

- 垂直方向: `line-height`
- 水平方向: `letter-spacing`

## 41 如何垂直居中一个浮动元素？

CSS

/\*\*方法一： 已知元素的高宽\*\*/

```
#div1{
  background-color:#6699FF;
  width:200px;
  height:200px;
  position: absolute;          //父元素需要相对定位
  top: 50%;
  left: 50%;
  margin-top:-100px ;    //二分之一的height, width
  margin-left: -100px;
}
```

/\*\*方法二:\*\*/

```
#div1{
  width: 200px;
  height: 200px;
  background-color: #6699FF;
  margin:auto;
  position: absolute;          //父元素需要相对定位
  left: 0;
  top: 0;
  right: 0;
  bottom: 0;
}
```

如何垂直居中一个 `<img>` ？(用更简便的方法。)

CSS

```
#container    /**<img>的容器设置如下**/
{
  display:table-cell;
  text-align:center;
  vertical-align:middle;
}
```

## 42 px和em的区别

- `px` 和 `em` 都是长度单位，区别是，`px` 的值是固定的，指定是多少就是多少，计算比较容易。`em` 得值不是固定的，并且 `em` 会继承父级元素的字体大小。
- 浏览器的默认字体高都是 `16px`。所以未经调整的浏览器都符合：`1em=16px`。那么 `12px=0.75em`，`10px=0.625em`。

## 43 Sass、LESS是什么？大家为什么要使用他们？

- 他们是 `CSS` 预处理器。他是 `CSS` 上的一种抽象层。他们是一种特殊的语法/语言编译成 `CSS`。
- 例如Less是一种动态样式语言。将CSS赋予了动态语言的特性，如变量，继承，运算，函数。`LESS` 既可以在客户端上运行（支持 `IE 6+`，`Webkit`，`Firefox`），也可一在服务端运行（借助 `Node.js`）

为什么要使用它们？

- 结构清晰，便于扩展。
- 可以方便地屏蔽浏览器私有语法差异。这个不用多说，封装对-浏览器语法差异的重复处理，减少无意义的机械劳动。
- 可以轻松实现多重继承。
- 完全兼容 `CSS` 代码，可以方便地应用到老项目中。`LESS` 只-是在 `CSS` 语法上做了扩展，所以老的 `CSS` 代码也可以与 `LESS` 代码一同编译

## 44 知道css有个content属性吗？有什么作用？有什么应用？

css的 `content` 属性专门应用在 `before/after` 伪元素上，用于来插入生成内容。最常见的应用是利用伪类清除浮动。

CSS

```
/**一种常见利用伪类清除浮动的代码**/  
.clearfix:after {  
    content:".";           //这里利用到了content属性  
    display:block;  
    height:0;  
    visibility:hidden;  
    clear:both;  
}  
.clearfix {  
    *zoom:1;  
}
```

## 45 水平居中的方法

- 元素为行内元素，设置父元素 `text-align:center`
- 如果元素宽度固定， 可以设置左右 `margin` 为 `auto` ；
- 如果元素为绝对定位，设置父元素 `position` 为 `relative` ，元素设  
`left:0;right:0;margin:auto;`
- 使用 `flex-box` 布局，指定 `justify-content` 属性为`center`
- `display` 设置为 `table-cell`

## 46 垂直居中的方法

- 将显示方式设置为表格， `display:table-cell` ,同时设置 `vertical-align: middle`
- 使用 `flex` 布局，设置为 `align-item: center`
- 绝对定位中设置 `bottom:0,top:0` ,并设置 `margin:auto`
- 绝对定位中固定高度时设置 `top:50%`, `margin-top` 值为高度一半的负值
- 文本垂直居中设置 `line-height` 为 `height` 值

## 47 如何使用CSS实现硬件加速？

硬件加速是指通过创建独立的复合图层，让GPU来渲染这个图层，从而提高性能，

- 一般触发硬件加速的 `CSS` 属性有 `transform` 、 `opacity` 、 `filter` ， 为了避免2D动画在 开始和结束的时候的 `repaint` 操作， 一般使用 `tranform:translateZ(0)`

