前端：面试题解析及答案
一 到 三：基础知识与浏览器相关题解析
四 到 六：Web 安全与性能优化题解析
七 到 九：CSS 布局、响应式与现代特性题解析
十 到 十二：现代工具链与模块化题解析
十三 到 十五：浏览器渲染与性能优化题解析
十六 到 十八：安全防御与现代安全实践题解析
十九 到 二十一：现代浏览器特性与实验性功能题解析
二十二 到 二十四：现代工具链与生态题解析
二十五 到 二十七：现代前端工程化与自动化题解析
二十八 到 二十九：现代前端测试与质量保证题解析
三十 到 三十二：现代前端性能优化与监控题解析
三十三 到 三十五：现代前端安全与隐私保护题解析

一 到 三：浏览器相关知识点与高频考题解析

Web 前端工程师的日常需要面对浏览器渲染、网络请求和安全防御等多维度的复杂场景。这些基础知识是构建稳定应用的基石。

知识点梳理

- 浏览器核心架构及渲染流程
- 性能优化
- 网络请求与缓存
- 安全防御

本小节将从浏览器的加载过程和页面渲染，到网络请求的缓存策略，最后到安全防御机制，全面剖析这些高频考题的解析思路。

加载页面和渲染过程

可将加载过程理解为浏览器向服务器发出请求，获取资源的过程，并解析这些资源，把它们渲染成屏幕上可见的内容。

题目： 浏览器从输入网页到完成渲染的流程是？

加载过程

- 浏览器解析 URL 获取协议、主机名、端口、路径和查询参数。
- 向域名服务器发送 DNS 解析请求，获取服务器的 IP 地址。
- 建立与服务器的 TCP 连接。
- 浏览器发送 HTTP 请求。
- 服务器处理请求并返回 HTTP 响应，包含状态码、响应头和响应体。

渲染过程

- 解析 HTML 文件，构建 DOM 树。
- 根据 CSS 规则，构建 CSSOM 树。
- 将 DOM 和 CSSOM 合并成渲染树（Render Tree）。
- 计算每个节点的精确位置和大小（布局/回流）。
- 将节点内容绘制到屏幕上（绘制）。

浏览器渲染、网络请求与缓存

- 解析 HTML 文件，构建 DOM 树。
- 解析 CSS 文件，构建 CSSOM 树。
- 将 DOM 和 CSSOM 合并成渲染树（Render Tree）。
- 计算每个可见元素的精确位置和大小（布局/回流）。
- 将元素内容绘制到屏幕上（绘制）。

性能优化

- 减少 HTTP 请求数量（合并文件、使用雪碧图）。
- 使用浏览器缓存（设置 HTTP 缓存头）。
- 压缩 CSS 和 JS 文件。
- 优化图片格式（使用 WebP 等）。
- 使用 CDN 以提升内容分发。

网络请求与缓存

- CDN 内容分发网络。
- 浏览器缓存（强缓存、协商缓存）。
- 服务器缓存（如 Redis）。
- 数据库缓存。

安全防御

- 防止 XSS（跨站脚本）攻击。
- 防止 CSRF（跨站请求伪造）攻击。
- 使用 HTTPS 加密。
- 内容安全策略（Content-Security-Policy）。

浏览器渲染

静态资源部分

如果不动作，每个请求会一直以前不动的状态发送

```
http/get urls.js?v=hash1
http/get urls.js?v=hash2
http/get urls.js?v=hash3
```

如果改动了，就只会一直请求文件夹

```
http/get urls.js?v=hash1
```

静态资源部分

通过缓存请求的时刻保存

```
http/get urls.js?v=hash1
```

只有内容改变的时候，缓存请求时才会改变

```
http/get urls.js?v=hash2
```

这个请求不用动的，可通过缓存请求文件夹的最大时间，为文件名即添加 MD5 哈希。

使用 CDN 分流浏览器资源

CDN 分散所有动态请求的文件夹，静态资源请求通过 CDN 上。例如：

```
http/get cdn/https/cdn.bootcdn.net/ajax/libs/ajax.js?v=hash1
```

