### 软件架构风格 △△△

**软件架构风格是描述某一类特定应用领域中软件系统组织方式和惯用方式。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。**

面向对象架构风格的特征是将数据表示和基本操作封装在对象中。这种模式的构件是对象，对象维护自身表示的完整性，对象之间通过消息机制进行通信，对象交互时需要知道彼此的标识，通过对象之间的协作完成计算过程。

控制环路架构风格是将过程输出的指定属性维护在一个特定的参考值（设定点）。控制环路风格包括过程变量、被控变量、输入变量、操纵变量和设定点等构件，通过收集实际和理想的过程状态信息，并能调整过程变量使得实际状态趋于理想状态。

主程序-子程序架构风格中，所有的计算构件作为子程序协作工作，并由一个主程序 顺序地调用这些子程序，构件通过共享存储区交换数据。

管道-过滤器架构风格中，每个构件都有一组输入和输出，构件接受数据输入，经过内部处理，然后产生数据输出。这里的构件称为过滤器，构件之间的连接件称为数据流传输的管道。

|  |  |  |
| --- | --- | --- |
| **比较因素** | **管道-过滤器风格** | **数据仓储风格** |
| 交互方式 | 顺序结构或有限的循环结构 | 星型 |
| 数据结构 | 数据流 | 文件或模型 |
| 控制结构 | 数据流驱动 | 业务功能驱动 |
| 扩展方法 | 接口适配 | 模型适配 |

### 软件质量属性△△

常见的软件质量属性有多种，例如性能（Performance)、可用性（Availability)、可 靠性（Reliability)、健壮性（Robustness)、安全性（Security)、可修改性（Modification)、可变性(Changeability)、易用性(Usability)、可测试性(Testability)、功能性(Functionality) 和互操作性（Inter-operation)等。

这些质量属性的具体含义是：

(1) 性能是指系统的响应能力，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事件的个数。

(2) 可用性是系统能够正常运行的时间比例。

(3) 可靠性是指软件系统在应用或错误面前，在意外或错误使用的情况下维持软件系统功能特性的基本能力。

(4) 健壮性是指在处理或环境中，系统能够承受压力或变更的能力。

(5) 安全性是指系统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。

(6) 可修改性是指能够快速地以较高的性能价格比对系统进行变更的能力。

(7) 可变性是指体系结构经扩充或变更成为新体系结构的能力。

(8) 易用性是衡量用户使用一个软件产品完成指定任务的难易程度。

(9) 可测试性是指软件发现故障并隔离、定位其故障的能力特性，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。

(10) 功能性是系统所能完成所期望工作的能力。

(11) 互操作性是指系统与外界或系统与系统之间的相互作用能力。

### 质量属性的架构设计策略△

(1) 在线交易平台必须在Is内完成客户的交易请求。该要求主要对应性能，可以采用的架构设计策略有增加计算资源、改善资源需求（减少计算复杂度等)、资源管理（并发、数据复制等）和资源调度（先进先出队列、优先级队列等）。

(2) 该平台必须严格保证客户个人信息和交易信息的保密性和安全性。该要求主要对应安全性，可以采用的架构设计策略有抵御攻击（授权、认证和限制访问等)、攻击检测（入侵检测等)、从攻击中恢复（部分可用性策略）和信息审计等。

(3) 当发生故障时，该平台的平均故障恢复时间必须小于10s。该要求主要对应可用性，可以采用的架构设计策略有Ping/Edio、心跳、异常和主动冗余等。

(4) 由于企业业务发展较快，需要经常为该平台添加新功能或进行硬件升级。添加新功能或进行平台升级必须在6小时内完成。该要求主要对应可修改性，可以采用的架构设计策略有软件模块泛化、限制模块之间通信、使用中介和延迟绑定等。

### 系统架构设计中的非功能性需求

系统性能需求（Performance Requirements）：指响应时间、吞吐量、准确性、有效性、资源利用率等与系统完成任务效率相关的指标。可靠性、可用性等指标归为此类。