## 48 重绘和回流（重排）是什么， 如何避免？

- DOM的变化影响到了元素的几何属性（宽高）,浏览器重新计算元素的几何属性， 其他元素的几何
- 属性和位置也会受到影响， 浏览器需要重新构造渲染树， 这个过程称为重排， 浏览器将受到影响的部分
- 重新绘制到屏幕上的过程称为重绘 。引起重排的原因有
  - 添加或者删除可见的DOM元素，
  - 元素位置、尺寸、内容改变，
  - 浏览器页面初始化，

- 浏览器窗口尺寸改变， 重排一定重绘， 重绘不一定重排，

减少重绘和重排的方法：

- 不在布局信息改变时做 DOM 查询
- 使用 `cssText` 或者 `className` 一次性改变属性
- 使用 `fragment`
- 对于多次重排的元素， 如动画， 使用绝对定位脱离文档流， 让他的改变不影响到其他元素

## 49 说一说css3的animation

- css3的 `animation` 是css3新增的动画属性， 这个css3动画的每一帧是通过 `@keyframes` 来声明的， `keyframes` 声明了动画的名称， 通过 `from`、 `to` 或者是百分比来定义
- 每一帧动画元素的状态， 通过 `animation-name` 来引用这个动画， 同时css3动画也可以定义动画运行的时长、动画开始时间、动画播放方向、动画循环次数、动画播放的方式，
- 这些相关的动画子属性有：  
`animation-name` 定义动画名、 `animation-duration` 定义动画播放的时长、 `animation-delay` 定义动画延迟播放的时间、 `animation-direction` 定义 动画的播放方向、 `animation-iteration-count` 定义播放次数、 `animation-fill-mode` 定义动画播放之后的状态、 `animation-play-state` 定义播放状态， 如暂停运行等、 `animation-timing-function`
- 定义播放的方式， 如恒速播放、艰涩播放等。

## 50 左边宽度固定，右边自适应

左侧固定宽度，右侧自适应宽度的两列布局实现

html结构

```
<div class="outer">
  <div class="left">固定宽度</div>
  <div class="right">自适应宽度</div>
</div>
```

html

在外层 `div`（类名为 `outer`）的 `div` 中，有两个子 `div`，类名分别为 `left` 和 `right`，其中 `left` 为固定宽度，而 `right` 为自适应宽度

方法1：左侧div设置成浮动：float: left，右侧div宽度会自拉升适应

```
.outer {  
  width: 100%;  
  height: 500px;  
  background-color: yellow;  
}  
.left {  
  width: 200px;  
  height: 200px;  
  background-color: red;  
  float: left;  
}  
.right {  
  height: 200px;  
  background-color: blue;  
}
```

方法2：对右侧:div进行绝对定位，然后再设置right=0，即可以实现宽度自适应

绝对定位元素的第一个高级特性就是其具有自动伸缩的功能， 当我们将 `width` 设置为 `auto` 的时候（或者不设置，默认为 `auto`），绝对定位元素会根据其 `left` 和 `right` 自动伸缩其大小

```
.outer {  
  width: 100%;  
  height: 500px;  
  background-color: yellow;  
  position: relative;  
}  
.left {  
  width: 200px;  
  height: 200px;  
  background-color: red;  
}  
.right {  
  height: 200px;  
  background-color: blue;  
  position: absolute;  
  left: 200px;  
  top: 0;  
  right: 0;  
}
```

方法3：将左侧div进行绝对定位，然后右侧div设置margin-left: 200px

CSS

```
.outer {  
  width: 100%;  
  height: 500px;  
  background-color: yellow;  
  position: relative;  
}  
.left {  
  width: 200px;  
  height: 200px;  
  background-color: red;  
  position: absolute;  
}  
.right {  
  height: 200px;  
  background-color: blue;  
  margin-left: 200px;  
}
```

方法4：使用flex布局

CSS

```
.outer {  
  width: 100%;  
  height: 500px;  
  background-color: yellow;  
  display: flex;  
  flex-direction: row;  
}  
.left {  
  width: 200px;  
  height: 200px;  
  background-color: red;  
}  
.right {  
  height: 200px;  
  background-color: blue;  
  flex: 1;  
}
```

