20230920

1. (接day 9.19)DOM

1.1 innerHTML&innerText

作用:

innerHTML: 获取节点元素内的html内容,能更改对应html元素

```
<body>
    <div id="id_div_01">
        <h1>试试</h1>
    </div>
    <script>
        var div1 = document.getElementById("id_div_01");
        console.log(div1)
        div1.innerHTML= "试完了"
        console.log(div);

    </script>
</body>
```

innerText: 获取html内的文本内容,同样也能修改

1.2 动态操作元素的HTML属性

```
<div>这是一个div</div>
    <script>
    var div = document.getElementsByTagName("div")[0];
    div.hidden = "hidden"

    </script>
```

1.3 动态操作元素的CSS样式

1.3.1 操作行内样式

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

</head>
<body>
    <div>这是一个div</div>
    <script>
    var div = document.getElementsByTagName("div")[0];
    div.style = "border: solid 1px gray; width: 200px; height: 300px"
    </script>
</body>
```

```
</html>
```

如果css3属性需要添加前缀, 按照这种方式添加做兼容

```
div.style.transform ="rotate(90deg)"
```

1.3.2 动态设置class属性

提前将style 样式规划好, 给html元素添加 class, 使style样式生效,

元素的class属性 节点对象.className

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        .banner{
            border: solid 1px gray; width: 200px; height: 300px
        }
    </style>
</head>
<body>
    <div>这是一个div</div>
    <script>
    var div = document.getElementsByTagName("div")[0];

    div.className  = "banner"

    </script>
</body>
</html>
```

1.4 操作自定义属性

html允许在标签上添加自定义属性 (没有特定功能)

获取: 元素对象.getAttribute("自定义属性名")

赋值: 元素对象.setAttribute("自定义属性名","属性值")

1.5 总结

DOM编程/DHTML (动态HTML)

js可以动态操作一切HTML属性(设置和读取/赋值和取值)

格式: 对象.属性

单一属性: 例如 readonly disabled 属性值 一律为boolean布尔值 true/false

特殊: class属性 className

js可以动态操作一切css样式

格式: 对象.style.样式

默认操作行内样式

获取html元素对象的几种方式:

document.getElementByid(); 单一节点对象

document.getElementsByName(); 以下为节点对象数组

document.getElementsByTagName();

document.getElementsByClassName();

操作自定义属性: H5要求 自定义属性 必须以data-开头(如data-icss)

getAttribute() setAttribute()

2. 事件机制

定义: 指用户通过外部设备与浏览器发生的一些特定交互, 这些事件会被浏览器监听, 监听后调用响应程序去处理本次交互

2.1 事件中的关键词

事件流: 事件触发的内部机制

事件的绑定方式: 如何将不同类型的事件 与 响应程序形成关联

事件对象: 由浏览器封装好的Object对象, 对象中包含本次事件的全部信息(事件目标, 事件位置, 事件类型, 事件操作方法)

响应函数: 由事件触发的程序/代码块(需要提前被加载)

2.2 事件流

事件流分为两种流程方式:

冒泡方式: 响应函数从底层最具体的元素开始执行, 向上层父节点传播, 由下至上

捕获方式: 由上层元素开始, 执行后向下层子节点传播, 由下至上 (早期流动方式)

2.3 事件绑定方式

绑定方式三种:

2.3.1 HTML级别绑定:

在html中内嵌js程序 (早期的绑定方式)

特点: 编码简洁明了, 不符合DOM标准, html代码和js代码耦合在一起, 没有实现技术分离

```
<body>

    <button onclick="func()">点击</button>
    <script>
        function func(){
            alert("点击了")
        }
    </script>
</body>
```

2.3.2 DOM 0级 事件绑定方式

定义: 将响应程序直接赋值为DOM元素对象的事件属性

格式:

元素对象.事件属性 = 函数名 !!! 不能写为'函数名()'

```
<button id="id_button_01">点击</button>
<script>
    document.getElementById("id_button_01").onclick = function (){
        alert("执行了");
    }
</script>
```

特点:

符合DOM早期版本规定

可以使用函数名/匿名函数作为响应函数

无法传递参数, 可以通过其他方式传递(全局变量)

可以通过 xxx.onclick = null 解除绑定

不符合 DOM2 标准

2.3.3 DOM 2级 事件绑定

添加事件绑定(添加监听器):

格式:

