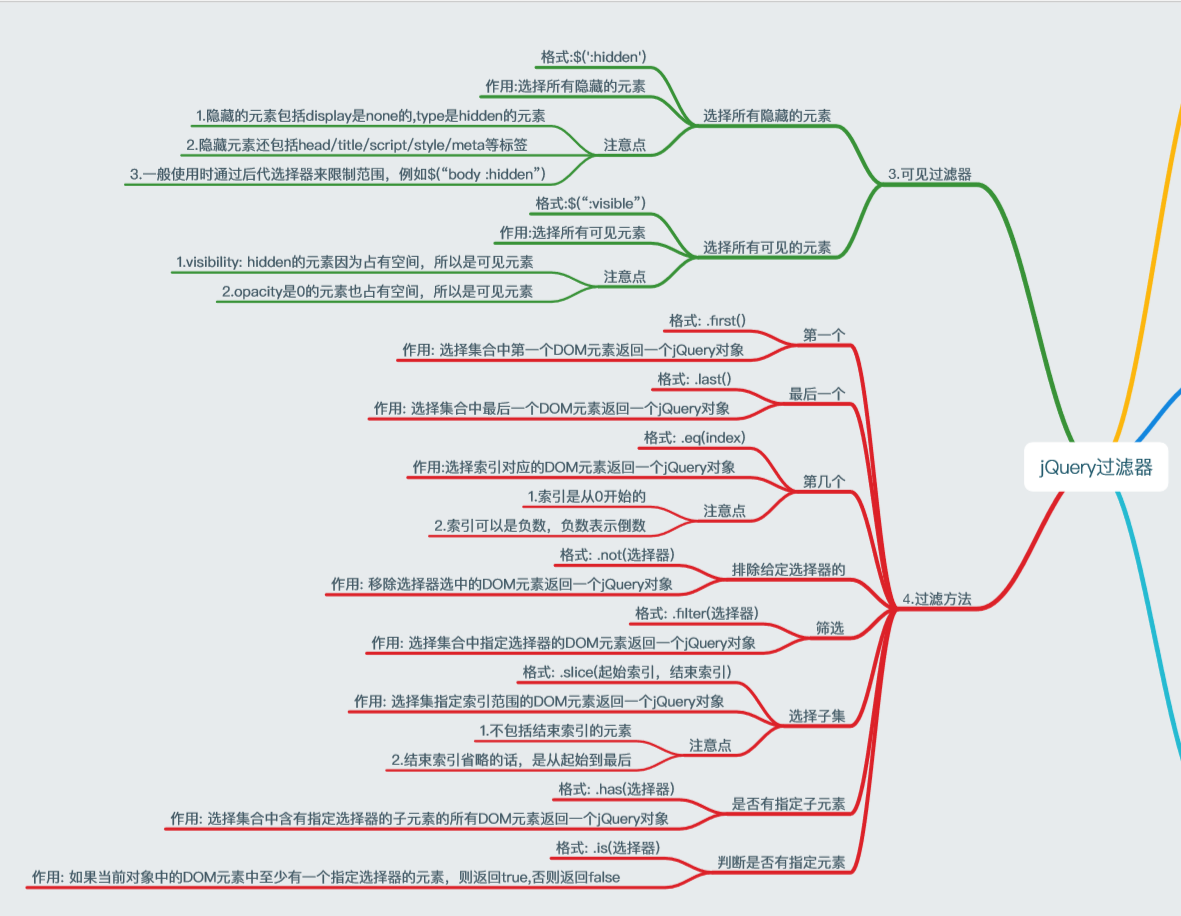


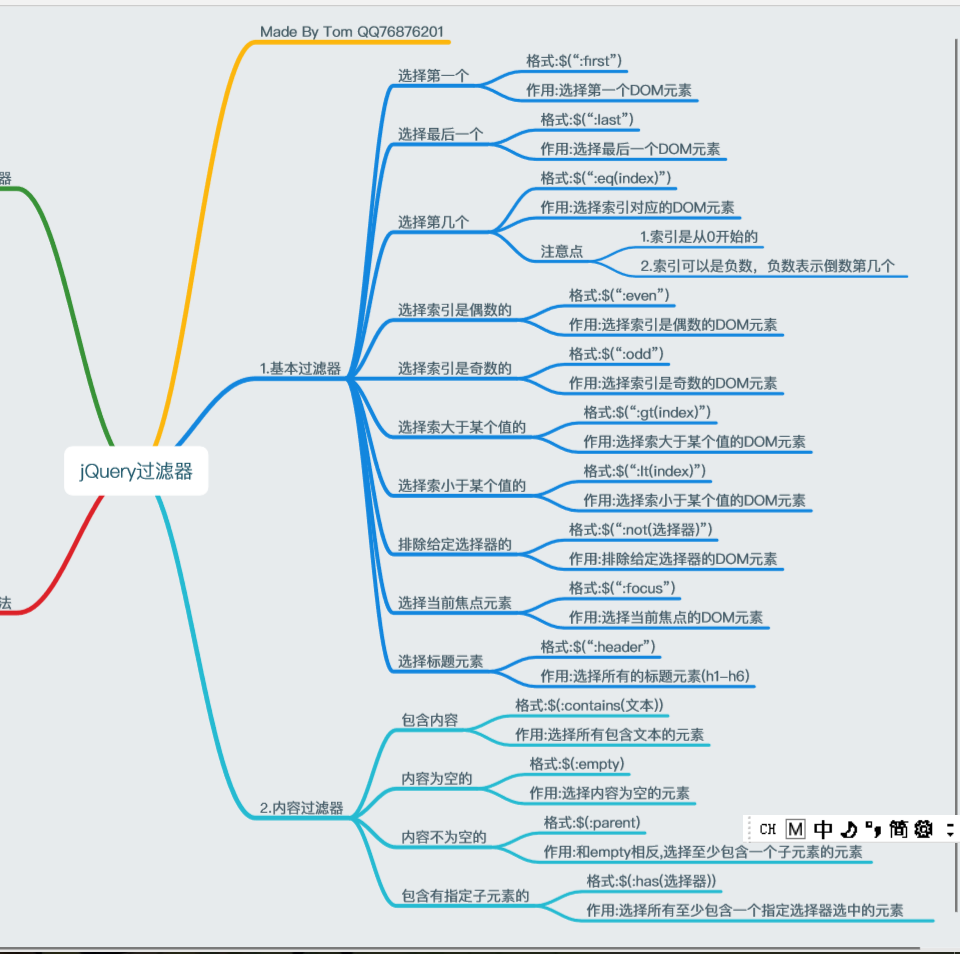


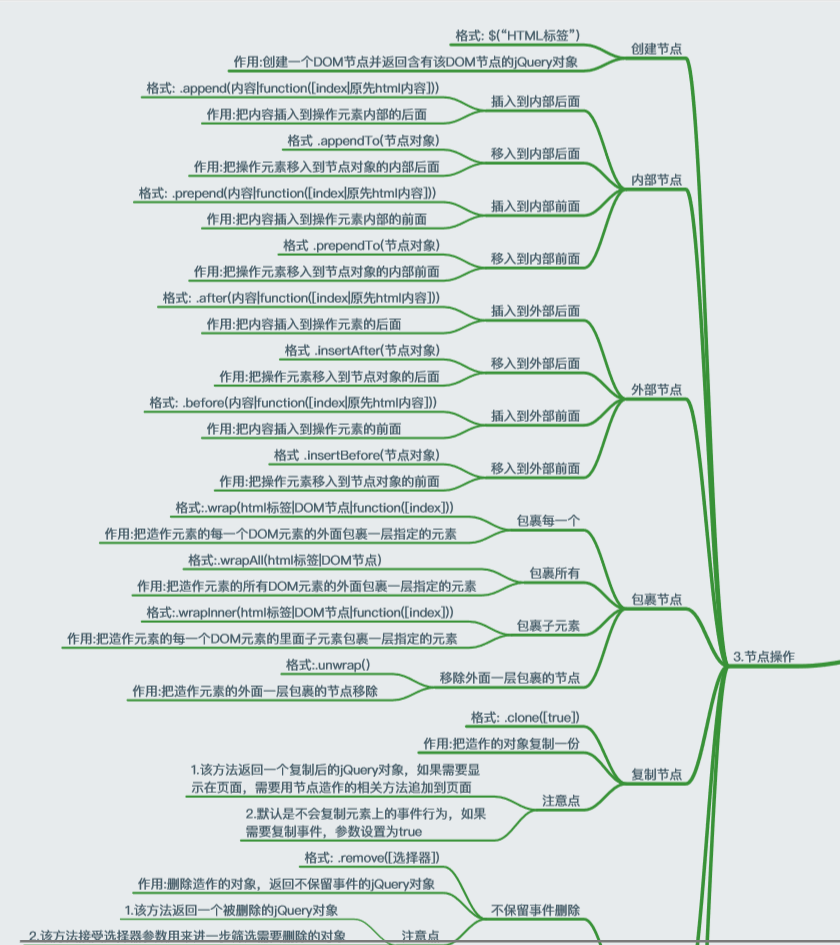




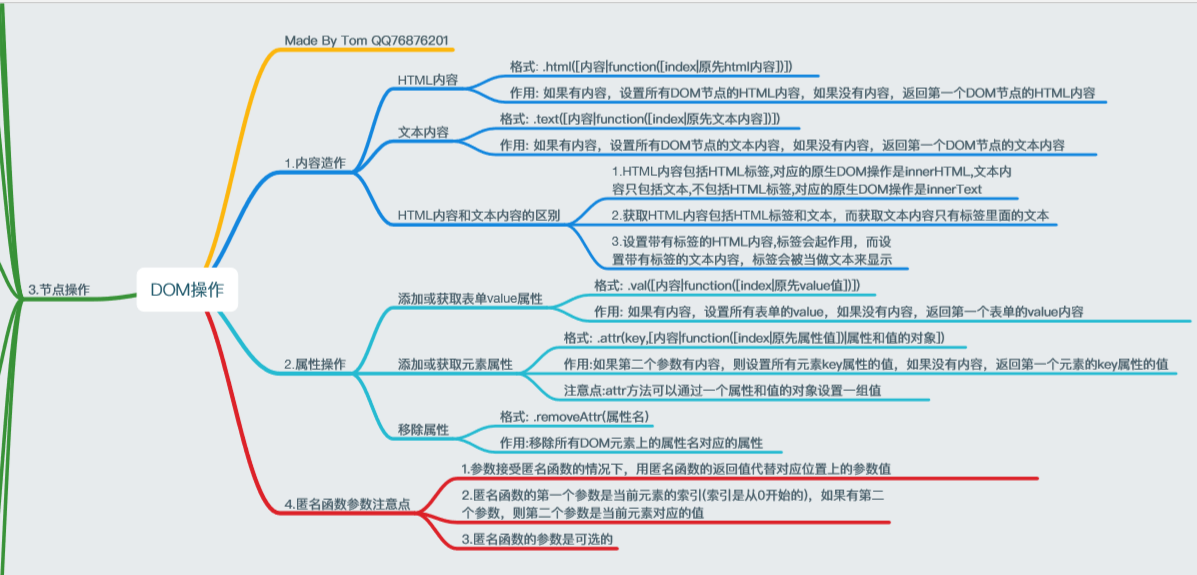


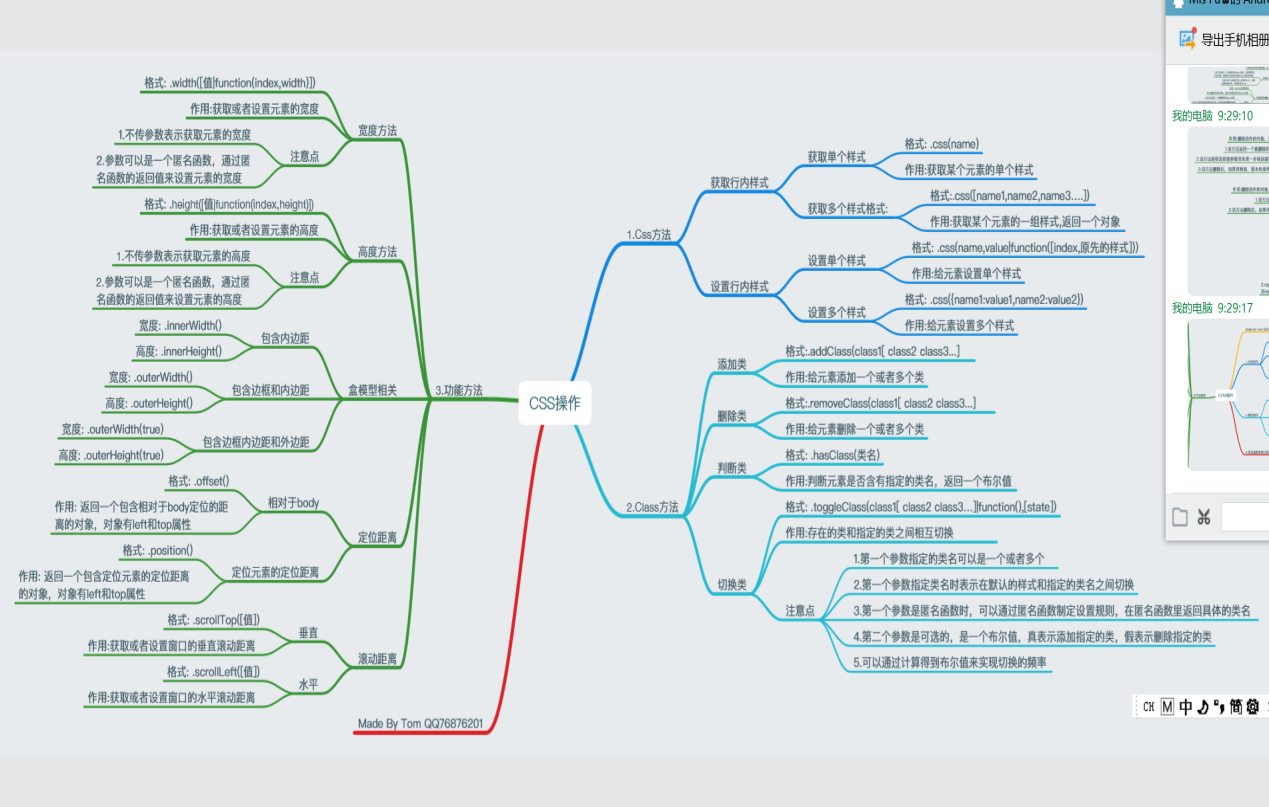


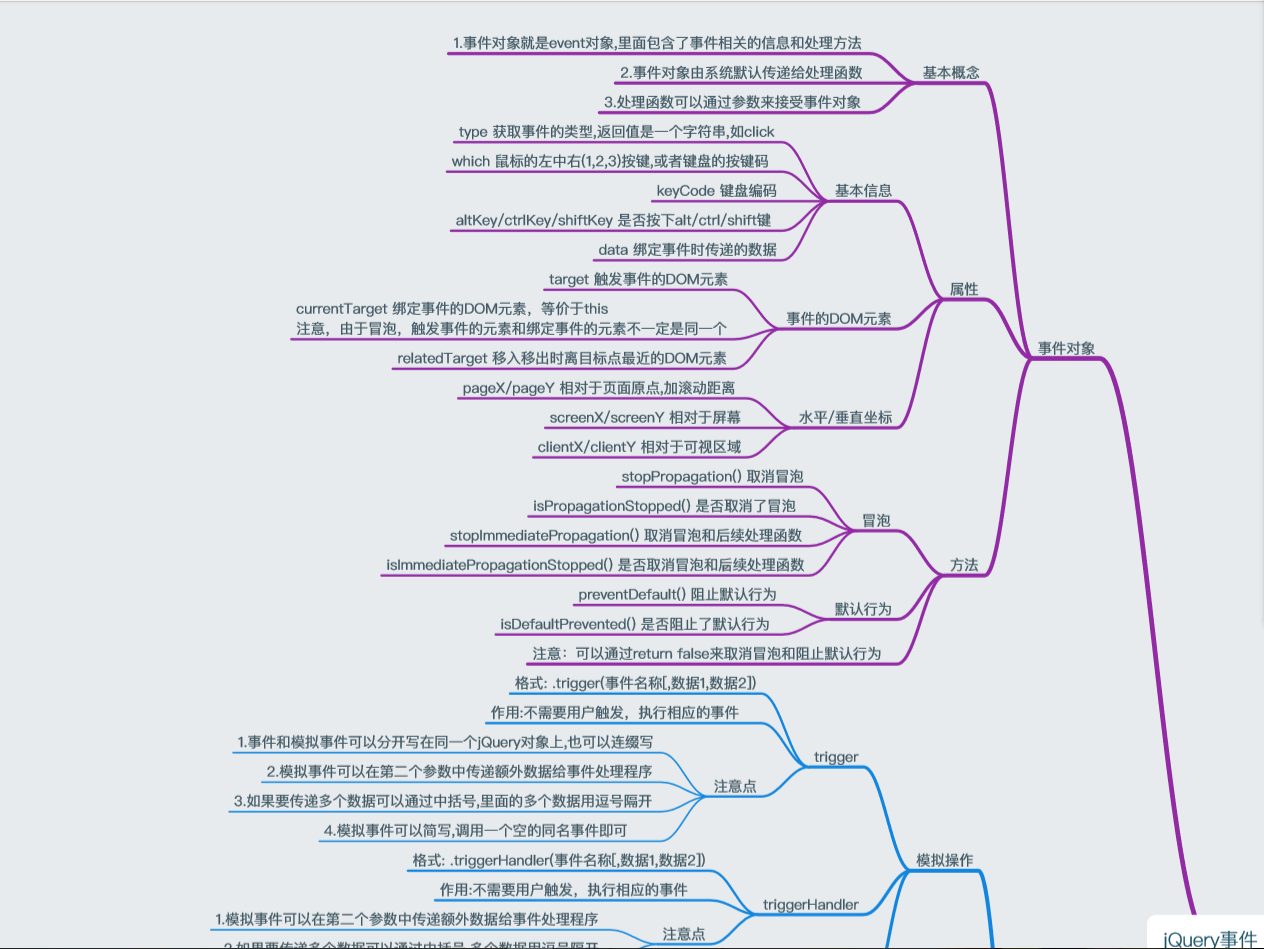


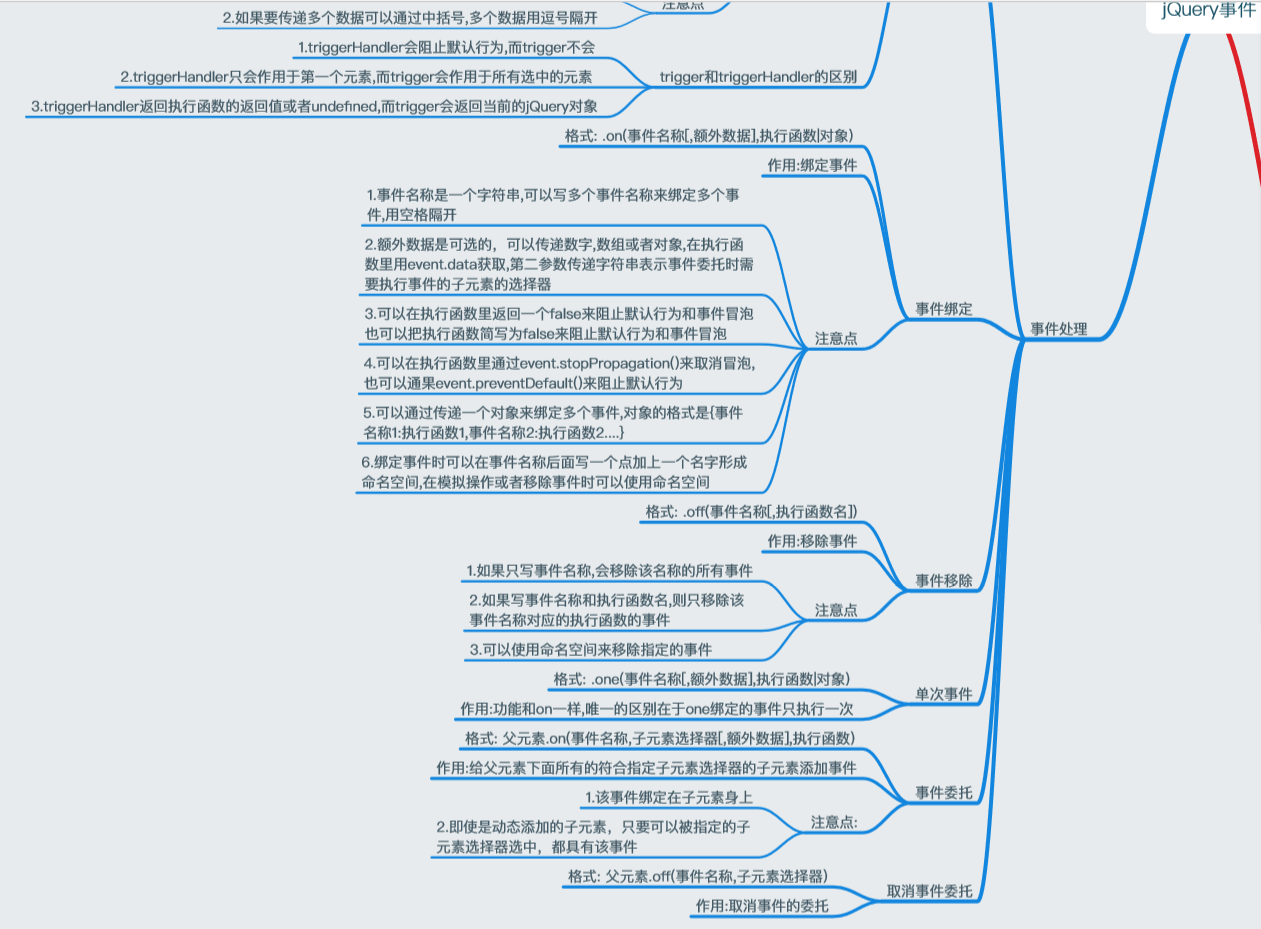