## 51 两种以上方式实现已知或者未知宽度的垂直水平居中

```
/** 1 **/  
.wrapper {  
  position: relative;  
  .box {  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    width: 100px;  
    height: 100px;  
    margin: -50px 0 0 -50px;  
  }  
}  
  
/** 2 **/  
.wrapper {  
  position: relative;  
  .box {  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
  }  
}  
  
/** 3 **/  
.wrapper {  
  .box {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: 100px;  
  }  
}  
  
/** 4 **/  
.wrapper {  
  display: table;  
  .box {  
    display: table-cell;  
    vertical-align: middle;  
  }  
}
```

## 52 如何实现小于12px的字体效果

`transform:scale()` 这个属性只可以缩放可以定义宽高的元素，而行内元素是没有宽高的，我们可以加上一个 `display:inline-block`；

```
transform: scale(0.7);
```

css 的属性，可以缩放大小

## 三、JavaScript

### 1 闭包

- 闭包就是能够读取其他函数内部变量的函数
- 闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量,利用闭包可以突破作用链域
- 闭包的特性：
  - 函数内再嵌套函数
  - 内部函数可以引用外层的参数和变量
  - 参数和变量不会被垃圾回收机制回收

说说你对闭包的理解

- 使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念
- 闭包 的最大用处有两个，一个是读取函数内部的变量，另一个就是让这些变量始终保持在内存中
- 闭包的另一个用处，是封装对象的私有属性和私有方法
- 好处：能够实现封装和缓存等；
- 坏处：就是消耗内存、不正当使用会造成内存溢出的问题

使用闭包的注意点



- 由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露
- 解决方法是，在退出函数之前，将不使用的局部变量全部删除

## 2 说说你对作用域链的理解

- 作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到 `window` 对象即被终止，作用域链向下访问变量是不被允许的
- 简单的说，作用域就是变量与函数的可访问范围，即作用域控制着变量与函数的可见性和生命周期

## 3 JavaScript原型，原型链？有什么特点？

- 每个对象都会在其内部初始化一个属性，就是 `prototype` (原型)，当我们访问一个对象的属性时
- 如果这个对象内部不存在这个属性，那么他就会去 `prototype` 里找这个属性，这个 `prototype` 又会有自己的 `prototype`，于是就这样一直找下去，也就是我们平时所说的原型链的概念
- 关系：`instance.constructor.prototype = instance.__proto__`
- 特点：
  - `JavaScript` 对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变
- 当我们需要一个属性的时，`Javascript` 引擎会先看当前对象中是否有这个属性，如果没有的
- 就会查找他的 `Prototype` 对象是否有这个属性，如此递推下去，一直检索到 `Object` 内建对象

## 4 请解释什么是事件代理

- 事件代理 ( `Event Delegation` )，又称之为事件委托。是 `JavaScript` 中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是DOM元素的事件冒泡。使用事件代理的好处是可以提高性能
- 可以大量节省内存占用，减少事件注册，比如在 `table` 上代理所有 `td` 的 `click` 事件就非常棒

- 可以实现当新增子对象时无需再次对其绑定

## 5 Javascript如何实现继承?

- 构造继承
- 原型继承
- 实例继承
- 拷贝继承
- 原型 `prototype` 机制或 `apply` 和 `call` 方法去实现较简单， 建议使用构造函数与原型混合方式

```
function Parent(){  
    this.name = 'wang';  
}
```


```
function Child(){  
    this.age = 28;  
}
```

```
Child.prototype = new Parent();//继承了Parent, 通过原型
```

```
var demo = new Child();  
alert(demo.age);  
alert(demo.name); //得到被继承的属性
```

js

## 6 谈谈This对象的理解

- `this` 总是指向函数的直接调用者（而非间接调用者）
- 如果有 `new` 关键字， `this` 指向 `new` 出来的那个对象
- 在事件中， `this` 指向触发这个事件的对象， 特殊的是，  中的 `attachEvent` 中的 `this` 总是指向全局对象 `Window`

## 7 事件模型

W3C 中定义事件的发生经历三个阶段：捕获阶段（`capturing`）、目标阶段（`targetin`）、冒泡阶段（`bubbling`）

- 冒泡型事件：当你使用事件冒泡时，子级元素先触发，父级元素后触发
- 捕获型事件：当你使用事件捕获时，父级元素先触发，子级元素后触发
- DOM 事件流：同时支持两种事件模型：捕获型事件和冒泡型事件
- 阻止冒泡：在 W3c 中，使用 `stopPropagation()` 方法；在 IE 下设置 `cancelBubble = true`
- 阻止捕获：阻止事件的默认行为，例如 `click` - `<a>` 后的跳转。在 W3c 中，使用 `preventDefault()` 方法，在 IE 下设置 `window.event.returnValue = false`

## 8 new操作符具体干了什么呢？

- 创建一个空对象，并且 `this` 变量引用该对象，同时还继承了该函数的原型
- 属性和方法被加入到 `this` 引用的对象中
- 新创建的对象由 `this` 所引用，并且最后隐式的返回 `this`

## 9 Ajax原理

- Ajax 的原理简单来说是在用户和服务器之间加了一个中间层( AJAX 引擎)，通过 `XmlHttpRequest` 对象来向服务器发异步请求，从服务器获得数据，然后用 `javascript` 来操作 DOM 而更新页面。使用户操作与服务器响应异步化。这其中最关键的一步就是从服务器获得请求数据
- Ajax 的过程只涉及 `JavaScript`、`XMLHttpRequest` 和 `DOM`。 `XMLHttpRequest` 是 `ajax` 的核心机制

```

/** 1. 创建连接 */
var xhr = null;
xhr = new XMLHttpRequest()
/** 2. 连接服务器 */
xhr.open( 'get', url, true)
/** 3. 发送请求 */
xhr.send(null);
/** 4. 接受请求 */
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4){
        if(xhr.status == 200){
            success(xhr.responseText);
        } else {
            /** false */
            fail && fail(xhr.status);
        }
    }
}

```

js

## ajax 有那些优缺点?

- 优点：
  - 通过异步模式，提升了用户体验.
  - 优化了浏览器和服务端之间的传输，减少不必要的往返，减少了带宽占用.
  - Ajax 在客户端运行，承担了一部分本来由服务器承担的工作，减少了大用户量下的服务器负载。
  - Ajax 可以实现动态不刷新（局部刷新）
- 缺点：
  - 安全问题 AJAX 暴露了与服务器交互的细节。
  - 对搜索引擎的支持比较弱。
  - 不容易调试。

## 10 如何解决跨域问题?

首先了解下浏览器的同源策略 同源策略 /SOP (Same origin policy) 是一种约定，由Netscape公司1995年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到 XSS 、 CSFR 等攻击。所谓同源是指"协议+域名+端口"三者相同，即便两个不同的域名指向同一个ip地址，也非同源

那么怎样解决跨域问题的呢?

- 通过jsonp跨域

```
var script = document.createElement( 'script' );
script.type = 'text/javascript';

// 传参并指定回调执行函数为onBack
script.src = 'http://www.....:8080/login?user=admin&callback=onBack';
document.head.appendChild(script);

// 回调执行函数
function onBack(res) {
    alert(JSON.stringify(res));
}
```

js

- document.domain + iframe跨域

此方案仅限主域相同，子域不同的跨域应用场景

### 1) 父窗口：(http://www.domain.com/a.html)

```
<iframe id="iframe" src="http://child.domain.com/b.html"></iframe>
<script>
    document.domain = 'domain.com';
    var user = 'admin';
</script>
```

html

### 2) 子窗口：(http://child.domain.com/b.html)

```
document.domain = 'domain.com';
// 获取父窗口中变量
alert( 'get js data from parent ---> ' + window.parent.user);
```

js

- nginx代理跨域
- nodejs中间件代理跨域
- 后端在头部信息里面设置安全域名

## 11 模块化开发怎么做？

- 立即执行函数,不暴露私有成员

```
var module1 = (function(){
    var _count = 0;
    var m1 = function(){
        //...
    };
    var m2 = function(){
        //...
    };
    return {
        m1 : m1,
        m2 : m2
    };
})();
```

js

## 12 异步加载JS的方式有哪些？

- `defer`，只支持 `IE`
- `async`：
- 创建 `script`，插入到 `DOM` 中，加载完毕后 `callBack`

## 13 那些操作会造成内存泄漏？

- 内存泄漏指任何对象在您不再拥有或需要它之后仍然存在
- `setTimeout` 的第一个参数使用字符串而非函数的话，会引发内存泄漏
- 闭包使用不当

## 14 XML和JSON的区别？

- 数据体积方面
  - `JSON` 相对于 `XML` 来讲，数据的体积小，传递的速度更快些。
- 数据交互方面
  - `JSON` 与 `JavaScript` 的交互更加方便，更容易解析处理，更好的数据交互
- 数据描述方面
  - `JSON` 对数据的描述性比 `XML` 较差
- 传输速度方面
  - `JSON` 的速度要远远快于 `XML`

## 15 谈谈你对webpack的看法

- `WebPack` 是一个模块打包工具，你可以使用 `WebPack` 管理你的模块依赖，并编译输出模块们所需的静态文件。它能够很好地管理、打包 `Web` 开发中所用到的 `HTML`、`Javascript`、`CSS` 以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源，`webpack` 有对应的模块加载器。`webpack` 模块打包器会分析模块间的依赖关系，最后生成了优化且合并后的静态资源

## 16 说说你对AMD和Commonjs的理解

- `CommonJS` 是服务器端模块的规范，`Node.js` 采用了这个规范。`CommonJS` 规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。`AMD` 规范则是非同步加载模块，允许指定回调函数

- **AMD** 推荐的风格通过返回一个对象做为模块对象， **CommonJS** 的风格通过对 `module.exports` 或 `exports` 的属性赋值来达到暴露模块对象的目的

## 17 常见web安全及防护原理

- **sql** 注入原理
  - 就是通过把 **SQL** 命令插入到 **Web** 表单递交或输入域名或页面请求的查询字符串， 最终达到欺骗服务器执行恶意的SQL命令
- 总的来说有以下几点
  - 永远不要信任用户的输入， 要对用户的输入进行校验， 可以通过正则表达式， 或限制长度， 对单引号和双 "-" 进行转换等
  - 永远不要使用动态拼装SQL， 可以使用参数化的 **SQL** 或者直接使用存储过程进行数据查询存取
  - 永远不要使用管理员权限的数据库连接， 为每个应用使用单独的权限有限的数据库连接
  - 不要把机密信息明文存放， 请加密或者 **hash** 掉密码和敏感的信息

### XSS原理及防范

- **Xss(cross-site scripting)** 攻击指的是攻击者往 **Web** 页面里插入恶意 **html** 标签或者 **javascript** 代码。比如：攻击者在论坛中放一个看似安全的链接，骗取用户点击后，窃取 **cookie** 中的用户私密信息；或者攻击者在论坛中加一个恶意表单， 当用户提交表单的时候，却把信息传送到攻击者的服务器中， 而不是用户原本以为的信任站点

### XSS防范方法

- 首先代码里对用户输入的地方和变量都需要仔细检查长度和对 "<", ">", ";", "'", " " 等字符做过滤；其次任何内容写到页面之前都必须加以encode， 避免不小心把 **html tag** 弄出来。这一个层面做好， 至少可以堵住超过一半的XSS 攻击

### XSS与CSRF有什么区别吗？

- **XSS** 是获取信息，不需要提前知道其他用户页面的代码和数据包。 **CSRF** 是代替用户完成指定的动作， 需要知道其他用户页面的代码和数据包。要完成一次 **CSRF** 攻击， 受害者必须依次完成两个步骤
- 登录受信任网站 **A**， 并在本地生成 **Cookie**
- 在不登出 **A** 的情况下， 访问危险网站 **B**

### CSRF的防御

- 服务端的 **CSRF** 方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数
- 通过验证码的方法

## 18 用过哪些设计模式？

- 工厂模式：
  - 工厂模式解决了重复实例化的问题，但还有一个问题，那就是识别问题，因为根本无法
  - 主要好处就是可以消除对象间的耦合，通过使用工程方法而不是 **new** 关键字
- 构造函数模式
  - 使用构造函数的方法，即解决了重复实例化的问题，又解决了对象识别的问题，该模式与工厂模式的不同之处在于
  - 直接将属性和方法赋值给 **this** 对象；

## 19 为什么要有同源限制？

- 同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议
- 举例说明：比如一个黑客程序，他利用 **Iframe** 把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过 **Javascript** 读取到你的表单中 **input** 中的内容，这样用户名，密码就轻松到手了。

## 20 offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别

- **offsetWidth/offsetHeight** 返回值包含content + padding + border，效果与 `e.getBoundingClientRect()`相同
- **clientWidth/clientHeight** 返回值只包含content + padding，如果有滚动条，也不包含滚动条
- **scrollWidth/scrollHeight** 返回值包含content + padding + 溢出内容的尺寸

## 21 javascript有哪些方法定义对象

- 对象字面量： `var obj = {};`
- 构造函数： `var obj = new Object();`
- `Object.create()`: `var obj = Object.create(Object.prototype);`



## 22 常见兼容性问题？

- `png24` 位的图片在IE6浏览器上出现背景，解决方案是做成 `PNG8`
- 浏览器默认的 `margin` 和 `padding` 不同。解决方案是加一个全局的 `*`  
`{margin:0;padding:0;}` 来统一，但是全局效率很低，一般是如下这样解决：

```
body,ul,li,ol,dl,dt,dd,form,input,h1,h2,h3,h4,h5,h6,p{  
margin:0;  
padding:0;  
}
```

CSS

- IE 下, `event` 对象有 `x` , `y` 属性,但是没有 `pageX` , `pageY` 属性
- Firefox 下, `event` 对象有 `pageX` , `pageY` 属性,但是没有 `x,y` 属性.

## 23 说说你对promise的了解

- 依照 `Promise/A+` 的定义, `Promise` 有四种状态：
  - `pending`: 初始状态, 非 `fulfilled` 或 `rejected`.
  - `fulfilled`: 成功的操作.
  - `rejected`: 失败的操作.
  - `settled`: `Promise` 已被 `fulfilled` 或 `rejected` , 且不是 `pending`
- 另外, `fulfilled` 与 `rejected` 一起合称 `settled`
- `Promise` 对象用来进行延迟( `deferred` ) 和异步( `asynchronous` ) 计算

### Promise 的构造函数

- 构造一个 `Promise` , 最基本的用法如下：

```
var promise = new Promise(function(resolve, reject) {  
  
    if (...) { // succeed  
  
        resolve(result);  
    }  
}
```

js

```
    } else {    // fails

        reject(Error(errMessage));

    }
});
```

- `Promise` 实例拥有 `then` 方法（具有 `then` 方法的对象，通常被称为 `thenable`）。它的使用方法如下：

```
promise.then(onFulfilled, onRejected)
```

- 接收两个函数作为参数，一个在 `fulfilled` 的时候被调用，一个在 `rejected` 的时候被调用，接收参数就是 `future`，`onFulfilled` 对应 `resolve`，`onRejected` 对应 `reject`

## 24 你觉得jQuery源码有哪些写的好的地方

- `jquery` 源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入 `window` 对象参数，可以使 `window` 对象作为局部变量使用，好处是当 `jquery` 中访问 `window` 对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问 `window` 对象。同样，传入 `undefined` 参数，可以缩短查找 `undefined` 时的作用域链
- `jquery` 将一些原型属性和方法封装在了 `jquery.prototype` 中，为了缩短名称，又赋值给了 `jquery.fn`，这是很形象的写法
- 有一些数组或对象的方法经常能使用到，`jQuery` 将其保存为局部变量以提高访问速度
- `jquery` 实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率

## 25 vue、react、angular

- `Vue.js` 一个用于创建 `web` 交互界面的库，是一个精简的 `MVVM`。它通过双向数据绑定把 `View` 层和 `Model` 层连接了起来。实际的 `DOM` 封装和输出格式都被抽象为了 `Directives` 和 `Filters`
- `AngularJS` 是一个比较完善的前端 `MVVM` 框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，自带了丰富的 `Angular` 指令
- `react` `React` 仅仅是 `VIEW` 层是 `facebook` 公司。推出的一个用于构建 `UI` 的一个库，能够实现服务器端的渲染。用了 `virtual dom`，所以性能很好。