一切回答要注重结合实例

什么是软件架构

1. 什么是架构

名词：体系结构，核心是结构，与结构相关，将产品分为一系列组件和模块交互

动词：应该构建什么样的设计愿景以便进行架构和做出恰当设计决策

架构基础：需求驱动架构

2. 架构的种类

软件架构、硬件架构、DB 架构、信息架构、安全架构...

不同种类的架构特点：结构和愿景（未来的发展），如结构上需要足够多的电缆，愿景上需要满足环境约束和非功能特性。

3. 什么是软件架构

①应用程序架构：软件设计的较低层面，只考虑单一技术栈

②系统架构：从组件和服务到子系统等较高层次的抽象，系统架构的定义大多数包括了软件和硬件

③软件架构：应用程序架构和系统架构的结合，即从代码结构和基础到将代码部署到生产环境。与软件系统重要元素相关的所有东西就软件架构。既是逻辑上高层次的概念和表达，又是物理抽象，图应该反映现实。

结构性+蓝图+折中选择（Every thing is a trade off，why is more important than how）

④软件架构的高层和低层：

低层：如面向对象的原则，接口、类、重构

高层：安全、性能、操作、审计、运维、登录和异常处理

4. 什么是敏捷软件架构

结构分析（SDLC）

传统：

系统规划（Systems planning）：问题或需要的改变，初步调研、可行性报告

系统分析（Systems Analysis）：需求文档

系统设计（Systems Design）：设计文档

系统实施（Systems Implementation）

系统测试（Systems testing）

系统支持和安全（Systems support and security）：安全、可靠、规模

敏捷：

规划（Planning）：确定目标、约束、交付目标

风险分析（Risk analysis）：确定风险和指定相应解决方案

实施（Engineering）：开发一个包含所以交付目标的原型

评估（Evaluation）：开展评估测试，制定下一个迭代的目标

敏捷方法：动快速，适应变化，持续交付，接受反馈，团队动态。

敏捷软件架构：

①与传统的区别：快速迭代，文档少。

启动→规划→执行→监控→收尾→迭代

②敏捷是相对的，按时间衡量，跟上环境变化就是敏捷

5. 架构 vs 设计
①所有的架构都是设计，但并非所有的设计都是架构。架构反映了使一个系统成型的重要决策设计，重要性通过改变的成本衡量。
②设计是一个系统内命名的结构或行为，助于解决系统中的问题，设计代表潜在决策空间的一个点

6. 软件架构重要吗
缺乏软件架构：非理想结构、不安全
拥有软件架构：明确的结构和愿景，减少风险
每一个软件项目都需要软甲架构

7. 为软件项目/软件系统的架构决策做一个清单，例子
①系统的形态（如客户端－服务器、基于 Web）
②软件系统的结构（如组件、层、交互）
③技术选择（即编程语言、部署平台、数据库）
④框架选择（例如，Web MVC[插图]框架、持久性/ORM 框架）
⑤设计方法/模式选择（如针对性能、可伸缩性、可用性）
①基于 web 无需安装其他软件，易于访问跨平台好②前后端分离，前端展示页面，后端处理数据③选用 Java 语言、Mysql 数据库，技术稳定，社区支持广④选用 Spring Boot 框架，能够快速开发⑤缓存机制减轻数据库压力

软件架构的角色

8. 软件架构的角色/软件开发者升级为架构师要做到哪些
架构驱动力：理解目标，抓住、提炼、挑战需求和限制
设计软件：建立技术战略、愿景和路线图
技术风险：发现、减轻和承担风险，保证架构的正常运行
架构演化：贯穿软件交付过程，持续的技术领导和对架构承担
编写代码：参与到软件交付的实践过程
质量保证：引入并坚持标准、指导、原则等

SA 角色：架构师、技术主管、首席设计师
合作：软件架构的角色或个人需要理解和认同架构的开发团队合作
SA 角色向软件团队引入技术领导：晋级软件架构师

9. 软件架构师应该编码
软件架构师是负责设计软件系统高层次结构的专业人士，构建原型、框架、基础，先完成架构的本职工作，有余力时编码。掌握新技术和应用进行代码评审，通过开源项目、新语言和新框架用于保持技术水平。

10.软件架构师应该是建造大师
软件架构师 += 工程师 + 建造师
成长路径：软件工程师→分析和设计师→架构师
象牙塔通常用来形容那些过于理想化、脱离实际、不关心社会现实或缺乏实践经验的学术环境或学者。软件架构不是象牙塔。

11.从开发者到架构师
①区分设计和架构的因素：规模扩大、抽象层级增加、做出正确设计决策的意义
②软件架构师总揽全局：看清大局才能理解软件系统整体如何工作
③开发者升级为架构师：经验、架构驱动力（非功需求）、架构演化、设计软件

12.拓展技术
大局观和底层细节自由切换，技术领导中编码最好，编码中领导力最好
①Technique：编程语言方法、API、框架、自动化单元测试。技术可以用于判断方案是否有效，技术可以回答需要这样架构吗
②避免一概而论，如只有一把锤子，一切问题都像钉子，要有多套方案。
③宽知识面：技术选型是否合适，设计和构建的选项是否采用通用模式

13.软技能
领导力、沟通力、影响力、信心、合作能力、责任感（个人品行）

14.软件架构不是接力运动
①敏捷团队：核心专长+知识与经验通才构成　　需求→架构→编码→部署
大团队更需技术领导
②解决方案/技术架构师问题：设计软件、编写文档方案、交付开发团队。原项目开发团队到新项目不能推脱架构责任。
③技术领导负责大局：创建愿景，交流，演化。避免接力后不管不问

15.软件架构要引入控制
①提供指导，追求一致性：若缺乏控制，则项目混乱不一致，引入控制和约束才能实现指导和一致性
②控制不能独断专行或完全放手
③控制是操控杆而非按钮，控制取决于团队经验、合作、规模、需求复杂性等。从部分控制开始，倾听反馈，微调。

16.避免鸿沟
①团队内的鸿沟：团队成员参差不齐，只关注底层代码，忽视全局
②闭门造车的独裁架构师：脱离团队，造成鸿沟
③消除鸿沟：让团队理解大局和决策理由，讨论架构合理性

17.未来软件架构师
指导辅导和师徒关系：开会、技术、职业生涯指导等
技术导师的流失：技术岗到管理岗

18.每个人都是架构师，除非另有身份
①每个人都是架构师：大局内外自如切换、会动手的架构师、能成功解决非功需求。
②除非另有身份：生存性（需直接指挥和控制的领导）、学习型（需要指导的领导）、自组织型（需简化以保持平衡）

③敏捷需要架构：复杂的非功需求和约束需要架构，集体代码所有制，每个人在架构的层次上工作。

19.软件架构咨询师
领域知识：由单一领域扩张到多领域知识。敏锐的分析能力，理解关键业务领域。
权威：技术领导责任需有决策权，软技能。

设计软件

20.架构驱动力
功能需求（需求驱动架构，容易识别，比较清楚，如功能清单、需求规格书、验收标准）+质量属性+约束+原则

21.质量属性（非功能需求，需要识别，不清楚，与服务质量有关）※
性能：比如延迟、吞吐、响应时间
可伸缩性：软件处理更多用户、请求、数据、消息等的能力，与并发性密不可分
可用性：比如运行时间、停机时间、定期维护、全天候等
安全性：如认证、授权、数据在运输和存储中的保密性等
可扩展性：如功能可扩展
友好性：系统的设计考虑到对残疾人、色盲等
灵活性：不能做死，不能只是一个具体的问题
互操作性：和系统的交互，如蓝屏提示故障码
审计：谁在什么时间干了什么事
业务连续性：如新旧系统之间
遵守法律法规
约束：技术选型、部署平台
原则：代码原则（编码规范，使用自动化测试）、架构原则（分层策略、架构模式）

22.处理非功能需求
捕获：客户只提功能需求或隐含非功需求，架构师应主动询问捕获
提炼：模糊→量化（如用户提快，要衡量具体快的指标）
沟通：成本 vs 实现高规格非功需求

23.约束
时间、预算、法律法规、道德约束，具有两面性，约束可减少设计工作。
①技术约束：批准的技术清单（许可证）、系统整合和互操作性、技术成熟度、内部知识产权
②人员与组织约束：团队规模、技能、扩展性、维护团队匹配性，软件系统是否战术或战略实施，组织政治。
③约束优先级：时间紧迫时时间优先高于技术清单。

24.原则 ※
①开发原则（代码级）：编码标准和规范、自动化单元测试、静态分析工具
②架构原则：分层策略（UI 层、业务层、数据访问层）、业务逻辑的位置、无

状态组件（可复制组件对系统横向扩展分担负载）、存储过程、域模型（丰富与贫瘠）
③高内聚、低耦合 SOLID※
单一职责原则（SRP）：一个类或模块只负责一个职责
开闭原则（OCP）：对扩展开放、对修改关闭
里氏替换原则（LSP）：子类必须能完全替代父类，且不改变原有程序行为
接口隔离原则：（ISP）：客户端不应被迫依赖其不需要的接口，将大接口拆解为更小更具体的接口。
依赖倒置原则（DIP）：高层模块不应依赖于底层，二者共同依赖抽象
约束是强加于你的，原则是将标准方法和一致性引入构建软件的方式而想采用的。
25.架构图应包含技术选型，但技术不是实现细节
①复杂非功需求（不考虑时，选择任何技术均可。考虑时，如可伸缩，不是所有技术均可）
②约束：影响软件架构，可挑战约束但不可忽视。
③一致性：团队选择技术/框架/方法的一致性，否则有副作用

26.更多分层等于更多复杂度
金融风险系统案例
需求：分发数据到企业局域网用户子集
解决方案：允许用户通过内部 web 程序访问数据
分层不是越多越好

27.协同设计是一把双刃剑
三层 web 应用：
克制自负，专注交付，满足需求的解决方案
Web 开发者：JSON 数据、JQuery 动态处理数据
服务端开发者：重用和扩展中间服务层、业务逻辑
DB 开发者：存储过程为主
避免凭知识、经验和偏好各自为战，合作就是沟通和挑战

28.软件架构是对话的平台
敏捷方法：定期在团队内与客户交流，防止跑偏
软件开发不只是交付特性，还有许多利益相关者：软件使用者，当前开发团队、未来开发团队、其他团队、安全团队、执行/支持人员

29.SharePoint 项目需要的
Sharepoint 为微软开发的企业级协作平台和文档管理系统。
新旧技术集成、复杂、非功能需求复杂
需技术领导、软件架构

可视化软件

30.软沟通障碍
敏捷团队：高效表达与沟通，故事墙或看板，可视化工作进展
老方法中的好东西：架构驱动保持整体一致性、统一软件开发过程

抛弃 UML 中的问题：失去视觉化沟通能力（没有统一标准）

31.对草图的需求
①分析瘫痪与难于理解的 UML 图之间挣扎
②测试驱动开发与图标：TDD（测试驱动开发），Red-Green-Refactor
画草图：非 UML、地图（高层次结构、非技术细节）
银弹（Sliver buller）：一种能够彻底解决软件开发复杂性、显著提升生产力和质量的“万能方法”或“终极技术”，是一种不存在的万能解决方案
Red-Green-Refactor：先编写一个失败的测试用例证明当前代码没有实现所需功能。编写最少量的代码，使测试通过，保证测试结果变为绿色。重构提升质量属性

32.无效的草图
如无具体内容、只有框没有连线、连线太乱没有逻辑
正确的草图有实线、虚线、不乱，逻辑层次分明，标题短而表意，图有图号和图题。

33.C4 图：语境、容器、组件、类
架构作图工具：M.S.Vision、Rational software
通用的抽象集合：OOP：软件系统→容器→组件→类，即软件系统由多个容器构成，容器又由多个组件构成，组件由一个或多个类实现
①语境图：
设定场景的高层次图，包括关键的系统依赖和参与者
意图：系统是什么？用户是谁？如何融入 IT 环境？
结构：中间画系统框图，周围是交互的用户和系统
②容器图（高层次的技术选择） container 图：
高层次的技术选择，容器如何分担职责、如何通信
意图：系统整体形态、高层次技术决策、职责分布、容器交互、在哪里写代码
结构：简单框图展示关键技术选择
③组件图 component 图：
每个容器的关键逻辑组件及之间的关系
意图：系统组件/服务、高层次如何工作是否清晰、组件/服务所在容器
④类图 class 图：
可选的细节层次，解释某个模块或组件怎样实现
不具体到详细的技术实现

34.软件架构和编码
职责驱动设计和组件分解：①深入组件、服务或模块②组件是接口后的一组相关行为，可用一个或多个类实现③好组件和好类共性，高内聚、低耦合、良好封装、良好定义公共接口
防止架构和编码脱节
用层封装代码，用特性封装、用组件封装
对齐软件架构和编码

35.你不需要 UML 工具

不同类型的 UML 工具：只有图（Vision）、逆向工程（从代码创建 UML 图，但太多细节）、往返工程（code→UML，UML→改变 code）

为软件生成文档

36.代码不会讲完整故事
①重构要求方法变得更小和可复用以及自文档化
②有人偏好自注释、有人偏好注释
③易读、理解和维护的好代码不是完整故事（如为何选择正使用的技术，各组件运行时部署在哪，如何沟通）
④拥有 TDD 测试驱动开发
37.软件文档即指南
敏捷也需要文档：不能走极端，没有文档或复杂的文档
文档即指南：图表起导航作用、C4、UML
文档应短小而简洁：12+1（语境、功能性概览、质量属性、约束、原则、软件架构、外部接口、代码、数据、基础设施架构、部署、运营和支持+决策日志）

38.功能概览
意图：系统关键功能，重要特性，重要用户的需求是否满足系统主要流程
结构：功能规格、用例文档、用户故事清单、图表、流程图
受众：开发团队内外技术和非技术人员

39.质量属性※
满足 SMART 原则：
Specific 具体（必须明确、具体，而不是模糊的描述，如不能上不封顶、尽快、尽可能小）
measurable 可衡量（必须能够通过指标或度量来验证如性能指标吞吐量 ≥ 500 TPS）
achieveable 可达成（必须在技术、资源和成本条件下可实现）
relevant 相关（必须与系统的业务目标和用户需求紧密相关）
time-bounded 及时（必须有时间约束，明确在何时达成或在何种时间范围内满足如上线前必须完成安全测试）
质量属性是软件系统在功能之外必须具备的特性，用来衡量系统的“表现如何”，而不是“能做什么”。

40.外部接口
有哪些外部系统接口
关键接口有哪些，有没有经过审查，哪些开放给客户
接口的技术定义是什么，消息格式是什么，同步还是异步
接口是否幂等（结果一样，如 x 的三次方和 y 的二次方结果一致）

开发生命周期中的软件架构

41.敏捷和架构的冲突：神话 or 现实
对立统一
敏捷和架构总体兼容：架构是结构和愿景，敏捷是交流和快速迭代

具体冲突：团队结构（架构师 vs 少文档化）
流程和产出（大型设计目标是蓝图前对全部知识达成共识，敏捷为随机应变）
架构提供 TDD、BDD、DDD、RDD（Test Driven Development、Behavior Driven Development、Domain Driven Design、Responsibility Driven Design）

42.量化风险
作用：识别风险是恰当预告设计的关键部分
概率和影响：风险发生概率与影响
设定风险的优先级：概率大、影响大的优先级高

43.风险风暴
①从 C4 图开始化架构图：C4 可用于标注架构中的不同风险
②识别风险：团队每人认识风险并量化（概率和影响）如第三方系统的数据格式意外变更、外部系统不可用、组件运行过慢、组件无法伸缩、关键组件崩溃、磁盘满
③汇总图中风险：把风险小贴士置于正确地方
④缓解策略：根据风险级别，修改架构，原型
⑤集体所有制：把技术风险所有权交给软件架构角色

44.避免预先设计太多或太少
太少：不了解系统边界是什么，在哪里。团队对大局没形成意识，没有考虑非功需求，没考虑现实约束。尚未确认重大问题及答案，解决问题方案不一致
太多：太多图标、过于死板、文档中写代码，所有抽线层的决策已做出，详细的 ER 图和 DB 设计

结构：理解主要结构元素以及它们如何基于架构驱动力结合在一起。 容器图和组件图
风险：识别并缓解最高优先级的风险
愿景：创建并交流团队开发工作的愿景 语境图、组件图

网站
前/后端开发工具：Hbuilder、Dreamweaver/vscode
版本控制系统：Git、CVS、SVN
代码调试与测试工具：Jet、Selenium、Postman
数据库管理工具：MongoDB、Mysql、SQLserver
项目管理与协作工具：YesDev、Asana、Jira
性能监测工具：Lighthouse、Analytics、Aws cloud watch
服务器配置和域名管理：注册购买——搭建——控维
网页设计与用户体验工具：Sketch、Figma、Invision
网站开发：Req-UI、UX-front-end

1. 数据库管理工具概述
Mysql：支持 E-R 图建模，高性能、可扩展、高可靠，采用客户端/服务器架构，以行为主。

Navicat：支持 mysql、mariaDB，数据的导入和导出
SQLyog：跨平台的 MySQL 管理，多服务器同步

2. 关系型数据库管理系统
MySQL：以行为主的关系型数据库，支持 E-R 图建模，高性能、可扩展、高可靠，采用客户端/服务器架构，
Oracle Database：高稳定、高扩展、兼容强
PostgreSQL：支持多种数据分区（hash、list 分区）

3. 非关系型数据库
MongoDB：游戏存储（高效查询）、查询
HBase：以列为主的非关系型数据库，可伸缩，灵活，自动分区，高并发读写。

非关系型和关系型比较：（下述描述是基于非关系型的）
数据模型：灵活的数据模型，无需预先定义表结构
扩展性：通过水平方向扩展处理大规模数据、读写需求
查询：使用非结构化的查询语言
事务支持：采用最终一致性，关系型保证强一致性

4. 数据库开发工具
数据库设计工具定义：创建、修改、管理数据库结构的软件应用程序。
重要性：简化数据库设计和调整过程，自动化生成定义语言
发展趋势：自动化、兼容化

SQL 编写和调试工具:
数据库管理工具：MySQL workbench、SQL server Management studio
SQL 代码编辑器：SQL server Management studio
数据库测试工具：SQL Fiddle、SQL Test
数据库性能分析工具：Perfmon、DBA Monitor

数据库性能优化工具：
监控工具：Perfom、SQL server Management studio
调优工具：MySQLTuner
健康检查工具：Sysbench
性能分析工具：Perfmon
负载测试工具：Apache JMeter

5. 数据库开发流程
①需求分析与需求文档撰写
需求收集方法：面对面询问、调查问卷、参考其他项目
需求分类与优先级设定：需求验证就与评审、需求文档撰写规范
需求跟踪与变更管理
②概念模型设计
实体-关系 ER 图，便于进一步逻辑和物理模型设计 现实→计算机

③逻辑模型设计
确保模型的一致性、完整性，如关系模型，确定数据表、字段、键
④物理模型设计
设计数据库表、索引、存储过程等，优化性能和存储空间
⑤数据库测试与实施
功能、性能、安全测试

数据库实施与配置
数据库设计（需求分析，概念、逻辑、物理模型设计）→数据库配置（定义参数、设置空间、访问权限、保证数据安全和完整性）→数据库安装与调试→数据导入和初始化→试运行与评价

6. 数据库维护与管理
数据库备份与恢复：完全备份与增量备份、冷备份与暖备份与热备份、冷恢复与暖恢复与热恢复　暖：数据库处于只读或有限运行状态　热：数据库正常运行
数据库性能监控与调优：性能监控工具、调整索引与优化查询语句与配置参数、性能测试与评估
数据库安全管理与权限配置：用户与权限管理、最小权限原则、密码策略和加密技术、安全审计与备份

APP
1. 需求分析与规划
确定用户需求、制定功能清单、市场定位与竞品分析、设定可量化目标
①市场定位与竞品分析：目标市场定义、竞争对手分析、市场需求评估、市场规模和增长趋势、市场定位和差异化
②制定详细需求文档和项目计划：明确项目目标、识别用户需求、制定功能规格、编写详细需求文档、制定项目时间表和里程碑

2. 原型设计与 UI/UX 开发
低保真、高保真原型设计、原型设计规范、原型测试与反馈、设计用户界面和交互流程
低：基本功能、用户流程
高：视觉效果、交互细节
设计规范：布局、颜色

3. 技术选型与架构设计
确定开发语言：java
技术栈：原生如 ios 的 swift、安卓的 java、跨平台的 flutter
UI/UX 设计工具：sketch、Figma
开发工具：Android Studio、IOS Xcode
系统架构设计和数据库设计：
应用架构设计概述（定义系统组件、之间的依赖关系以及如何部署提高性能）、系统组件与模块划分（确定系统核心组件和模块）、API 设计与服务端实现（通过定义 GraphQL API 确保前后端高效通信、安全性、性能优化）、数据库设计原

则（标准、一致、高效）、数据库安全与备份策略
云服务平台选择（考虑平台性能、稳定性、安全性、价格）、云存储解决方案（云存储为 APP 存储大量数据、提供管理、开发者根据数据类型和访问频率选择合适的云存储方案如 Amazon S3 或 Google cloud storage）、API 集成方法（REST、RPC 等允许 APP 快速访问外部数据和服务器）、身份验证与授权机制（常见验证方法有 JWT、授权机制有 RBAC 等）
数据库设计原则（标准、一致、高效）
标准：数据库设计要遵循统一的规范，包括命名规则、字段类型选择、范式化设计和文档化，即保持结构清晰、命名统一、规范化，便于维护和扩展
一致：数据库中的数据和结构要保持逻辑一致性，避免冗余和冲突，确保完整性约束生效，即保证数据在不同表和操作中始终正确、协调，不出现矛盾
高效：在满足规范和一致性的前提下，设计要兼顾性能，合理使用索引、分区、缓存和适度反范式化，即让数据库既能保证数据质量，又能在查询和更新时快速响应

4. 前端与后端开发
前端界面与功能实现：JS 交互...
后端数据处理与接口开发：数据库设计配置、API 设计与实现、数据传输和格式选择、接口文档编写、安全性与错误处理

5. 测试与调试 ※
①Debug、Test 区别：Test 主要关注于功能、性能，Debug 主要用于调通代码
②单元测试与集成测试:
单元测试：对软件最小单元进行测试，确保每个函数如期工作。单元测试框架与工具（Junit、TestNG、unittest，提供简洁的 API 和报告机制，便于编写和维护测试用例）
集成测试：检查不同模块或组件之间的接口是否正确，确保无缝协作。方法有黑盒测试（测试人员不关心程序内部逻辑，只根据输入和输出验证功能是否符合需求）、白盒测试（测试人员需要了解程序内部结构和逻辑，通过覆盖代码路径来验证正确性）、灰盒测试（结合黑盒和白盒测试，测试人员对系统内部有部分了解，但仍主要从外部验证功能）
自动化测试在单元测试和集成测试中应用、功能测试（界面、业务功能、兼容性测试、用户满意度）、性能测试（压力、负载、上传速度测试）、安全测试（数据泄露、恶意攻击、未授权访问）
Bug 修复和性能优化：Bug 修复策略（定期的代码审查、单元测试、用户反馈机制）、性能监控工具（Memory 分析师）、内存管理优化、代码优化技巧（精简冗余代码、避免重复设计、优化算法逻辑）、热更新和动态修复

6. 发布与上线
准备应用商店提交材料
应用描述与关键词优化、应用截图和视频制作、隐私政策和用户协议准备、应用商店账号注册与认证

7. 运营与维护

实施监控工具（Google Analytics）、错误跟踪与分析（Sentry）、定期更新与维护、收集用户反馈进行产品迭代（定义反馈目标、多渠道接受反馈）、数据分析和优化用户体验（用户行为分析、转化率分析如投放广告与购买率、事件追踪）

微信小程序

1. 微信小程序概述

①定义：无需手动下载、需宿主平台、轻量级应用

②特点：即用即走、快速启动、低内存占用

③优势：既继承网页应用的快速启动与低内存优势，又有原生应用出色体验

④场景：电商、餐饮

⑤发展历程：概念提出→正式上线→功能与服务扩展→生态体系完善→未来发展方向

2. 开发工具选择

微信开发者工具：提供代码编辑、调试、预览及上传、定期更新支持新技术，适应多种操作系统

可视化操作工具：Figma、Taro

代码编辑与补全：Wechat snippets 智能补全、wxapp-helper 页面与组件生成

3. 注册与账号管理

访问公众平台→填写信息→验证身份信息→获取 APPID 与 AppSecert→激活小程序账号

账号权限设置与管理：角色与权限配置、获取用户授权状态、提前发起授权请求、管理用户授权关系

开发者模式切换：开发者与游客模式、模拟器与真机调试

4. 开发环境搭建

开发则工具安装→配置开发工具→环境搭建与验证→更新与升级

本地开发环境搭建：下载安装工具、创建新项目与配置、本地服务器配置、开发者权限配置

5. 程序页面与组件

xml 与 wxml

目标：XML 为通用数据描述语言，存储和传输文件。WXML 为微信小程序界面描述语言

灵活：XML 语法严格、标签属性可自定义。WXML 语法相对简化

功能：XML 仅用于描述数据结构、无渲染能力。WXML 支持数据绑定、渲染能力

应用：XML 用于跨平台数据交换。WXML 仅用于小程序界面开发

标签：XML 无固定含义，可自定义。WXML 为微信封装组件，固定功能。

常用组件介绍与用法：

View 组件：容器视图　scroll-view：滚动效果 swiper：滑动组件　button：表单组件，界面按钮　image：插入图片

页面布局样式：flex 布局、栅格系统

6. API 与功能调用
小程序 API 分类与用法：网络请求 API、数据存储与缓存 API、设备 API、媒体 API、用户身份验证 API
微信支付接口请求：申请商户号→API 密钥→引入微信支付 SDK→配置微信支付
第三方服务与能力接入：第三方平台接入流程、数据接口使用、云开发能力利用、安全与性能服务

7. 小程序开发流程
①需求分析与产品规划：需求收集与调研、功能性需求定义、非功能性需求定义、需求评审与确认
②原型设计与用户测试：原型设计重要性（创建可交互的模型用来展示应用程序的界面和功能）、常用原型设计工具（Sketch、Figma）、用户测试方法与实践（用户使用原型的行为和反馈、A/B 测试）、收集用户反馈、优化迭代过程
③前后端开发过程：前端、后端联调

8. 测试与上线
①功能测试（界面测试、功能模块测试）、性能测试（响应时间、内存占用、承载能力和稳定性）、安全测试（身份验证、数据加载、代码审计）、兼容性测试
错误排查与问题修复：错误类型识别（语法错误、逻辑错误、运行错误）、控制台输出检查（错误描述、位置、行数）、调试工具使用（利用断点）、错误日志分析（具体错误信息和上下文）、版本与机型兼容测试
②发布与上线流程：提交小程序审核、配置服务器域名、上传代码和设置版本号、获取小程序账号权限
③开发常见问题与解决方案：页面加载慢（优化代码、减少 HTTP 请求次数）、兼容性（响应式设计、灵活运用样式）、数据绑定错误（数据与视图同步、双向数据绑定）、网络超时（配置网络时间、优化服务器）

QA：
①架构是什么

②IT 领域有很多不同类型的架构，有什么共同之处
如软件架构、系统架构、网络架构、企业架构，都关注结构（定义一系列的组件和模块交互），以需求为导向，注重非功能属性

③软件架构的标准定义

④如果用敏捷来描述一个软件的架构是什么意思，如何面向敏捷进行设计
4+使用模块化单体，便于独立修改，推进迭代。先满足最小可行性再逐步扩展，避免前期行动缓慢，文档多。

⑤把当前软件项目所做的架构决策列一个清单，他们被视为重要的原因明显吗
整体架构（B/S）、后端技术栈（SpringBoot）、数据库（Navicat）、前端技术栈

（Vue.js）

⑥从代码后退一步，你的软件系统中的大局包含哪些事情
架构模式、组件和模块、非功能需求、运维

⑦所在组织的技术职业发展怎么样，企业架构会是你的出路吗

⑧软件架构重要吗，为什么，好处是什么，你的软件项目架构足够吗，大还是小

⑨软件架构和软件开发角色的区别是什么

⑩软件架构的角色都做些什么，这个定义是你团队现状或理想？若是后者，你的团队可以做些什么改变？

①①为什么承担软件架构角色的人理解所用的技术很重要

①②如果你是所在项目的软件架构师，编码在工作中占多大比例，太多还是太少

①③作为一个软件架构师，若不能编码还有其他方式接触到项目中的底层工作吗，有其他方式更新技术吗

①④为什么知识的广度和深度都很重要

①⑤你认为自己掌握了承担软件架构角色所需的软技能吗，若没有在哪些方面进行改进，为什么，怎么做

①⑥从软件架构的观点来说，你目前的软件项目有足够的指导和控制吗，是不是太多

①⑦为什么合作是软件架构角色的一个重要组成部分？你的团队在这方面做的够不够，为什么

①⑧在你的团队中，对软件架构有定义好的参考案例吗，若有，每个人都明白吗，若没有，那么为了捋清架构师的角色和责任创建一个条例是否有价值

①⑨软件架构的角色如何融入敏捷项目和自组织团队