前端架构设计简介

讲师：张欢

2019年9月

目录

CONTENTS

课程背景

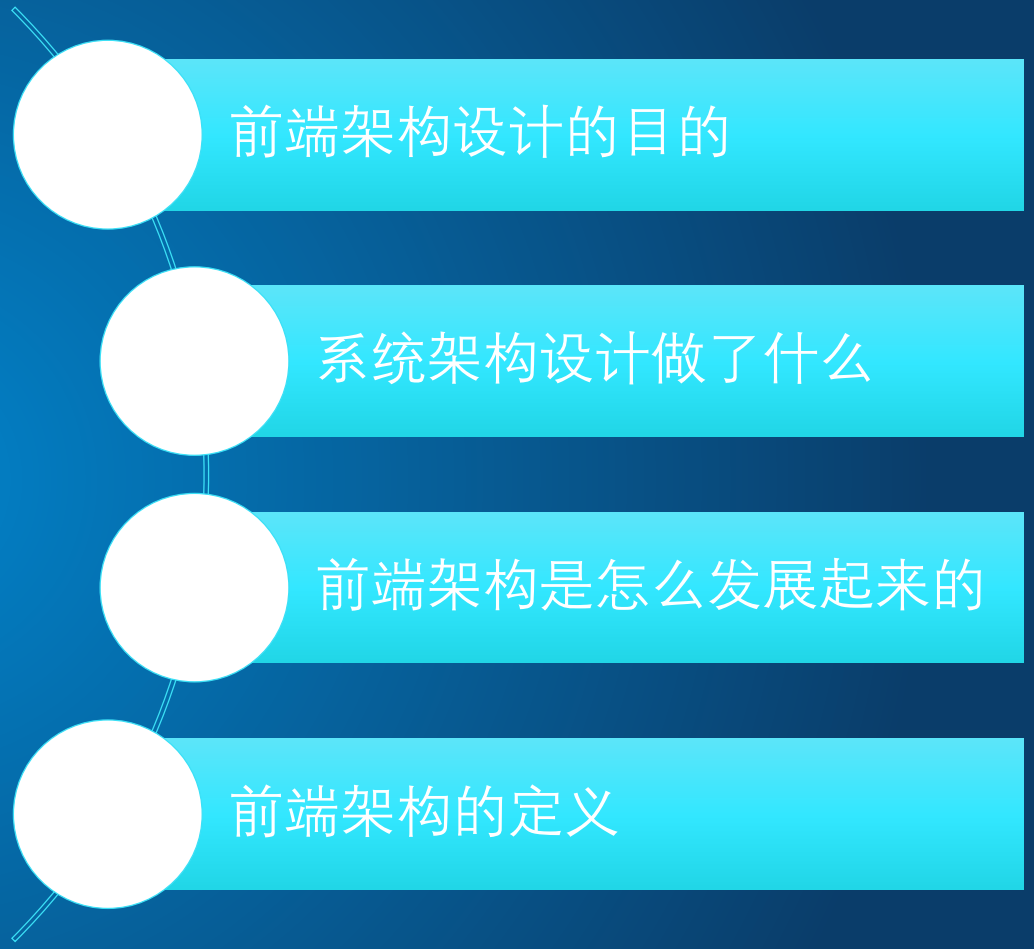
- ToC系统的用户越来越多，业务逻辑也在像前端转移，用户体验和前端交互越来越重要。对前端开发的能力要求越来越高，需要掌握一定的架构能力。
- 业内对于前端架构设计的方法，没有非常统一的说法。在实际工作中，架构师们由于出身不同，设计方法各不相同，产物的形式多种多样，不利于横向交流和后续指导。
- 不同企业对架构师岗位理解不尽相同，架构师的职责范围也因此不尽相同，架构师需要有一定的适应能力。

