爱创课堂前端培训

# HTML5

第1天课堂笔记（本课程共1天）

班级：北京前端训练营19期

讲师：彭帅伟

日期：2019年1月3日

彭帅伟老师

QQ: 284337908

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

[HTML5 1](#_Toc7407)

[复习： 3](#_Toc9544)

[一、 HTML5 4](#_Toc10107)

[1.1 存储 4](#_Toc18203)

[1.2 本地存储API 6](#_Toc19906)

[1.3 会话存储 6](#_Toc16552)

[二、 服务器 7](#_Toc27925)

[2.1 http协议 7](#_Toc19295)

[三、 多线程 8](#_Toc27905)

[3.1 postMessage 8](#_Toc23820)

[3.2 onmessage 8](#_Toc11977)

[下午复习： 9](#_Toc12773)

[四、 WebSql 10](#_Toc26056)

[1.1 openDatabase 10](#_Toc24561)

[1.2 transaction 11](#_Toc1967)

[1.3 SQLTransaction 11](#_Toc1719)

[五、 事件推送 13](#_Toc15366)

[5.1 初始化 13](#_Toc1367)

[六、 history 13](#_Toc11348)

[6.1 history.forward 13](#_Toc12925)

[6.2 history.back 14](#_Toc25300)

[6.3 pushState 14](#_Toc11464)

[6.4 replaceState 14](#_Toc5039)

[七、 拖拽事件 15](#_Toc20385)

[7.1 拖拽事件的分类 15](#_Toc19718)

# 复习：

2d是平面：

坐标系是: x + y

x: 正方向向右

y: 正方向向下

3d是立体

坐标系是：x + y + z

x: 正方向向右

y: 正方向向下

z: 垂直于屏幕

translateZ: 该属性是transform的一个子属性， 表示沿着自身Z轴移动了多少像素

perspective: 景深 表示父元素到屏幕的距离

transform-style: perserve-3d： 让子元素保留3d效果

backface-visibility: hidden 背面不可以见

改变旋转轴中心：

transform-origin： 接收两个参数

可以是像素法

可以是单词法

可以是百分比

# HTML5

HTML5是html的第五个版本

里面规定了一些废弃标签，新增了一些标签，新增了一些功能

## 存储

存储是前端仓库的意思

每一个网站分配一个，只要浏览器不卸载， 电脑不崩溃， 它将永远存在

它的生命周期是永久

查看：

|  |
| --- |
|  |

现在该对象中没有任何数据

既然是一个对象，我们就可以使用对象的语法添加数据

比如：

|  |
| --- |
|  |

但是该对象的原型上提供了相应的API，所以建议我们使用原型中的方法

存储数据：

localStorage.setItem(key, value)

key: 要存储的数据名称

value: 要存储的数据 （该方法会默认转为字符串）

比如：

|  |
| --- |
|  |

在存储数组的时候：

|  |
| --- |
|  |

同样的存储对象的时候：

|  |
| --- |
|  |

解决方案：

在存储的时候先转为字符串，在提取的时候再转为原类型

|  |
| --- |
| 1. // 定义一个对象 2. var obj = { 3. age: 20, 4. sex: "男" 5. } 6. // 在存储的时候转为字符串 7. localStorage.setItem("obj", JSON.stringify(obj)) 8. // 提取数据， 转为原类型 9. var obj1 = JSON.parse(localStorage.getItem("obj")); 10. console.log(obj1); |

结果：

|  |
| --- |
|  |

## 本地存储API

设置数据：

使用方式：

localStorage.setItem(key, value)

key: 要存储的数据名称

value: 要存储的数据

获取数据：

使用方式：

localStorage.getItem(key)

key: 要获取的数据名称

删除某一项数据：

使用方式：

localStorage.removeItem(key)

key: 要删除的数据名称

删除全部：

使用方式：

localStorage.clear()

