系统架构设计师

架构知识1

姜美荣

目录

直播内容：架构知识

1. 传统架构风格相关案例☆☆☆☆☆

- 分类
- 优缺点
- 适用场景
- 对应案例

2. 架构质量属性相关案例☆☆☆☆☆

- 质量属性分类
- 质量属性场景
- 质量属性辨识
- 对应案例

传统架构风格–定义

软件架构的概念

软件体系结构（架构）风格是描述某一特定应用领域中系统组织方式的惯用模式。体系结构风格定义一个系统家族，即一个**体系结构定义一个词汇表和一组约束**。

① 词汇表中包含一些构件和连接件类型。

② 约束指出系统是如何将这些构件和连接件组合起来的。

软件架构风格反映了领域中**众多系统所共有的结构和语义特性**，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

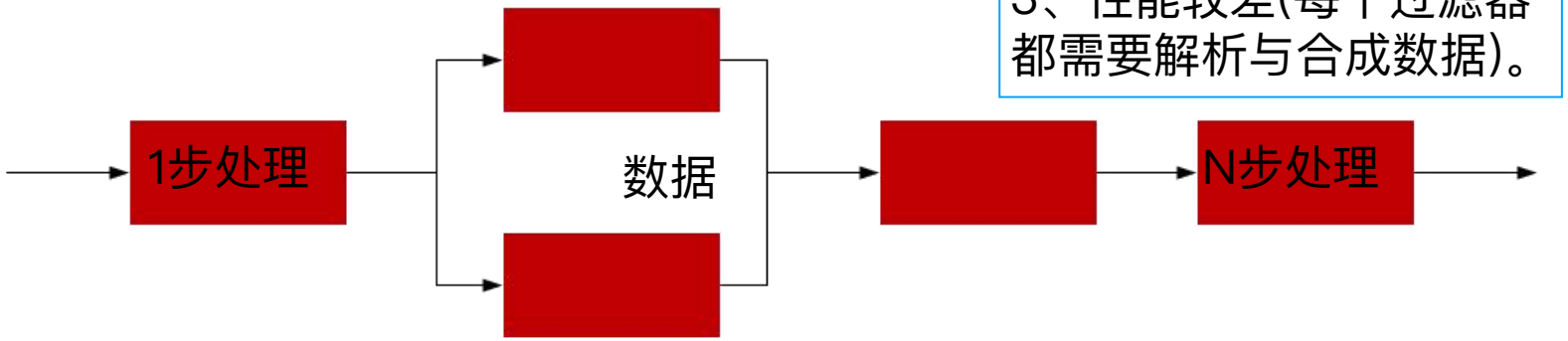
传统架构风格–通用架构风格

软件架构设计的一个**核心问题是能否达到架构级的软件复用**。架构风格定义了一个系统“家族”，即**一个架构定义、一个词汇表和一组约束**。词汇表中包含一些构件和连接件类型，而约束指出系统 是如何将这些构件和连接件组合起来的。**架构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个构件有效地组织成一个完整的系统**。

五大架构风格	子风格
数据流风格 【Data Flow】	批处理【Batch Sequential】、 管道–过滤器【Pipes and Filters】
调用/返回风格 【Call/Return】	主程序/子程序【Main Program and Subroutine】 面向对象【Object–oriented】、 分层架构【Layered System】
独立构件风格 【Independent Components】	进程通信【Communicating Processes】 事件驱动系统(隐式调用)【Event system】
虚拟机风格 【Virtual Machine】	解释器【interpreter】、 规则系统【Rule–based System】
以数据为中心 【Data–centered】	数据库系统【Database System】 黑板系统【Blackboard System】 超文本系统【Hypertext System】

数据流体系结构风格

1. 数据流风格



缺点：
1、交互性较差；
2、复杂性较高；
3、性能较差(每个过滤器都需要解析与合成数据)。

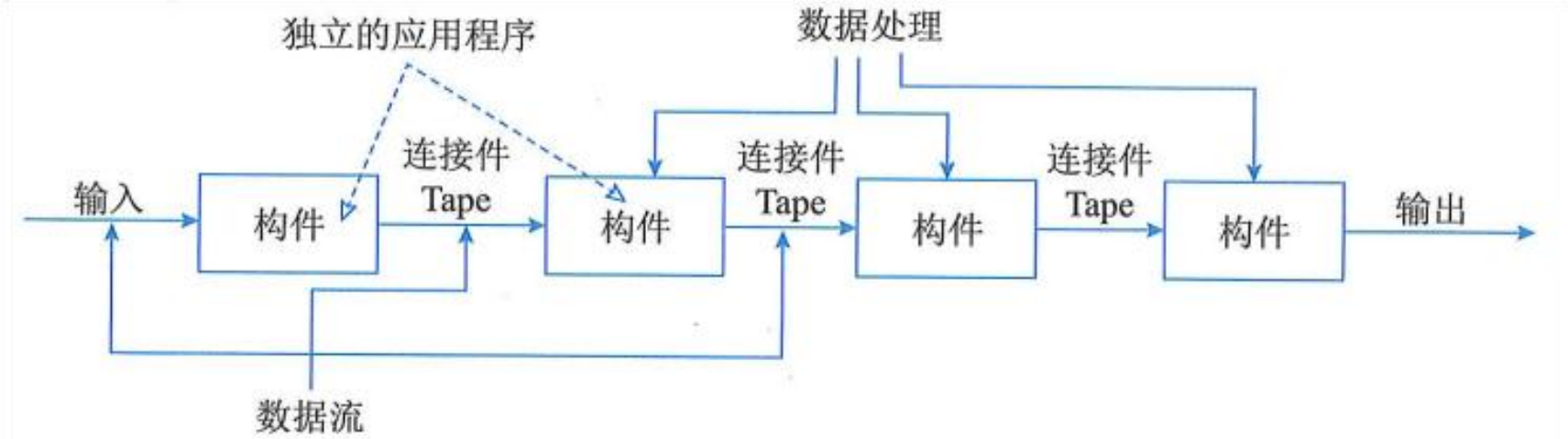
优点：
1、松耦合【高内聚–低耦合】；
2、良好的重用性/可维护性；
3、可扩展性【标准接口适配】；
4、良好的隐蔽性；
5、支持并行。

实例：
传统编译器、网络报文处理

数据流体系结构风格

1. 数据流风格包括批处理序列和管道/过滤器两种风格。

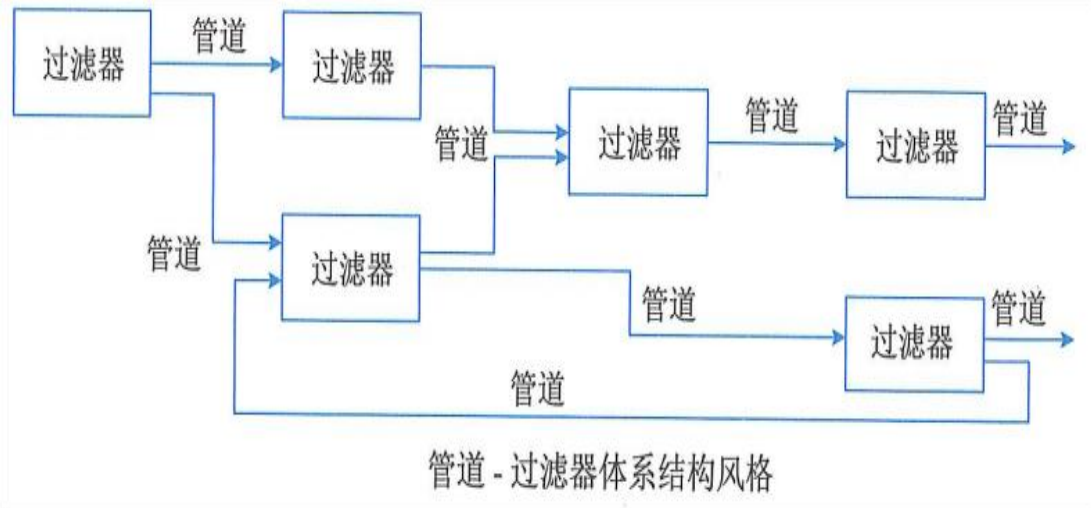
（1）批处理风格。构件为一系列固定顺序的计算单元，构件之间只通过数据传递交互。每个处理步骤是一个独立的程序，**每一步必须在其前一步结束后才能开始，数据必须是完整的，以整体的方式传递**。它的基本构件是独立的应用程序，连接件是某种类型的媒介。连接件定义了相应的数据流图，表达拓扑结构。



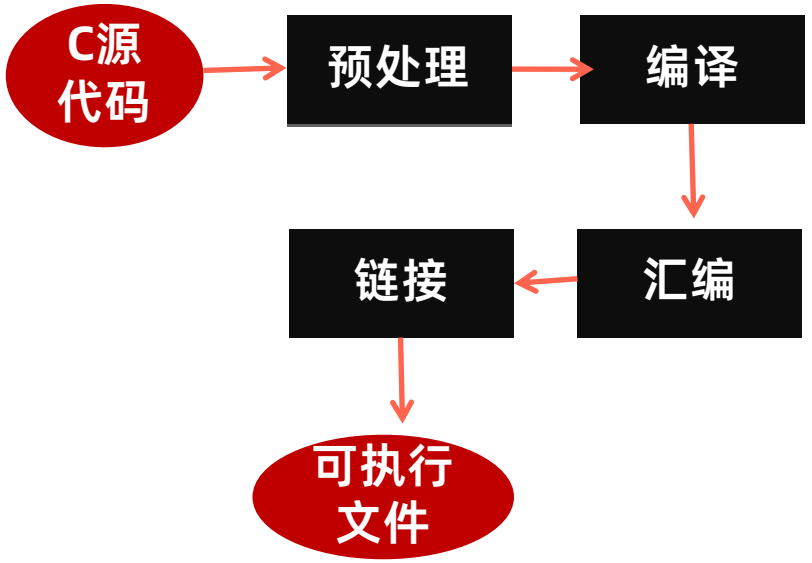
批处理体系结构风格

数据流体系结构风格

（2）**管道/过滤器**。每个构件都有一输组入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。



管道 - 过滤器体系结构风格



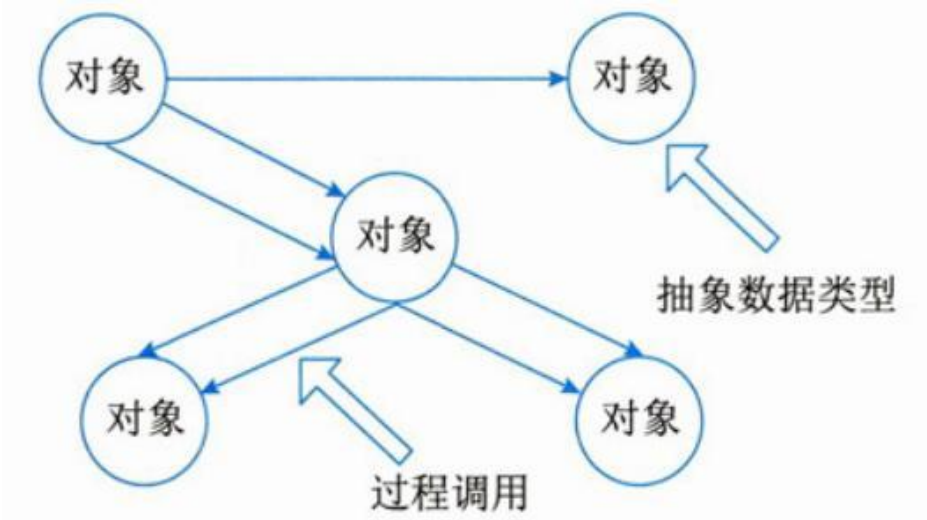
传统编译器

调用/返回体系结构风格

2. 调用/返回风格包括**主程序/子程序、面向对象、层次型以及客户端/服务器风格。**

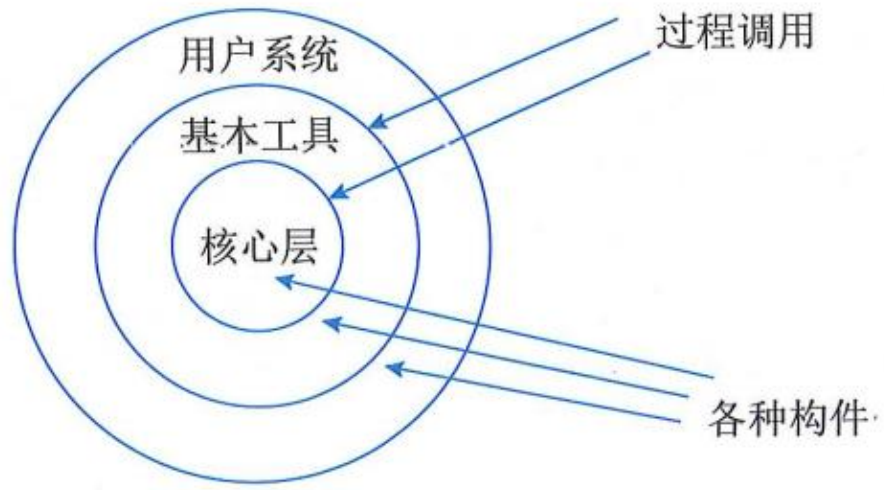
（1）主程序/子程序风格。单线程控制，把问题划分为若干个处理步骤，**构件即为主程序和子程序**，子程序通常可合成为模块。过程调用作为交互机制，即充当连接件的角色。 调用关系具有层次性，其语义逻辑表现为**主程序的正确性取决于它调用的子程序的正确性**。

（2）面向对象体系结构风格。**构件是对象**，对象是抽象数据类型的实例。 在抽象数据类型中，数据的表示和它们的相应操作被封装起来，对象的行为体现在其接受和请求的动作中。对象具有封装性，一个对象的改变不会影响其他对象。



调用/返回体系结构风格

（3）**层次型体系结构风格**。层次系统的构件组织成一个层次结构，连接件通过层间交互的协议来定义。该风格的特点：**每层为上一层提供服务，使用下一层的服务，只能见到与自己邻接的层**。将大的问题分解为若干个渐进的小问题逐步解决，可以隐藏问题的复杂度。



优点：

1、良好的重用性，只要接口不变可用在其它处；

2、可维护性好；

3、可扩展性好，支持递增设计。

缺点：

1、并不是每个系统都方便分层；

2、很难找到一个合适的、正确的层次抽象方法；

3、不同层次之间耦合度高的系统很难实现。

特点：

1、各个层次的组件形成不同功能级别的虚拟机；

2、多层相互协同工作，而且实现透明。

调用/返回体系结构风格

（4）**客户端/服务器体系结构风格**。

二层C/S架构：“分而治之，各司其职”客户端和服务器都有处理功能，现在已经不常用。

C/S 架构的优点：**系统的客户应用程序和服务器构件分别运行在不同的计算机上**，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析， 而服务器的开发则织中于数据的管理。

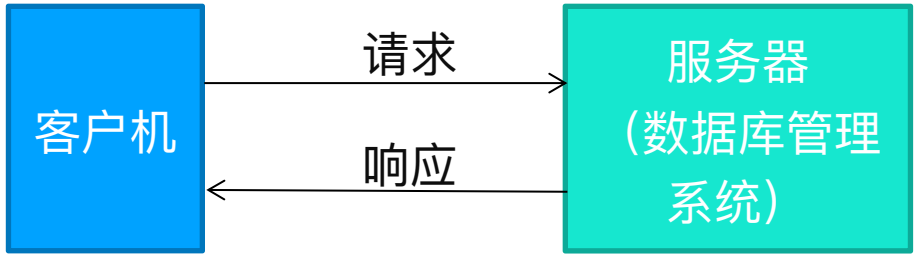
缺点：

（1）开发成本较高　　**（2）客户端程序设计复杂。**

（3）用户界面风格不一　　**（4）软件移植困难**

（5）软件维护和升级困难　　**（6）新技术不能轻易应用**

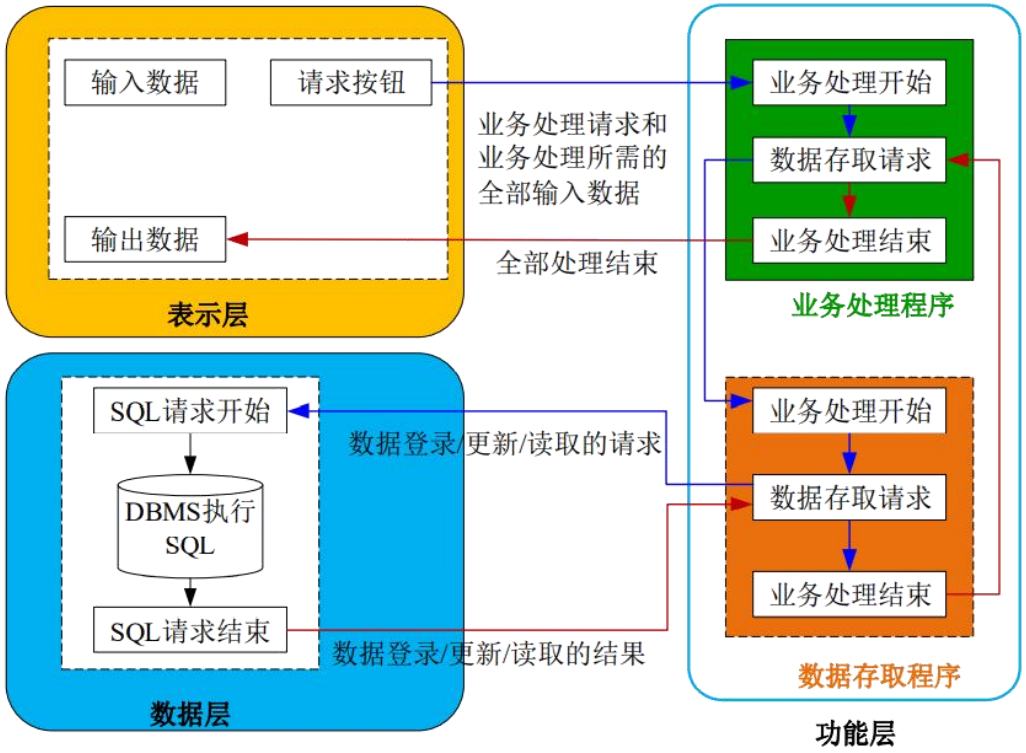
（7）可扩展性差　　**（8）系统安全性难以保证**



调用/返回体系结构风格

与二层C/S架构相比，在三层C/S架构中，增加了一个应用服务器。**可以将整个应用逻辑驻留在应用服务器上，而只有表示层存在于客户机上**。这种客户机称为瘦客户机。三层C/S架构将应用系统分成表示层、功能层和数据层三个部分。