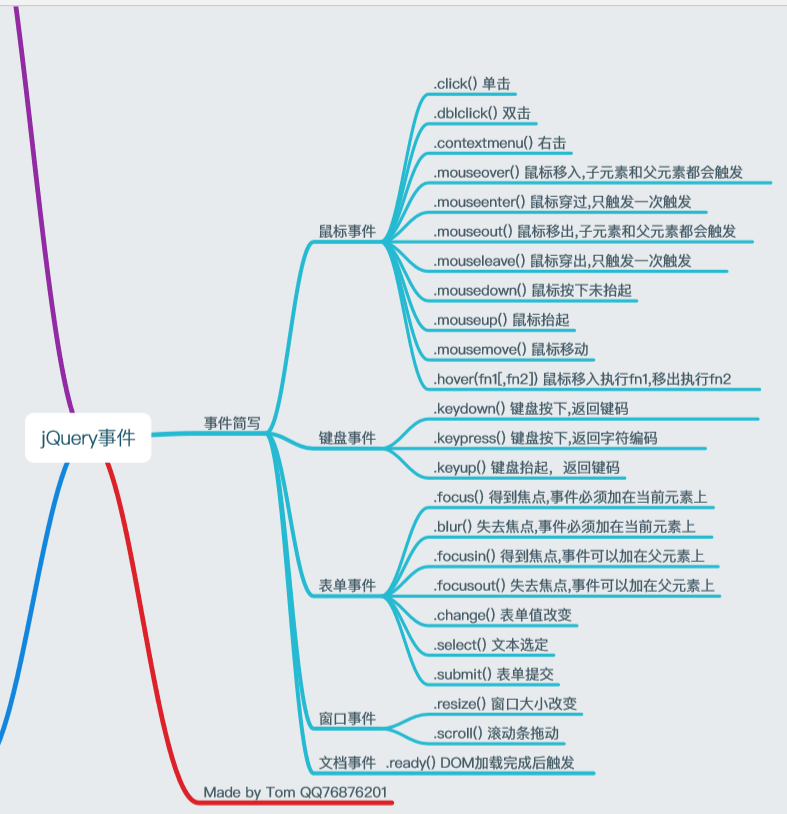


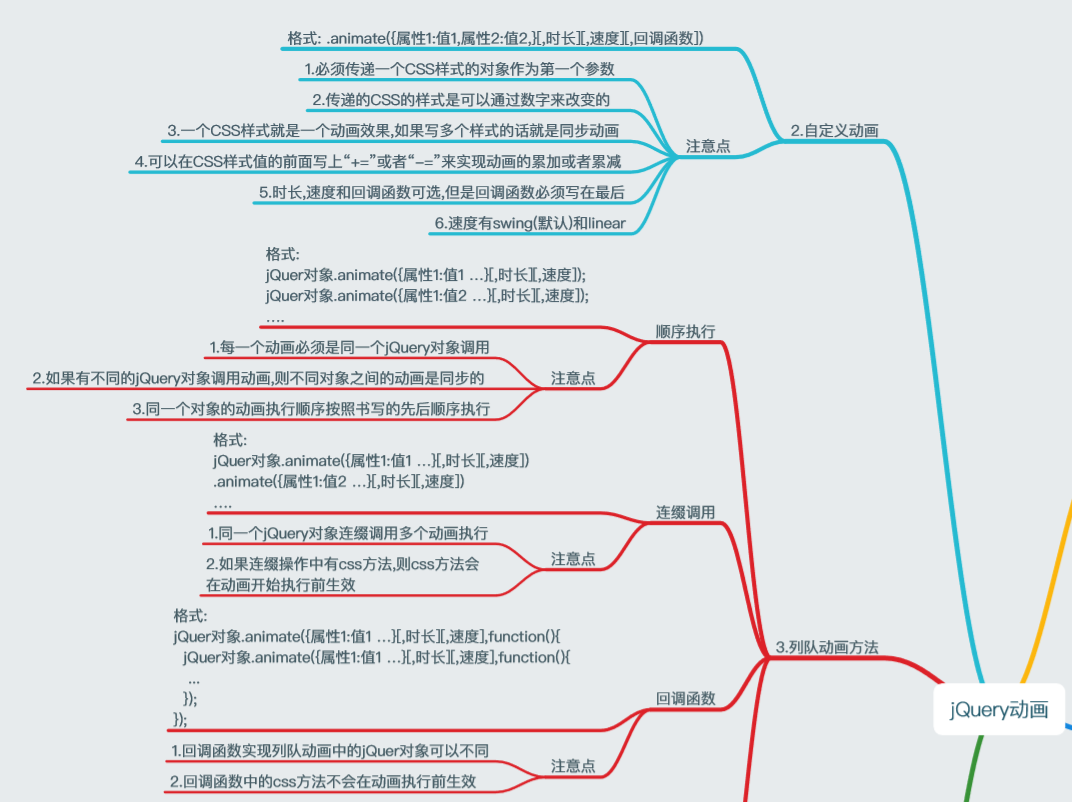


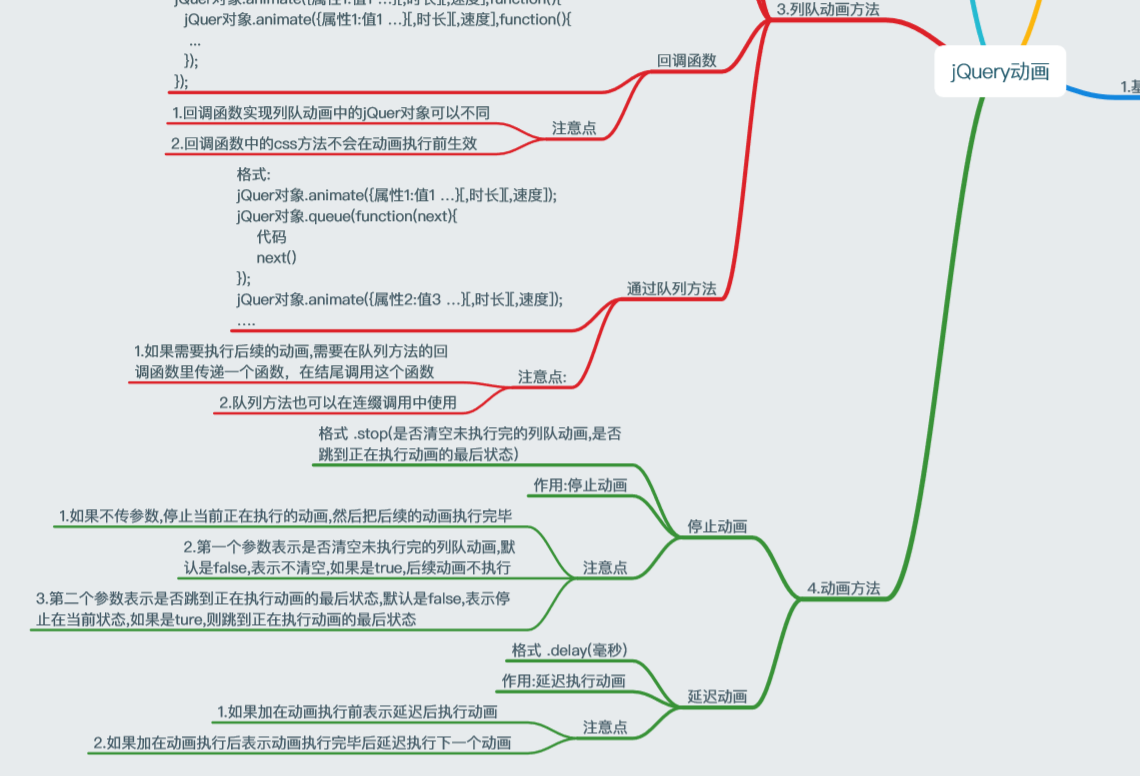


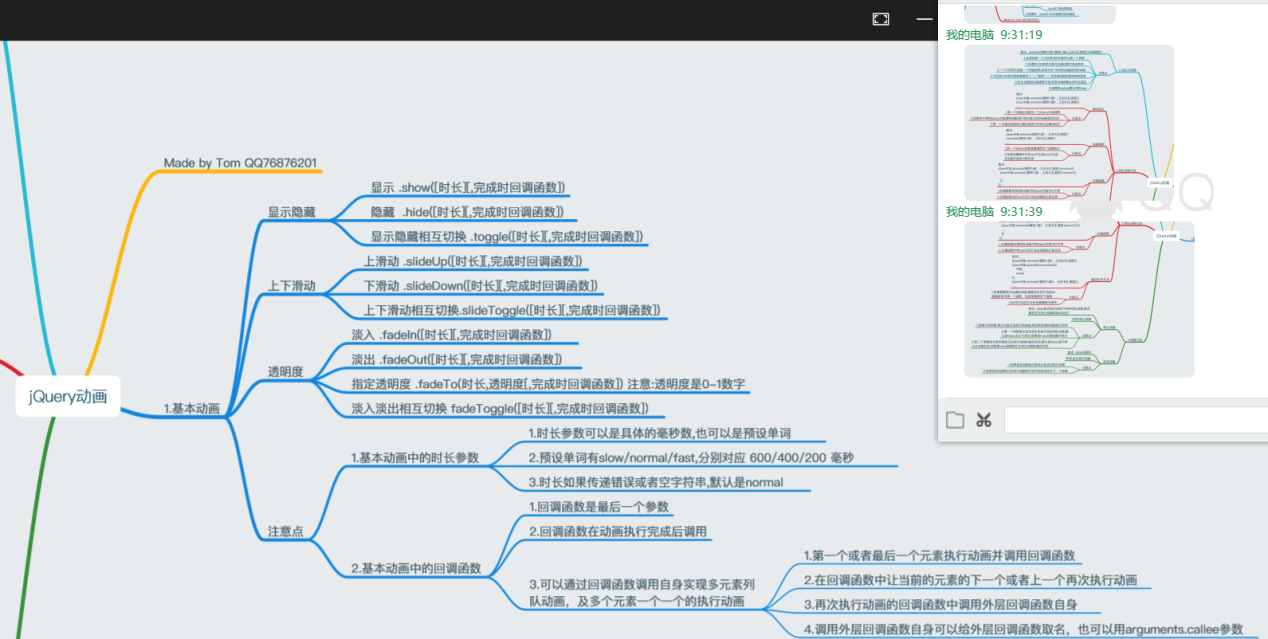




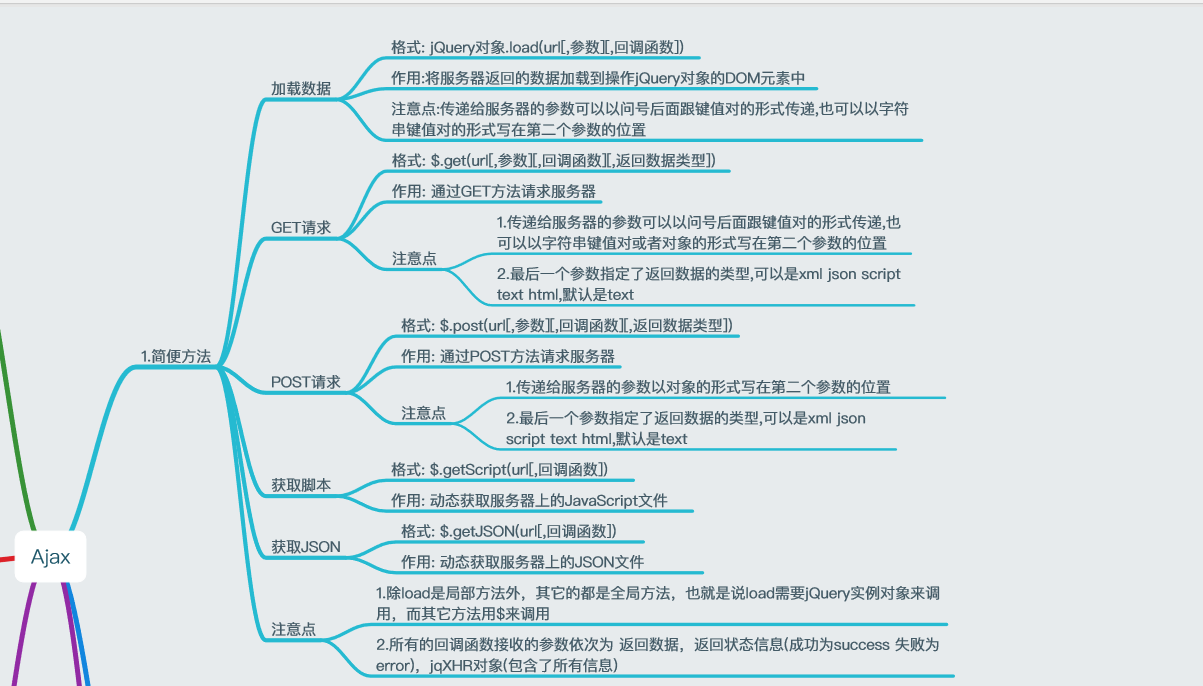


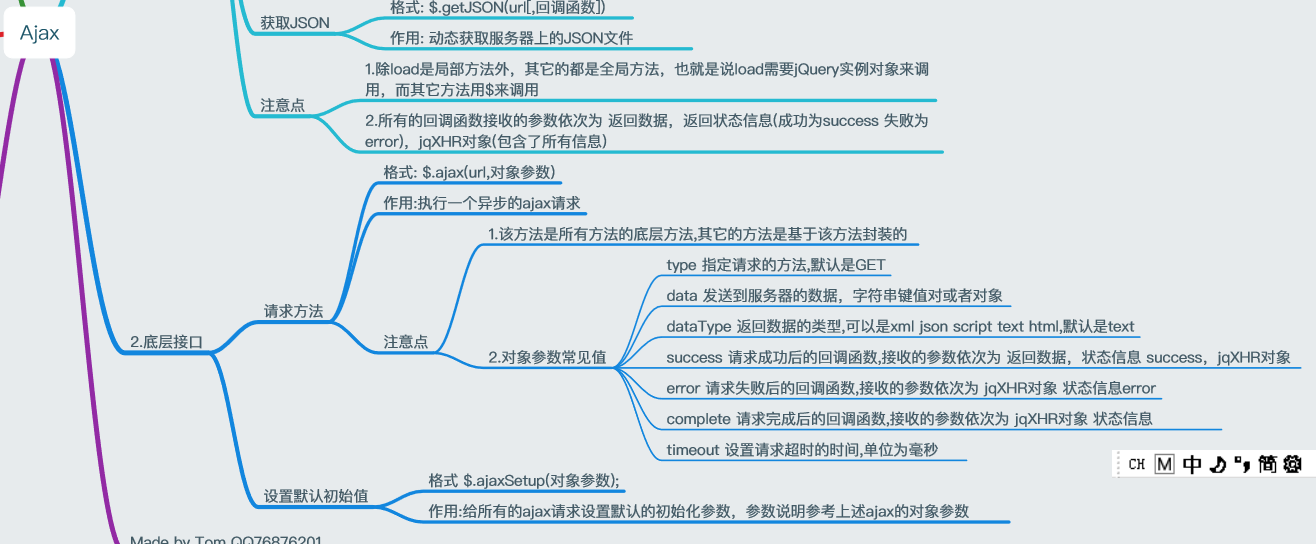


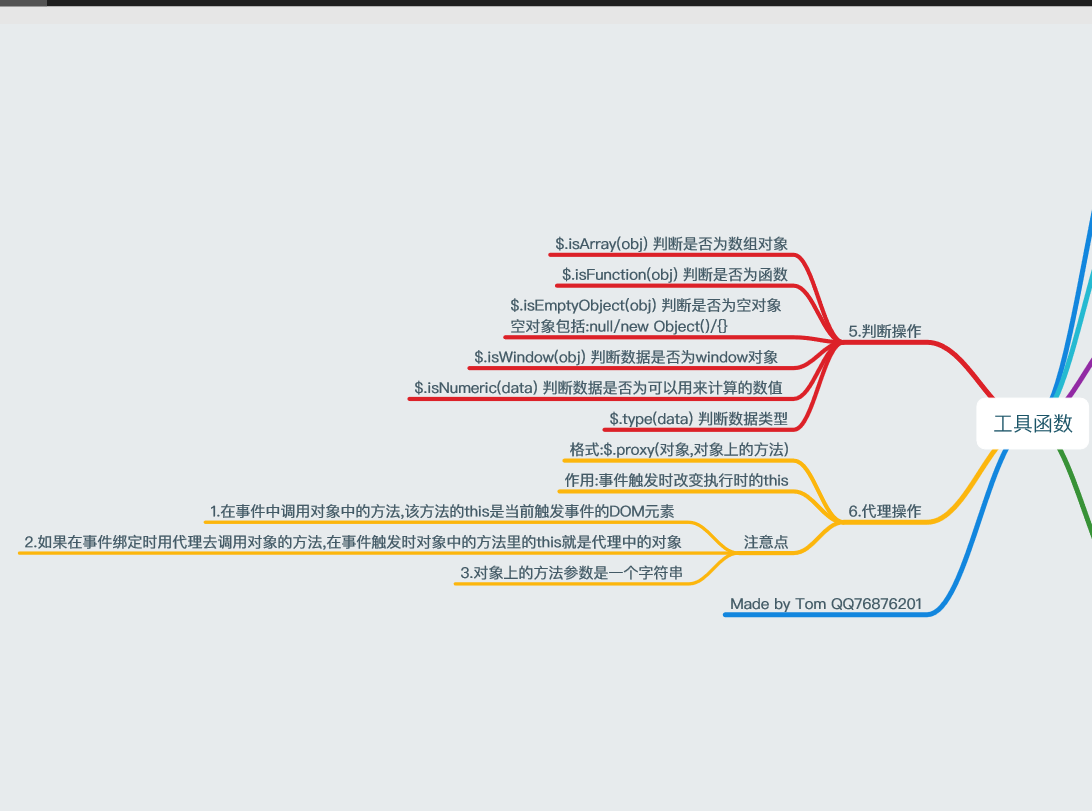


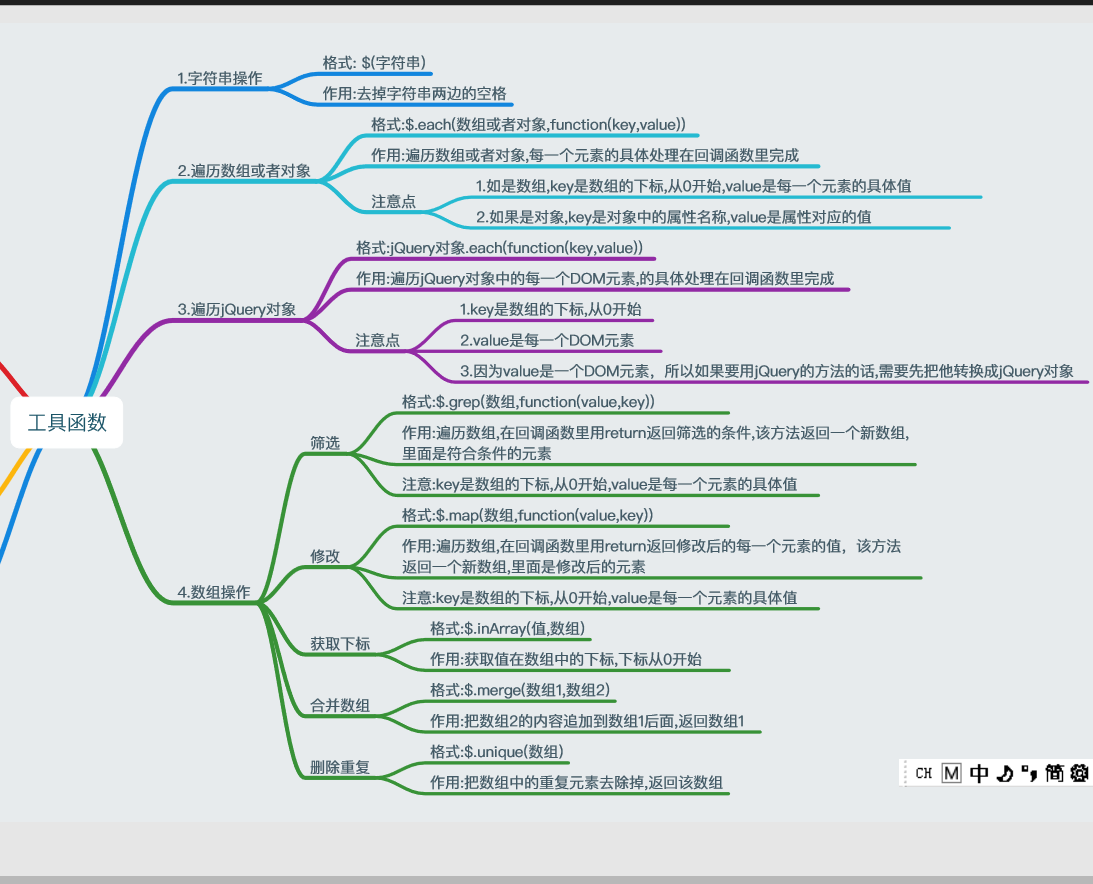












### 001-本地存储.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### cookie

* cookie的设置

document.cookie = 'key1=value1';

document.cookie = 'key2=value2';

注意,后面的不会覆盖前面的,在ff下测试

* cookie有效期的设置

var oDate = new Date();

oDate.setDate(oDate.getDate() + 14);

document.cookie = "name=Tom;expires="+oDate;

* 封装cookie相关的函数

function setCookie(key,value,iDate){

var oDate = new Date();

oDate.setDate(oDate.getDate() + iDate);

document.cookie = key+"="+value+";expires="+oDate;