使用 SSR 后端渲染

用一个模板 HTML 文件，不再动态渲染数据内容之后，再通过 Ajax 加载数据，再通过 JS，例如使用 smarty，Jsp 等 API。

CSS 渲染部分，内容区

上文内容的渲染请参考以前的时间线图，不再赘述。

网络请求

一个网络时间 Ajax 就是一个通知时间线图，下面的时间点的最后之时间线，如下，

```
http/get.php 数据请求，动态的，加数据时，用动态请求数据的时间线 http/get.php，加数据 API 时，用静态请求，数据请求数据的时间线 Ajax 的 data-hash 的值。
```

```
http/get/article/id/ data-hash=hash.php?
```

同时，动态加载的数据 Ajax 请求数据请求 ——— 同样 HTML 中的动态加载数，都是请求 Ajax 的 Ajax，同时 Ajax 的数据请求的时间线图，都用请求时间。

DOM 渲染部分

两段代码第一天可以：

```
var arr = document getElementsTagName('p') // 只要每一个 DOM 渲染时 arr 变了
var a = 1
var b = 0; b < arr length; b++ {
}
```

```
var a = 1
var b = 0; b < document getElementsTagName('p') length; b++ { // 每次循环，都会请求 DOM 渲染时
}
```

但是，DOM 渲染，无论最后是否修改，都要在每次循环时修改，造成重大。

点击 DOM 插入

DOM 渲染是要每次循环修改时，因此第八多个时间线，当插入 Fragment 时后再统一插入 DOM。

```
var listNode = document getElementById('list')
// 假设有 1000 个 li 元素
var frag = document createDocumentFragment();
var i = 1;
for (; i < 1_000; i++) {
  var li = document createElement("li");
  li innerHtml = 'Item ' + i;
  frag appendChild(li);
}
listNode appendChild(frag);
```

操作方法

例如要在文字改变时触发一个 change 事件，通过 keyup 事监听，然后代码。

```
var textarea = document getElementById('text')
var timeout
textarea addEventListener('keyup', function () {
  clearTimeout(timeout)
  // 清除之前的计时器，防止上一次输入还在等待执行
  timeout = setTimeout(function () {
  // 设置一个新的计时器，500ms 后执行
  console log('检测到输入，500ms 内没有新输入，现在执行');
  }, 500);
});
```

以下为正确作

```
window addEventListener('load', function () {
  // 页面加载完成后，执行初始化操作
  console log('页面加载完成，开始初始化');
});
document addEventListener('DOMContentLoaded', function () {
  // DOM 结构解析完成，可以安全地操作 DOM 了
});
```

性能优化方法

上面提到的渲染性能优化的操作方式，性能优化还涉及网络实现缓存，还涉及到下面步骤做法：

1. 通过浏览器控制台查看，性能面板，分析渲染时间。
2. 使用 Lighthouse 工具，生成性能报告，获取优化建议。
3. 合并和压缩文件，减少网络请求数。
4. 优化图片格式，使用 WebP 等现代格式。
5. 实现缓存策略，利用浏览器缓存和 CDN。

性能优化是个长期过程，不是一蹴而就，需要不断测试、分析和迭代。

Web 安全

题目： 前端开发时应该注意哪些安全？

Web 前端安全问题，都是前端开发时的不小心，这个是前端最容易忽略了，不知之前，先来一个最简单的注入方式 ——— SQL 注入。

上面的时间线图第一个（SQL注入）的动态方式，例如第一个表的数据请求图，输入用户名时的时，图示之后，动态数据请求数据的时间线 SQL 语句会直接拼接，比如说 SELECT * FROM users WHERE name = '用户输入的内容'，如果用户输入的内容是 ' OR '1'='1，那不就完蛋了，这个数据请求图形时间线图，就会变成 SELECT * FROM users WHERE name = '' OR '1'='1'，这个时间线图时间线图，就会变成 SELECT * FROM users，这样数据请求图时间线图，就会变成数据请求图时间线图，就会变成数据请求图时间线图，就会变成