正课:

1. \*\*\*定时器

2. BOM常用对象: location navigator

3. \*\*\*event

1. \*\*\*定时器:2种:

1. 周期性定时器:

什么是: 让程序每隔一段时间间隔，反复执行同一函数

何时: 只要让程序自动每隔一段时间间隔反复执行任务

如何: 3件事:

1. 任务函数: 定时器反复调用的函数

2. 启动定时器: var timer=setInterval(task,间隔ms)

强调: task是回调函数，不止执行一次，不加括号！

3. 停止定时器: clearInterval(timer)

timer是定时器的序号(一个从1开始整数)，唯一标识

只能在启动定时器时获得

2种:

1. 用户手动停止

2. 自动停止: 在任务函数中，设定临界值，一旦到达临界值，就自动调用clearInterval

2. 一次性定时器:

什么是: 让程序先等待一段时间后，自动执行一次。执行后，自动停止。

何时: 只要让程序先等待一段时间后，再执行一次。

如何: 3件事:

1. 任务函数:

2. 启动定时器: var timer=setTimeout(task,等待ms)

3. 停止定时器: clearTimeout(timer)

指在执行之前，停止等待，取消执行。

2. BOM常用对象:

location: 封装当前窗口正在打开的url的对象

属性:

.href 获取或设置当前窗口正在打开的完整url

.protocol: 协议

.host: 主机+端口号

.hostname: 主机

.port: 端口号

.pathname: 相对路径

.hash: 锚点地址

.search: 查询字符串

方法:

.replace("新url") 用新url代替现有url，实现在当前窗口打开，禁止后退

.assign("新url") 修改location的href属性，实现在当前窗口打开，可后退

在当前窗口打开,可后退:

open("新url","\_self")

location.href="新url"

location.assign("新url") 等效于location.href="新url"

location="新url"

.reload() 重新加载当前页面——刷新!

普通刷新: 在刷新时，优先使用浏览器本地缓存的文件

本地缓存没有，或者过期，才从服务器重新下载

history.go(0)

location.reload()

F5

强制刷新: 无论浏览器缓存中是否有想要的文件，每次都强迫从服务器重新下载新文件

location.reload(true)

navigator: 封装浏览器软件配置信息的对象

.cookieEnabled: 获取当前浏览器是否启用cookie

什么是cookie: 客户端硬盘上持久保存用户私密数据的小文件

为什么: 内存中的所有数据，只要程序关闭，就丢失

何时: 只要希望持久保存一个数据时

如何: 下个阶段讲

.plugins: 封装浏览器插件信息的集合

插件: 为浏览器添加新功能的小软件

何时: 只要判断浏览器是否安装了某个插件

如何判断:

navigator.plugins["插件名"]!==undefined 说明已安装

.userAgent: 保存浏览器名称，内核和版本号的字符串

3. \*\*\*event

什么是事件: 浏览器自动触发的或用户手动触发的页面内容状态的改变.

什么是事件处理函数: 在发生事件时，浏览器自动调用的函数

如何绑定事件处理函数: 3种:

1. 在html中绑定事件处理函数:

<ANY on事件名="js语句">

问题: 1. 不符合内容与行为分离的原则,不便于维护

2. 不可重用！

总结: 几乎不用

2. 在js中，为事件处理函数属性提前赋值一个函数对象:

ANY.on事件名=function(){

... this->ANY ...

}

问题: 一个事件，只能绑定一个处理函数, 因为是赋值

3. 在js中，为元素添加事件监听:

什么是事件监听: 记录元素的事件和要触发的处理函数之间关系的一种对象。

在绑定事件时手动添加

当事件发生时，浏览器会找到对应的事件监听，执行其中的处理函数.

如何添加事件监听:

ANY.addEventListener("事件名",处理函数)

如何移除事件监听:

ANY.removeEventListener("事件名",处理函数)

强调: 如果可能移除一个事件监听，在绑定时，不能使用匿名函数，必须使用有名的函数

因为在移除时，要求找到原处理函数对象，而匿名函数，时无法找到原处理函数对象的。

事件模型: 从触发事件，到所有处理函数执行完的过程

DOM: 3个阶段

1. 捕获: 由外向内，依次记录各级父元素上绑定的处理函数

2. 目标触发: 优先出发目标元素上的事件处理函数

目标元素: 实际最初出发事件的元素

3. 冒泡: 由内向外，按捕获顺序的反向，依次出发父元素上的事件处理函数

事件对象:

什么是: 事件发生时，自动创建的，封装事件信息的，并提供操作事件API的对象。

何时: 2种:

1. 想获得事件信息时

2. 修改事件的默认行为

如何:

获取: DOM: 事件对象，默认会作为事件处理函数的第一个参数传入！

ANY.on事件名=function(e){...e->自动获得事件对象...}

操作:

1. 取消冒泡/停止蔓延: e.stopPropagation();

2. 利用冒泡:

优化: 尽量少添加事件监听！

为什么: 浏览器在查找事件监听时，使用的是遍历方式。

事件监听的个数，直接影响查找速度，进而影响事件触发的响应速度。

何时: 只要多个子元素，需要绑定相同的事件时

如何: 将事件仅绑定在父元素上一份，所有子元素共用

难题: 2个:

1. 如何获得目标元素:

错误: this -> 父元素

解决: e.target->实际触发事件的子元素--目标元素

2. 目标元素一定是想要的吗？

解决: 事件处理函数中，先判断目标元素的特征，再决定是否执行操作

3. 取消事件/阻止默认行为

为什么: 有些事件的默认行为，不是想要的

何时: 2种:

1. 事件执行过程中出错了，希望不再执行

2. 事件本身的默认行为，不想要

比如:

1. a元素的锚点地址:

默认: 单击带有锚点地址的a，url末尾会自动添加#锚点地址.

总结: 今后凡是用a当按钮用时，都要阻止默认添加#锚点地址.

2. 阻止表单元素提交:

默认: 只要单击type="submit"的按钮，都会提交表单。

如何阻止表单提交:

事件: form.onsubmit事件: 在正式提交表单之前触发，是所有表单提交的最后一关.

总结: 自定义表单提交: 2种:

1. input type="button" + form.submit()

2. input type="submit"

+ form.onsubmit事件

+ e.preventDefault();

如何: e.preventDefault()