层次	功能
表示层	**用户接口**，检查用户输入的数据，显示输出数据。
功能层	业务逻辑层，是**将具体的业务处理逻辑编入程序中**。
数据层	相当于二层 C/S 架构中的服务器**对DBMS进行管理和控制**。

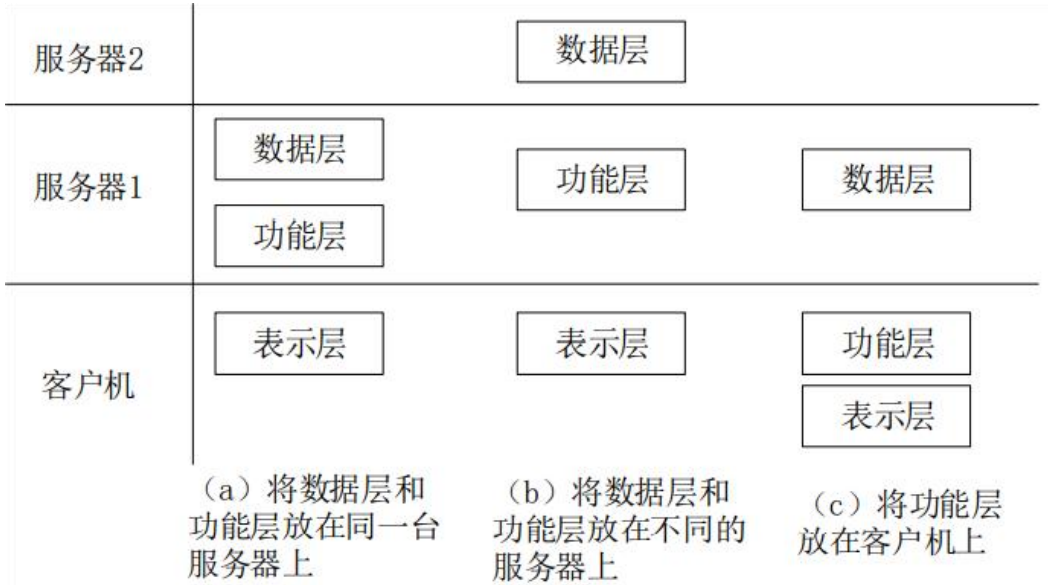


三层C/S架构的一般处理流程

调用/返回体系结构风格

与传统的二层架构相比，**三层C/S架构**具有以下优点：
（1）允许合理地划分三层的功能，使之在逻辑上保持相对独立性
（2）允许更灵活
（3）系统的各层可以并行开发
（4）利用功能层可以有效地隔离表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层

设计时必须慎重考虑三层间的通信方法、 通信频度和数据量



三层C/S架构的物理部署

调用/返回体系结构风格

浏览器/服务器(Browser/Server,B/S)架构是三层C/S架构的一种实现方式，其具体结构为“浏览器/Web服务器/数据库服务器”。B/S架构利用不断成熟的WWW浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。

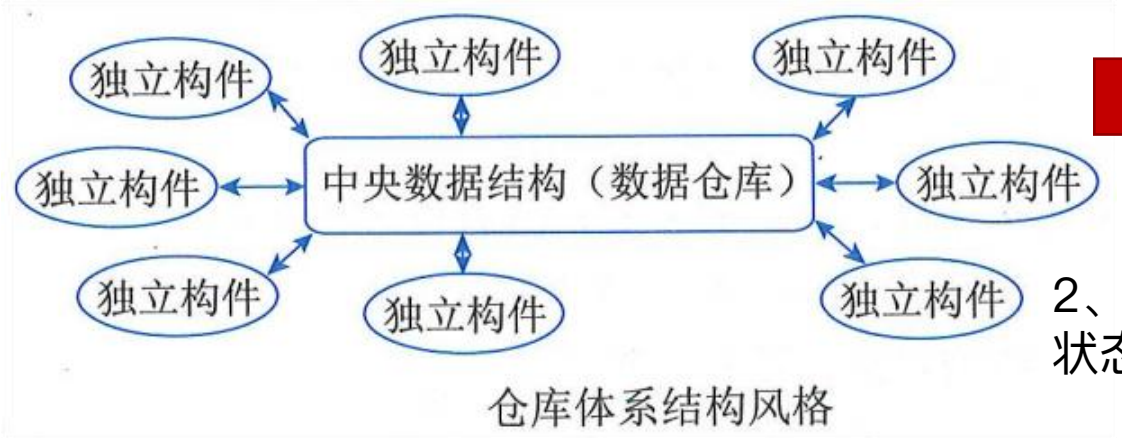
B/S优点：分层解耦、 0客户端架构仅用浏览器访问即可。

B/S缺点：**缺乏对动态页面的支持能力，没有集成有效的数据库处理功能；安全性难以控制；采用 B/S 架构的系统在数据查询等响应速度上远远低于 C/S 架构：B/S 架构的数据提交一般以页面为单位，数据的动态交互性不强，不利于 OLTP 应用**。