## 会话存储

除了本地存储之外还有一个对象是会话存储对象

它与本地存储之间的区别是生命周期

会话存储的生命周期是从页面打开到关闭

查看：

|  |
| --- |
|  |

使用方式与本地存储API是一致的

# 服务器

简单来说就是可以提供服务的机器

这个服务器可以买， 可以租，可以下载一些模拟软件

我们选择自己写的nodejs

如果想要使用nodejs，必须先下载node.ext程序

在平时查看代码的地址栏：

|  |
| --- |
| 1. file：// …… |

有了服务器之后查看地址栏：

|  |
| --- |
| 1. http://…… |

## http协议

http协议： 超文本传输协议， 里面规定了前端如何发送请求后端如何接收， 后端返回数据前端如何解析

当有了服务器之后，就可以通过http发送请求，请求服务器上的资源

在很久以前，浏览器的目的只是为了渲染页面，并且是静态页面

此时就面临一个问题， 前端发送一个请求后端如何接收，后端返回数据，前端如何解析

于是，http协议就出现了

比如，前端想要请要求一张图片到服务器端， 前端就必须按照http协议中的规定来发送请求， 后端接收请求并返回数据的时候，必须按照正确的格式返回数据

还有另外一个问题， 服务器的承载（连接）数量是有限的， 比如说一个服务器的承载数量是10， 此时前端发送过来一个请求，并且是有状态（持久连接），占用了一个连接， 此时服务器最多连接数量是10， 但是当时浏览器的目的只是为了渲染页面，所以没有必要是持久连接， 所以特意给http设置为无状态（没有持久连接）

连接过程：

浏览器发送http请求，服务器接受响应返回数据，断开连接，浏览器接收并渲染页面

由于断开了连接， 所以此时可连接的数量远大于有持久连接的

服务器最为重要的一方面可以提供服务器环境， 前端某些时候是需要服务器环境支持的

比如：多线程，事件推送， ajax，canvas相关功能

# 多线程

js是单线程，当js在执行的时候，页面是停止渲染的

HTML5提供了一个Worker函数， 用于开辟一个新的线程

使用方式：

1 将要计算的代码提取到一个js文件中

2 初始化Worker函数

特点：需要服务器环境

举例：

第一步:

computed.js

|  |
| --- |
| 1. var fibonacci =function(n) { 2. return n <2 ? n : arguments.callee(n -1) +arguments.callee(n -2); 3. }; 4. console.log("计算开始"); 5. fibonacci(40); 6. console.log("计算结束"); |

第二步：

|  |
| --- |
| 1. var worker = new Worker("computed.js"); |

## 3.1 postMessage

开启额外的线程用于计算，可以通过postMessage将计算的结果发送到主线程

|  |
| --- |
| 1. var i = 0; 2. // 开启定时器 3. setInterval(function() { 4. i++; 5. // console.log(i); 6. postMessage(i); 7. }, 1000) |

## 3.2 onmessage

额外的线程已经将结果发送到主线程， 在主线程可以通过onmessage来接受信息

|  |
| --- |
| 1. // 接收信息 2. worker.onmessage = function(e) { 3. // console.log(e); 4. // 具体的数据在e.data中 5. console.log(e.data); 6. } |

## 下午复习：

HTML5：

存储：

本地存储： localStorage

特点：它的生命周期是永久

API:

添加数据：

localStorage.setItem(key, value)

key: 要存储的数据名称

value: 存储的数据 （该方法会默认转为字符串）

获取数据：

localStorage.getItem(key)

key: 要获取的数据名称

删除某一项：

localStorage.removeItem(key)

key: 要删除的数据名称

删除全部：

localStorage.clear()

会话存储：

sessionStorage

特点： 它的生命周期是从页面打开到关闭

# WebSql

前端数据库，本来是HTML5的规范， 但是后来又没有了

## openDatabase

这是一个方法，是window的属性



作用：用于打开或者是创建数据库

使用方式：

openDatabase(dbname, version, descript, size)

dbname: 数据库名称

version: 版本号

descript: 描述

size: 大小

|  |
| --- |
|  |

观察数据库：

|  |
| --- |
|  |

执行openDatabase(“19期数据库”， 1， “这是19期数据库”, 1024 \* 1024)

|  |
| --- |
|  |

创建数据库之后返回一个数据库对象：

|  |
| --- |
|  |

## transaction

这是一个方法，接受一个参数是回调函数，函数中的参数就是事物对象

|  |
| --- |
| 1. db.transaction(function() { 2. console.log(arguments); 3. }) |

查看：

|  |
| --- |
|  |

## SQLTransaction

通过ta调用SQLTransaction原型上的方法，可以执行sql语句

使用方式：

ta.executeSql(sql, [], successCallback, failedCallback)

sql: sql语句

[]: 数组， 替代数组

successCallback: 成功时候执行的回调函数

failedCallback: 失败时候执行的回调函数

举例：

|  |
| --- |
| db.transaction(function(ta) {  ta.executeSql("create table student (name varchar(20), age int(10))", [], function() {  console.log("创建成功");  }, function() {  console.log("创建失败");  })  }) |