安全性需求（Security Requirements）：系统向合法用户提供服务并阻止非授权用户使用服务方面的系统需求。

操作性需求（Operational Requirements）:与用户操作使用系统相关的一些需求。

文化需求（Cultural Requirements）：带有文化背景因素的系统需求。

### 系统架构风险、敏感点和权衡点的定义

系统架构风险是指架构设计中潜在的、存在问题的架构决策所带来的隐患。

敏感点是指为了实现某种特定的质量属性，一个或多个系统组件所具有的特性。权衡点是指影响多个质量属性，并对多个质量属性来说都是敏感点的系统属性。

### 质量效用树（utility tree）

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。效用树主要关注**性能、可修改性、可用性和安全**4个方面

### 系统可靠性△

系统在规定的时间内及规定的环境条件下，完成规定功能的能力，就是系统无故障运行的概率。

根据国家标准《软件工程产品质量第1部分：质量模型》(GB/T16260.1—2006)的规定，系统可靠性包括：成熟性、容错性、易恢复性和可靠性的依从性4个子特性。

提高系统可靠性一般采用以下4类技术：

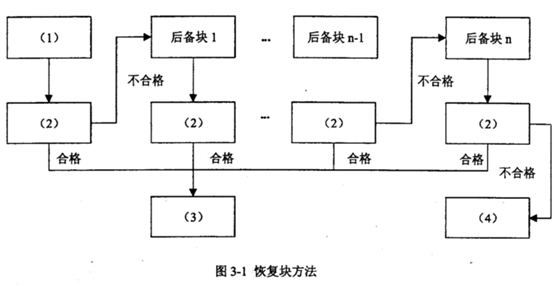
**(1) 冗余技术；**

**(2) 软件容错技术；**

**(3) 双机容错技术；**

**(4) 集群技术。**

**软件的可靠性设计主要包括了恢复块和N版本程序设计两种方法。**恢复块方法是一种反向恢复的方法，其核心原理是：对于可靠性要求高的软件，在程序运行的某时刻，将数据或程序进行备份，一旦发现主程序块有异常发生时，可将已备份的数据或程序进行恢复，保证程序的正确性。





1. 恢复块方法：

(1) 主块

(2) 验证测试

(3) 输出正确结果

(4) 异常处理

2. 恢复块方法与N版本程序设计的比较

(5) 表决

(6) 反向恢复

(7) 差

(8) 好

### 动态冗余和N版本程序设计技术

动态冗余又称为主动冗余，它是通过故障检测、故障定位及故障恢复等手段达到容错的目的。其主要方式是多重模块待机储备，当系统检测到某工作模块出现错误时，就用一个备用的模块来替代它并重新运行。各备用模块在其待机时，可与主模块一样工作，也可以不工作。前者叫热备份系统（双重系统），后者叫冷备份系统（双工系统、双份系统)。N版本程序设计是一种静态的故障屏蔽技术，其设计思想是用N个具有相同功能的程序同时执行一项计算，结果通过多数表决来选择。其中N个版本的程序必须由不同的人独立设计，使用不同的方法、设计语言、开发环境和工具来实现，目的是减少N个版本的程序在表决点上相关错误的概率。

M2采用动态冗余后的可靠度为：R=1- (1-0.99) 3 = 0.999999

李工给出的方案同时采用了串联和并联方式，其计算方法为首先计算出中间M2和M3两个并联系统的可靠度，再按照串联系统的计算方法计算出整个系统的可靠度。

R = 0.99 \* 0.999999 \* 0.999999 \* 0.99 = 0.98

### 检错技术的优缺点，常见的实现方式和处理方式。

检错技术实现的代价一般低于容错技术和冗余技术，但有一个明显的缺点，就是不能自动解决故障，出现故障后如果不进行人工干预，将最终导致软件系统不能正常运行。

检错技术常见的实现方式：最直接的一种实现方式是判断返回结果，如果返回结果 超出正常范围，则进行异常处理；计算运行时间也是一种常用技术，如果某个模块或函数运行时间超过预期时间，可以判断出现故障；还有置状态标志位等多种方法，自检的实现方式需要根据实际情况来选用。

检错技术的处理方式，大多数都采用“査出故障-停止软件运行-报警”的处理方式。 但根据故障的不同情况，也有采用不停止或部分停止软件系统运行的情况，这一般由故障是否需要实时处理来决定。