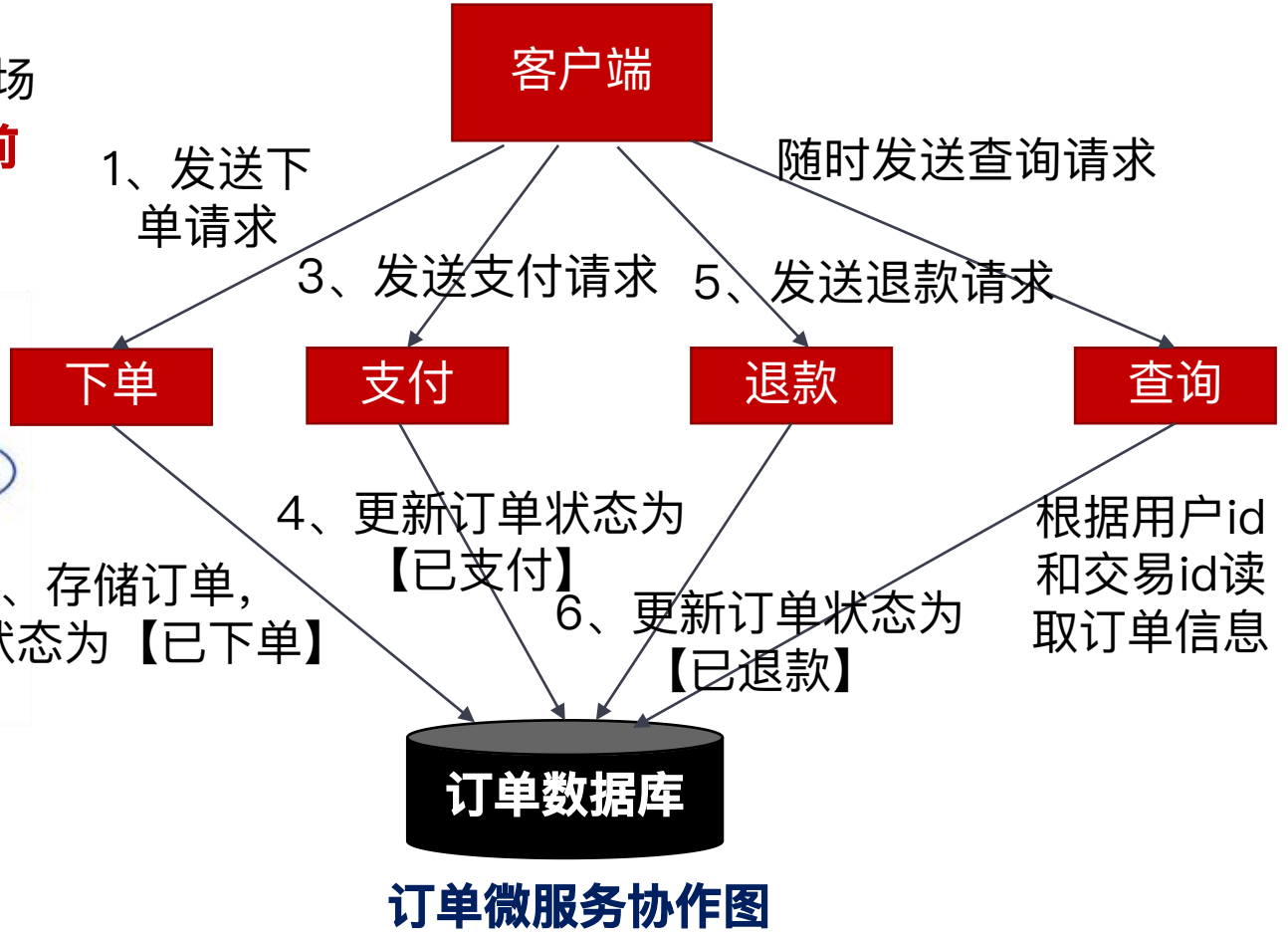
以数据为中心的体系结构风格

3. 以数据为中心的体系结构风格主要包括**仓库风格和黑板风格。**

（1）仓库(repository)是存储和维护数据的中心场所。有两种不同的构件：**中央数据结构(说明当前数据的状态)，对中央数据进行操作的独立构件**。



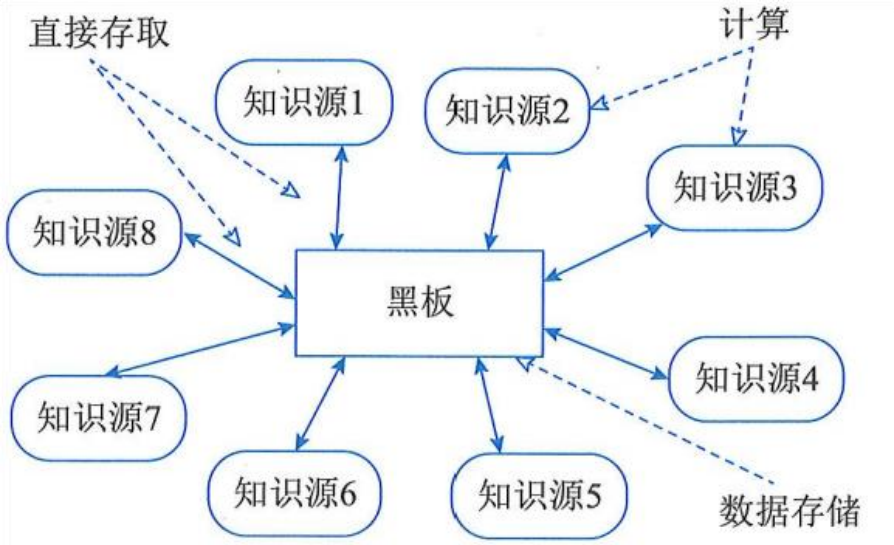
仓库体系结构风格



订单微服务协作图

以数据为中心的体系结构风格

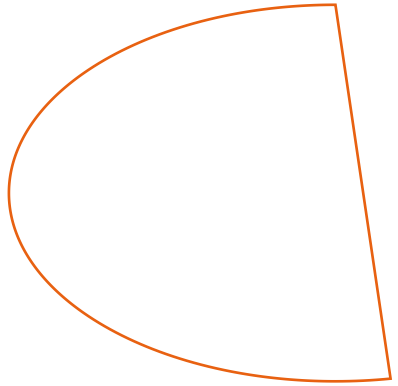
（2）**黑板体系结构风格**适用于**解决复杂的非结构化的问题**，能在求解过程中综合运用多种不同知识源，使得问题的表达、组织和求解变得比较容易。黑板系统是一种问题求解模型，是组织推理步骤、控制状态数据和问题求解之领域知识的概念框架。黑板系统的传统应用是**信号处理领域**，如**语音和模式识别**。另一应用是**松耦合代理数据共享存取**。



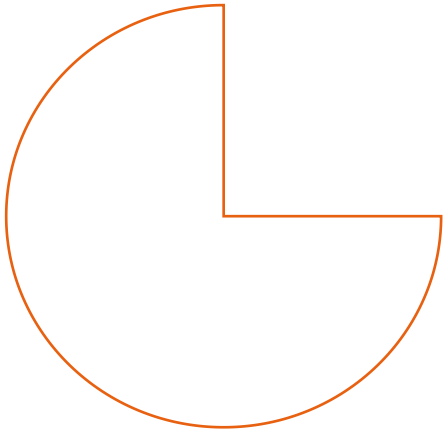
黑板体系结构风格

黑板风格：与数据库系统相反，一个或多个构件以被动触发的方式以不确定的顺序去更新共享数据存储区。每个构件可能多次参与执行流程，但流程本身无法事先确定。

以数据为中心的体系结构风格



以数据为中心的体系结构风格–仓库风格



以数据为中心的体系结构风格–黑板风格

优点：可更改性和可维护性；可重用的知识源；容错性和健壮性。

缺点：测试困难；不能保证有好的解决方案；难以建立好的控制策略；低效；开发困难；缺少并行机制。

特点：在以数据为中心的基础上，使用**中心数据触发业务逻辑部件。**

使用场景：语音识别；模式识别；图像处理；知识推理。

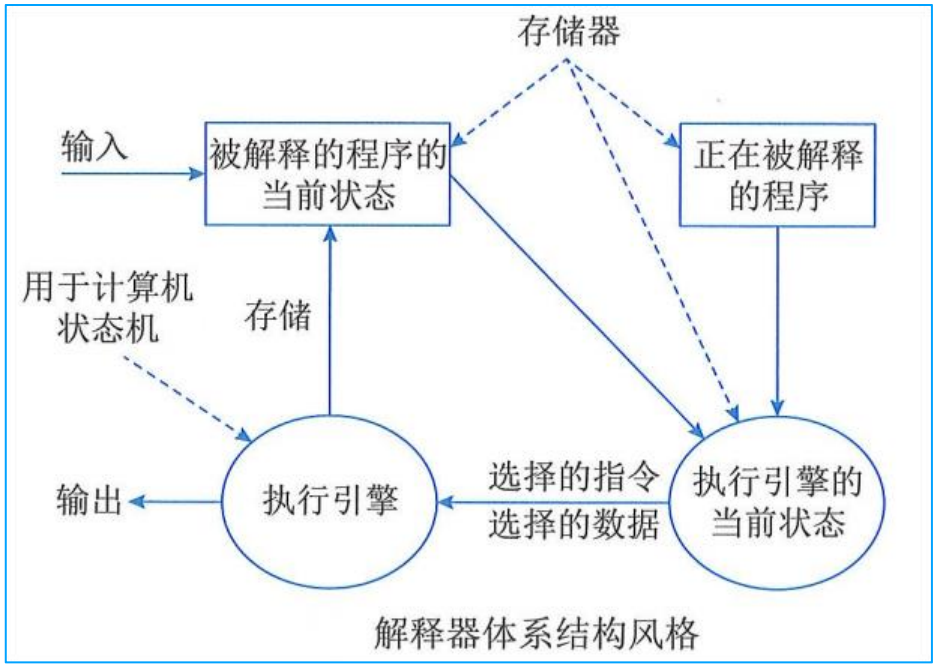
虚拟机体系结构风格

4. 虚拟机体系结构风格主要包括**解释器风格和规则系统风格**。

（1）解释器风格

一个解释器通常包括**完成解释工作的解释引擎，一个包含将被解释的代码的存储区，一个记录解释引擎当前工作状态的数据结构，以及一个记录源代码被解释执行进度的数据结构。**

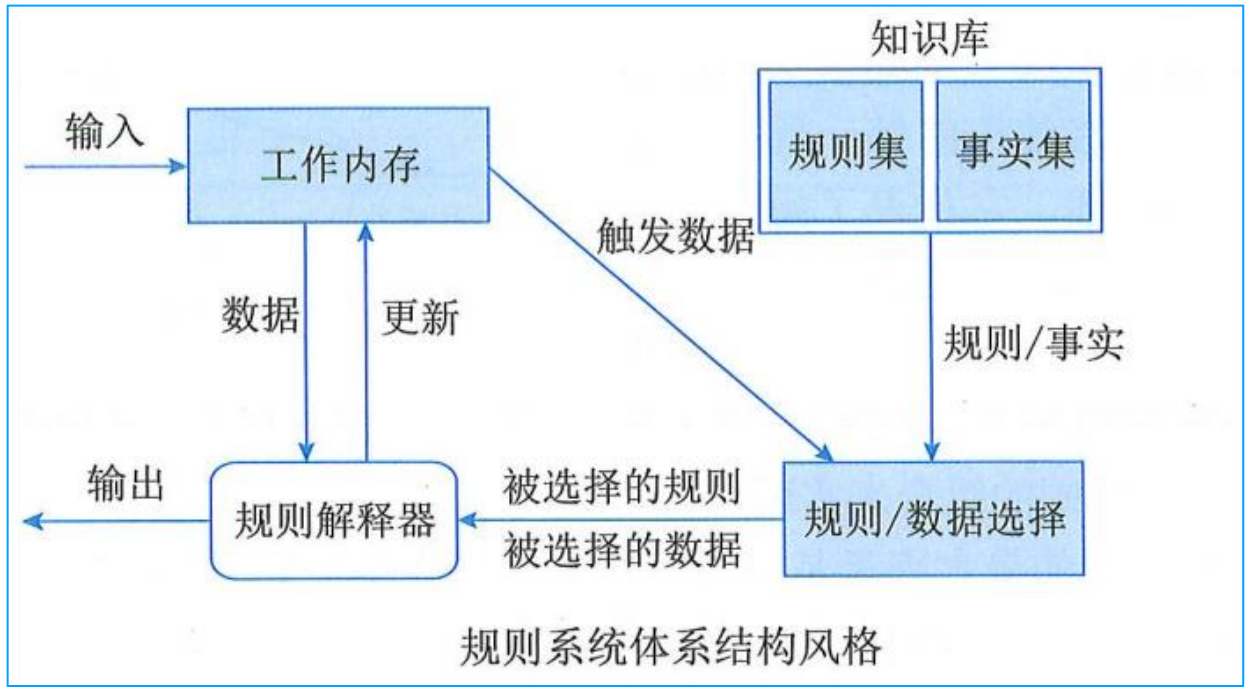
解释器通常被用来**建立一种虚拟机以弥合程序语义与硬件语义之间的差异**。其**缺点是执行效率较低**。典型的例子是**专家系统**。