}

function getCookie(name){

var str = document.cookie;

var arr1 = str.split(";");

var result = "";

for(var i=0;i<arr1.length;i++){

var arr2 = arr1[i].split("=");

if(arr2[0].trim() == name.trim()){

result = arr2[1];

break;

}

}

return result;

}

function removeCookie(name){

setCookie(name,false,-1);

}

* cookie的特点:
* cookie是document对象上的一个属性
* cookie的大小限制是4k
* cookie会存在于请求头中
* 主域名中的cookie会在子域名中存在

### Storage

* localStorage 和 sessionStorage的api相同
* setItem(key,value) 设置
* getItem(key) 获取
* removeItem(key) 删除
* clear() 清除所有
* length 内容个数
* key(num) 获取key的值
* localStorage 和 sessionStorage的存储时效不同
* localStorage 持久化
* sessionStorage 网页会话结束失效
* localStorage 和 sessionStorage的存储容量不同
* localStorage 2-5Mb左右
* sessionStorage 2-10Mb左右,部分浏览器 没有限制
* 注意点:
* Storage仅能存储字符串
* Storage的数据不能在不同的域中共享

### 002-拖拽.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 拖拽

被拖拽元素添加 draggable=true

绑定拖拽事件

被拖拽元素相关

* ondragstart 当元素开始拖动时触发
* ondrag 被拖动元素拖动时触发
* ondragend 拖拽时松开鼠标时触发

容器相关

* ondrop 当放置被拖动元素时触发
* ondragover 被拖动元素正在到放置到容器时触发,默认无法把元素放置到其他元素中,如果需要设置允许放置,我们必须阻止对元素的默认处理方式。
* ondragenter 当被拖动元素进入容器时触发
* ondragleave 当被拖动元素离开容器时触发
* ondragend 被拖动元素在容器中松开鼠标时触发
* ondrag 容器中的被拖动元素拖动时触发
* dataTransfer对象
* 设置数据 dataTransfer.setData(key,val)
* 获取数据 dataTransfer.getData(key)
* 设置拖动的图标图片: dataTransfer.setDragImage(图片DOM节点,x,y)

注意:dataTransfer对象存在在原生的event对象上,如果用jQuery的event对象,需要使用jQuery的event对象的originalEvent

### 拖拽插件

* jquery.pep.js 官网: http://pep.briangonzalez.org/

### 003-SVG体验.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### SVG 介绍

SVG(Scalable Vector Graphics)可缩放矢量图形,一种用来描述二维矢量图形的XML语言

### SVG视口(viewport)

* SVG视口是指可见区域的大小(画布的大小)

<svg width='800' height='600'></svg>

svg默认的宽高是300px\*150px

### SVG视区盒子(viewBox)

viewBox = 'x y width height'

//x:水平坐标 y:垂直坐标 width:宽度 height:高度

* viewBox会根据坐标点和宽高来截取视口,把截取后的内容全屏

<svg width='800' height='600' viewBox='0 0 80 60'>

<rect >

</svg>

### 图形

直线 <line x1="0" y1="0" x2="300" y2="300" style="stroke:rgb(99,99,99);stroke-width:2"/>

矩形 <rect x='100' y='100' width='400' height='300' fill="#ff0">

### 004-canvas.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### canvas

canvas是一个画布,用来在网页上绘制图像

### canvas基础

* 创建canvas

<canvas></canvas>

canvas默认的宽高是300px\*150px

* 画布的大小
* 在canvas标签中设置的width和height是指画布大大小,不用带单位
* 在css中设置的width和height是指画布拉伸缩放后的大小
* 一般建议在js中设置画布的大小
* 画直线

cxt.moveTo(0,0);//指定起始坐标

cxt.lineTo(100,100);//直线终点坐标

cxt.lineTo(100,200);

cxt.strokeStyle='#f00';//指定描边的样式

cxt.lineWidth = 10;//设置线条的宽度

cxt.closePath();//封闭路径

cxt.stroke(); //描边

cxt.fillStyle = 'rgba(0,255,0,0.5)';//设置填充的颜色

cxt.fill();//闭合路径后填充

cxt.beginPath();重新开始路径,绘制时不会绘制之前的路径

圆形

cxt.arc(圆心x,圆心y,半径,起始角度,终止角度,是否是逆时针(true为逆时针))

3点钟时0PI 6点钟是0.5PI 9点钟是1PI 12点钟是1.5PI

矩形 cxt.strokeRect(x,y,宽,高) cxt.fillRect(x,y,宽,高)

平移坐标原点 cxt.translate(x,y);

旋转坐标系 cxt.rotate(弧度) 以顺时针为方向旋转

缩放变换 cxt.scale(x轴比例,y轴比例);

所有的变换都是针对坐标系的 各种变换之间是相互叠加的

save和restore环境 cxt.save(); 保存之前的环境 cxt.restore();恢复到上一次保存时的环境

线性渐变(LinearGradient) var linearGradient = cxt.createLinearGradient(起始x,起始y,终止x,终止y); linearGradient.addColorStop(百分比,颜色)

径向渐变(RadialGradient) var radialGradient = createRadialGradient(起始圆心x,起始圆心y,起始圆半径,终止圆心x,终止圆心y,终止圆半径);

文本

图像绘制

图形画刷

剪辑区

绘制阴影

绘制曲线

* 圆弧: context.arc(圆心x,圆心y,半径,起始角度,终止角度,是否逆时针)
* 二次样条曲线: context.quadraticCurveTo(qcpx,qcpy,qx,qy)

http://blogs.sitepointstatic.com/examples/tech/canvas-curves/quadratic-curve.html

* 贝塞尔曲线: context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)

http://blogs.sitepointstatic.com/examples/tech/canvas-curves/bezier-curve.html

### 推荐插件

* http://echarts.baidu.com/
* https://antv.alipay.com/zh-cn/index.html
* http://www.chartjs.org/
* https://www.highcharts.com
* https://d3js.org/

### 001-Less.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### Less介绍

Less 是一门 CSS 预处理语言，它扩展了 CSS 语言，增加了变量、Mixin、函数等特性，使 CSS 更易维护和扩展。

### Less编译

* 浏览器

<!--

1. 引入.less文件,注意rel="stylesheet/less"

2. 在less文件后面引入编译less的js文件,可以从http://lesscss.cn/中下载

-->

<link rel="stylesheet/less" type="text/css" href="less/001.less">

<script src="js/less.js" type="text/javascript"></script>

* 客户端

http://koala-app.com/

* 服务端

### Less语法

* 注释

//我是注释一 编译被删除

/\*我是注释二 编译被保留\*/

变量 @变量名:值

混合(Mixin)

* 基本混合,直接把选择器写在另外一个选择中
* 带参数
* 带参数有默认值

匹配模式 选择符合条件的混合模式,根据传入的参数来决定使用哪个混合

嵌套 在选择器中嵌套选择器

运算 +-\*/

避免编译 属性:~'css的值'

!important关键字 在混合模式后添加!important关键字,会为混合模式中的所有样式添加!important

\*混合参数做为属性和设置计算后的属性值

.set(@attr,@val){ @{attr}:unit(@val/10,px); }

### 002-移动端.md

×

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 移动基础

* 像素
* 设备的物理像素,屏幕的物理像素,任何设备的物理像素是固定的,例如电脑屏幕的分辨率是1366 X 768,表示在电脑屏幕上有1366 X 768个小格子,一个小格子代表一个物理像素
* CSS像素,又称逻辑像素,是为WEB开发者创造出来的一个抽象层
* 只有在页面不缩放的情况下,一个CSS像素才等于一个物理像素
* 如果页面放大,根据放大的比例,多个物理像素合成一个CSS像素
* 如果页面缩小,根据缩小的比例,一个物理像素分成多个CSS像素
* 因为像素的合成,当网页放大后,CSS定义的像素并没有改变,但是视觉效果上变大了
* DPI/PPI
* DPI（Dot Per Inch）每英寸的墨汁点，多用于印刷行业
* PPI（Pixels Per Inch）屏幕每英寸的物理像素数量，即单位英寸的物理像素密度
* 1 英寸(inch)=25.4毫米(millimetres)=2.54厘米

<img src="img/ppi.png">

PPI值越高,一个实际1英寸上容纳的物理像素就越多,就能够展示更多的画面细节,也就意味着画面更细腻

DPR

* 手机分辨率很高,但手机屏幕尺寸有限,所以可以让手机屏幕上的多个物理像素合成一个CSS像素来更好的显 示内容
* DPR(device Pixel Ratio) 设备像素比 = 设备物理像素 / CSS像素,这个DPR就决定了多少个物理像素合成一个CSS像素
* 以iPhone6为例
* iPhone6 的分辨率为1334X750
* iPhone6 的CSS像素为667X375
* 所以iPhone6 的DPR为 1336/667 = 2,即2个物理像素对应一个css像素
* DPR的设置是根据人体手臂的长度来设定的,也就是说将 1334X750的物理像素合成667X375的CSS像素,视觉效果是最好的
* 视口 viewport

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

* width,设置视口的宽度,表示把移动设备的宽缩放为多少个CSS像素,为一个正整数,或字符串"device-width",device-width表示缩放到设备合成后的CSS像素
* initial-scale 设置页面的初始缩放值,为一个数字,可以带小数
* minimum-scale 允许用户的最小缩放值,为一个数字,可以带小数
* maximum-scale 允许用户的最大缩放值,为一个数字,可以带小数
* height,设置视口的高度
* user-scalable,是否允许用户进行缩放,值为"no"或"yes", no 代表不允许,yes代表允许
* 这些属性可以同时使用，也可以单独使用或混合使用，多个属性同时使用时用逗号隔开

### 屏幕尺寸适配

* 像素px,绝对单位,由于单位固定
* 无法做屏幕的适配
* 百分比%,相对于父元素的宽高
* 可以实现布局的适配,但是字体不容易适配
* em,当前元素字体的大小,1em=当前元素的font-size
* 可以根据不同的屏幕来设置元素的font-size,再利用em来适配,但是元素的font-size可以被继承,同时js和css经常会修改font-size,如果用em来适配的话容易造成混乱
* 注意:font-size最小值是12px;
* 1vw=当前设备宽度的1%,vh=当前设备高度的1%
* 可以适配布局和文字,但兼容性不好
* rem,根元素字体的大小,默认值是16px
* 可以根据不同的屏幕来设置根元素的font-size,再利用rem来适配
* 根元素的font-size需要动态设置,一般有JS和媒体查询两种方法来动态设置

### 响应式布局(Responsive)

* 媒体类型 (media type)
* 媒体类型是从CSS2和HTML4开始,可以根据设备的类型来决定使用什么样式
* 使用方法一

<link rel="stylesheet" media="print" href="css/print.css" >

<link rel="stylesheet" media="screen" href="css/screen.css" >

* 使用方法二

<style type="text/css">

@media screen{

h1{

color: blue;

}

}

@media print{

h1{

color: red;

}

}

</style>

* 常见的媒体类型有(还有其他的但基本已被抛弃,所以在此不在列举)

screen 屏幕 print 打印机 all 所有

* 媒体查询(media query)
* 媒体查询是由媒体类型(media type)和媒体特征(media features)共通决定使用什么样式
* 常见的媒体特征有
* width, 浏览器窗口的宽度,可以有min/max前缀
* height, 浏览器窗口的高度,可以有min/max前缀
* device-width,分辨率宽度,可以有min/max前缀
* device-height,分辨率高度,可以有min/max前缀
* orientation,移动设备窗口的方向,portrait(纵向),landscape(横向)
* min/max的解释
* (min-width:400px) 浏览器窗口的宽度最小是400px样式生效,换句话说是大于等于400px时样式生效
* (max-width:800px) 浏览器窗口的宽度最大是800px样式生效,换句话说是小于等于800px时样式生效
* 使用方法一

<link rel="stylesheet" href="css/blue.css">

<link rel="stylesheet" href="css/yellow.css" media="screen and (max-width: 500px)">

* 使用方法二

<style>

@media screen and (min-width: 500px){

body{

background: yellow;

}

}

</style>

### 弹性布局(Flexible Box)

* 把容器指定为弹性布局
* 块状元素 display: flex;
* 行内元素 display: inline-flex;

指定了弹性布局的元素称为Flex容器(flex container),它的所有子元素自动成为容器成员,称为 Flex项目(flex item)

容器属性

* flex-direction 决定项目的排列方向,有以下取值
* row(默认值):水平方向,起点在左端
* row-reverse:水平方向,起点在右端
* column:垂直方向,起点在上沿
* column-reverse:垂直方向,起点在下沿
* flex-wrap 决定项目一行排不下，如何换行,有以下取值
* nowrap(默认值):不换行,元素的宽度按照指定款的的比例从新计算
* wrap:换行,第一行在上方
* 换行后项目的垂直对齐方式由align-items/align-content来决定
* wrap-reverse:换行,第一行在下方
* flex-flow:flex-direction属性和flex-wrap属性的简写形式,默认值为row nowrap。
* justify-content 决定项目的水平对齐方式,有以下取值
* flex-start(默认值):左对齐
* flex-end:右对齐
* center:居中
* space-between:两端对齐,项目之间的间隔都相等。
* space-around:每个项目两侧的间隔相等。所以,项目之间的间隔比项目与边框的间隔大一倍。
* align-items/align-content 决定项目的垂直对齐方式,有以下取值
* stretch(默认值):换行后,如果项目未设置高度或设为auto,将占满整个容器的高度
* flex-start：起点对齐
* flex-end：终点对齐
* center：中点对齐
* baseline: 项目的第一行文字的基线对齐
* align-items/align-content的区别在于:
* align-item换行后会根据父容器的高度填充整个行
* align-content换行后按项目的的高度来填充行
* 项目属性
* order 项目的排列顺序,数值越小,排列越靠前,默认为0
* flex-grow 项目的放大比例,默认为0,即如果存在剩余空间,也不放大
* 如果所有项目的flex-grow属性都为1,则它们将等分剩余空间(如果有的话)
* 如果一个项目的flex-grow属性为2,其他项目都为1,则前者占据的剩余空间将比其他项多一倍
* flex-shrink 项目的缩小比例,默认为1,即如果空间不足,该项目将缩小
* 如果所有项目的flex-shrink属性都为1,当空间不足时,都将等比例缩小
* 如果一个项目的flex-shrink属性为0,其他项目都为1,则空间不足时,前者不缩小
* flex-basis 定义了在分配多余空间之前,项目占据的大小,浏览器根据这个属性,计算是否有多余空间
* 它的默认值为auto,即项目的本来大小
* flex 是flex-grow,flex-shrink,flex-basis的简写,默认值为0 1 auto,后两个属性可选
* align-self 允许单个项目有与其他项目不一样的对齐方式,可覆盖align-items属性

### 001-变量.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### let

* 作用:定义变量
* 注意点:
* let定义过的变量不能再定义,var定义过的变量可以重新定义
* let定义的变量不会声明提升,var定义的变量会发生声明提升
* let定义的变量在let命令所在的代码块内有效,var定义的变量在函数范围内有效

### const

* 作用:定义常量(一旦定义,值就不能改变)
* 注意点:
* 定义常量的时候必须赋初值
* 常量不可以再次赋值
* const定义的常量保证的是常量的地址不可以改变
* 对于简单数据来说值就保存在变量指向的那个内存地址,因此不可以再赋值
* 对于复杂数据来说常量指向数据保存的地址,const保证的是这个地址不变,但地址指向的数据是可以修改的
* const定义的常量在const命令所在的代码块内有效
* const定义的常量不会声明提升

### 结构解析

作用:从数组或者对象中提取值然后对变量进行赋值

数组的结构解析:按照对应位置,把等号右边数组中的值赋给等号左边数组中的变量,数组可以嵌套

* 注意点
* 如果解构不成功,变量的值就等于undefined
* 如果等号的右边不是数组,会报错
* 可以在等号右边数组中设置默认值,当等号左边数组中对应位置的值是undefined时默认值才会生效。
* 对象的结构解析:按照对应属性名,把等号右边对象中的属性的值赋给等号左边对象中的同名属性
* 注意点
* 等号左边对象中的属性的值是真正被赋值的变量名,如果属性的值省略,属性的值和属性的名称相同

### 模版字符

语法:字符 ${javascript表达式}

### 002-扩展运算.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 扩展运算

扩展运算符三个点...

数组的扩展运算就是将一个数组转为用逗号分隔的参数序列

数组的扩展运算的应用

* 函数调用时使用把数组转换为参数列表

function add(n1,n2){

return n1+n2;

}

let arr = [11,22];

add(...arr);

* 函数定义时使用把参数列表转换为数组

function add(...arr){

let result = 0;

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

result += arr[i]

}

return result;

}

add(11,22);

* 求数组的最大数

let arr = [11,22,44,88,77];

//ES5

Math.max.apply(null,arr)

//ES6 用扩展运算

Math.max(...arr)

* 深拷贝数组

let arr1 = [11,22,44,88,77];

//ES5

let arr2 = arr1.concat();

//ES6 用扩展运算

let arr2 = [...arr1];

对象的扩展运算符...用于取出参数对象的所有可遍历属性,拷贝到当前对象之中

对象的扩展运算符的应用

* 拷贝对象(浅拷贝)

let obj1 = {name:'Tom',age:18};

let obj2 = {...ojb1};

如果对象中有复杂数据类型,拷贝的是数据的地址

### 003-Set和Map.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 遍历器

* 遍历器(Iterator)是一种接口,实现了该接口的数据结构就可以完成遍历操作
* 遍历器接口主要供for...of循环来使用,也就是说实现了遍历器接口的数据类型都可以用for...of循环来遍历数据

### Set

类似于数组,但值唯一的数据结构

Set对象的size属性返回Set实例的值的总数

Set对象的操作方法

* add(value):添加某个值,返回 Set 结构本身
* delete(value):删除某个值,返回一个布尔值,表示删除是否成功
* has(value):返回一个布尔值,表示该值是否为Set的成员
* clear():清除所有值,没有返回值
* Set对象的遍历方法
* keys():返回键名的遍历器
* values():返回键值的遍历器
* entries():返回键值对的遍历器
* forEach(function(value,key){}):使用回调函数遍历
* 注意点:
* Set只有值没有键,所以key和value一样
* Set的遍历顺序就是插入顺序

### Map

类似于对象,但键的范围不限于字符串,各种类型的值(包括对象)都可以当作键的数据结构

Map对象的size属性返回Map实例的键值对的总数

Map对象的操作方法

* set(key,value):设置键名key对应的键值为value，然后返回整个 Map 结构,如果key已经有值，则键值会被更新，否则就新生成该键。
* get(key):读取key对应的键值，如果找不到key，返回undefined。
* delete(key):delete方法删除某个键，返回true。如果删除失败，返回false。
* has(key):has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中。
* clear():清除所有键值,没有返回值
* Map对象的遍历方法
* keys():返回键名的遍历器
* values():返回键值的遍历器
* entries():返回键值对的遍历器
* forEach(function(value,key){}):使用回调函数遍历
* 注意点:
* Map的遍历顺序就是插入顺序

### 004-Symbol数据类型.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### Symbol

* Symbol是一种新的数据类型,表示独一无二的值
* Symbol值通过Symbol函数生成
* Symbol函数可以接受一个字符串作为参数,表示对Symbol实例的描述
* Symbol的主要作用是用来表示对象的属性名
* Symbol值作为对象属性名时,不能用点运算符,在对象的内部页不可以

### 005-类.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### class类

class类是ES5面向对象的语法糖

举例

class Person{

constructor(name,age){

this.name = name;

this.age = age;

}

show(){

console.log('I am '+this.name+' and I am '+this.age+' years old');

}

showName(){

console.log('I am '+this.name);

}

showAge(){

console.log('I am '+this.age+' years old');

}

static sayHi(){

console.log('Hi...')

}

}

class Student extends Person{

constructor(name,age,major){

super(name,age);

this.major = major;

}

show(){

super.show();

console.log('I am from '+this.major)

}

showMajor(){

console.log('I am from '+this.major)

}

}

* 定义注意点
* 用class定义的类也是函数
* constructor方法是类的默认方法,通过new命令生成对象实例时,自动调用该方法。
* 一个类必须有constructor方法,如果没有显式定义,系统默认添加一个空的constructor方法
* constructor方法默认返回实例对象
* 在class 中定义的方法实际上是给原型对象上添加方法
* 类不存在变量提升
* 静态方法前面添加static关键字,调用时用 类名.方法名 的形式调用
* 继承注意点
* 父类的静态方法,可以被子类继承
* 通过extends关键字实现继承
* 子类必须在constructor方法中调用super方法,该super方法实际上是调用父类的constructo
* 子类如果没有定义constructor方法,系统默认添加一个空的constructor方法,并且在这个方法会被默认添加一个空的super方法
* 在子类的构造函数中,只有调用super之后,才可以使用this关键字

### 006-函数.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 函数参数默认值

* 直接在参数定义的时候赋值来为为函数的参数设置默认值

function show2(x,y="world"){

console.log(x,y);

}

show2("hello");//hello world

* 参数默认值不是传值的,而是每次都重新计算默认值表达式的值

let x = 1;

function show(i=x+1){

console.log(i);

}

show();//2

x = 2;

show();//3

* 结构解析和默认参数结合

function show2(url,{method="GET",body="name=Tom"} = {}){

console.log(method,body);

}

show2();//GET name=Tom

show2('http://kuazhu.com');//GET name=Tom

* 默认值参数应该是参数的尾部参数,否则不能省略

### rest参数

* rest参数就是在函数定义时使用扩展运算(...变量名)把参数列表转换为数组

### 箭头函数

箭头函数就是使用"箭头"(=>)来定义函数

箭头函数的基本用法

let fn = (x,y)=>{

console.log(x,y);

return x + y;

}

* 如果函数体只有返回值,函数体的花括号和return语句可以省略,简写:

let fn = (x,y)=> x + y

* 如果只有一个参数,圆括号可以省略,简写:

let fn = x =>{

console.log(x);

return x + 10;