元素对象. addEventListener(eventType,func,eventFlow);

eventType: 事件类型, 如click(省略前缀on)

func: 响应函数 可以为函数名也可以为匿名函数(之后无法接触绑定)

eventFlow: 设定事件流. 默认为false. true冒泡 false捕获(通常为默认值)

```
document.getElementById("id_button_01").addEventListener("click",function (){
    alert("点击")
})
```

移除事件绑定(移除监听器):

格式:

元素对象.removeEventListener("已添加监听器")

```
document.getElementById("id_button_01").removeEventListener("click",function (){
    alert("点击")
})
```

2.4 事件类型

定义: 用户交互的方式, 如单机,双击,光标进入等

分类:

焦点事件: 跟光标有关

鼠标事件: 鼠标动作

键盘事件: 键盘

窗口事件: 针对窗口的动作

变动事件: 如, 下拉框选中, 单选多选的选中/反选

2.4.1 焦点事件

focus: 得到焦点事件（光标进入)

onfocusin

onfocusin 焦点进入事件不能和弹窗一起使用 会出现死循环

blur: 失去焦点事件（光标移出)

2.4.2 鼠标事件

事件名称	事件含义
click	点击事件
dblclick	双击事件
mousedown	鼠标按下触发
mouseup	鼠标按钮松开时触发
mouseover	鼠标移动到某元素上方时触发 (鼠标滑过)
mousemove	鼠标在元素范围内移动时触发
mouseout	鼠标离开对象范围时触发
mousewheel	鼠标滚轮滚动时触发
contextmenu	鼠标右键菜单弹出时触发

2.4.3 键盘事件

事件名称	含义
keydown	键盘按下时调用, 如果不松手则反复调用
keyup	键盘送开始调用
keypress	键盘按下时调用, 如不松手则反复调用

可以通过事件对象.keyCode获取字符码

```
document.getElementById("ipt_01").addEventListener("keydown",function (e){
    document.getElementById("div_01").innerText = e.keyCode
})
```

2.4.4 窗口事件

resize 重置窗口大小

2.4.5 变动事件

change 譬如下拉框内容会发生改变

```
<select onchange="console.log(this.value)">
    <option>这里</option>
    <option>那里</option>
    <option>到底在哪里</option>
</select>
```

2.5 关键字this

this 在事件中, 代表当前事件的对象

谁触发了this,那么this就代表谁

通过this在函数中, 获得触发文本框的内容

```
<input type="text" id="ipt_01" onblur="method(this)">

<script>
    function method(data){
        console.log(data);
        console.log(data.value);
    }
</script>
```

2.6 onload事件

定义: 预加载事件 由window调用

功能: 在页面加载完成后 立即执行(通常用于body标签, 作用为等页面加载完毕, 包括html,css文件再执行脚本代码, 如js)

文本会先读取js代码再执行html和css(是个bug)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
    console.log(data);
        console.log(data.value);
        </script>
</head>
```

```
<body>
<select onchange="console.log(this.value)">
    <option>这里</option>
    <option>那里</option>
    <option>到底在哪里</option>
</select>
</body>
</html>
```

使用onload事件可以消除

方法1: 所有js代码写在函数中,由来调用

方法2:

```
window.onload = function(){

}
```

本质: 绑定方式不同

2.7 事件对象

event: 事件触发过程中, 自动生成的对象, 包含事件所有信息

作业

1.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <input type="text" value="4" id="id_ipt_01">
    <button onclick="func01()">点击累加</button>
    <input type="text" readonly value="1" id="id_ipt_02">

    <script>




        function func01(){
        var ipt1 = parseInt(document.getElementById("id_ipt_01").value);
        var ipt2 = parseInt(document.getElementById("id_ipt_02").value);
        console.log( ipt1+ipt2);

        document.getElementById('id_ipt_02').value = ipt1+ipt2
        }

    </script>
</body>
```

```
</html>
```

2.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="../../../../js/vue.js"></script>
</head>
<body>
    <div id="app">
        <img :src="currentImage" alt="Image" @click="nextImage" />
      </div>

      <script>

var vm = new Vue({
  el: '#app',
  data: {
    images: [
      'img/img1.jpg',
      'img/img2.jpg',
      'img/img3.jpg',
    ],
    currentIndex: 0
  },
  computed: {
    currentImage() {
      return this.images[this.currentIndex];
    }
  },
  methods: {
    nextImage() {
      this.currentIndex = (this.currentIndex + 1) % this.images.length;
    }
  }
});


      </script>
</body>
</html>
```