虚拟机体系结构风格

(2) 规则系统体系结构风格

基于规则的系统包括**规则集、规则解释器、规则/数据选择器及工作内存**。用在人工智能领域和DSS中。在解释器的基础上**增加了经验规则**



规则系统体系结构风格

独立构件体系结构风格

5.独立构件风格主要强调系统中的每个构件都是相对独立的个体，它们之间不直接通信，以降低耦合度，提升灵活性。独立构件风格主要包括**进程通信和事件系统风格**。

（1）进程通信。构件是独立的过程，连接件是消息传递。这种风格的特点是，构件通常是命名过程，消息传递的方式可以是点对点、异步或同步方式，以及远程过程（方法）调用等。

同步通信

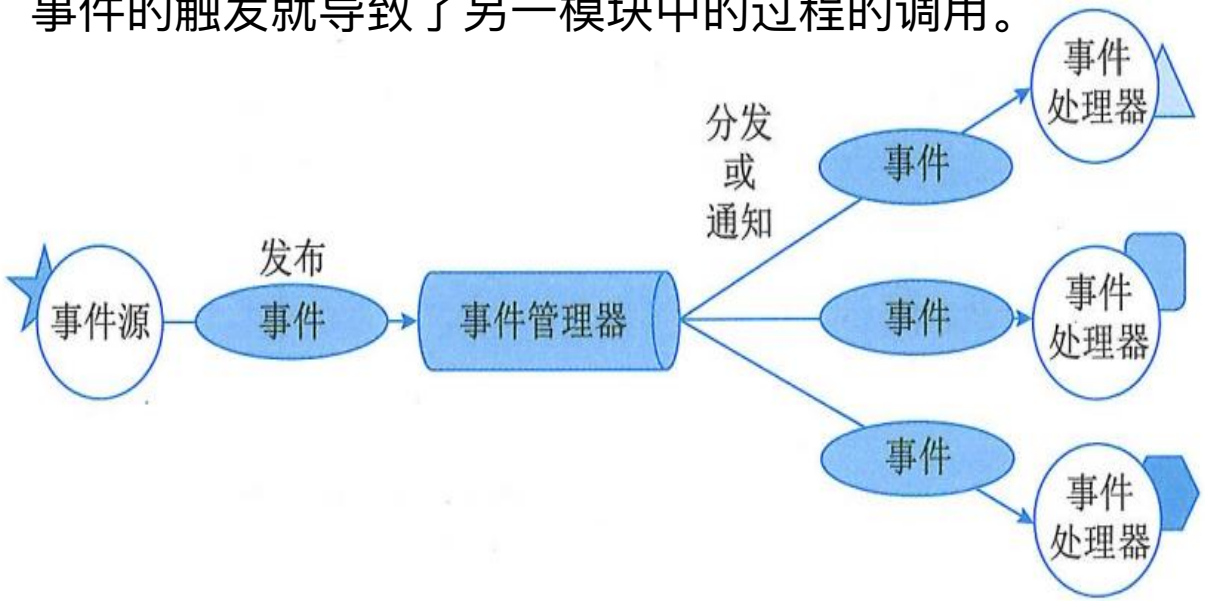
间接通信（共享内存、消息队列、Redis发布订阅模式）

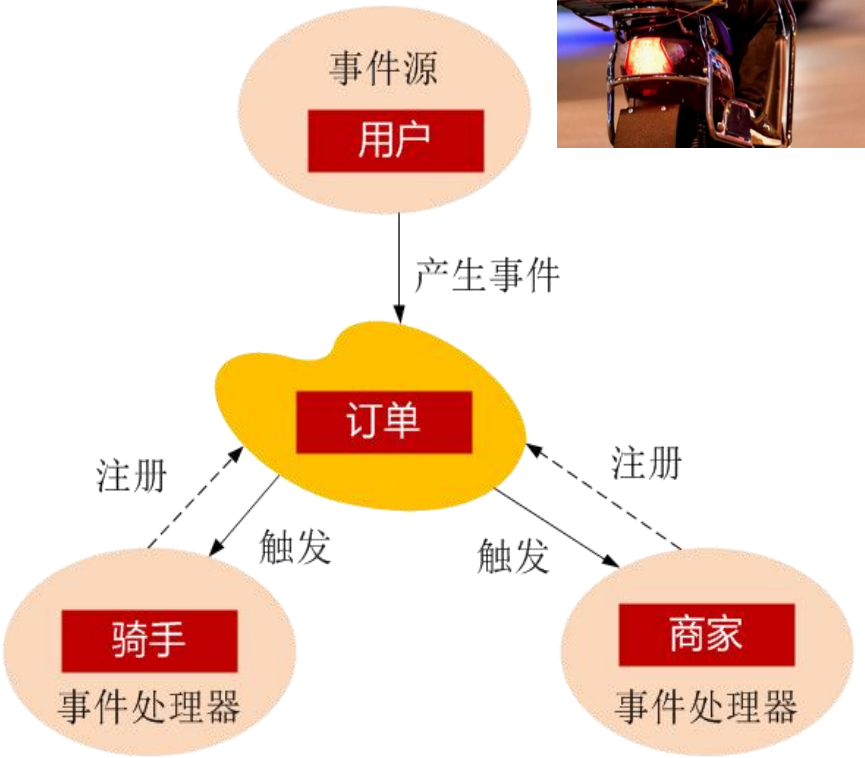
异步通信

独立构件体系结构风格

（2）**事件系统风格**。

基于事件的隐式调用风格的思想，构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中其他构件中的过程在一个或多个事件中注册，**当一个事件被触发，系统自动调用在这个事件中注册的所有过程**，这样，一个事件的触发就导致了另一模块中的过程的调用。





典型例题（202505）

一、某公司开发一个在线大模型训练平台，支持Python 代码编写、模型训练和部署，用户通过 python 编写模型代码，将代码交给系统进行模型代码的解析，最终由系统匹配相应的 计算机资源进行输出，用户不需要关心底层硬件平台。李工提出，该平台适合用解释器风格架构。