**可靠性是实时系统的关键特性之一，区分软件的错误(Error)、缺陷（Defect）、故障(Fault)和失效(Failure)概念是软件可靠性设计工作的基础。请简要说明错误、缺陷、故障和失效的定义**

软件错误：软件错误是指在软件生存期内的不希望或不可接受的人为错误，其结果是导致软件缺陷的产生。

软件缺陷：软件缺陷是存在于软件（文档、数据、程序）之中的那些不希望或不可接受的偏差。

软件故障：软件故障是指软件运行过程中出现的一种不希望或不可接受的内部状态。

软件失效：软件失效是指软件运行时产生 的一种不希望或不可接受的外部行为结果。

### 设计模式△

**创建型模式主要用于创建对象，为设计类实例化新对象提供指南。**

1. **单例（Singleton）模式**：某个类只能生成一个实例，该类提供了一个全局访问点供外部获取该实例，其拓展是有限多例模式。
2. **原型（Prototype）模式**：将一个对象作为原型，通过对其进行复制而克隆出多个和原型类似的新实例。
3. **工厂方法（FactoryMethod）模式**：定义一个用于创建产品的接口，由子类决定生产什么产品。
4. **抽象工厂（AbstractFactory）模式**：提供一个创建产品族的接口，其每个子类可以生产一系列相关的产品。
5. **建造者（Builder）模式**：将一个复杂对象分解成多个相对简单的部分，然后根据不同需要分别创建它们，最后构建成该复杂对象。

**结构型模式主要用于处理类或对象的组合，对类如何设计以形成更大的结构提供指南。**

1. **代理（Proxy）模式**：为某对象提供一种代理以控制对该对象的访问。即客户端通过代理间接地访问该对象，从而限制、增强或修改该对象的一些特性。
2. **适配器（Adapter）模式**：将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类能一起工作。
3. **桥接（Bridge）模式**：将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现的，从而降低了抽象和实现这两个可变维度的耦合度。
4. **装饰（Decorator）模式**：动态地给对象增加一些职责，即增加其额外的功能。
5. **外观（Facade）模式**：为多个复杂的子系统提供一个一致的接口，使这些子系统更加容易被访问。
6. **享元（Flyweight）模式**：运用共享技术来有效地支持大量细粒度对象的复用。
7. **组合（Composite）模式**：将对象组合成树状层次结构，使用户对单个对象和组合对象具有一致的访问性。

**行为型模式主要用于描述类或对象的交互以及职责的分配，对类之间交互以及分配责任的方式提供指南。**

1. **模板方法（Template Method）模式**：定义一个操作中的算法骨架，将算法的一些步骤延迟到子类中，使得子类在可以不改变该算法结构的情况下重定义该算法的某些特定步骤。
2. **策略（Strategy）模式**：定义了一系列算法，并将每个算法封装起来，使它们可以相互替换，且算法的改变不会影响使用算法的客户。
3. **命令（Command）模式**：将一个请求封装为一个对象，使发出请求的责任和执行请求的责任分割开。
4. **职责链（Chain of Responsibility）模式**：把请求从链中的一个对象传到下一个对象，直到请求被响应为止。通过这种方式去除对象之间的耦合。
5. **状态（State）模式**：允许一个对象在其内部状态发生改变时改变其行为能力。
6. **观察者（Observer）模式**：多个对象间存在一对多关系，当一个对象发生改变时，把这种改变通知给其他多个对象，从而影响其他对象的行为。
7. **中介者（Mediator）模式**：定义一个中介对象来简化原有对象之间的交互关系，降低系统中对象间的耦合度，使原有对象之间不必相互了解。
8. **迭代器（Iterator）模式**：提供一种方法来顺序访问聚合对象中的一系列数据，而不暴露聚合对象的内部表示。
9. **访问者（Visitor）模式**：在不改变集合元素的前提下，为一个集合中的每个元素提供多种访问方式，即每个元素有多个访问者对象访问。
10. **备忘录（Memento）模式**：在不破坏封装性的前提下，获取并保存一个对象的内部状态，以便以后恢复它。
11. **解释器（Interpreter）模式**：提供如何定义语言的文法，以及对语言句子的解释方法，即解释器。

