前端体系

- 渲染
  - jpg、png、jpeg、webp、apng图片
    - 编码原理
    - 特点和优势
    - 适用场景
  - iconfont/svg
    - 矢量图原理
    - 适用场景
  - opengl
    - oepngl api
  - canvas
  - 图片、文字渲染过程
- web端
  - html/jt/css/sass/less常用语法和使用
  - em,rem原理与实现
  - 脚手架
    - 打包构建工具
      - npm、jspm、bower包管理工具
    - 浏览器适配
    - node
    - nginx
    - 开源框架
      - vue
      - angular
      - react
  - 跨域
    - jsonp
      - 动态创建<script>标签，前后端约定回调函数名称
    - cors跨域资源共享
      - 普通跨域请求：只服务端设置Access-Control-Allow-Origin即可，前端无须设置，若要带cookie请求：前后端都需要设置
    - nginx代理跨域
  - 存储
    - cookie
      - 如果不在浏览器中设置过期时间，cookie被保存在内存中，生命周期随浏览器的关闭而结束，这种cookie简称会话cookie。如果在浏览器中设置了cookie的过期时间，cookie被保存在硬盘中，关闭浏览器后，cookie数据仍然存在，直到过期时间结束才消失。
      - Cookie是服务器发给客户端的特殊信息，cookie是以文本的方式保存在客户端，每次请求时都带上它
      - 单个cookie保存的数量不能超过4kb
      - cookie只能作用于当前页面
    - sessionStorage
      - 大小5m
      - 关闭浏览器或当前页面删除
      - 只能作用于当前页面
    - localStorage 本地存储
      - 本地存储，除非手动删除，否则永久
      - 同一浏览器可以数据共享
      - 大小20m
  - 安全
    - XSS-跨脚本攻击
      - 对重要的Cookie字段使用HTTPOnly参数：禁止js获取cookie
      - 1). 将cookie等敏感信息设置为httponly，禁止javascript通过document.cookie获得
      - 2). 对所有的输入做严格的校验尤其是在服务器端，过滤掉任何不合法的输入，比如手机号必须是数字，通常可以采用正则表达式
      - 3. 净化和过滤掉不必要的html标签，比如：<iframe>, alt,<script> 等
      - 4. 净化和过滤掉不必要的javascript的事件标签，比如：onclick, onfocus等
      - 5. 转义单引号、双引号、尖括号等特殊字符，可以采用htmlencode编码 或者过滤掉这些特殊字符
      - 6. 设置浏览器的安全设置来防范典型的XSS注入
    - CSRF，也就是跨站请求伪造
      - 防范方法：
        - 1). 在客户端进行cookie的hashing，并在服务端进行hash认证
        - 2). 提交请求是需要填写验证码
        - 3. 使用One-Time Tokens为不同的表单创建不同的伪随机值
    - SQL注入
      - 攻击方法：
        - 编写恶意字符串，比如' or 1=1--等。
        - 手动查试目标网站上所有涉及数据库操作的地方
      - 1). 禁止目标网站利用动态拼接字符串的方式访问数据库
      - 2). 减少不必要的数据库露出的错误信息
      - 3. 对数据库的操作赋予严格的权限控制
      - 4. 净化和过滤掉不必要的SQL保留字，比如：where, or, exec 等
      - 5). 转义单引号、上引号、尖括号等特殊字符，可以采用htmlencode编码 或者过滤掉这些特殊字符
  - 浏览器渲染原理
    - 解析html构建dom树
      - 浏览器将HTML解析成树形的数据结构
    - 解析css生成cssom树
      - 浏览器将CSS解析成树形的数据结构
    - 根据dom树和cssom树生成render树
      - DOM和CSSOM合并后生成Render Tree
    - 布局render树
      - 有了Render Tree，浏览器已经能知道网页中有哪些节点，各个节点的CSS定义以及他们的从属关系，从而去计算出每个节点在屏幕中的位置。
    - 绘制render树上的节点
      - 按照算出来的规则，通过显卡，把内容画到屏幕上。
    - 当浏览器发现某个部分发生了点变化影响了布局，需要倒回去重新渲染，内行称这个回退的过程叫 reflow。reflow 会从 <html> 这个 root frame 开始进行往下，依次计算所有的结点几何尺寸和位置。reflow 几乎是无法避免的。现在界面上流行的一些效果，比如树状目录的折叠、展开（实质上是元素的显示与隐藏）等，都将引起浏览器的 reflow。鼠标滑过、点击……只要这些行为引起了页面上某些元素的占位面积、定位方式、边距等属性的变化，都会引起它内部，周围甚至整个页面的重新渲染。通常我们都无法预估浏览器到底会 reflow 哪一部分的代码，它们都彼此相互影响着。
    - reflow回流
    - repaint重绘
      - 改变某个元素的背景色、文字颜色、边框颜色等等不影响它周围或内部布局的属性时，屏幕的一部分要重画，但是元素的几何尺寸没有变
- app端
- 小程序端
- 开发技巧和调试
  - vscode
  - 常用代码片段
  - 代理
  - eslint代码检查
  - 代码格式化
  - 自动化测试
  - postmain网络调试
  - rap2在线接口文档
- 前端数据监控
  - 用户数据统计
    - 友盟/百度/腾讯
  - 前端性能分析与上报
    - performance timing测速上报
    - profile分析
    - 打点性能上报
    - 页面资源加载timeline分析
  - 异常数据上报
    - 捕获异常try-catch、onerror
    - onerror可以捕获语法错误与运行时错误，可以拿到出错信息、堆栈、出错文件行号、列号，当前页面执行的脚本出错都可以捕捉到。如果报错文件与html 不同源直接上报脚本错误需要在script crossorigin属性和服务器跨域权限
    - try-catch:无法捕获语法错误，只能捕获运行时错误，可以拿到出错的信息，堆栈，出错文件，行号，列号。
    - 原理：根据onload特定文件成功加载，对失败的结果进行上报统计
  - js加载失败优化方案
    - 失败重发机制
    - https反劫持
- 前端性能优化
  - 逻辑性能优化
  - 网络优化
  - 渲染优化
  - 减少请求资源和大小优化
    - 压缩和合并js和css文件，使用webpack等打包工具
    - 小于10kb的资源使用base64编码，减少资源请求
    - 把相对比较小的资源做成雪碧图或精灵图
    - 前后端协商请求资源使用压缩资源
    - 避免使用iframe
      - 不仅不好管控样式，而且相当于在本页面又嵌套其他页面，消耗性能会更大，因为还得去加载这个嵌套页面的资源
    - 减少dom操
      - 主要是减少DOM的重绘与回流
      - 数据异步加载，图片懒加载
- webpack
  - 入口
    - 入口起点(entry point)指示 webpack 应该使用哪个模块，来作为构建其内部依赖图的开始。进入入口起点后，webpack 会找出有哪些模块和库是入口起点（直接和间接）依赖的。
  - 出口
    - output 属性告诉 webpack 在哪里输出它所创建的 bundles，以及如何命名这些文件。默认值为 ./dist。基本上，整个应用程序结构，都会被编译到你指定的输出路径的文件夹中。你可以通过在配置中指定一个 output 字段，来配置这些处理过程：
  - 配置
    - loader
      - loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只理解 JavaScript）。loader 可以将所有类型的文件转换为 webpack 能够处理的有效模块，然后你就可以利用 webpack 的打包能力，对它们进行处理。
    - 插件
      - loader 被用于转换某些类型的模块，而插件则可以用于执行范围更广的任务。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量。插件接口功能极其强大，可以用来处理各种各样的任务。
  - 优化
    - 缓存
      - 大部分loader提供cache配置
      - 如果loader不提供cache，则用cache-loader
    - 多核
      - happypack多核编译
    - 抽离
      - 把不常变更的静态依赖，比如react全家桶
      - externals
      - webpack-dll-plugin
- 前端架构
  - 模块化
  - 工程化
    - 工具
      - 开发工具:vscode
      - 版本工具:git
      - 代码风格editorconfig
      - 代码编译:babel/sass/less等预编译
      - 代码质量eslint
      - 构建工具webpack
      - 脚手架
      - 代码模版
      - vscode插件
      - npm打包
  - 开发规范
    - 目录规范、编码规范、命名规范
    - 模块化方案，开发模版
    - 构建工具，开发工具，组件化设计
    - 开发和部署打通
  - 项目部署
  - 前端统计
    - 数据统计、性能统计、访问统计、用户行为统计、错误统计、安全监控
  - 安全测试
    - 安全意识
  - 自动化测试
    - 自动化真效测试
  - 规范化
    - 目录规范
    - 编码规范
    - 命名规范
  - 部署和构建-提升性能
    - 预编译、es6/7语法转译、css预编译、如babel
    - 依赖打包-分析依赖关系，将同步依赖的文件打包一起，减少http请求
    - 资源嵌入
      - 比如小于10kb的资源编译为base64 格式嵌入文档，减少http请求
    - 文件压缩
      - 减小文件体积
    - hash指纹
      - 通过给文件名增加hash指纹，防止浏览器对缓存引起的静态资源更新问题
    - 代码审查
      - 避免上线文件的低级错误
    - 模板构建
- 网络及网络安全
  - 短连接http、https
    - https原理
  - 长链接socket
    - web socket
    - select io
    - epoll
    - iocp
  - restful协议
    - GET
    - POST
    - PUT
    - DELETE
    - OPTIONS
  - cdn与dns技术
  - fetch api
- 软技能
  - sketch
  - ps
  - xmind
  - 团队管理
  - 知识管理/总结分享
  - 产品思维
  - 软件工程体系思想
- hybird移动开发设计
  - web->native交互
    - scheme协议
      - WebView拦截页面跳转
      - jsbridge://className:callbackAddress#methodName?jsonObj
    - JavascriptInterface
    - WebChromeClient.onJsPrompt
  - native->web交互
    - WebView.loadUrl("javascript:function()")
  - 更新
- 框架及组件
  - mvc/mvp/mvvm框架设计和原理，vue/react/angular 等
  - 分层
    - 模块化
      - 将一个复杂业务实现，根据功能，页面或者其他进行不同粒度的划分程不同的模块，模块之间解耦。分别进行实现，也就是编写的模块化思想。
      - 模块化的特点是：模块之间解耦，可以独立管理。
    - 组件化
    - 插件化
  - commonjs/AMD/CMD
    - 模块引入
    - 模块定义
    - 模块标识
    - 模块懒执行(CMD)与预执行(AMD)
    - UMD解决不同规范兼容性问题
  - load js模块化加载原理与实现
  - virtual Dom
    - 1、生成虚拟dom，用js对象树表示dom树结构，将html字符串解析成js对象，涉及词法解析。
    - 2、对比差异化虚拟dom，状态改变时，重新构建新的虚拟dom树，和旧的dom树对象进行对比，记录二者的差异，对比差异是多叉树遍历算法。
    - 3、渲染最小的差异树，将对比的差异渲染到页面上
    - 小结：js对象模拟dom树，用diff算法对比差异，差异化渲染dom
  - shadow dom
    - 隔离外部环境封装组件：结构、样式、行为
    - 实现形式：新标签,class类属性+构架编译
  - web compontes
  - web worker 与service worker
  - es6 + 开发体系
    - es6+ 特性与编码规范
    - babel构建打包
    - typescript
  - browserify运行原理
  - performance timing 来进行性能优化
  - promise原理与使用
  - vue、angular2
    - mvvm语法
    - 组件化方案
    - 状态管理方案
  - react 如react导图
  - 缓存机制
    - cache-control
      - no-cache
        - 表示必须先与服务器确认返回的响应是否被更改，然后才能使用该响应来满足后续对同一个网址的请求。因此，如果存在合适的验证令牌 (ETag)，no-cache 会发起往返验证缓存的响应，如果资源未被更改，可以避免下载。
      - no-store
        - 禁止缓存任何响应，也就是说每次用户请求资源时，都会向服务器发送一个请求，每次都会下载完整的响应
      - public
        - 如果响应被标记为public，即使有关联的 HTTP 认证，甚至响应状态码无法正常缓存，响应也可以被缓存
      - private
        - 浏览器可以缓存private响应，但是通常只为单个用户缓存，因此，不允许任何代理服务器对其进行缓存 。比如，用户浏览器可以缓存包含用户私人信息的 HTML 网页，但是 CDN 不能缓存。
      - max-age
        - 用来设置资源被缓存的最长时间(单位是秒)
    - etag
      - 验证码，一般文件内容hash，服务器通过 ETag HTTP 头传递验证码，大概是像"a123cef"这样的字符串。当浏览器在资源过期后再次请求时，浏览器就认会通过If-None-Match传递Etag的验证码，通过验证码可以进行高效的资源更新检查：如果资源未更改，则不会传输任何数据。
  - mvvm数据绑定-数据变更检测
    - 手动设置
    - 脏数据检测
    - 前端数据劫持
    - es6代理

XMind: ZEN 试用模式