1. js事件应该很清楚吧、那js事件流了解吗?

- **事件流**：指从页面中接收事件的顺序，有冒泡流和捕获流。
- **事件捕获**：当鼠标点击或者触发dom事件时，浏览器会从根节点开始由外到内进行事件传播。即先触发父元素绑定的事件，再触发子元素绑定的事件。

  事件冒泡与事件捕获想法，由内而外，先触发子元素绑定的事件，再出发父元素绑定的事件。

  dom标准事件流的触发的先后顺序为先捕获再冒泡。`IE10及以下不支持捕获型事件`

  如果给目标节点同时注册冒泡和捕获事件，事件触发会按照注册的顺序执行.

  ==由此题可以主动拓展出下题中的事件委托，这都是一套的==

2. 事件绑定方法有哪些?

- 直接在html标签中绑定事件。但代码耦合严重
- 在DOM元素上采用 `on+事件名称` 的方法绑定。
  - 但多个事件处理程序无法同事绑定在一个DOM对象上;
  - 而且动态生成的DOM 对象无法绑定事件。
  - 当目标元素过多时，需要一一绑定，繁琐且性能差。
- 采用addEventListener和attachEvent函数绑定
  - `addEventListener(event,function,boolean)` 函数是W3C标准规定的，IE8及IE8以下不支持；boolean默认值false:冒泡阶段；true:捕获阶段
  - `attachEvent()` 函数是IE特有的，IE8以下浏览器可以使用，可添加多个事件处理函数，只支持冒泡阶段;

兼容各个浏览器的事件绑定和解除:

```
// 事件绑定

function addEvent(element, type, handler) {
    if(element.addEventListener){ //如果支持addEventListener
        element.addEventListener(type, handler, false);
    }else if(element.attachEvent){ //如果支持attachEvent
        element.attachEvent("on"+type, function(){
            handler.call(element); // 将this指向当前DOM对象
        });
    }else{ //否则使用兼容的onclick绑定
        element["on"+type] = handler;
    }
}
```

```
// 事件解绑

function removeEvent(element, type, handler) {
    if(element.addEventListener){
        element.removeEventListener(type, handler, false);
    }else if(element.attachEvent){
        element.detachEvent("on"+type, handler);
    }else{
        element["on"+type] = null;
    }
}
```

这两种绑定方式可以给DOM对象绑定多个事件处理程序

当遇到给多个元素或者给动态生成的元素绑定操作时，可以采用**事件委托（事件代理）**的方式绑定在父节点上。

事件委托原理是事件冒泡，有这样一个结构 `ul>li`; 比如给li加一个click点击事件，那么这个事件就会一层一层的向外冒泡，执行顺序先li再ul。那么我们给外面的ul绑定点击事件，那么里面li触发点击事件的时候，都会冒泡到外层的ul上，触发ul上的事件操作程序。这就是事件委托，委托它们父级代为执行事件。

```
适合用事件委托的事件：
    click, mousedown, mouseup, keydown, keyup, keypress。

本身没有冒泡的特性，不能使用事件委托的事件：
    focus, blur
```

3. 我看你简历上写着了解jsonp跨域，给我讲讲你还知道哪些跨域的解决方案?

因为浏览器出于安全考虑，有同源策略。也就是说，如果协议、域名或者端口有一个不同就是跨域，Ajax 请求会失败。

跨域方式

- **JSONP**

  原理：Web页面上调用js文件不收跨域影响，凡是具有src属性标签都不受同源策略的影响，正是这个特性，我们把资源放到script标签里的src里面，把数据放到服务器上，并且是json形式（js很容易操作json）。script标签中src属性的格式是'url+参数'，比如 url?cd=doJSON，cd是我们和后台协商好的类似于形参的东西，是一个留给我们写处理函数接口，doJSON就是我们事先定义好的函数，也就是回调函数，所有数据会以参数的形式传递给该函数（json是一种数据格式，jsonp是一种约定俗成的非正式传输协议）。