***(1) 策略模式***

***解决方案：在具有公共接口的独立类中定义每个计算。可以利用该模式创建各种促销类，它们从同一个超类继承。每个类都有相同名称的标准接口方法，用于根据订单编号计算将要折扣的金额总数。***

***(2) 适配器模式***

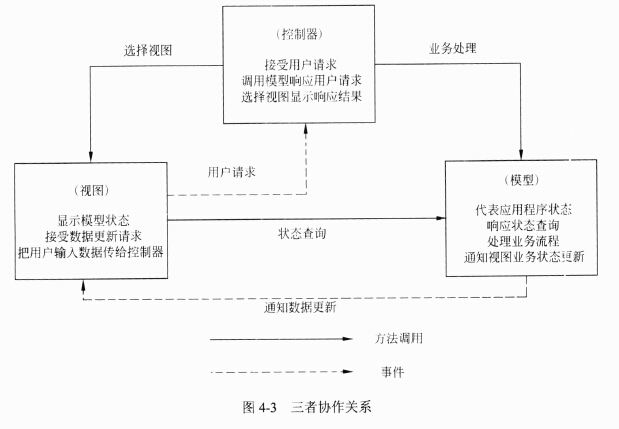
***解决方案：增加一个类作为适配器，转换类的接口到客户端类期望的另一个接口。实现一个适配器类，这个类为系统的其他部分提供了一个不变的方法供调用，为了集成不同商品供应商提供的税率计算类，编写一个适配器类的子类，包含调用购买类所需的代码。该子类将系统的调用映射到某个供应商的税率计算类。如果要更换供应商，那么只需要写一个新的适配器子类，其他保持不变。***

### 工厂设计模式的优点和应用场景，在数据访问层中的应用。

工厂模式分抽象工厂与工厂方法，题目中的场景适合采用抽象工厂设计模式。

抽象工厂设计模式提供一个接口，可以创建一系列相关或相互依赖的对象，而无需指定它们具体的类。其优点是可以非常方便的创建一系列的对象，其使用场景也是创建系列对象的情况。在本题中，可以针对Oracle、MySQL、SQLServer分别建立抽象工厂，若指定当前工厂为Oracle工厂，则创建出来的数据库连接，数据集等一系列的对象都是符合Oracle操作要求的。这样便于数据库之间的切换。

### MVC模式△



软件体系结构设计中，层次设计是一种常见的架构设计方法，使设计的系统结构清晰，便于提高复用能力和产品维护能力。

MVC是一种目前广泛流行的软件设计模式。MVC强制性地将一个应用处理流程按照模型、视图、控制的方式进行分离，形成了控制器、模型、视图三个核心模块。

(1) 控制器：接受用户的输入并调用模型和视图去完成用户的请求。一方面接受视图的输入，将其转为对模型特定方法的调用；一方面处理来自模型的事件，调用适当的视图反馈给用户。

(2) 模型：应用程序的主体部分，表示业务数据和业务逻辑，可以为多个视图提供数据。

(3) 、见图：用户看到并与之交互的界面。视图可以向模型查询业务状态，接收模型的数据更新事件，同步更新界面。

三者协作关系如图4-3所示。

使用MVC设计表现层，具有以下优点：

(1) 允许多种用户界面的扩展。在MVC模式中，视图与模型没有必然的联系，都是通过控制器发生联系，如果增加新类型的用户界面，只需修改响应的控制器和视图即可，模型无需变动；

(2) 易于维护。控制器和视图随着模型的扩展而扩展，只要保持公共接口，控制器和视图的旧版本可以继续使用；

(3) 支持功能强大的用户界面。用户界面与模型方法调用组合起来，使程序的使用更清晰，可将友好的界面发布给用户。

MVC架构风格最初是Smalltalk-80中用来构建用户界面时采用的架构设计风格。其中M代表模型（Model)，V代表视图（View)，C代表控制器（Controller)。在该风格中，模型表示待展示的对象，视图表示模型的展示，控制器负责把用户的动作转成针对模型的操作。模型通过更新视图的数据来反映自身的变化。