}

* 箭头函数中的this是在定义时确定的(定义时属于谁),而不是在运行时确定(谁调用)的

function Timer(id){

this.id = id;

setTimeout(function(){

console.log(this,this.id);//window 99

},100)

}

var id = 99;

let t = new Timer(100);

* function 定义的函数,this是在运行时确定的,运行时函数属于谁this就是谁,上面例子中的延迟函数会在100毫秒后由window对象来调用,所以里面的this是window对象
* function 定义的函数,可以通过call/bind/apply来改变函数的内部this

function Timer(id){

this.id = id;

setTimeout(()=>{

console.log(this,this.id);//Timer {id: 100} 100

},100);

}

var id = 99;

let t = new Timer(100);

* 箭头函数导致this总是指向函数定义生效时所在的对象
* 上面例子中延迟函数中匿名函数在定义的时候是属于Timer的实例对象,所以里面的this就是Timer对象
* 用let定义的函数不会声明提升

### 007-promise.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### Promise的概念

* Promise 是一种异步编程解决方案
* 将异步操作以同步操作的流程表达出来,避免了层层嵌套的回调函数
* Promise对象提供统一的接口,使得控制异步操作更加容易

### Promise的特点

* Promise有三种状态
* pending（进行中）
* fulfilled（已成功）
* rejected（已失败）

只有异步操作的结果,可以决定当前是哪一种状态,任何其他操作都无法改变这个状态

一旦状态改变,就不会再变

Promise对象的状态改变，只有两种可能：

* 从pending变为fulfilled
* 从pending变为rejected

### Promise的使用

* Promise构造函数接受一个函数作为参数,该函数的两个参数分别是resolve和reject
* resolve函数在异步操作成功时(从pending变为fulfilled)调用,并将异步操作的结果作为参数传递出去
* reject在异步操作失败时(从pending变为rejected)调用,并将异步操作报出的错误作为参数传递出去
* Promise实例生成以后,可以用then方法分别指定resolved状态和rejected状态的回调函数,rejected状态的回调函数可以省略
* Promise 新建后就会立即执行

### Promise对象的方法

* then(resolved状态的回调,rejected状态的回调)
* then方法返回的是一个新的Promise实例
* catch(回调)
* 如果异步操作抛出错误,状态就会变为rejected,当then中没有指定rejected状态的回调时就会调用catch方法指定的回调函数
* 如果运行中抛出错误,也会被catch方法捕获
* finally(回调)
* 不管Promise对象最后状态如何,都会执行的操作
* finally方法的回调函数不接受任何参数

### Promise静态方法

* Promise.resolve()
* 如果参数是Promise实例,那么Promise.resolve将不做任何修改、原封不动地返回这个实例
* 如果参数是一个具有then方法的对象,resolve方法会将这个对象转为 Promise 对象然后就立即执行该对象的then方法
* 如果参数参数不是具有then方法的对象,或根本就不是对象,返回一个新的 Promise 对象,状态为resolved
* 不带有任何参数,直接返回一个resolved状态的 Promise 对象
* Promise.reject()
* 返回一个新的Promise实例,该实例的状态为rejected,参数会原封不动地作为reject的理由
* Promise.all()
* 接受一个Promise实例数组作为参数,返回一个新Promise实例
* 当数组中的所有Promise实例的状态都变成fulfilled时,Promise实例的状态才会变成fulfilled,此时,返回值组成一个数组传递给新Promise实例的回调函数
* 当数组中有一个Promise实例被rejected,新Promise实例的状态就变成rejected,此时第一个被reject的实例的返回值会传递给新Promise实例的回调函数
* Promise.race()
* 接受一个Promise实例数组作为参数,返回执行最快的那个promise对象

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 什么是NodeJs

NodeJs是基于V8引擎的JavaScript运行环境

### NodeJs特点

* 单线程
* 非阻塞式I/O
* 事件驱动

### NodeJs 简介

* 2009年5月 Ryan Dahl发布了最初版本的部分Node.js包
* 目前的LTS版本是8.11.3

### NodeJs 安装

https://nodejs.org/en/

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 模块

* 在NodeJs中,一个文件就是一个模块,这个模块的本质其实就是一个函数,每个模块都有自己的属性

### 模块化

CommonJS exports 和 require

ES6 export 和 import

AMD define 和 require 语句

### NodeJs模块分类

* 核心模块
* 系统核心模块在Node源代码编译过程中被编译成了二进制文件,在node启动时被直接加载到了内存中
* 自定义模块
* 在目录node\_modules下的文件或者包,例如用npm安装的模块
* 自定义模块的加载路径可以通过module.paths查看
* 文件模块
* 文件模块是用户自己编写的模块,以相对或者绝对路径形式引入
* 注意点
* Node对所有加载过的模块对象都会缓存
* Node对二次加载的模块一律采用缓存优先
* 核心模块的优先级高于自定义模块

### NodeJs文件模块

* 模块定义
* module.exports, 在module.exports对象上添加属性导出值
* exports,在exports对象上添加属性导出值
* exports对象和module.exports对象是同一个对象
* exports对象添加属性来导出值
* module.exports对象可以赋值一个对象来导出值
* 导出的始终是module.exports指向的对象
* 模块引用
* require('文件名'),执行对应的文件并且返回该文件对应的modeule.exports对象
* 模块标识
* 模块标识其实就是传递给require()方法的参数
* 模块标识可以是绝对路径也可以是相对路径
* 用绝对路径的话系统会直接加载路径指定的文件
* 用相对路径时注意,当前目录一定要写“./”,否则表明要加载系统的核心模块
* 模块加载规则
* 首先按照模块的文件名进行查找
* 如果没有找到,则会在模块名称后面加上.js后缀进行查找
* 如果还没有找到,则会在模块名称后面加上.json后缀进行查找
* 如果还没有找到,则会在模块名称后面加上.node 后缀进行查找
* 如果还没有找到,则会报错

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### NPM(Node Package Manager)

* NPM是一个工具,安装node时会自动安装
* NPM主要用来完成第三方模块的发布,安装和依赖

### NPM 常用命令

查看版本 npm -v

搜索包 npm search 包名关键字

初始化 npm init [--yes]

* 参数--yes表明不出现提问设置,所有参数采用默认值
* 安装包 npm install [-g] 包名 [--save] [--save-dev]
* 如果带参数-g,表明全局安装,这样安装的包在/usr/local/lib/node\_modules文件夹中，可以在命令行中使用,但不可以在项目中require,使用-g参数时没有--save
* 如果不带参数-g表明本地安装,包文件安装在当前文件夹的node\_modules文件夹中,只能在当前项目中使用require引入模块
* 参数--save:把包的版本信息添加到package.json中dependencies 中,一般是需要最终需要打包到业务代码中的依赖,如果执行npm install就会安装 package.json中dependencies的依赖
* 参数--save-dev: 依赖会加在package.json的devDependencies中,一般是辅助开发的依赖,不会打包上线的
* 卸载包 npm uninstall [-g] 包名 [--save] [--save-dev]

安装用什么参数卸载就用什么参数

yarn

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 什么是全局变量

在所有的模块中都可以使用,不需要引入

### 作用域在模块内的变量

* \_\_dirname 当前模块的文件夹名次
* \_\_filename 当前模块文件的绝对路径
* module 当前的模块信息
* exports module.exports对象,用来导出模块
* require() 引入模块

### global全局的命名空间对象

### process进程对象

* process 和 global.process是同一个对象
* process.argv
* process.env
* process.pid
* process.nextTick()

### 定时器

* setInterval()/clearInterval()
* setTimeout()/clearTimeout()
* setImmediate()/clearImmediate()

### 控制台对象

### Buffer类

* Buffer是用来存放二进制数据的容器
* Buffer.from
* Buffer.alloc

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### event

* 概念
* Node中的event没有默认行为和事件冒泡
* const EventEmitter = require('events');//返回的是一个类
* 通常我们需要继承EventEmitter类来实现事件
* 一个EventEmitter对象默认最大可以有10个监听,可以通过emitter.setMaxListeners(n)来设置最大监听数
* 大多数 Node.js 核心对象都继承事件类
* 添加监听
* emitter.addListener(eventName, listener)
* emitter.on(eventName, listener)
* emitter.once(eventName, listener)
* 注意点:
* 和浏览器端的事件不同,监听函数的第一个参数就是触发时传入的参数而不是event对象
* emitter.addListener和emitter.on(eventName, listener)是同一个方法
* 触发事件
* emitter.emit(eventName[, ...args])
* 注意点:
* 和浏览器端的事件不同,传入参数不用数组而是参数列表
* 移除事件
* emitter.removeListener(eventName, listener)
* emitter.off(eventName, listener)
* 注意点:
* emitter.removeListener和emitter.off是同一个方法
* emitter.off 新增于: v10.0.0
* 事件

'newListener'事件,当有新的监听被添加是触发,回调函数接受两个参数分别是添加的事件名称和函数的

### stream

* 概念
* 流stream是一种中处理数据的抽象接口,是将数据分割成一段一段的来处理,提高效率
* 流可以是可读的、可写的、或是可读写的,所有的流都是 EventEmitter 的实例

readable.pipe(writable) 将可读流的数据传递给可写流

writable 流指的是只能流进不能流出的流

readable 流是产出数据的流

process.stdin 代表控制台的可读流

process.stdout 代表控制台的可写流

### fs(File System)

同步读写文件

异步读写文件

读写文件流

### http

* http.createServer((req,res)=>{}) 创建一个服务器实例
* req 它实现了 可读流 接口
* res 它实现了 可写流 接口

### url

### path

### querystring

### crypto

### 001-介绍和安装.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### 数据库介绍

* 什么是数据库

数据库(DataBase,简称DB),是一种专门用来管理数据的软件

* 为什么要数据库

因为应用程序中的数据是保存在内存中的,随着程序的执行完毕或者退出,数据就会销毁,而对于大多数情况下我们是需要把用户的数据保存下来的,一个最简单的方法是把用户数据写入文件,但是随着程序的功能越来越复杂,数据量越来越大,用文件管理数据就显得力不从心了,所以需要一个专门来管理数据的软件来帮忙,他就是数据库

* 数据的分类和选择
* SQL数据库
* Oracle ,SQL Server,DB2,Sybase
* MySQL,PostgreSQL,sqlite
* NoSQL(not only sql)数据库
* MongoDB
* CouchDB
* HBase
* Redis
* SQL数据库结构化,适合中大型系统
* NoSQL数据库数据模型比较简单,但操作比较灵活,适合微架构
* SQL数据库和NoSQL数据库不是二选一的关系,而是相互辅助和补充的关系

### mongodb 的安装

https://www.mongodb.com/

https://docs.mongodb.com/manual/reference/

http://www.mongodb.org.cn/tutorial/

* 注意环境变量的配置

### 002-基本使用.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### mongodb 存储基本组成

* database 数据仓库,数据仓库中存放集合,一个mongodb中可以有多个数据仓库
* collection 集合,集合中存放文档,一个集合可以有多个文档
* document 文档,文档是有字段(field)和值组成,数据结构和JSON基本一样,是数据库中的最小单位,是存储和操作的内容