课程目标

- 理解前端架构设计的作用
- 掌握架构设计的步骤和方法
- 了解架构设计优化的方法

什么是前端架构

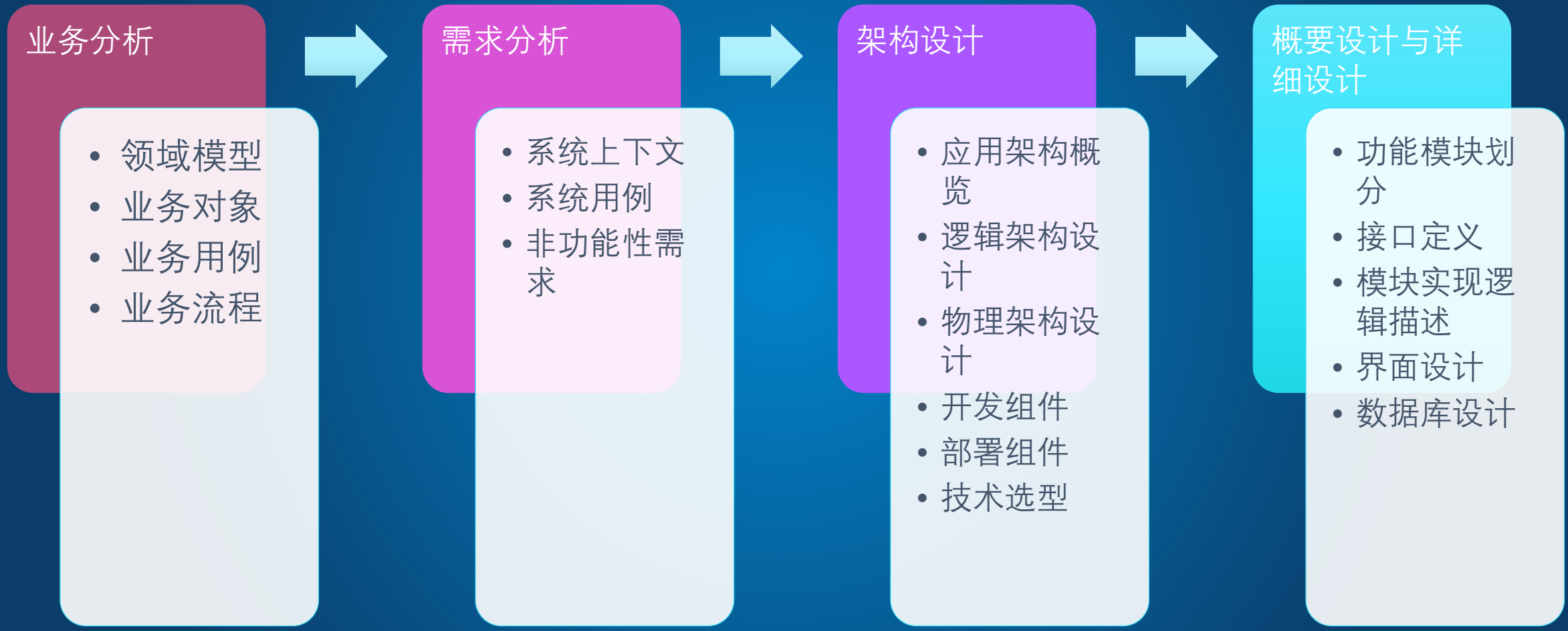
前端架构设计的目的和定义



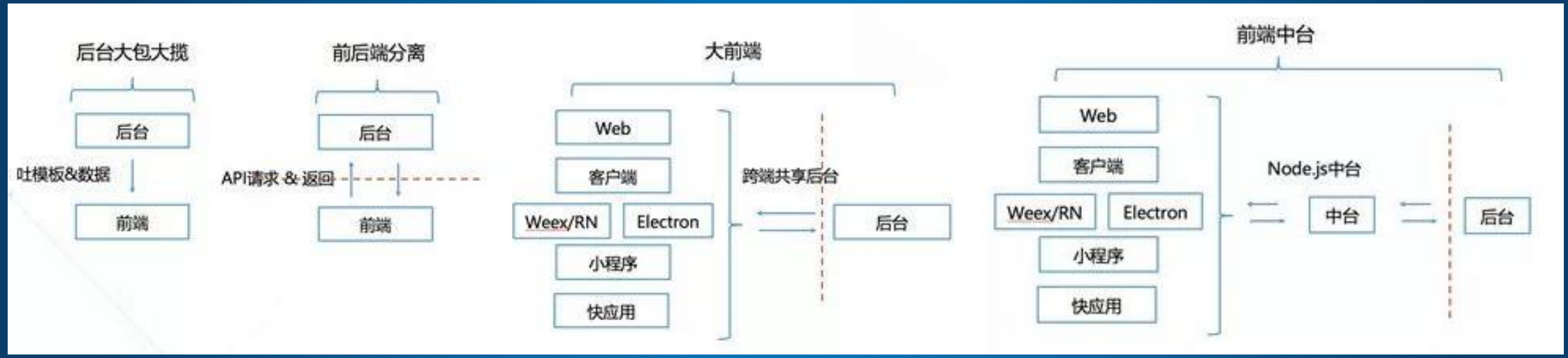
前端架构设计的目的

- 解决问题
  - 解决已存在或者未来可能发生的技术问题，增加项目的可管理性、稳定性、可扩展性。
- 提升人效比
  - 降低花费的人力成本，节约的时间和金钱
  - 避免的项目风险与资损、提高对业务的支撑能力以带来在业务上可衡量的更高的价值、以及其他价值。

系统架构设计做了什么


业务分析
领域模型
业务对象
业务用例
业务流程
需求分析
系统上下文
系统用例
非功能性需求
架构设计
应用架构概览
逻辑架构设计
物理架构设计
开发组件
部署组件
技术选型
概要设计与详细设计
功能模块划分
接口定义
模块实现逻辑描述
界面设计
数据库设计

前端架构是怎么发展起来的



前端架构的定义

前端架构是一系列工具、流程和设计的集合，旨在提升前端代码的质量，并实现高效、可持续的工作流。

体系设计

建立系统设计的规范，清晰描绘产品和代码的最终形态

架构设计

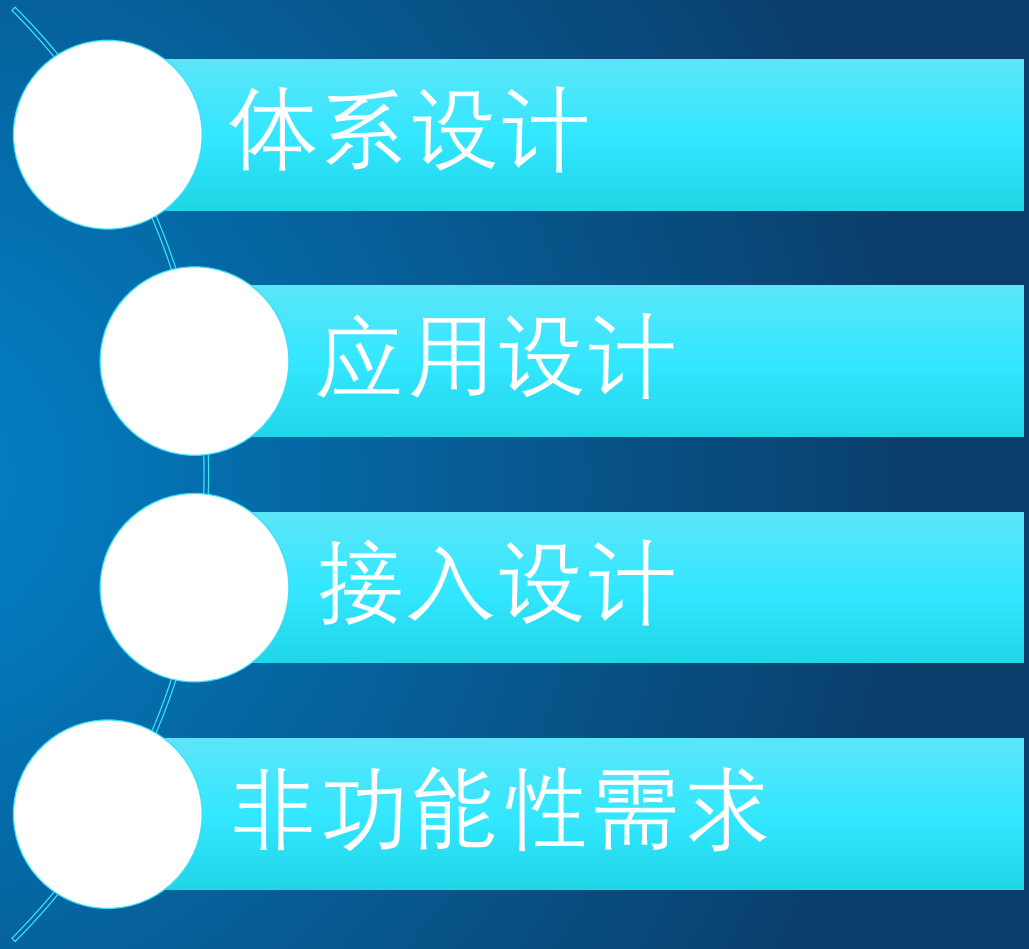
设计出能流畅运转的系统，制定完整开发工作流；

架构优化

监督跟进，能够持续地优化工作流程，保证项目高效率完成。

前端构架的内容

建立系统设计的规范，
清晰描绘产品和代码的最终形态



1

体系设计

建立系统设计的规范

体系设计

开发语言

TypeScript / JavaScript

开发规范

版本管理

工具选择、提交规范等

开发模式

脚手架

体系设计

输出文档

- 规范、约定及相关检查工具
- Mock方案
- 测试方案
- 持续集成与继续发布方案
- 日志与埋点方案
- 文档管理方案

2

应用设计

设计出能流畅运转的系统

子应用划分

页面架构

技术选型

功能模块

接口定义

接入设计

子应用划分

- 项目规模较大时，应该按业务来划分子应用
  - 方便进行版本管理;
  - 可以独立发布子应用;
- 意义:
  - 规范项目，增加代码的安全性，降低项目维

页面架构选择

- 单页
  - 优点
    - 用户体验更好，界面流畅；
    - 框架友好；
    - 多端适配，可移植性好；
  - 缺点
    - 初次加载时耗时较多；
- 多页
  - 优点：
    - 页面和页面之间独立，对有长期历史的项目来说，可维护性、可重构性更好；
    - 版本控制更简单；
    - 灰度发布更友好；
    - 可以减少内存泄漏的风险；
  - 缺点
    - 页面跳转时的白屏；
    - 跨页状态管理成本较高；

技术选型

开发框架	UI框架	状态管理	数据存储
• Angular • React • Vue • 不要框架	• AntDesign • Bootstarp • MaterialDesign • ElementUI	• DvaJs • Redux • mobx • Vuex	• localStorage • sessionStorage • cookie • indexDb

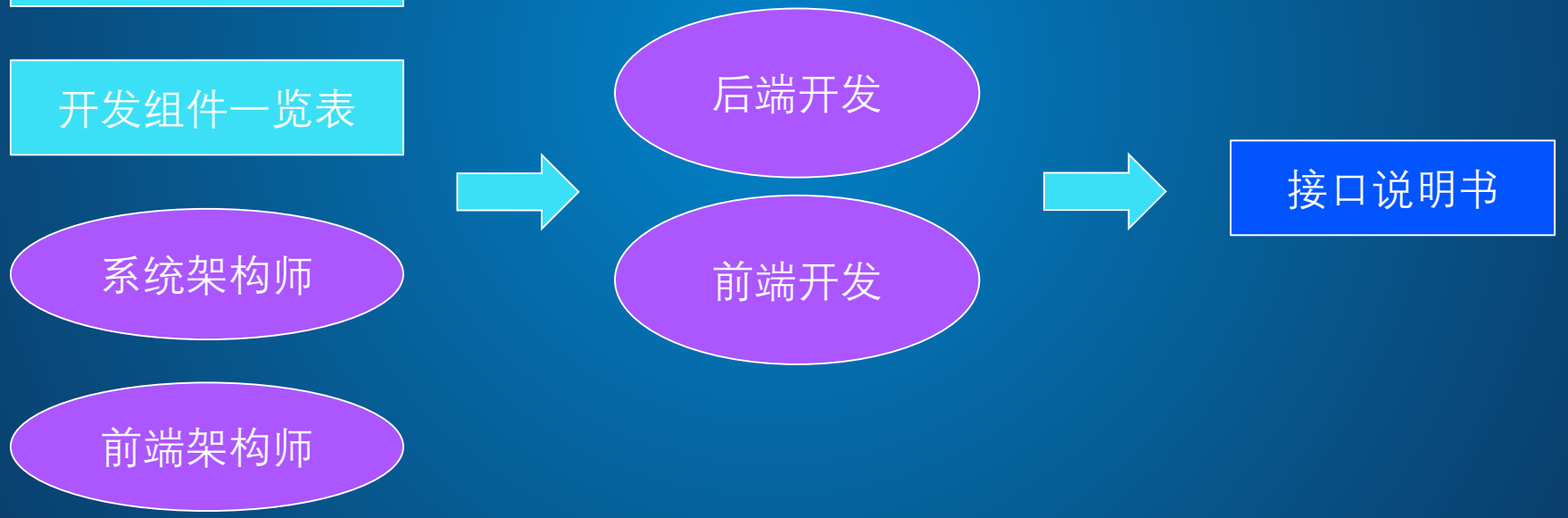
功能模块设计

- 模块划分
  - 结合系统用例以及具体页面
- 可复用组件
  - 独立模式
  - 配置模式
  - 组合模式

接口定义

- 根据分解出的模块来确定模块间的接口，

包括接口 系统用例 参数、返回值



输出文档

- 应用概览图
- 技术选型一览表
- 重要模块详细设计
- 接口说明书

3
接入设计

打通后台服务与用户

- 传统后端
- node.js中间层
- 反向代理
- 服务端渲染

接入设计

- 不分离，传统后端，ajax请求数据
- 初步分离，nginx反向代理或API网关
- 进一步解耦，node.js中间层，过滤合并请求
- 服务端渲染，直出页面

输出文档

- 前端系统接入文档
- 前端系统部署文档
- 中间层或SSR设计文档

4

非功能性需求

需求背后的需求

兼容性

浏览器兼容性是前端开发中头疼的事情，从IE到微信webview，无论技术发展到哪个时代都逃不掉。

那么那些事情是需要确认的呢？

- 各种浏览器内核具体的型号，而不是讨论搜狗、360这类壳浏览器。
- APP内部的webview，需要收集相关安卓或IOS的版本号。
- 是否允许一定程度上的降级策略，比如关闭过渡动画等。

安全性

- 身份校验和权限
- 表单验证
- XSS
- CSRF
- 文件上传

性能相关

- 首屏加载时间
- 响应时间
- 实时消息通知
- 分布式系统延迟

输出文档

- 兼容性列表
- 安全红线与典型处理方案
- 性能数据一览表

架构优化

持续监督跟进，
保证项目高效稳定的完成

架构跟踪

子模块的详细设计
代码Review

设计优化

- 增量设计
  - 重要的需求变更
  - 在不改变现有架构的基础上增加或者扩展设计
- 渐进式重构
  - 老的架构已经不能满足需求，但是不能一次性重构
  - 设定重构工作计划表
  - 按照独立的功能模块来进行

性能优化

- 优化与过度优化
  - 是指性能出现问题，流量出现瓶颈才进行的优化，不要为了优化而优化
- 弱依赖，强依赖与过度依赖
  - 弱依赖就是，你依赖的包，运行慢或者报错，都不影响功能
  - 强依赖本身意味着一荣俱荣，一损俱损，要进行转化
  - 不要依赖底层逻辑去解决上层业务问题
- 整体与部分
  - 降低单个部分对全局影响 —— 微前端