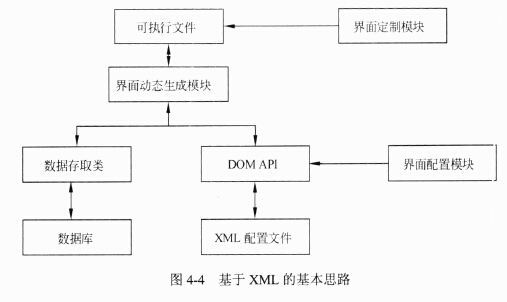
视图（View）：视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并能接收用户的输入数据，但是它并不进行任何实际的业务处理。

控制器（Controller）：控制器接受用户的输入并调用模型和视图去完成用户的需求。该部分是用户界面与Model的接口。一方面它解释来自于视图的输入，将其解释成为系统能够理解的对象，同时它也识别用户动作，并将其解释为对模型特定方法的调用；另一方面，它处理来自于模型的事件和模型逻辑执行的结果，调用适当的视图为用户提供反馈。

模型（Model）：模型是应用程序的主体部分。模型表示业务数据和业务逻辑。一个模型能为多个视图提供数据。

**请从设计模式的角度，简要说明设计方案采用XML作为GUI描述语言的机制。**

从设计模式的角度来说，整个XML表现层解柝的机制是一种策略模式。在调用显示GUI时，不是直接调用特定的表现技术的API,而是装载GUI对应的XML配置文件，然后根据特定的表现技术的解析器解析XML，得到GUI视图实例对象。这样，对于GUI开发人员来说，GUI视图只需要维护一套XML文件即可。



界面定制是对用户界面的动态修改过程，在软件运行过程中，用户可按照需求和使用习惯，对界面元素的属性进行修改。软件运行结束后，界面定制的结果被保存。

界面动态生成是系统通过DOMAPI读取XML配置文件的表示层信息，通过数据存取类读取数据库中的数据层信息，运行时由界面元素动态生成界面。界面配置和定制模块在软件运行前后修改配置文件、更改界面内容。

界面配置是对用户界面的静态定义，通过读取配置文件的初始值对界面配置。由界面配置对软件功能进行裁剪、重组和扩充，以实现特殊需求。

基于XML的界面管理技术实现的管理信息系统实现了用户界面描述信息与功能实现代码的分离，可针对不同用户需求进行界面配置和定制，能适应一定程度的数据结构改动。只需要对XML文件稍加修改，即可实现系统的移植。

### 数据库分区

数据分区是一种物理数据库的设计技术，它的目的是为了在特定的SQL操作中减少数据读写的总量以缩减响应时间。

分区并不是生成新的数据表，而是将表的数据均衡分摊到不同的硬盘，系统或是不同服务器存储介子中，实际上还是一张表。另外，分区可以做到将表的数据均衡到不同的地方，提高数据检索的效率，降低数据库的频繁IO压力值，分区的优点如下：

1、相对于单个文件系统或是硬盘，分区可以存储更多的数据；

2、数据管理比较方便，比如要清理或废弃某年的数据，就可以直接删除该日期的分区数据即可；

3、精准定位分区查询数据，不需要全表扫描查询，大大提高数据检索效率；

4、可跨多个分区磁盘查询，来提高查询的吞吐量；

5、在涉及聚合函数查询时，可以很容易进行数据的合并；

采用**水平分区**机制可根据用户标识将用户数据进行水平分割，用户操作时先将请求分发到不同数据库分区，再进行具体数据库操作，以提高数据库访问效率。

**垂直分区**方式一般来说是通过对表的垂直划分来减少目标表的宽度，使某些特定的列被划分到特定的分区，每个分区都包含了其中的列所对应的行。

### 主从复制机制的好处。

①避免数据库单点故障：主服务器实时、异步复制数据到从服务器，当主数据库宕机时，可在从数据库中选择一个升级为主服务器，从而防止数据库单点故障。

②提高査询效率：根据系统数据库访问特点，可以使用主数据库进行数据的插入、删除及更新等写操作，而从数据库则专门用来进行数据査询操作，从而将査询操作分担到不同的从服务器以提高数据库访问效率。

### 引入Memcached后系统访问数据库的基本过程为：

系统需要读取后台数据时，先检査数据是否存在于Memcached中，若存在则直接从Memcached中读取，或不存在则从数据库中读取并保存在Memcached中；当系统数据库中数据发生更新时，需要将更新后的内容同步到Memcached缓存实例中。