a.系统发生错误时，异常请求能够不影响用户正常工作，并发送一个消息通知系统管理员。

b.平台应保护用户隐私，防止未授权访问。

c. 系统界面能调整屏幕，适配用户提供的屏幕尺寸比例。

d.用户提交训练任务时应该在一分钟内提供硬件和资源，启动训练。

e.支持远程修改，供远程用户进行连接操作，仅提供给系统注册用户使用。

f.在训练时，应对请求5秒钟提供队列信息。

g.支持多语音界面，操作指南和文档。

h.发生故障时应在15秒钟内定位故障、解决故障。

i.系统发生故障时，要能提供操作日志。

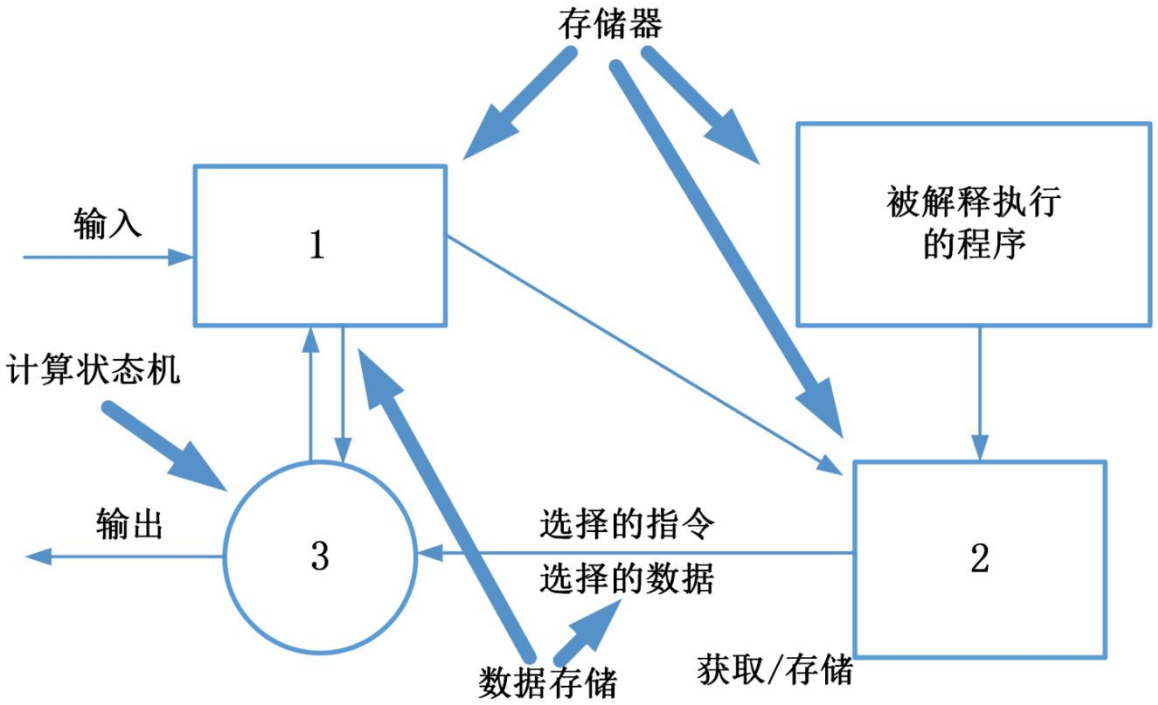
j.具备扩展能力，能够3天内完成新功能部署。

k. 数据库发生故障后，自动切换到备用数据库，保证训练不中断 。

l.符合用户习惯默认的快捷键设置。

典型例题（202505）

2、解释器风格架构的组成图填空(6分)



解释器体系结构风格

【参考答案】：

(1) 程序执行的当前状态。

(2) 解释器引擎的内部状态。

(3) 解释器引擎。

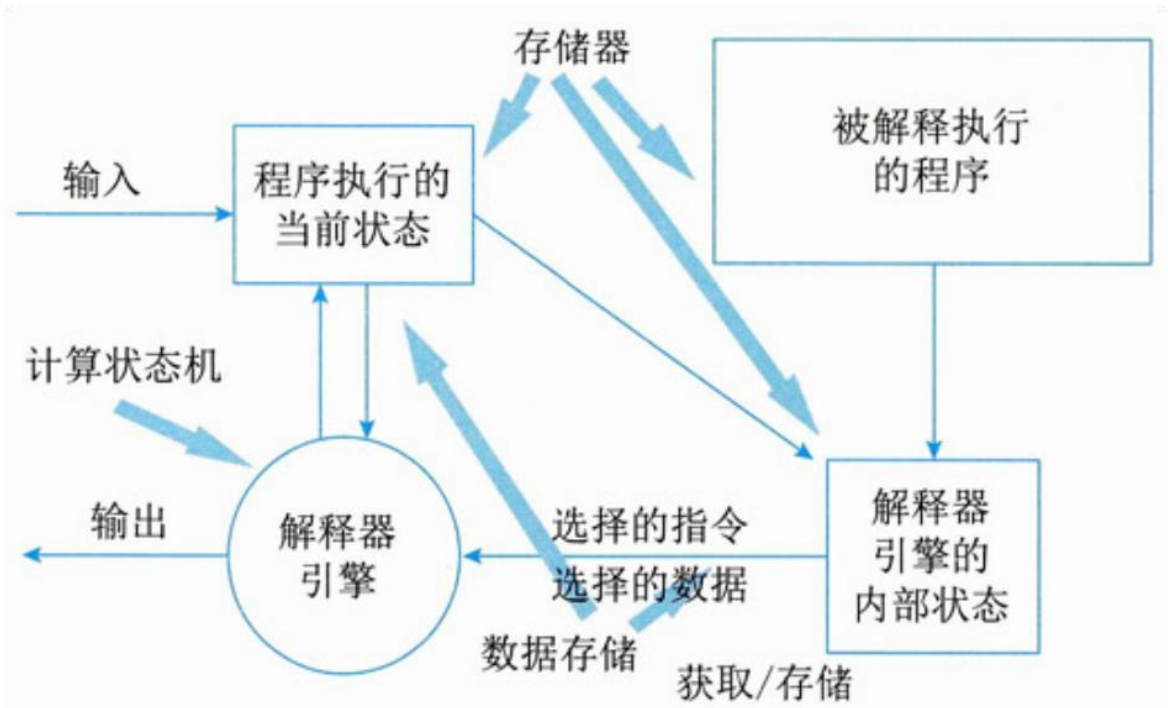


图 7-15　解释器体系结构风格

典型例题（2025 05）

3、请解释为什么该模型平台适合解释器风格(7分)。

1. 支持多语言扩展与灵活性。

解释器架构允许平台将用户编写的Python代码转换为统一的中间表示（IR），再映射到不同硬件资源（CPU/GPU/TPU）。这种设计解耦了代码解析与硬件执行，未来可轻松扩展支持其他语言（如 R、Julia），无需修改底层执行引擎。

2. 资源调度与优化。

平台可通过解释器分析代码中的计算密集型操作（如矩阵乘法），动态分配最优硬件资源。例如：

将深度学习训练任务自动调度到 GPU 集群。

对推理任务进行量化优化后部署到边缘设备。

解释器可在中间层插入优化逻辑（如自动并行化、内存管理），用户无需手动调整代码适配硬件。

3. 安全性与隔离性。

解释器架构可实现细粒度的权限控制：

沙箱化执行环境，防止恶意代码访问敏感资源。

限制资源使用上限（如内存、CPU 时间），避免单点任务拖垮集群。

用户代码不直接操作底层硬件，降低系统风险。

4. 统一抽象层。

通过解释器构建统一的硬件抽象层，屏蔽底层基础设施差异。用户只需编写标准 Python 代码，平台自动处理：

分布式训练的并行策略。

多机多卡的通信优化。

不同框架（TensorFlow/PyTorch）的兼容性。

作业

某软件公司为其新推出的字处理软件设计了一种脚本语言，专门用于开发该字处理软件的附加功能插件。为了提高该语言的编程效率，公司组织软件工具开发部门为脚本语言研制一套集成开发环境。软件工具开发部门根据字处理软件的特点，对集成开发环境进行了需求分析，总结出以下3项核心需求：

(1)集成开发环境需要提供对脚本语言的编辑、语法检查、解释、执行和调试等功能的支持，并要实现各种功能的灵活组合、配置与替换。

(2)集成开发环境需要提供一组可视化的编程界面，用户通过对界面元素拖拽和代码填充的方式就可以完成功能插件核心业务流程的编写与组织。

(3)在代码调试功能方面，集成开发环境需要实现在脚本语言编辑界面中的代码自动定位功能。具体来说，在调试过程中，编辑界面需要响应调试断点命中事件，并自动跳转到当前断点处所对应的代码。

针对上述需求，软件工具开发部门对集成开发环境的架构进行分析与设计，王工认为该集成开发环境应该采用管道–过滤器的架构风格实现，李工则认为该集成开发环境应该采用以数据存储为中心的架构风格来实现。公司组织专家对王工和李工的方案进行了评审，最终采用了李工的方案。

作业

【问题1】请用200字以内的文字解释什么是软件架构风格，并从集成开发环境与用户的交互方式、集成开发环境的扩展性、集成开发环境的数据管理三个方面说明为什么最终采用了李工的设计方案。

【问题2】在对软件系统架构进行设计时，要对架构需求进行分析，针对特定需求选择最为合适的架构风格，因此实际的软件系统通常会混合多种软件架构风格。请对核心需求进行分析，说明为了满足需求(2)和(3)，分别应采用何种架构风格，并概要说明采用相应架构风格后的架构设计过程。

作业

【问题1】

软件架构风格是指描述特定软件系统组织方式的惯用模式。组织方式描述了系统的组成构件和这些构件的组织方式，惯用模式则反映众多系统共有的结构和语义。

从集成开发环境与用户的交互方式看，用户通常采用交互式的方式对脚本语言进行编辑、解释执行与调试。在这种情况下，采用以数据存储为中心的架构风格能够很好地支持交互式数据处理，而管道–过滤器架构风格则对用户的交互式数据处理支持有限。

从集成开发环境的扩展性来看，系统核心需求要求实现各种编辑、语法检查、解释执行等多种功能的灵活组织、配置与替换。在这种情况下，采用以数据存储为中心的架构风格，以数据格式解耦各种功能之间的依赖关系，并可以灵活定义功能之间的逻辑顺序。管道–过滤器架构风格同样以数据格式解耦数据处理过程之间的依赖关系，但其在数据处理逻辑关系的灵活定义方面较差。

从集成开发环境的数据管理来看，集成开发环境需要支持脚本语言、语法树(用于检查语法错误)、可视化模型、调试信息等多种数据类型，并需要支持数据格式的转换。以数据存储为中心的架构将数据存储在统一的中心存储器中，中心存储器能够表示多种数据格式，并能够为数据格式转换提供各种支持。管道–过滤器架构风格通常只能支持有限度的数据格式，并且在数据格式转换方面的灵活性较差。

