js-web-api

题目

- DOM是哪种基本的数据结构?
  - 树
- DOM操作的常用 `API` 有哪些?
- DOM节点的 `attribute` 和 `property` 有何区别?
  - `property` 只是js对象的属性的修改
  - `attribute` 是对html 标签的修改和获取
- 编写一个通用的事件监听函数
- 描述事件冒泡流程
- 对于一个无限下拉加载图片的页面，如何给每一个图片绑定一个事件
- 手动编写一个 ajax ,不依赖第三方库
- 跨域的几种实现方式
- 描述一下 `cookie` , `sessionStorage` 和 `localStorage` 的区别

知识点

- DOM的本质

> `HTML` 是一种特殊的 `XML` , `DOM` (文档对象模型)是针对 `HTML` 和 `XML` 文档的一个 `API` (应用程序编程接口)。

- DOM节点操作
  - `getElementById`
  - `getElementsByTagName`
  - `getElementsByClassName`
  - `querySelector`
  - `querySelectorAll`
- DOM结构操作
  - 新增节点

```
var p=document.createElement('p');
p.innerHTML='Hello World';
document.body.appendChild(p);
```

  - 获取父节点

```
var p=document.createElement('p');
p.innerHTML='Hello World';
p.setAttribute("id", "child");
document.body.appendChild(p);
var pEle = document.getElementById('child');
console.log(pEle.parentElement);
```

  - 获取子节点

```
var body = document.getElementsByTagName('body')[0];
var childNodes = body.childNodes;
console.log(childNodes);
```

- 删除节点

```
var body = document.getElementsByTagName('body')[0];
var childNodes = body.childNodes;
body.removeChild(childNodes[0]);
```

- BOM
  - `navigator`

```
var ua = navigator.userAgent;
var isChrome = ua.indexOf('Chrome');
console.log(isChrome);
```

  - `screen`

```
console.log(screen.width);
console.log(screen.height);
```

  - `location`

```
console.log(location.href);
console.log(location.protocol); // 协议
console.log(location.host); // 域名
console.log(location.pathname); // 路径
console.log(location.search); // ?后面的查询字符串
console.log(location.hash); // #后面的内容
```

  - `history`

```
history.back(); // 返回
history.forward(); // 前进
```

- 事件
  - 通用事件绑定

```
var btn = document.getElementById('btn1');
btn.addEventListener('click', function(e) {
  console.log(e);
})

function bindEvent(elem, type, fn) {
  elem.addEventListener(type, fn)
}
var a = document.getElementById('link1');
bindEvent(a, 'click', function(e) {
  e.preventDefault();
  alert('click')
})
```

```
function bindEvent(elem, type, selector, fn) {
  if (!fn) {
    fn = selector;
    selector = null;
  }
  elem.addEventListener(type, function(e) {
    var target;
    if (selector) {
      // 事件代理
      target = e.target;
      if (target.matches(selector)) {
        fn.call(target, e)
      }
    } else {
      // 不使用代理
      fn(e);
    }
  })
}
```

- 事件冒泡

> 事件开始由最具体的的元素接收，然后逐级向上传播到较为不具体的节点(文档),事件一直冒泡到 `window` 对象。
> - 事件捕获
> > 事件开始由最不具体的节点接收事件，而最具体的节点应该最后收到事件。
> - DOM事件流
> > `DOM2级事件` 规定事件流包括三个阶段：事件捕获阶段、处于目标阶段、事件冒泡阶段。
> - 事件代理

> 利用事件冒泡，将监听的函数绑定在父元素上

```
<div id="div1">
  <a href="#">a1</a>
  <a href="#">a2</a>
  <a href="#">a3</a>
  <a href="#">a4</a>
  <a href="#">a5</a>
  <!-- ... -->
</div>
```

```
// js
var div1 = document.getElementById('div1');
// 第一种方式
bindEvent(div1, 'click', function(e) {
  var target = e.target;
  if (target.nodeName === 'A') {
    console.log(target.innerHTML);
  }
})
// 第二种方式
bindEvent(div1, 'click', 'a', function(e) {
    e.preventDefault();
    console.log(this.innerHTML)
```

```
})
```

- Ajax
  - `XMLHttpRequest`

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/api', false);
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
      console.log(xhr.responseText);
    }else{
      console.log(xhr.status);
    }
  }
}
xhr.send(null);
```

- 跨域
  - 浏览器有同源策略，不允许 `ajax` 访问其他域接口
  - 跨域条件：协议、域名、端口，有一个不同就算跨域
  - 三个允许跨域加载资源的标签
    - `<img src='xxxx'>`

> `<img>` 可用于打点统计，统计网站可能是其他域
> - `<link rel="stylesheet" href="xxxxx">`
> - `<script src="xxxx"></script>`
> `<link>` `<script>`可以使用 CDN ，CDN 也可以用其他域；
> `<script>` 可以用于 `JSONP`；
> - 所有的跨域请求都必须经过信息提供方允许
> - 如果未经允许即可获取，那就是浏览器同源策略出现漏洞

- 跨域的解决方案
  - `JSONP`

> 利用 `script` 标签可以跨域访问资源，获取服务器上动态生成的内容（例如一个函数执行语句），在浏览器端定义函数，当获取了数据后，在浏览器端执行即可，完成跨域访问；

```
window.callback = function(data) {
    console.log(data);
  }
```

```
<script src = 'http://xxx.com/api.js' > < /script>
// 请求这个js，返回 callback({x:100,y:200})
```

  - 服务端设置 http header

```
// 不同的服务端写法不一样
// 第二个参数填写允许跨域的域名，不建议填写 *
response.setHeader('Access-Control-Allow-Origin',
'http://a.com,http://b.com');
```

```
  response.setHeader('Access-Control-Allow-Headers', 'X-Requested-With');
  response.setHeader('Access-Control-Allow-Methods',
'PUT,POST,GET,DELETE,OPTIONS');
  // 设置跨域的cookie
  response.setHeader('Access-Control-Allow-Credentials', 'true');
```

- 存储
  - `cookie`
    - 本身用于客户端和服务端通信
    - 但是他有本地存储的功能，于是被 `借用`
    - 使用 `document.cookie = 'xxx'` 来获取或设置
    - 缺点
      - 存储量太小，只有 4KB
      - 所有的http请求都带着，会影响获取资源的效率
      - API简单，只是一个字符串，存储多个数据时需要拆解和封装才能用 `document.cookie = 'xxx'`
  - `localStorage` 和 `sessionStorage`
    - HTML5专门为存储而设计，最大容量为5M,本地存储
    - API简单易用
      - `localStorage.setItem(key,value)`
      - `localStorage.getItem(key)`
    - 区别
      - `sessionStorage` 浏览器关闭，就会清除，`localStorage` 则需要手动清除
    - 坑
      - `IOS` `safari` 隐藏模式下，`localStorage.getItem` 会报错，建议统一使用 `try-catch` 语句封装