与MySQL查询缓存相比，**使用Memcached机制存在以下优势**：

(1) 缓存架构：数据库查询缓存通常每个数据库只有一个实例，因此存储内容受数据库服务器可用内存限制，可缓存数据有限；而Memcached可采用高速分布式缓存服务器结构，不受数据库服务器约束，可扩展性更好。

(2) 缓存有效性：数据库查询缓存只要在发生写操作时就会失效，即使更新的是数据库中的其他行；而Memcached可通过键值将数据进行散列缓存，有效降低缓存的更新频率，从而提高缓存的有效性。

(3) 缓存数据类型：数据库杏询缓存只能缓存数据库行，对社交网站好友动态显示等典型业务所需要的组合数据缓存缺乏有效支持，而Memcached理论上可缓存任何内容，因此可以将分散在数据库中的关系或者列表组合后进行缓存，以提高缓存数据的针对性和效率。

### 关系型数据库管理系统和文件系统

文件系统具有以下特点：

•针对特定应用系统设计，难度较小；

•数据冗余较大，可能在多个文件中复制相同的数据属性；

•以应用系统为中心组织、管理数据；

•符合特定应用系统要求的文件数据很难在不同的应用系统之间共享。

关系型数据库具有以下特点。

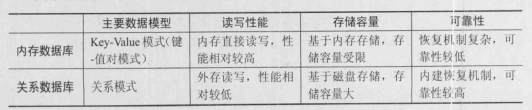
•数据结构需要符合关系模式，设计难度较大；

•遵守数据库范式，数据冗余较少；

•以数据库为中心组织、管理数据；

•数据独立于应用系统，很容易在不同的应用系统之间共享数据。

### 内存数据库和关系数据库



### SQL语句设计时，影响查询效率的设计原则是：

•查询时尽量不要返回不需要的行、列；

•需要进行多表连接査询时，尽量使用连接查询，避免使用子查询结构；

•尽量避免采用NOTIN、NOTEXIST、LIKE等使用全表查询的操作；

•尽量避免使用DISTINCT关键字。

### 增加数据访问层的原因：

（1）由于涉及到多种异构数据库平台，数据访问复杂性增加，不宜与业务逻辑混合在一起

（2）数据管理变复杂之后，需要使用的代码量增加，分单独层次有利于让逻辑更清晰。

（3）业务逻辑应以相同的方式应对异构的数据库，此时需要单独的数据访问层屏蔽差异性。

### 什么是数据持久层，使用数据持久层能够为项目开发带来哪些好处？

数据持久层是根据分层思想，通过建立逻辑数据操作接口，采取一定的对象/关系映射策略，隐藏数据库访问代码细节，向业务开发人员提供透明的对象持久化操作机制。能够为项目开发带来的好处：

(1)分离业务逻辑层和数据层，降低两者之间的耦合；

(2) 通过对象/关系映射向业务逻辑提供面向对象的数据访问；

(3) 简化数据层访问，隐藏数据库链接、数据读写命令和事务管理细节。

Hibernate和iBatis是轻量级JavaEE框架中两种数据持久层技术，两者都是优秀的开源项目。iBatis相对简单易学而且更灵活，但开发工作量较大，数据之间是关联关系；Hibernate框架相对复杂，所生成的持久化对象能够表达面向对象中的继承和聚合等关系，开发工作量较小，Hibernate使用更广泛更成熟，能够适应目前所有主流的关系型数据库。