作业

【问题2】

为了满足需求(2)，应该采用解释器架构风格。具体来说，需要：①为可视化编程元素及其拖拽关系定义某种语言，并描述其语法与语义；②编写解释器对该语言进行解释；③生成对应的脚本语言程序。

为了满足需求(3)，应该采用隐式调用架构风格。具体来说，首先需要定义“断点在调试过程中命中”这一事件，并实现当断点命中后的屏幕定位函数。集成开发环境维护一个事件注册表结构，将该事件与屏幕定位函数关联起来形成注册表中的一个记录项。在调试过程中，集成开发环境负责监听各种事件，当“断点在调试过程中命中”这一事件发生时，集成开发环境查找事件注册表，找到并调用屏幕定位函数，从而实现脚本语言编辑界面与调试代码的自动定位。

软件架构质量属性

属性		作用及要点
性能		处理任务所需时间或单位时间内的处理量。
可靠性		软件系统在应用或系统错误面前，在意外或错误使用的情况下维持软件系统的功能特性的基本能力。
可用性		系统能够正常运行的时间比例，用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。
安全性		系统向合法用户提供服务并阻止非法用户的能力。
可修改性	可维护性	在错误发生后“修复”软件系统，对其他构件的负面影响最小化。
	可扩展性	使用新特性来扩展软件系统，以及使用改进版本方式替换构件并删除不需要或不必要的特性和构件。
	结构重组	重新组织软件系统的构件及构件间的关系，例如通过将构件移动到一个不同的子系统来改变它的位置。
	可移植性	软件系统适用于多种硬件平台、用户界面、操作系统、编程语言或编译器。
功能性		系统完成所期望工作的能力。
可变性		架构经扩充或变更而成为新架构的能力。
互操作性		与其他系统或自身环境相互作用。

容错性：包容错误，出错后系统能继续运行。

健壮性：出错后系统不能继续运行，但能按已定义好的方式终止执行。

软件架构质量属性

性能：指系统的**响应能力**，即要经过多长时间才能对某个事件做出响应，或者在某段时间内系统所能处理的事件的**个数**。

常见场景分析：

(1) 同时支持1000并发。
(2) 响应时间应小于1s。
(3) 显示分辨率达到4k。

性能战术

设计策略：

- 资源需求：**提高计算效率**、减少处理事件流所需的资源、减少所处理事件的数量、控制资源的使用
- 资源管理：**引入并发**、维持数据或计算的多个副本、增加可用资源
- 资源仲裁：**资源调度策略**、先进先出、固定优先级调度、动态优先级调度、静态调度

软件架构评估–质量属性–可用性

可用性：是系统能够**正常运行的时间比例**，经常用两次故障之间的时间长度或在出现故障时系统能够恢复正常的速度来表示。如故障间隔时间。

常见场景分析：
（1）主服务器故障，1分钟内切换至备用服务器。
（2）系统故障，1小时内修复。
（3）系统支持7*24小时。

可用性战术

- 错误检测：**命令/响应**、心跳线、异常处理
- 错误恢复：**表决**（通过冗余构件与表决器相联）、主动冗余、被动冗余、备件、状态再同步、检查点/回滚
- 错误预防：从服务中删除、事务（事务保证一致性）、**进程监视器**

软件架构评估–质量属性–安全性

安全性：是指系统在向合法用户提供服务的同时能够阻止非授权用户使用的企图或拒绝服务的能力。如保密性、完整性、不可抵赖性、可控性。

常见场景分析：
（1）可抵御SQL注入攻击。（2）对计算机的操作都有完整的记录。（3）用户信息数据库授权必须保证 99.9%可用。

设计策略：入侵检测、用户认证、用户授权、追踪审计。
抵抗攻击：对用户身份进行验证、授权、维护数据的机密性、完整性、限制暴露的信息、限制访问。
检测攻击：“**入侵检测**”。
从攻击中恢复：识别攻击者（**审计追踪**）、恢复（冗余）。

软件架构评估–质量属性–可修改性

可修改性：指能够**快速地以较高的性价比**对系统进行**变更**的能力。通常以某些具体的变更为基准，通过考查变更的代价来衡量可修改性。可修改性包含以下 4个方面：
(1)可维护性　(2)可扩展性　(3)结构重组　(4)可移植性

常见场景分析
（1）更改系统报表模块，必须在2人周内完成。
（2）对Web界面风格进行修改，修改必须在4人月内完成。

可修改性战术

- 局部化修改：**维持语义的一致性**、预期期望的变更、泛化该模块、限制可能的选择
- 防止连锁反应：**信息隐藏**、维持现有的接口、限制通信路径、仲裁者的使用
- 推迟绑定时间：运行时注册、配置文件、多态、构件更换、遵守已定义的协议。

系统架构评估–概念

敏感点(Sensitivity Point)和权衡点 (Tradeoff Point)。

- 敏感点是**一个或多个构件(和或构件之间的关系)的特性**。研究敏感点可使设计师明确在搞清楚如何实现质量目标时应注意什么。
- 权衡点是**影响多个质量属性的特性，是多个质量属性的敏感点**。例如，**改变加密级别可能会对安全性和性能产生非常重要的影响**。提高加密级别可以提高安全性，但可能要耗费更多的处理时间，影响系统性能。如果某个机密消息的处理有严格的时间延迟要求，则加密级别可能就会成为一个权衡点。

风险承担者 (Stakeholders)或者称为利益相关人。系统的架构涉及很多人的利益，这些人都对架构施加各种影响，以保证自己的目标能够实现。

场景

在进行架构评估时，一般首先要精确地得出具体的质量目标，并以之作为判定该架构优劣的标准。为得出这些目标而采用的机制称之为场景。

场景是从风险担者的角度对与系统的交互做的简短描述。在架构评估中，一般采用**刺激、环境和响应**三方面来对场景进行描述。

典型例题（202505）

一、某公司开发一个在线大模型训练平台，支持Python 代码编写、模型训练和部署，用户通过 python 编写模型代码，将代码交给系统进行模型代码的解析，最终由系统匹配相应的 计算机资源进行输出，用户不需要关心底层硬件平台。李工提出，该平台适合用解释器风格架构。

a.系统发生错误时，异常请求能够不影响用户正常工作，并发送一个消息通知系统管理员。

b.平台应保护用户隐私，防止未授权访问。

c. 系统界面能调整屏幕，适配用户提供的屏幕尺寸比例。

d.用户提交训练任务时应该在一分钟内提供硬件和资源，启动训练。

e.支持远程修改，供远程用户进行连接操作，仅提供给系统注册用户使用。

f.在训练时，应对请求5秒钟提供队列信息。

g.支持多语音界面，操作指南和文档。

h.发生故障时应在15秒钟内定位故障、解决故障。

i.系统发生故障时，要能提供操作日志。

j.具备扩展能力，能够3天内完成新功能部署。

k. 数据库发生故障后，自动切换到备用数据库，保证训练不中断 。

l.符合用户习惯默认的快捷键设置。

典型例题（202505）

1、(12分)列举了所有需求列表，填写对应的质量属性(第一问是12个质量属性场景，问这12个分别属于哪个质量属性)。

【参考答案】：

a–可靠性
b–安全性
c–易用性
d–性能
e–安全性
f–性能
g–易用性
h–可用性
i–可维护性
j–可修改性
k–可用性
l–易用性

作业

试题一 (共25分)

阅读以下关于软件架构设计与评估的叙述，在答题纸上回答问题1和问题2。

【 说 明 】

某电子商务公司拟升级其会员与促销管理系统，向用户提供个性化服务，提高用户的粘 性。在项目立项之初，公司领导层一致认为本次升级的主要目标是提升会员管理方式的灵活 性，由于当前用户规模不大，业务也相对简单，系统性能方面不做过多考虑。新系统除了保 持现有的四级固定会员制度外，还需要根据用户的消费金额、偏好、重复性等相关特征动态 调整商品的折扣力度，并支持在特定的活动周期内主动筛选与活动主题高度相关的用户集 合，提供个性化的打折促销活动。

在需求分析与架构设计阶段，公司提出的需求和质量属性描述如下：

(a) 管理员能够在页面上灵活设置折扣力度规则和促销活动逻辑，设置后即可生效；(b) 系统应该具备完整的安全防护措施，支持对恶意攻击行为进行检测与报警；

(c) 在正常负载情况下，系统应在0.3秒内对用户的界面操作请求进行响应；

(d) 用户名是系统唯一标识，要求以字母开头，由数字和字母组合而成，长度不少于6 个字符 。

(e) 在正常负载情况下，用户支付商品费用后在3秒内确认订单支付信息；(f) 系统主站点电力中断后，应在5秒内将请求重定向到备用站点；

(g) 系统支持横向存储扩展，要求在2人 · 天内完成所有的扩展与测试工作；(h) 系统宕机后，需要在10秒内感知错误，并自动启动热备份系统；

(i) 系统需要内置接口函数，支持开发团队进行功能调试与系统诊断；

(j) 系统需要为所有的用户操作行为进行详细记录，便于后期查阅与审计；

(k) 支持对系统的外观进行调整和配置，调整工作需要在4人 · 天内完成。

在对系统需求、质量属性描述和架构特性进行分析的基础上，系统架构师给出了两种候 选的架构设计方案，公司目前正在组织相关专家对系统架构进行评估。

作业

【问题1】(12分)

