**PC**

**1、html5几种存储形式**

本地存储（localStorage && sessionStorage）

离线缓存（application cache）

indexedDB 和 webSQL

**2、localStorage && sessionStorage**

过期时间：localStorage 永久存储，永不失效除非手动删除

sessionStorage 浏览器重新打开后就消失了

大小：每个域名是5M

**3、localStorage API和sessionStorage API一致**

getItem //取记录

setIten//设置记录

removeItem//移除记录

key//取key所对应的值

clear//清除记录

**4、存储的内容**

数组，图片，json，样式，脚本。。。（只要是能序列化成字符串的内容都可以存储）

**本地存储**

存储用户的一些临时数据

本地存储：

**webStorage**  **客服端存储（前端）** 本地存储html5新增的一种API

**localStorage -> 本地永久存储**

**sessionStorage -> 只存储一次性浏览器会话记录 浏览器关闭清除所有值**

只能用于客户端

浏览器定义了一些方法可以直接使用

html5新增的API

Storage = localStorage || sessionStorage;

**添加属性 属性值默认转为字符串**

Storage.setItem(key,value) Storage['key'] = value Storage.key = value

**获取值 返回对应value**

Storage.getItem(key) Storage['key'] Storage.key

如果要存对象的话，要转成字符串 JSON.stringify(obj);

如果要取的话。 JSON.parse(obj)

**Storage不是对象 有length属性 （储存信息个数）**

**Storage.key(1) 根据下标返回对应的属性key**

**替换**

后写的覆盖先写的

**删除值**

Storage.removeItem(key)

**删除全部**

Storage.clear()

获取key 返回下标为index的key

Storage.key(index)

**Cookie**

**可以用于客户端也可以用于服务端**

**是一种能够让网站把少量数据储存到客服端的一种技术；**

**没有 自己封装 jquery.cookie.js**

前后端交互的一种存储方式： 4kb；

**设置**

document.cookie=’key=val’; (open with live...打开浏览器)

**获取**

document.cookie (字符形式，所有信息汇成一个字符串)

**Session** **服务端储存（后端）**

**深拷贝**

JSON.parse(JSON.stringify());

**浅拷贝 es6（合并对象）**

Object.assign({}, defaults, opt); 后面对象覆盖默认对象

有{}表示不覆盖默认值 没有则会覆盖

**异常抛错**

try{

//此处**发生错误**，接着抛出错误

}catch(e){ （e相当于error）

console.log(e) 表示错误类型信息

console.log(e.meaaage) 表示错误信息

//此处用于**接收错误**

}

try catch **可以嵌套**

try :语句测试代码块的错误，一般把可能会出错的代码放在这里

catch：只有try里面的代码块发生错误是，才会执行这里的代码，参数err记录着try里面代码的错误信息

finaally：**无论有无异常里面代码都会执行**

**throw new Error(‘’); 手动抛出错误信息 可以报中文报错信息**

**常见错误**

EvalError 表示错误的原因：与 eval() 有关。

InternalError 代表Javascript引擎内部错误的异常抛出的实例。

RangeError 数值变量或参数超出其有效范围。

ReferenceError 表示错误的原因：无效引用。

SyntaxError 表示错误的原因：语法错误。

TypeError 表示错误的原因：变量或参数不属于有效类型。

URIError　　　　 表示错误的原因：给 encodeURI()或 decodeURl()传递的参数无效

**display:flex; 让所有子元素在同一行显示（不包括子孙元素）**

**:animated 匹配正在动画的元素 is(‘:animated’)判断元素是否动画**

**table.rows 获取表格的行(tr)**

**$.ajax({**

**}).done(function(a,b,c){**

**a==>反复结果**

**b==>状态**

**C==>xhr状态**

**})**

**//$.ajax执行成功则执行done**

**geuBoundingClientRect() 包括四个属性：left right top bottom**

**分别表示元素各边与页面上边与左边的距离**

**find() 函数用来查找目标元素，找到就返回该元素，**

**找不到返回undefined；**

**findIndex() 函数也是查找目标元素，找到就返回元素的位置，找不到就返回-1；**

**[1,2,3].find(funciton(value,index,arr){**

**value:每一个迭代查找的数组元素**

**index：每一次迭代查找的数组元素的索引**

**arr：被查找的数组  
})**

**swiper**

new swiper(‘父元素’，配置项（参数可选）)；

配置项{

**autoplay**:false|true 默认三秒轮播一次

**direction**：‘horizontal’（水平）|‘vertical’（垂直） 轮播方向

**loop**：false|true 无缝轮播

**autoplay**：{

delay：1000|自动时间 ms

}

**pagination**{

el：分页器的元素，

clickable：false|true，点击分页切换轮播

}

**navigation**：{

nextEl：下一张按钮元素

prevEl：上一张按钮元素

}

**scrollbar**:{

el:轮播的元素，

}

**slidesPerView:2， 一屏显示两张图片**

**SlidesPerGroup：2, 每次走两张**

}

口蹄疫引用多个swiper但是要保留swiper-container,这个类多个之间用id或class进行区分

**Require 模块化**

**是AMD规范定义的**

优点：  **实现js文件的异步加载，避免网页失去响应**

**管理模块之间的依赖性，便于代码的编写和维护**

载入模块

async 属性表明这个文件需要异步加载，避免网页失去响应, IE：defer;

data-main 属性的作用是，指定网页程序的主模块

**require() 函数 接受两个参数 第一个参数是一个数组 表示所依赖的模块；**

**第二个参数是一个回调函数**

**require.config({ //配置项，设置文件路径**

**baseUrl: //文件基路径 从根本路径找 ‘scripts/libs’,**

**paths:{ //在不设置基路径的情况下是根据入口文件的路径去找的**

**‘modeA’:’libs/modeA’ (.js不写)**

**}**

**shim:{ //shim参数来进行相应配置不符合AMD规范的组件**

**有两个属性 （deps，exports） deps为数组 表示其依赖的库，**

**exports 表示输出的对象名**

**}**

**})**

**require([‘modeA’,’modeB’],function(){ }) 中括号表示载入模块**

**Function 表示每一个模块的返回值 没有返回值的模块可以不写**

**modeA.js 里面写 define(function(){ })**

**define([‘jquery’],function($){ })**

**插件**

**对象级别插件 （this指向当前调用的jq对象 也就是实例对象）**

$.fn.方法名=Fn;=======>jquery.prototype.方法名=Fn;

在jquery的原型上面扩展一个方法

**单个方法**

$.fn.方法名=function(){ }

**多个方法** (**extend扩展对象**)

$.fn.extend({

方法1:function(){ },

方法2:function(){ }

})

$.fn.conm=function(options){ //**options 是接受当前jq对象传过来的参数**

var defaults={

} **//设置一组默认的参数**

**var data=$.extend({},defaults,options)**  **//合并参数**

**||**

**Object.assign({},defaults,options)**

**($.extend用来扩展对象 和assign作用一样)**

**为了确定this指向谁 return this.each(function(){**

**(这里的this关键字都是指一个不同的dom元素)**

**})**

}

**对象级别插件 必须通过jQuery对象调用 eg: $(‘.box’).方法名()**

**类级别插件直接调用**

**类级别插件**

**全局插件 this指向window**

**单个方法**

$.方法名=function(){ }

**多个方法**

$.extend({

方法1：function(){ },

方法2:function(){ }

....

})

方法名直接调用

**mock**

**生成随机数据，拦截ajax请求**

有返回值 是一个对象 对象里有属性mock

//创建数据

mock.mock(有两个参数：

**1**.地址‘data.json’==> 随机写json数据

**2**.请求方式 对象或者函数

{

@name 默认英文 @cname 中文

|18-30 18至30以内的随机整数

|+1 数字：属性值自动加一

‘date’: @date 表示随机日期

@date('2018-9-10') 固定日期

‘titer’:’@titer’ 随机生成英文标题

’@ctiter’ 随机生成中文标题

"city":"@city()",//表示城市 @city(tru省份 城市

"color":"@color",//16进制颜色

"flag|1":true, 存储的为Boolean

"obj|2":obj,//随机从对象中抽取两个元素

"obj2|1-3":obj,//从obj对象中随机获取1至3个属性

"array|1":arr,//随机从数组中抽取一个元素

（抽取的值不是数组）

"array2|2":arr,//表示把数组中的元素重复两遍

"imgUrl|+1":["four.jpg","five.jpg","six.jpg"]

//array中 按顺序选取元素作为结果

}

)

**对象表示抽取数据不会进行重复**

**数组1表示随机抽取一个元素**

**+1表示按着数组元素的顺序抽取**

**其它情况下表示对元素进行重复**

//请求数据 拦截ajax

$.ajax({

url:’data.json’,

detaType:’json’,

Success:function(data){

Console.log(data)

}

}) 不需要用ajax那个服务器打开

**文件域.files**

**返回上传的文件对象**

src: ['png', 'jpg', 'jpeg', 'gif', 'bmp', 'svg']

})

{

size:文件大小 b字节kb千字节mb兆字节 **1kb=1024b 1mb=1024kb**

name:文件名称

type:文件类型

lastModified:文件创建时间

}

**Filereader对象可以读取本地文件**

new FileReader()

**设置要上传的文件对象**

对象. readAsDataURL(本地文件对象)

**上传成功触发的方法**

对象. onload

**上传结果**

对象. result

返回base64格式的字符串