软件体系结构与设计模式复习参考

题型： 选择题 20% 简答题 20% 分析题 40%

波哥语录：设计模式只会考单独一个，不会几个设计模式复合应用来考，这样难度太高。设计模式只会考基本的，深层的不会考。比如企业应用架构分层模式中的数据访问层如何支持多种数据库产品？这个可以用设计模式中的某设计模式解决这个详细设计问题。分析题就考软件体系结构风格与设计模式，本来有两道，我降低难度只考了一道。

第一章 软件体系结构基础

1. 软件体系结构（Software Architecture）出现的原因？

软件体系结构（Software Architecture）正是：面对日益复杂的系统的设计与构造问题，寻求理解大型软件系统结构的更好的方法，寻求构造更大更复杂的软件系统的更有效的方法的过程中，对实践中的工程经验加以总结，再上升到理论研究的层面进行系统化和规范化后出现的一门软件设计新技术与软件工程新研究方向。

2. 什么是软件体系结构（SA）？

SA={components, connectors}

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组组合连接起来。

3. 如何正确理解软件体系结构？

a) 宏观与微观

i. 软件体系结构主要关注的是系统的宏观结构（Macro-Architecture），它是对系统结构的高层描述，也称为系统概要设计。其设计结果体现了系统宏观的结构蓝图（Blueprint）；

ii. 对系统结构的局部（Components & Connectors）设计则是对系统微观结构（Micro-Architecture）的设计，也就是详细设计的主要设计内容，其设计结果体现为能具体指导编码实现的系统设计规范（Specification）；

b) 自顶向下（Top-down）与自底向上（Bottom-up）

i. 对系统的设计过程应先由总体到局部(Top-down)，而实现的过程则是先局部再整体(Bottom-up)。

4. 软件体系结构与软件开发过程

需求分析>需求规范>系统体系结构设计>体系结构文档>系统详细设计>详细设计文档>系统编码>系统测试

5. 系统体系结构的设计过程

在明确系统需求的基础上，对系统体系结构的设计主要包括分解与组合两个步骤：

分解（Decomposition）：按照系统功能分解的方式，将系统功能进行分解，每一个相对

独立的功能分解模块形成系统结构的一个功能构件；

组合（Composition）：考虑系统的各功能构件间的相互关系，通过什么样的连接机制实现将各个独立的构件形成一个整体系统；

6. Component（构件）

定义

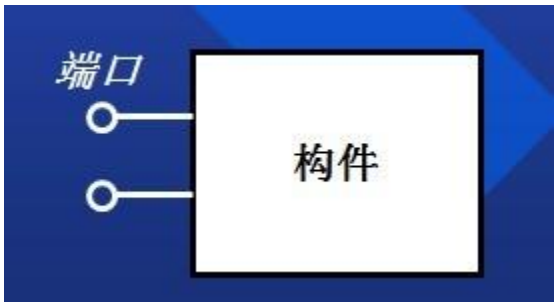
系统的逻辑与功能结构单元（A logical and functional unit of the system）

- 构件是一种抽象概念表述。是对以各种具体实现技术的系统结构组成元素（如子系统、模块、包等）的统称。
- 构件作为系统体系结构的一个结构组成单元，其是对系统功能集中的一个功能子集的结构化封装与实现。因此每个构件对于系统而言，应具有一定功能性。
- 构件作为一个封装的实体，通过其端口（Port）与外部环境交互。构件的端口（Port）表示了构件和外部环境的交互点。

构件的端口（Port）

正如构件这一概念一样，端口也是一个抽象概念。

其根据构件的实现形式具体体现为不同形式。如：对于包（Package）而言，端口指包中所包含的对象的公共（public）成员。对于模块而言，端口则是指模块中申明为外部（external）的过程或函数。



构件的分解

- 根据系统分析与设计的需要，一个构件是可以进行进一步的分解。
- 原子（atom）构件与复合（composite）构件

在一个系统体系结构设计中不再需要进行分解的构件称作原子构件，反之则称为一个复合构件。

构件的分类

根据构件的功能层次，分为： 基础构件 中层构件 高层构件

根据构件的功能类别，分为：数据构件 界面构件 控制构件 安全构件

根据构件的复用度，分为：通用构件 专用构件

7. Connector（连接件）

❖ 定义

构件间相互交互的机制与规则（The interaction rules and mechanisms among components.）

连接件包含两个方面的含义：

- ❑ 机制：是指连接件的具体实现形式：如过程调用、共享存储区
- ❑ 规则：是指构件使用连接件应遵循的规范。如对“过程调用”这种连接件其规则是指调用的接口参数形式、共享存储区则是指其本身的数据存储结构。

