其他

1 负载均衡

> 多台服务器共同协作，不让其中某一台或几台超额工作，发挥服务器的最大作用

- `http`重定向负载均衡：调度者根据策略选择服务器以302响应请求，缺点只有第一次有效果，后续操作维持在该服务器 dns负载均衡：解析域名时，访问多个`ip`服务器中的一个（可监控性较弱）
- 反向代理负载均衡：访问统一的服务器，由服务器进行调度访问实际的某个服务器，对统一的服务器要求大，性能受到 服务器群的数量

2 CDN

> 内容分发网络，基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。

3 内存泄漏

> 定义：程序中己动态分配的堆内存由于某种原因程序未释放或无法释放引发的各种问题。

js中可能出现的内存泄漏情况

> 结果：变慢，崩溃，延迟大等，原因：

- 全局变量
- `dom`清空时，还存在引用
- `ie`中使用闭包
- 定时器未清除
- 子元素存在引起的内存泄露

避免策略

- 减少不必要的全局变量，或者生命周期较长的对象，及时对无用的数据进行垃圾回收；
- 注意程序逻辑，避免“死循环”之类的；
- 避免创建过多的对象 原则：不用了的东西要及时归还。
- 减少层级过多的引用

4 babel原理

> ES6、7代码输入 -> babylon进行解析 -> 得到AST（抽象语法树） -> plugin用babel-traverse对AST树进行遍历转译 ->得到新的AST树->用babel-generator通过AST树生成ES5代码

5 js自定义事件

> 三要素： document.createEvent() event.initEvent() element.dispatchEvent()

```
// (en:自定义事件名称，fn:事件处理函数，addEvent:为DOM元素添加自定义事件，triggerEvent:触发自定义事件)
window.onload = function(){
    var demo = document.getElementById("demo");
    demo.addEvent("test",function(){console.log("handler1")});
    demo.addEvent("test",function(){console.log("handler2")});
    demo.onclick = function(){
        this.triggerEvent("test");
    }
}
Element.prototype.addEvent = function(en,fn){
    this.pools = this.pools || {};
    if(en in this.pools){
        this.pools[en].push(fn);
    }else{
        this.pools[en] = [];
        this.pools[en].push(fn);
    }
}
Element.prototype.triggerEvent  = function(en){
    if(en in this.pools){
        var fns = this.pools[en];
        for(var i=0,il=fns.length;i<il;i++){
            fns[i]();
        }
    }else{
        return;
    }
}
```

6 前后端路由差别

- 后端每次路由请求都是重新访问服务器
- 前端路由实际上只是`JS`根据`URL`来操作`DOM`元素，根据每个页面需要的去服务端请求数据，返回数据后和模板进行组合