在架构评估过程中，质量属性效用树 (Utility Tree) 是对系统质量属性进行识别和优先级排序的重要工具。请将合适的质量属性名称填入图1-1中(1)、(2)空白处，并选择题干 描述的(a)～(k) 填入(3)～(6)空白处，完成该系统的效用树。

【问题2】(13分)

针对该系统的功能，李工建议采用面向对象的架构风格，将折扣力度计算和用户筛选分别封装为独立对象，通过对象调用实现对应的功能；王工则建议采用解释器 (Interpreters) 架构风格，将折扣力度计算和用户筛选条件封装为独立的规则，通过解释规则实现对应的功 能。请针对系统的主要功能，从折扣规则的可修改性、个性化折扣定义灵活性和系统性能三 个方面对这两种架构风格进行比较与分析，并指出该系统更适合采用哪种架构风格。

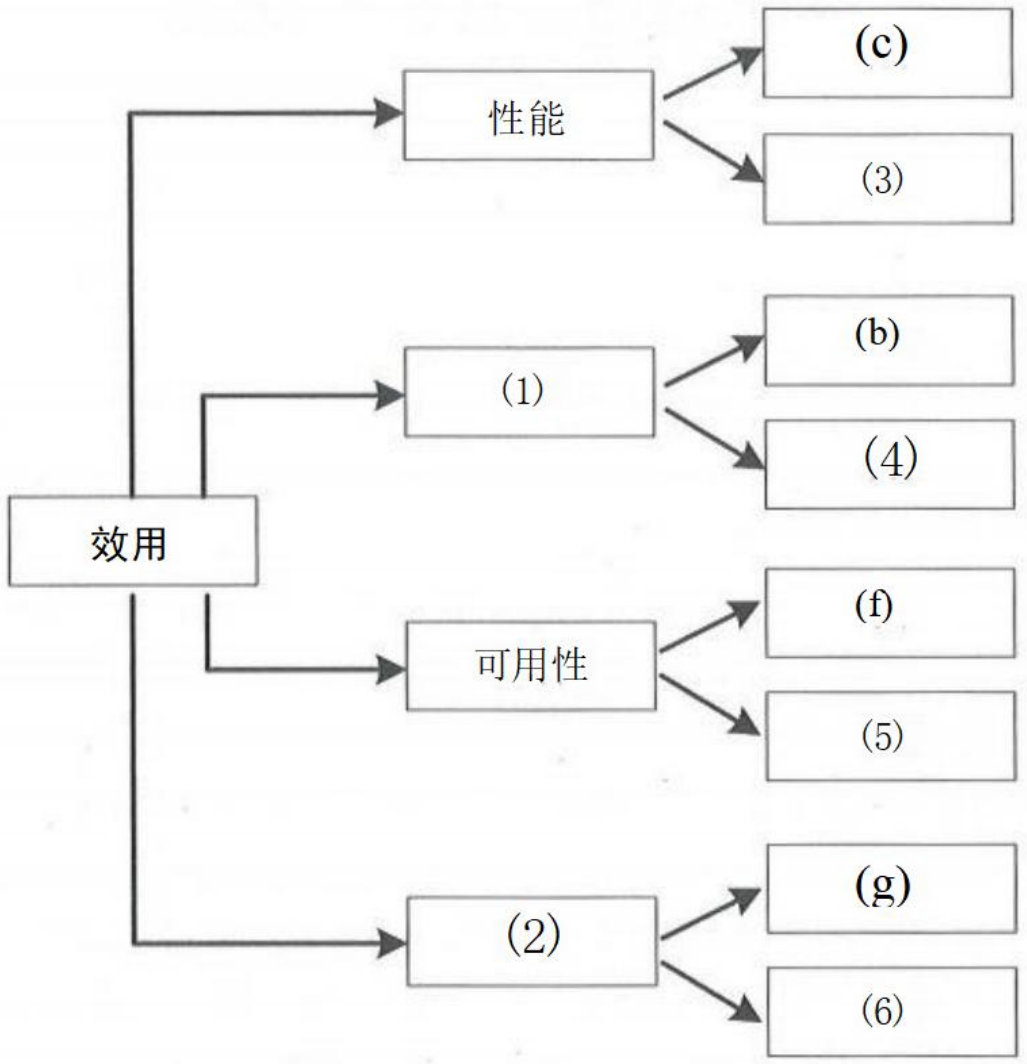


图1-1　会员与促销管理系统效用树

作业

【问题1】分析过程

在架构评估过程中，质量属性效用树是对系统质量属性进行识别和优先级排序的重要工 具。质量属性效用树主要关注性能、可用性、安全性和可修改性等四个用户最为关注的质量 属性，考生需要对题干的需求进行分析，逐一找出这四个质量属性对应的描述，然后填入空 白处即可。

经过对题干的分析，可以看出：

(a) 管理员能够在页面上灵活设置折扣力度规则和促销活动逻辑，设置后即可生效；(功能需求)
(b) 系统应该具备完整的安全防护措施，支持对恶意攻击行为进行检测与报警；(安全性)
(c) 在正常负载情况下，系统应在0.3秒内对用户的界面操作请求进行响应；(性能)
(d) 用户名是系统唯一标识，要求以字母开头，由数字和字母组合而成，长度不少于6 个字符；(功能需求)
(e) 在正常负载情况下，用户支付商品费用后在3秒内确认订单支付信息；(性能) (f) 系统主站点电力中断后，应在5秒内将请求重定向到备用站点；(可用性)
(g) 系统支持横向存储扩展，要求在2人 · 天内完成所有的扩展与测试工作；(可修改性) (h) 系统宕机后，需要在10秒内感知错误，并自动启动热备份系统；(可用性)
(i) 系统需要内置接口函数，支持开发团队进行功能调试与系统诊断；(可测试性)
(j) 系统需要为所有的用户操作行为进行详细记录，便于后期查阅与审计；(安全性)
(k) 支持对系统的外观进行调整和配置，调整工作需要在4人 · 天内完成。(可修改性)

作业

参考答案 【问题1】
(1)安全性
(2)可修改性
(3)(e)
(4)(j)
(5)(h)
(6)(k)

作业

【问题2】分析过程

本题考查考生对影响系统架构风格选型的理解与掌握。针对该系统的功能，李工建议采 用面向对象的架构风格，将折扣力度计算和用户筛选分别封装为独立对象，通过对象调用实 现对应的功能；王工则建议采用解释器 (Interpreters)　架构风格，将折扣力度计算和用户筛 选条件封装为独立的规则，通过解释规则实现对应的功能。考生需要针对系统的主要功能， 从折扣规则的可修改性、个性化折扣定义灵活性和系统性能三个方面对这两种架构风格进行 比较与分析。

(1)在折扣规则的可修改性方面，面向对象的可修改性弱于解释器，具体分析如下：

面向对象架构风格需要实现确定用户特征与折扣力度，将其封装为对象，折扣规则调整 时需要修改源代码。

解释器风格支持将用户的特征与折扣力度封装为规则形式，系统通过解释规则确定折扣 力度，调整时仅需要调整规则描述，无须修改源代码。

(2)在个性化折扣定义灵活性方面，面向对象的灵活性弱于解释器，具体分析如下：

面向对象架构风格需要将用户筛选条件封装为对象，调整筛选条件时需要修改源代码， 灵活性较低。

解释器风格以规则数据的方式描述用户筛选条件，支持规则的动态加载与处理，灵活性 较高。

(3)在系统性能方面，面向对象的性能强于解释器，具体分析如下：

面向对象架构风格将折扣力度和筛选条件内置在系统源代码中，处理速度快，性能高。

解释器风格将折扣力度和筛选条件的计算以规则文件的方式表达，系统需要动态解释规 则文件，处理速度慢，性能低。

经过上述分析与权衡，可以看出该系统更适合采用解释器架构风格。

作业

参考答案【问题2】

(1)在折扣规则的可修改性方面：

面向对象架构风格需要实现确定用户特征与折扣力度，将其封装为对象，折扣规则调整 时需要修改源代码。

解释器风格支持将用户的特征与折扣力度封装为规则形式，系统通过解释规则确定折扣 力度，调整时仅需要调整规则描述，无须修改源代码。

(2)在个性化折扣定义灵活性方面：

面向对象架构风格需要将用户筛选条件封装为对象，调整筛选条件时需要修改源代码， 灵活性较低。

解释器风格以规则数据的方式描述用户筛选条件，支持规则的动态加载与处理，灵活性 较高。

(3)在系统性能方面：

面向对象架构风格将折扣力度和筛选条件内置在系统源代码中，处理速度快，性能高。

解释器风格将折扣力度和筛选条件的计算以规则文件的方式表达，系统需要动态解释规 则文件，处理速度慢，性能低。

经过上述分析与权衡，该系统更适合采用解释器架构风格。

THANKS