结果：

|  |
| --- |
|  |

举例：

|  |
| --- |
| 1. // 插入数据 2. db.transaction(function(ta) { 3. // console.log(arguments); 4. // console.log(ta); 5. ta.executeSql("insert into student values('小明', 12)", [], function() { 6. console.log("插入成功"); 7. }, function() { 8. console.log("插入失败"); 9. }) 10. }) |

结果：

|  |
| --- |
|  |

# 事件推送

正常情况下，只能够从浏览器端主动推送信息到服务器端， 服务器端接受请求并返回数据， 关闭连接， 无法从服务器端主动推送信息到浏览器端， 所以HTML5提供了一个EventSource函数，用于主动推送信息到浏览器端

## 5.1 初始化

使用EventSource函数与Worker一样都需要初始化

使用方式：

var es = new EventSource(path)

path: 接口

都需要服务器环境支持

接口：

|  |
| --- |
| 1. // 判定是否是hehe 2. if(pathname === "/hehe") { 3. // 设置响应头 类型 4. res.setHeader("content-type", "text/event-stream"); 5. // 返回内容 6. res.end("data:" + "呵呵" + "\n\n"); 7. // 终止函数继续执行 8. return; 9. } |

每3s推送过来一个呵呵：

|  |
| --- |
|  |

Worker函数与EventSource函数的报错情况：

在没有传递参数的时候：

|  |
| --- |
|  |

没有在服务器环境下的报错：

|  |
| --- |
|  |

# history

## history.forward

该方法会加载历史记录列表中的下一个URL

调用该方法等价于点击了前进按钮或者是调用history.go(1)

|  |
| --- |
| 1. // 添加点击事件 2. next.onclick = function() { 3. // history.forward(); 4. history.go(1); 5. } |

## history.back

该方法会加载历史记录列表中上一个URL

调用该方法等价于点击了后退按钮或者是调用history.go(-1)

|  |
| --- |
| 1. pre.onclick = function() { 2. // history.back(); 3. history.go(-1); 4. } |

## pushState

作用：添加新的历史记录

使用方式：

history.pushState(obj, title, url)

obj: 是一个对象

title: 新的网页标题 （一般省略）

url：新的网页url

举例：

|  |
| --- |
| 1. // 添加新的历史记录 2. history.pushState({"name": "老王"}, null, "index.html#aaa"); |

## replaceState

作用: 替换当前历史记录

使用方式：

history.replaceState(obj, title, url)

obj: 是一个对象

title: 网页标题

url: 新的url

replaceState与pushState之间的区别：

pushState用来添加新的历史记录

replaceState替换当前历史记录

举例：

|  |
| --- |
| 1. // 添加历史记录 2. history.pushState({"name": "老李"}, null, "index.html#aaa"); 3. history.pushState({"name1": "老李"}, null, "index.html#bbb"); 4. history.replaceState({"name2": "老王"}, null, "index.html#ccc"); |

# 七、 拖拽事件

## 拖拽事件的分类

拖拽时候的样式：

|  |
| --- |
|  |

拖拽事件：

|  |
| --- |
| 1. // ondrag 2. img.ondrag = function() { 3. console.log("拖拽开始"); 4. } |

当拖拽着图片的虚影的时候会触发该事件， 如果拖拽着图片的虚影并且在移动中会加快触发该事件的频率

拖拽开始： ondragstart

举例：

当拖拽图片虚影的时候，只触发一次ondragstart事件

|  |
| --- |
|  |

拖拽进入： ondragenter

举例：

|  |
| --- |
|  |

当图片的虚影进入到垃圾桶范围的时候会触发该事件

拖拽离开事件： ondragleave

举例：

|  |
| --- |
|  |

当图片的虚影离开垃圾桶范围的时候会触发该事件

悬浮事件：ondragover

举例：

|  |
| --- |
|  |

当图片的虚影进入垃圾桶范围的时候会触发该事件

拖拽结束： ondragend

举例：

|  |
| --- |
|  |

当拖拽垃圾桶图片并且在垃圾桶范围中松开鼠标会触发该事件

丢弃事件: ondrop

举例：

|  |
| --- |
|  |

当松开鼠标的时候

|  |
| --- |
|  |

该事件有一个Bug, 该事件的ondragover事件中的默认行为阻止ondrop事件的执行， 所以想要给一个元素添加ondrop事件，必须给该元素添加ondragover事件并且阻止默认事件

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |