## 1、什么是架构？

（1）作为名词：架构是体系结构，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。

（2）作为名词：包括了理解你需要构建什么、设定愿景以便于进行构建和做出恰当的设计决策。架构是关于交流愿景以及引入技术领导力的，这样参与构建产品的每个人都能理解这个愿景，并为产品的成功做出积极贡献。

## 2、架构的种类？

（1）常见架构类型：

①基础设施架构②安全架构③技术架构④网络架构⑤系统架构⑥软件架构

（2）共同点：无论哪种架构，核心都包含结构和愿景：

①结构：涉及该领域的核心组件、模块及它们之间的交互关系，比如基础设施架构中的路由器、软件架构中的组件分层。

②愿景：明确架构要实现的目标，同时为参与构建的人员提供清晰方向，确保所有努力围绕共同目标推进。

## 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。

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：

①系统规划（Systemsplanning）：问题或需要的改变，初步调研、可行性报告

②系统分析（SystemsAnalysis）：需求文档

③系统设计（SystemsDesign）：设计文档

④系统实施（SystemsImplementation）

⑤系统测试（Systemstesting）

⑥系统支持和安全（Systemssupportandsecurity）：安全、可靠、规模

（1）敏捷：

①规划（Planning）：确定目标、约束、交付目标

②风险分析（Riskanalysis）：确定风险和指定相应解决方案

③实施（Engineering）：开发一个包含所有交付目标的原型

④评估（Evaluation）：开展评估测试，制定下一个迭代的目标

（2）敏捷方法：动快速，适应变化，持续交付，接受反馈，团队动态。

（3）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、架构与设计？

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？**

①缺乏软件架构：非理想结构、不安全

②拥有软件架构：明确的结构和愿景，减少风险

③总结：软件架构重要，每一个软件项目都需要软件架构。

**7、为软件项目/软件系统的架构决策做一个清单？**

①系统的形态（基于 Web）：基于 web 无需安装其他软件，易于访问跨平台好

②软件系统的结构（如组件、层、交互）：前后端分离，前端展示页面， 后端处理数据

③技术选择（即编程语言、部署平台、数据库）：选用 Java 语言、Mysql 数据库，技术稳定

④框架选择：选用 Spring Boot 框架，能够快速开发

⑤设计方法/模式选择（如针对性能、可伸缩性、可用性）：缓存机制减轻数据库压力。

**8、软件架构的角色/软件开发者升级为架构师要做到哪些？**

①架构驱动力：理解目标，抓住、提炼、挑战需求和限制

②设计软件：建立技术战略、愿景和路线图

③技术风险：发现、减轻和承担风险，保证架构的正常运

④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

⑤编写代码：参与到软件交付的实践过程

⑥质量保证：引入并坚持标准、指导、原则。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。

**9、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。

**10、软件架构要引入控制吗？**

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。

④从部分控制开始，倾听反馈，微调。

**11、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。

**12、质量属性？※**

①性能：比如延迟、吞吐、响应时间

②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

③可用性：比如运行时间、停机时间、定期维护、全天候等

④安全性：如认证、授权、数据在运输和存储中的保密性等

⑤可扩展性：如功能可扩展

⑥友好性：系统的设计考虑到对残疾人、色盲等。约束：技术选型、部署平台。原则：

①代码原则，包括编码规范，使用自动化测试
②架构原则，包括分层策略、架构模式。

**13、约束。**

定义：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。

**14、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、业务逻辑的位置、无状态组件、存储过程、域模型
（3）高内聚、低耦合 SOLID※
①单一职责原则（SRP）：一个类或模块只负责一个职责
②开闭原则（OCP）：对扩展开放、对修改关闭
③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
④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。
⑤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。
（4）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。

**15、C4 图。**

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：
①设定场景的高层次图，包括关键的系统依赖和参与者
②意图：系统是什么？用户是谁？如何融入 IT 环境？③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：
①高层次的技术选择，容器如何分担职责、如何通信
②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
③结构：简单框图展示关键技术选择
（3）组件图 component 图：
①每个容器的关键逻辑组件及之间的关系
②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。

**16、质量属性※满足 SMART 原则：**

①Specific 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）
②measurable 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）
③achieveable 可达成（必须在技术、资源和成本条件下可实现）
④relevant 相关（必须与系统的业务目标和用户需求紧密相关）

⑤time-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）。

**17、关系型数据库和非关系型数据库比较：**

①关系型数据库 采用表格模型，结构严谨，强调数据一致性和完整性（ACID）。适合复杂查询和事务处理，如金融系统，但在海量数据和高并发场景下扩展性较弱。

②非关系型数据库 数据模型灵活（如文档、键值对），易于水平扩展，适合处理大规模、高速或半/非结构化数据，如社交网络和物联网。通常在一致性与可用性之间权衡（BASE 原则）。

③核心差异：前者以结构和一致性优先；后者以灵活性和扩展性优先。

**18、数据库开发流程**

①需求分析与需求文档撰写：通过面对面询问、调查问卷、等方法，再进行需求分类与优先级设定

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

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

⑤数据库测试与实施：功能、性能、安全测试。

**19、数据库设计原则**

①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，便于维护和扩展

②一致：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，不出现矛盾

③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。

**20、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。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）。