### mongodb 常用终端命令

* 启动Mongodb: mongod

注意点:

* 启动前需要创建默认数据库存储目录
* 如果不用默认的存储目录,可以用 mongod --dbpath 自定义路径 来指定数据库存储目录
* 连接MongoDB: mongo

### 数据库操作

创建数据库:use DATABASE\_NAME,如果数据库不存在，则创建数据库，否则切换到指定数据库。

查看数据库:show dbs

注意:

* 只有数据库中插入了数据后用show dbs才能查看到
* 默认的数据库为test,如果你没有创建新的数据库,集合将存放在test数据库中

查看当前数据库名:db

删除数据库:db.dropDatabase()

注意:

* 删除前要切换到需要删除的数据库

### 集合操作

显示当前数据中的集合: show collections

删除当前数据库中指定的集合:db.COLLECTION\_NAME.drop()

### 文档操作

* 插入文档:db.COLLECTION\_NAME.insert(document)

注意:

* 新插入的数据系统会自动添加一个\_id的主键,也可以自己指定
* db.COLLECTION\_NAME.save(document),不指定 \_id 和insert一样,指定 \_id 则是更新该 \_id 的数据
* 查看文档:db.COLLECTION\_NAME.find(query,projection)

注意:

* query是查询的条件,是一个对象,不传的话是查询所有
* query查询可以用查询操作符,常见的查询操作符有
* 比较查询操作符
* $lt 小于
* $lte 小于等于
* $gt 大于
* $gte 大于等于
* $in 包含
* 逻辑查询操作符
* $and 与
* $or 或
* $not 非
* 该方法返回一个返回指向符合条件的所有文档的游标对象
* 游标对象的常用方法有
* skip(num)
* limit(num)
* toArray()
* length()
* count()
* sort({ field: value }),1升序,-1降序
* projection是指定那些字段显示或者隐藏,格式为{ field1: <value>, field2: <value> ... },1显示,0隐藏
* 更新文档:db.COLLECTION\_NAME.update(query,update)

注意:

* query是更新的条件,是一个对象
* update是有更新操作符和更新后的值组成的对象
* 常见更新操作符有
* $set,设置某一个字段的值,如果匹配到的文档有该字段则修改没有则新增
* $unset,删除字读
* update默认更新复合条件的第一条文档,如果需要更新所有
* 可以给update方法增加一个参数{multi:true}
* 也可以用updateMany方法
* 删除文档:db.COLLECTION\_NAME.remove(query,justOne)

注意:

* query是删除的条件,是一个对象,如果是一个空对象则删除所有
* justOne如果设置为true表示只删除第一条

### 集合间关系

* 一对一 1:1, 例如老公和老婆

可以通过文档关联或者内嵌文档的形式来实现

* 一对多 1:n, 例如老爸和孩子

可以通过文档关联或者内嵌文档的形式来实现

* 多对多 n:n, 例如老师和学生

可以通过文档关联的形式来实现

### 索引

为了提高查询效率,通常会为经常需要查询的字段创建索引,索引是对文档中的字段排序的一种数据结构,可以理解为书的目录

创建索引 db.COLLECTION\_NAME.createIndex( {field1: 1|-1, field2: 1|-1,….}<,{name:”indexName”, unique:true|fase}> )

* 1是升序,-1是降序
* 第二个参数是可选的，可以指定索引的名字,设置索引是否唯一
* 如果设置了unique为true,则该字段不能重复插入
* 如果新建索引时字段中有重复值，新建会失败

查看索引 db.COLLECTION\_NAME.getIndexes()

删除索引

* db.COLLECTION\_NAME.dropIndex( {field1: 1|-1} 删除指定的索引
* db.COLLECTION\_NAME.dropIndexes() 删除所有索引(除了\_id)

### 003-Driver和mongoose.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **NodeJs Driver**

http://mongodb.github.io/node-mongodb-native/3.1/quick-start/quick-start/

mongodb NodeJs Driver 安装

CRUD 体验

### **mongoose 基本使用**

mongoose 安装

CRUD 体验

连接DB 定义Schema 用Schema生成Model

* CRUD 进阶
* 新增
* Model.insertMany()
* Model.prototype.save()
* Model.create()
* 查找
* Model.find()
* Model.findById()
* Model.findOne()
* 更新
* Model.update()
* Model.updateMany()
* Model.updateOne()
* 删除
* Model.remove()
* Model.deleteOne()
* Model.deleteMany()
* 获取去重后的值
* Model.distinct()
* Schema 定义文档的模型
* 常见定义类型
* String
* Number
* Date
* Boolean
* ObjectId(mongoose.Schema.Types.ObjectId)
* Array
* 注意点
* 在存储和更新数据时,如果数据的类型和定义字段的类型不一致,mongoose内部会尝试将数据转换为定义的字段类型,如果转换失败则操作失败
* 定义类型的方法有两种,一种是直接用类型,另一种是用一个对象,类型是对象type属性的值
* MongoDB存储的是格林尼治标准时间(GMT时间),和们的时区错8个小时,存储时会减去8小时
* 通常我从数据库中取出的时间需要做格式化,可以借助 moment 包来格式化时间
* MongoDB会自动添加ObjectId的id,如果字段类型是ObjectId,插入时会把字符串转化为ObjectId
* 验证
* 内置验证
* 所有的类型都有required(必须)验证
* Number类型有min(最小)和max(最大)值验证
* String有enum(枚举),maxlength(最大长度)和minlength(最小长度)验证.
* 每一个验证都可以写为: 规则:[值,错误消息]的格式,也可以写为:规则:值
* 自定义验证举例

validate:{

validator:function(v){

return /1[358]\d{9}/.test(v)

},

message:'{VALUE} 不是合法电话号码'

}

* 方法
* 自定义实例方法举例

//注意不要用箭头函数

UserSchema.methods.findMyBlogs = function(callback){

// this是 UserModel的一个实例

// 在Model的原型上有Model.prototype.model()方法,该方法返回一个指定的Model

this.model('Blog').find({author:this.\_id},(err,docs)=>{

callback(null,docs)

})

}

* 自定义模型静态方法举例

//注意不要用箭头函数

UserSchema.statics.findByPhone = function(phone,callback){

//this 是 UserModel

//Model.model()方法返回一个指定的Model,因此this和this.model('User'))相等

this.find({phone:phone},(err,docs)=>{

callback(null,docs)

})

}

* 关联查询 populate

\*\*多个集合之间的关联，可以给允许populate的集合属性添加ref:’User//所写为创建模型时的集合’,指的是其中的属性可以被更改，

Ex:::

Blog.findOne({}).populate(’author’//所写为你所要关联的字段,’name -age’).then(callback).catch(callback);

### 001-express介绍.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **什么是express**

express是nodeJS的WEB开发框架,除了express外,还有其他比较流行的框架,比如koa,egg,hapi等 http://expressjs.com/ http://www.expressjs.com.cn/

### **express 安装**

$ npm install express --save

### **基本使用**

### 002-路由.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **路由结构 app.METHOD(PATH, HANDLER)**

app 是一个 express 实例

METHOD 是某个 HTTP 请求方式中的一个

* get 查询
* post 新增
* put 修改
* delete 删除
* PATH 是服务器端的路径
* 匹配字符串
* 使用正则中的?、+、\* 和 ()
* 使用正则
* HANDLER 是当路由匹配到时需要执行的回调函数
* 使用多个回调函数处理路由需要指定next
* app.all(PATH, HANDLER)

all 不是一个http请求,主要用来加载中间件

### **路由模块化**

* 用express.Router实例化一个router对象
* 使用router.METHOD(PATH, HANDLER)来处理路由
* 导出router对象
* 用app.use(PATH,router对象)来使用导出的router对象

### 003-静态资源.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **托管静态文件**

* app.use(express.static(静态资源目录));

所有的静态资源的请求路径都会相对于静态资源目录, 例如:如果请求路径的是/css/index.css,设置的静态资源目录是static,则服务器端获取文件的路径是static/css/index.css

* app.use('虚拟路径',express.static(静态资源目录))

设置了虚拟路径后,静态资源请求时需要加上虚拟路径 虚拟路径是根本不存在的目录

### 004-中间件.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **中间件介绍**

* 中间件其实就是一个函数,在收到请求和返回响应之间处理一些操作,这个函数可以访问请求对象(req), 响应对象(res)
* 如果当前中间件没有终结请求-响应循环，则必须调用 next() 方法将控制权交给下一个中间件，否则请求就会挂起。
* next()代表中间件数组中的下一个函数,在express内部有一个专门存放发出响应之前要执行的所有函数的数组,即中间件数组,每次执行app.use(fn)都会把函数添加到数组中

### **常用中间件**

* body-parse 处理post请求中间件
* 安装并引入
* 使用中间

//解析json

app.use(bodyParser.json());

//解析urlencoded内容

app.use(bodyParser.urlencoded({ extended: false }));

app.post('/add',(req,res)=>{

console.log(req.body)//post数据

})

### 005-模板引擎.md

[源码模式](https://pan.baidu.com/s/javascript:;" \o ")

### **模板的使用**

* 安装

npm i swig --save

* 集成

swig.setDefaults({

cache: false

})

//1. 配置应用模板

//第一个参数是模板名称,同时也是模板文件的扩展名

//第二个参数是解析模板的方法

app.engine('html', swig.renderFile);

//2.配置模板的存放目录

//第一参数必须是views

//第二个参数是模板存放的目录

app.set('views', './views')

//3.注册模板引擎

//第一个参数必须是view engine

//第二个参数是模板名称,也就是app.engine的第一个参数

app.set('view engine', 'html')

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

//4.渲染模板

//第一个参数是相对于模板目录的文件

//第二个参数是传递给模板的数据

res.render('index',{

title:'跨猪网',

content:'我是内容'

})

})

### **swig的使用**

* 变量

{{ name }}

{{ obj.name }}

* 判断

{ % if name === 'Tom' % }

hello Tom

{ % endif % }

{ % if name === 'Tom' % }

hello Tom

{ % elseif name === 'Leo' % }

hello Leo

{ % else % }

hello guy

{ % endif % }

* 循环

{ % for key, val in arr % }

<p>{ { key } } -- { { val } }</p>

{ % endfor % }

* 继承模板
* 定义父模板
* 父模板里定义block块,子模板可以对这block继承

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title %}{% endblock %}</title>

{% block head %}{% endblock %}

</head>

<body>

{% block content %}{% endblock %}

</body>

</html>

{% extends './layout.html' %}

{% block title %} index {% endblock %}

{% block content %}

<div>

<h1>hello kuazhu</h1>

<div>

{% endblock %}

* 包含模板

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>{% block title %}{% endblock %}</title>

{% block head %}{% endblock %}

</head>

<body>

{% block content %}{% endblock %}

</body>

{% include "./footer.html" %}

</html>

///

\*href=’/url/add’;其中第一个斜杠代表根路径重新走，不带第一个斜杠则是在当前路径下。

、、