❖ 连接件的理解

- ❑ 构件通过不同的连接件实现彼此间的交互时，构件在交互过程中体现为不同的角色。

如：在过程调用时，角色有调用方和被调用方；

客户/服务器连接中，角色有客户方和服务器方；

中断连接中，角色有中断源和中断响应；

- ❑ 一个连接件所涉及的角色可能是二元的，也有多元的

如：过程调用中涉及的角色只有“调用者(caller)”和“被调用者(callee)”

消息队列中涉及的脚色包括“消息源（source）”和可能多个“消息接收者/处理者（receivers/handlers）”

- ❑ 方向性：单向和双向

如：有返回值的函数调用为双向连接；

无返回值的函数调用为单向连接。

- ❑ 性能特性

如：同步/异步连接、开放/安全连接、串行/并行…

如果连接的请求在得到相应并处理完成之前，请求方构件一直等待，直到收到被请求方构件的处理结果后才退出连接，这种连接称为同步连接。

如果请求构件只需发送请求，而不需立即等待请求的处理结果，这种连接称为异步连接。

8. 问题：下面哪些是同步连接件？哪些是异步连接件？哪种属于开放连接件？哪种属于安

全连接件？

TCP 协议	同步连接件
UDP 协议	异步连接件
HTTPS 协议	同步、安全连接件
Web Service	开放连接件
RPC（Remote Process Call）	同步连接件

以上答案仅供参考!

第二章 软件体系结构建模

1.为什么要进行体系结构建模？

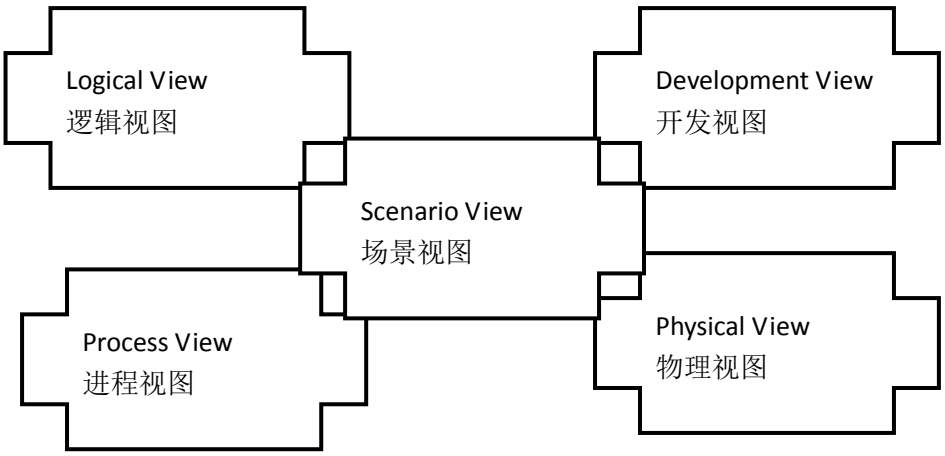
模型能够帮助人们建立对系统体系结构的认识，分析与验证设计的正确性与有效性。

为了减少不同人不同表达方式所造成的差异，形成一种较为规范与一致的系统体系结构的表达方式，方便设计人员之间的相互交流，需要采用一种大家均能接受的体系结构模型表达方法。

2.多视角建模

从不同的视角可以刻画系统体系结构的不同侧面，一个视角的刻画就是一个描述。

3.4+1 视图模型



3.1 各视图关注的主要问题

Logic View（逻辑视图）：关注系统的功能结构，它是系统的静态结构视图；

Development View（开发视图）：系统逻辑视图在软件详细设计与实现过程中的映射结构；

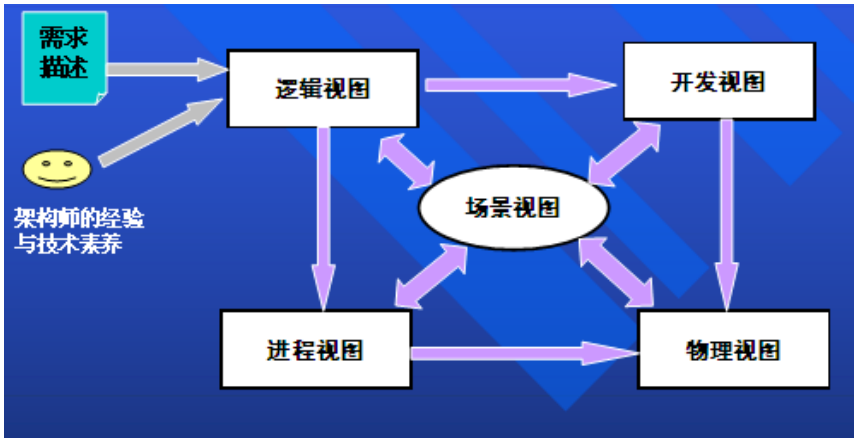
Process View（进程视图）：系统逻辑视图在运行环境（进程/线程）上的映射，它是系统的运行态的结构视图；