  JSONP 使用简单且兼容老版本浏览器，但是只限于 get 请求。

- **CORS**

  原理：需要浏览器和后端同时支持。要在服务器端的response header里面加一个 Access-Control-Allow-Origin: 指定域名 | | (表示所有域名都可以跨域), 浏览器端便可以发起post的跨域请求。浏览器会自动进行 CORS 通信，实现CORS通信的关键是后端。只要后端实现了 CORS，就实现了跨域。

IE 8 和 9 需要通过 XDomainRequest 来实现。它实现了CORS的部分规范，只支持GET/POST形式的请求。在服务器端，依旧要求在响应报头添加"Access-Control-Allow-Methods"标签（这点跟CORS一致）。创建一个XDomainRequest的实例，调用open()方法，再调用send()方法，请求返回之后，会触发load事件，相应的数据也会保存在responseText属性中

- **Document.domain**

  该方式只能用于二级域名相同的情况下，比如 a.test.com 和 b.test.com 适用于该方式。只需要给页面添加 document.domain = 'test.com' 表示二级域名都相同就可以实现跨。

  一个网页被攻击，另外一个站点就会引起安全漏洞。

- **postMessage**

  h5中有一种功能就是跨文档消息传输

  `getMessageHTML.postMessage(message,targetOrigin)`

  getMessageHTML是接受信息的页面引用，可以是iframe的contentWindow，window.open的返回值 targetOrigin是用于限制getMessageHTML的 填*的时候不做限制

- **服务器代理**

  服务器不受同源策略的影响，把所有资源放在服务器上，让后让服务器上的网页获取这些资源

```
// 发送消息端
window.parent.postMessage('message', 'http://test.com');
```

```
// 接收消息端

var mc = new MessageChannel();
mc.addEventListener('message', (event) => {
    var origin = event.origin || event.originalEvent.origin;
    if (origin === 'http://test.com') {
        console.log('验证通过')
    }
});
```

- **Iframe**

  `iframe+location.hash:`

  利用location.hash来传值，改变hash不会刷新页面，

  缺点：数据直接暴露在url中，数据长度和类型都有限制

  `iframe+window.name:`

  和location.hash方法差不多，主页面有一个iframe，通过修改引入的子页面（同源）来获取window.name值来达到跨域。

  缺点：需要借助同源其他子页面

4. setimeout(function(){},0),先执行括号里面的函数 还是先执行外部代码?

JS 是门非阻塞单线程语言，因为在最初 JS 就是为了和浏览器交互而诞生的。如果 JS 是门多线程的语言话，我们在多个线程中处理 DOM 就可能会发生问题（一个线程中新加节点，另一个线程中删除节点），当然可以引入读写锁解决这个问题。

JS 在执行的过程中会产生执行环境，这些执行环境会被顺序的加入到执行栈中。如果遇到异步的代码，会被挂起并加入到 Task（有多种 task） 队列中。一旦执行栈为空，Event Loop 就会从 Task 队列中拿出需要执行的代码并放入执行栈中执行，所以本质上来说 JS 中的异步还是同步行为。虽然 setTimeout 延时为 0，其实还是异步。这是因为 HTML5 标准规定这个函数第二个参数不得小于 4 毫秒，不足会自动增加。

5. 比如做个新闻编辑上传功能，有网络的情况下直接提交，没有网络的情况下可以离线保存。如何实现这个功能?

（想要把这道题答好，答案分两步。**数据交互, 离线缓存**）

- **数据交互** 利用ajax实现数据交互即可
- **离线缓存**

  cookie，localStorage，sessionStorage都可以用于离线缓存数据，那么选择一个合适的来使用就尤为重要了！下面是它们的区别：

从上表可以看到，cookie 已经不建议用于存储。如果没有大量数据存储需求的话，可以使用 localStorage 和 sessionStorage 。对于不怎么改变的数据尽量使用 localStorage 存储，否则可以用 sessionStorage 存储。所以此处可以采用localStorage进行存储。