架构设计总结

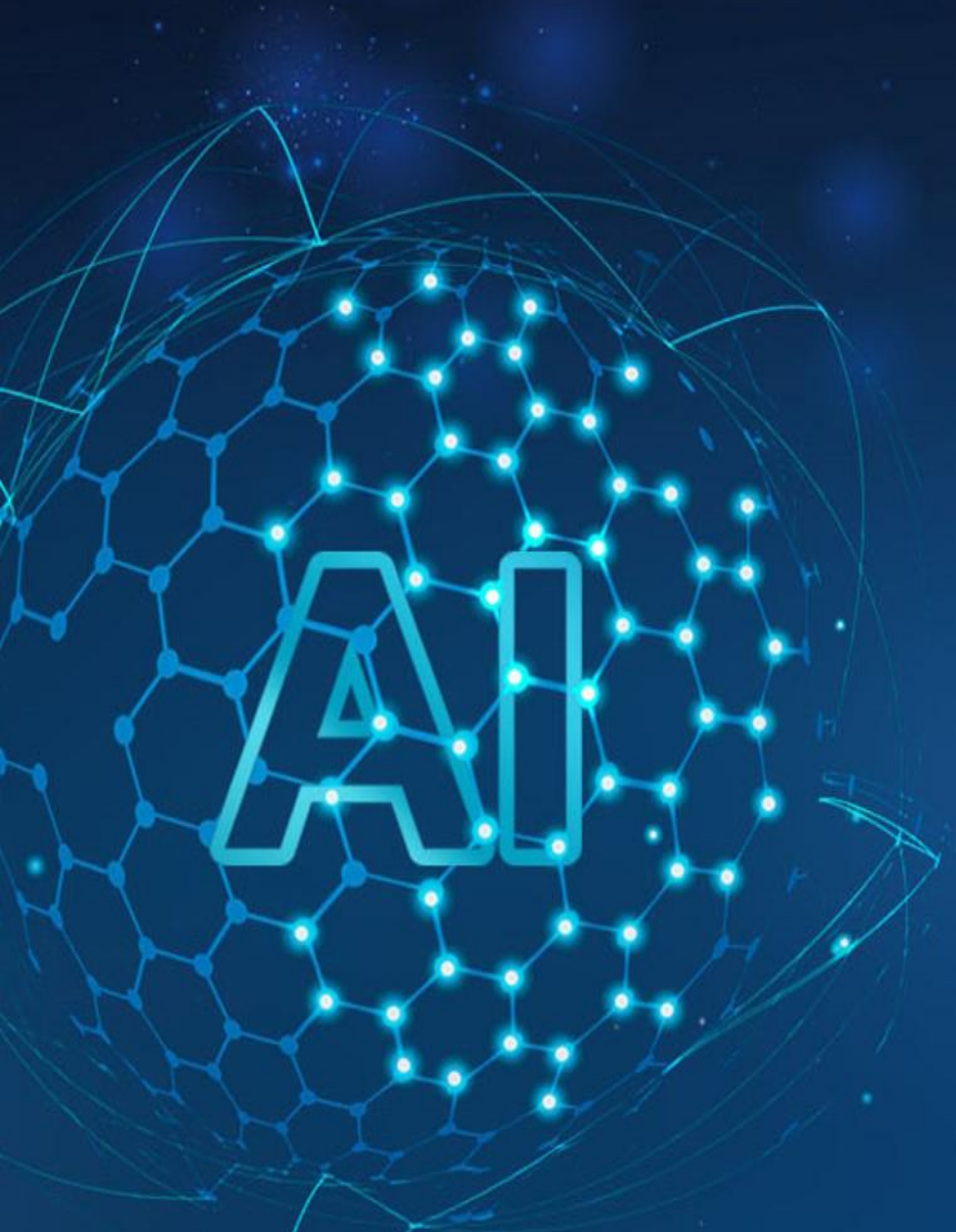
建立系统设计的规范，
清晰描绘产品和代码的最终形态

课程总结

- 体系设计
  - 解决前端本身所欠缺的规范与工程化问题
  - 提供便于开发测试所需的各种平台
- 应用设计
  - 做好子应用与模块的划分，设计出能够流畅运转的系统
  - 提供重要的模块详细设计
- 接入设计
  - 怎样把后台服务呈现到用户眼前
- 非功能性需求
  - 解决需求之外的需求
- 架构优化
  - 跟踪、Review你的设计，防止架构腐化
  - 重构过时的不满足需求的设计，进行架构演进

Q&A

提问时间



THANKS