Physical View（物理视图）：系统逻辑视图中的各功能构件物理环境（计算与存储设备）中的映射。它是系统的物理部署视图。

Scenarios View（场景视图）：从系统使用的角度对系统结构的描述。它反映的是在完成一

个系统功能时，系统各功能构件间的协作关系。

3.2 视图运用过程



3.3 各视图比较

视图	逻辑视图	进程视图	开发视图	物理视图	场景视图
构件	逻辑功能	进程/线程	模块、包、子系统	计算资源节点(分布式)	步骤、脚本
连接件	使用、依赖、聚集……	进程/线程间通讯	include 、import 、using	通讯连接（分布式）	
关注点	静态功能结构	运行时系统结构	实现划分	部署及维护环境	可理解性
主要涉及者	系统设计人员、用户	系统设计人员	系统开发人员	系统部署与维护人员	系统设计人员、用户

第四章　典型软件体系结构风格及其应用模式

1、 软件体系结构风格与应用模式：

（1）管道过滤器体系结构风格及其应用模式

管道—过滤器体系结构风格为处理数据流的软件系统架构提供了一种参考结构。它是由过滤器和管道组成的.每个处理步骤都被封装在一个过滤器组件中，数据通过相邻过滤器之间的管道进行传输。每个过滤器可以单独修改，功能单一，并且它们之间的顺序可以进行配置。

- 构件类型：过滤器（Filter）----数据处理构件
- 连接件类型：管道（Pipe）----过滤器间的连接件

适合解决的问题：

- 处理或者转换输入数据流.
- 对数据的处理可以容易地分成几个处理步骤.
- 系统的升级要求可以通过替换/增加/重组处理步骤实现. 有时甚至由使用者完成操作.
- 不同的处理步骤不共享信息.

过滤器(Filter)是数据流水线的处理单元,负责丰富,提炼或转换他的输入数据. 它以下面的三种方式工作:

- 随后的数据流水线单元从过滤器中拉出（pull）数据.
- 前面的数据流水线单元把新的输入数据压入(push)过滤器.
- 过滤器以循环的方式工作，从流水线中拉出输入数据并且将其输出数据压入流水线

管道(Pipe)表示过滤器之间的连接;数据源和第一个过滤器之间的连接;以及最后的过滤器和 data sink 之间的连接：

- 如果管道连接两个主动过滤器,那么管道需要进行缓冲和同步。
- 管道可以实现数据在过滤器之间的数据转换，将一个过滤器的输出数据格式转换为其后接的过滤器的数据输入格式。

应用模式 如：PageController

2、 分层系统结构风格与应用模式

Model:

- 将系统分成适当层次，按适当次序放置。
- 从最低抽象层次开始，以梯状把抽象层次 n 放在 n-1 层顶部。直到功能顶部。
- 第 n 层提供的绝大多数服务由第 n-1 层提供的服务组成。
- 每个层次是一个独立的组件。它的责任是：提供了由上层使用的服务，并且委派任务给下一层次。
- 不允许较高层次直接越级访问较低层次。
- 每个独立层可能由多个不同的相对独立的实体组成。这些实体之间可能也有相互调用的关系
- 可以通过层的接口保护层次之间的封装性。

“层”的设计

- 根据抽象准则定义抽象层次
  - 每个抽象层次对应模式中的一层。

- 是否把某些功能分在两个层次/或者合并成一个层次时需要权衡利弊
  - 过多的层次增加开销；
  - 层次太少则结构比较差；

❑给每个层次命名并指定任务

❑最高层的任务是整个系统的任务。如果较高层完成某个任务的时候，需要使用到下一个抽象层次的细节，那么给下一层次增加相应的任务。

❑分层的时候还需要考虑到将来的重用。设计底层时需要考虑可能的标准以增加重用的特性。

应用模式 如：The Tcp/IP Protocol suite also is a layered system Model

基于分层体系结构的企业应用系统开发模型

基于 J2EE 的分层企业应用架构

附《波哥语录》

设计模式部分不会区分 C++和 Java
设计不是编码，与语言无关
架构主要是基本概念
不写代码，但要求画 UML 设计类图
2-4-4 的分值比例
分析题只有一道
分析题就考软件体系结构风格与设计模式，本来有两道，我降低难度只考了一道
设计模式只会考一个模式，主要是为降低难度
每类设计模式我只会考 2-3 个
简答题只是考概念
设计模式大家注意模式的应用
比如企业应用架构分层模式中的数据访问层如何支持多种数据库产品？这个可以用设计模式中的某设计模式解决这个详细设计问题
设计模式只会考单独一个，不会几个设计模式复合应用来考，这样难度太高
其实不难，大家只要平时认真，考高分不难
单选、简答和应用分析各占 2-4-4

如有疑问，请到软件学院论坛提问：

http://new.cquhx.cn/forum.php?mod=forumdisplay&fid=150

波哥会及时跟大家沟通交流！