正课:

DOM:

1. 添加/删除:

2. HTML DOM常用对象

BOM:

1. BOM模型:

2. 打开和关闭窗口

3. history

4. location

5. \*\*\*定时器

DOM

1. 添加/删除:

添加:3步:

1. 创建空元素:

var elem=document.createElement("标签名");

比如: var a=document.createElement("a");

<a></a>

2. 设置关键属性:

比如: a.href="http://tedu.cn";

a.innerHTML="go to tedu";

<a href="http://tedu.cn">go to tedu</a>

3. 将新元素添加到DOM树: 3种:

在指定父元素末尾追加: parent.appendChild(elem)

在当前子元素前插入: parent.insertBefore(elem,child)

替换现有子元素: parent.replaceChild(elem,child)

\*\*\*优化: 尽量减少操作DOM树的次数

为什么: 每操作一次DOM树，就会导致重新layout和paint

原理:

html -> DOM Tree

↓

Render Tree->\*\*\*layout\*\*\*->paint

↑ 计算每个元素

css -> COM 的绝对布局位置

如何: 2种:

1. 如果同时添加父元素和子元素，应该先在内存中，将所有子元素添加到父元素中，最后，再将父元素一次性添加到DOM Tree

2. 如果同时添加多个平级子元素，应该先将所有子元素添加到文档片段中，再将文档片段一次性添加到DOM树

文档片段: 内存中，临时保存一棵DOM子树的虚拟父元素对象

何时: 只要添加多个平级元素时

如何: 3步:

1. 创建文档片段:

var frag=document.createDocumentFragment();

2. 将子元素添加到文档片段中

frag.appendChild(child);

3. 将文档片段添加到DOM Tree

parent.appendChild(frag);

强调: frag不占用页面元素位置，将子元素添加到父元素后，自动释放！

删除: parent.removeChild(child)

child.parentNode.removeChild(child)

2. HTML DOM常用对象:

Image: 代表一个img元素

创建: var img=new Image();

select/option

select:

属性: .value 获得当前选中项的值，

如果当前选中项没值，就用innerHTML代替

.selectedIndex 获得当前选中项的下标位置

.options 获得当前select下所有option的集合

.options.length 获得select下option的个数

.length => .options.length

固定套路: sel.length=0; =>sel.options.length=0

方法: .add(opt) => appendChild(opt)

问题:.add不支持文档片段!

.remove(i)

option:

创建: var opt=new Option(text,value);

属性: .text .value .index

table/...:

table:管着行分组

添加: var thead=table.createTHead();

var tbody=table.createTBody();

var tfoot=table.createTFoot();

删除: table.deleteTHead();

table.deleteTFoot();

获取: table.tHead

table.tFoot

table.tBodies[i]

行分组:管着行

添加: var tr=行分组.insertRow(i)

在i位置插入一个新行。原i位置的行向后挤压

固定套路:

1. 末尾追加: .insertRow()

2. 开头插入: .insertRow(0)

删除: 行分组.deleteRow(i)

删除i位置的行。

强调: i，要求是在行分组内的下标

问题: 行分组内的相对下标，无法自动获得

解决: 都用table.deleteRow(tr.rowIndex)

tr.rowIndex可自动获得当前tr相对于整个表格的下标位置

table.deleteRow(i) i刚好需要相对于整个表格的下标位置

获取: .rows

行管着格:

创建: var td=tr.insertCell(i)

在i位置插入一个格，原i位置的格向右挤压

固定套路: 不加i，表示末尾追加

强调: 只能创建td，不能创建th

删除: tr.deleteCell(i)

获取: .cells

form/...:

form:

获取: var form=document.forms[i/id/name];

属性: .elements 包含所有表单元素的集合

包括: input textarea button select

.elements.length获得表单中表单元素的个数

.length => .elements.length

固定套路: form.elements[form.length-1];

方法: .submit(); 手动提交表单

element:

获取: var elem=form.elements[i/name];

其实可以更简单: 如果一个元素上有name属性

var elem=form.name

方法: .focus() 让当前表单元素获得焦点

.blur()

BOM: Browser Object Model

1. BOM模型

什么是: 专门操作浏览器窗口的API——没有标准

包括:

window: 2个角色:

1. 代替ES中的global充当全局作用域对象

2. 封装所有BOM的API

包括:

history, location, document, navigator, screen, event

2. 打开和关闭窗口:

window.open("url","target"); 打开新窗口，并在新窗口中加载新url

window.close(); 关闭当前窗口

总结: 4种打开链接的方式:

1. 在当前窗口打开，可后退

html: <a href="url" target="\_self">

js: open("url","\_self");

2. 在当前窗口打开，不可后退

js:

3. 在新窗口打开，可打开多个

html:<a href="url" target="\_blank">

js: open("url","\_blank");

4. 在新窗口打开，只能打开一个

html: <a href="url" target="自定义的name">

js: open("url","自定义的name")

原理: 其实, 每个窗口都有一个唯一的name属性

浏览器规定: 同一个name属性的窗口，只能打开一个

后打开的同名窗口会替换先打开的。

其实, target属性，就是在为新窗口指定name属性

预定义: \_self: 用当前窗口自己的名字打开新窗口，新窗口自然后覆盖当前窗口

\_blank: 不指定窗口名，浏览器会随机生成窗口名，总是打开新窗口

3. history:

什么是: 保存当前窗口打开后，成功访问过得url的历史记录栈

API: history.go(n)

前进一次: go(1)

后退一次: go(-1)

后退2次: go(-2)

刷新: go(0)

4. location:

什么是: 封装当前窗口正在打开的url的对象

API:

属性: .href 完整url

.protocol 协议

.host 主机+端口

.hostname 主机

.port 端口

.pathname 相对路径

.search ?xxx=xxx&xxx=xxx...

.hash #锚点地址

鄙视: