前端开发

模块化

- CommonJS
  - `CommonJS` 有一个全局性方法 `require()` ,用于加载模块
  - `nodeJS` 模块化规范，现在被前端大量使用
  - 前端开发依赖的插件和库，都可以从 `npm` 中获取
  - 构建工具的高度自动化，使得使用 `npm` 的成本非常低
  - `CommonJS` 不会异步加载JS，而是同步一次性加载出来
- AMD
  - `AMD` 也采用 `require()` 语句加载模块，语法不同于 `CommonJS`

```
// 第一个参数[module]，是一个数组，里面的成员就是要加载的模块；第二个参数callback，则是加载成功之后的回调函数
require([module], callback);
```

  - 库：`require.js`

```
// math.js
define(function() {
  var add = function(x, y) {
    return x + y;
  };
  return {
    add: add
  };
});
```

```
// main.js
require(['math'], function(math) {
  alert(math.add(1, 1));
});
```

- `AMD` 和 `CommonJS` 的使用场景
  - 需要异步加载 `JS` ，使用 `AMD` ， `AMD` 比较适合浏览器环境
  - 使用了 `npm` 之后建议使用 `CommonJS`

构建工具

- gulp
- webpack

运行环境

- 从输入url到得到html的详细过程
  - 加载资源的过程
    - 在浏览器输入url
    - 浏览器查找DNS过程 浏览器缓存→系统缓存(hosts文件)→路由器缓存,如果找到对应解析，就会在浏览

器显示内容，如果没找到，就会进行下一步
    - 浏览器根据DNS服务器进行DNS解析得到域名的IP地址
    - 向这个IP的机器发送 `http/https` 请求，建立tcp连接
    - 服务器收到、处理并返回请求
    - 浏览器得到返回内容
  - 浏览器渲染页面的过程
    - 根据 `HTML` 结构生成 `DOM Tree`
    - 根据 `CSS` 生成 `CSSOM`
    - 将 `DOM` 和 `CSSOM` 整合形成 `RenderTree`
    - 根据 `RenderTree` 开始渲染和展示
    - 遇到 `<script>` 时,会执行并阻塞渲染
- `window.onload` 和 `DOMContentLoaded` 的区别

```
window.addEventListener('load',function(){
  // 页面的全部资源加载完才会执行，包括图片、视频等
})
document.addEventListener('DOMContentLoaded',function(){
  //  DOM渲染完即可执行，此时图片、视频还可能没有加载完
})
```

- 性能优化(综合性问题)
  - 原则
    - 多使用内存、缓存或者其他方法
    - 减少CPU计算、减少网络请求、减少IO操作
  - 入手
    - 加载页面和静态资源
    - 页面渲染
  - 加载资源优化
    - 静态资源的压缩合并(减少体积，合并请求)
      - 使用构建工具压缩js、css
    - 静态资源缓存
    - 使用CDN让资源加载更快（自己的小网站使用了 BootCDN,大公司可以自己搭建CDN)
    - 使用SSR后端渲染(服务端渲染)，数据直接输出到HTML中
  - 渲染优化
    - CSS放header,JS放在body尾部
    - 懒加载（图片懒加载，下拉加载更多）
    - 减少 DOM查询，对DOM查询做缓存

```
// 未缓存DOM查询
var i;
for (i = 0; i < document.getElementsBytagName('p').length; i++) {
  // todo
}
// 缓存DOM查询
var pList = document.getElementsBytagName('p');
var plen = pList.length;
```

```
var i;
for (i = 0; i < plen; i++) {
  // todo
}
```

- 减少DOM操作，多个操作尽量合并在一起执行

```
var listNode = document.getElementById('list');
// 插入10个li标签
var frag = document.createDocumentFragment();
var x, li;
for (x = 0; x < 10; x++) {
  li = document.creatElement('li');
  li.innerHTML = 'List item' + x;
  frag.appendChild(li);
}
listNode.appendChild(frag);
```

- 事件节流控制函数被触发的频率（`setTimeout`，`setInterval`，`clearTimeout`）
- 尽早执行操作（`DOMContentLoaded`）

- 安全性
  - XSS 跨站请求攻击
  - XSRF 跨站请求伪造