正课:

1. \*\*\*\*event:

1. \*\*\*\*event

什么是: 用户手动触发的或浏览器自己触发的页面状态的改变。

当事件发生时，都可以执行事件处理函数来响应事件的操作。

绑定事件处理函数: 3种:

1. 在HTML中绑定: <ANY on事件名="js语句"

比如: <button onclick="fun()">

问题: 不符合内容与行为分离的原则，不便于维护

2. 在js中绑定:

1. on事件名: elem.on事件名=function(){

this->elem

}

局限: 一个事件，是能绑定一个处理函数

2. addEventListener: 可一个事件同时绑定多个处理函数

elem.addEventListener("事件名",fn);

还可移除:

elem.removeEventListener("事件名",函数名)

问题: 绑定时如果使用匿名函数绑定，移除时无法找到原函数

解决: 如果一个处理函数可能被移除，就必须用有名的函数

事件模型/周期:

3个阶段:

1. 捕获: 由外向内，记录各级父元素上绑定的事件处理函数——只记录，不触发

2. 目标触发: 优先触发目标元素上的事件处理函数

3. 冒泡: 由内向外，按捕获顺序的反向，依次执行父元素上的事件处理函数

事件对象: e

什么是: 事件发生时，自动创建的，封装事件信息的对象

如何获取: 事件对象，默认作为处理函数的第一个参数传入: 事件处理函数(e){ e//自动获得事件对象 }

何时: 只要希望获取事件的数据，或修改事件的默认行为时

包括:

取消冒泡: e.stopPropagation();

利用冒泡:

问题: 浏览器通过遍历方式查找事件处理函数执行。如果添加的事件监听越多，遍历越慢，网页响应速度越慢

优化: 尽量少的添加事件监听

如何: 如果多个平级子元素绑定相同事件时，可在父元素仅添加一个事件监听，所有子元素共用！

问题: 1. 如何获得目标元素:

错误: this -> 父元素

正确: e.target

2. 可能目标元素不是想要的,就要先鉴别目标元素，再决定是否执行操作

另一个作用: 如果一个父元素下的子元素，需要动态生成，并绑定事件。则必须将事件绑定在父元素。动态生成的子元素，才能自动使用父元素的事件，而不需要反复单独绑定。

阻止默认行为: e.preventDefault();

2个典型应用:

1. 阻止a元素作为按钮时，自动添加#锚点地址

2. 表单验证未通过时，阻止默认的提交