项目组应该采用Hibernate框架的原因：

(1) Hibernate支持多种不同类型数据库，满足项目组数据库移植需求；

(2) Hibernate相对于iBatis减少了SQL语句开发的工作量；

(3) iBatis生成的P0是扁平化的，无法像Hibernate—样支持对象的继承和聚合等立体化关系。

### 数据库程序在线访问方式和ORM方式的优缺点

数据库程序在线访问方式优点：

1、性能比直接SQL好

2、可以处理复杂查询语句

数据库程序在线访问方式缺点：

1、要求程序员懂SQL语句

2、修改与维护相对困难

ORM优点：

1、使用ORM可以大大降低学习和开发成本。

2、程序员不用再写SQL来进行数据库操作。

3、减少程序的代码量。

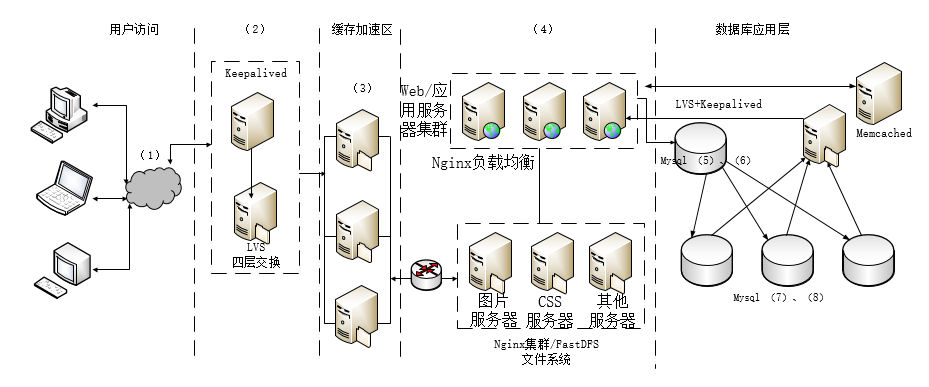
4、降低由于SQL代码质量差而带来的影响。

ORM缺点

1、不太容易处理复杂查询语句。

2、性能较直接用SQL差。

本题中的场景之所以选择ORM，主要考虑的是程序缺数据库开发经验，这样SQL语句质量有很大风险。同时学习成本很高。此外应用简单，不也担心ORM对性能的影响。



(a) Web 应用层

(b) 界面层

(c) 负载均衡层

(d) CDN 内容分发

(e) 主数据库

(f) 缓存服务器集群

(g) 从数据库

(h) 写操作

(i) 读操作

(j) 文件服务器集群

（1）（d）

（2）（c）

（3）（f）

（4）（a）

（5）（6）（e）（h）

（7）（8）（g）（i）

### 引入主从复制机制好处

1、提升性能

交易平台要求高并发，主从复制方式一主多从，不同的用户请求可以从不同的从数据库读取数据，提高并发度。

2、可扩展性更优

如果采用单台数据库服务器，则访问量持续增加时，数据库瓶颈暴露，且无法迅速解决问题。而主从结构可以快速增加从服务器数量，以满足需求。

3、提升可用性

一主多从，一台从服务器出现故障不影响整个系统正常工作。

4、相当于负载均衡

一主多从分担任务，相当于负载均衡

5、提升数据安全性

系统中的数据冗余存放多份，不会因为某台机器硬件故障而导致数据丢失。

|  |  |  |
| --- | --- | --- |
|  | Memcache | Redis |
| 数据类型 | 简单key/value结构 | key/value，list，set，hash，sorted |
| 持久性 | 不支持 | 支持 |
| 分布式存储 | 不支持 | 多种方式、主从、Sentinel、Cluster |
| 多线程支持 | 支持 | 不支持 |
| 内存管理 | 有 | 无 |
| 事务支持 | 不支持 | 有限支持 |

（1）Memcache没有持久化功能，所以掉电数据会全部丢失，而且无法直接恢复，这存在可靠性问题。

（2）Memcache不支持事务，所以操作过程中可能产生数据的不一致性。

同步方案：读取数据时，先读取Redis中的数据，如果Redis没有，则从原数据库中读取，并同步更新Redis中的数据，写回时，写入到原数据库中，并同步更新至Redis中。

### Redis分布式存储的2种常见方案：

1.主从

2.Cluster

### Redis集群切片的常见方式：

1.客户端切片，即在客户端就通过key的hash值对应到不同的服务器。

2.对数据根据key散列到不同的slot上，不同的slot对应到不同服务器。

### 数据库建模中的反规范化技术，优缺点。

规范化设计后，数据库设计者希望牺牲部分规范化来提髙性能，这种从规范化设计的回退方法叫做反规范化技术。反规范化设计允许保留或者新增一些冗余数据，从而减少数据查询中表连接的数目或简化计算过程，提高数据访问效率。

采用反规范化技术的益处：能够减少数据库查询时SQL连接的数目，从而减少磁盘I/O数据量，提高查询效率。

可能带来的问题：数据的重复存储，浪费了磁盘空间；为了保障数据的一致性，增加了数据维护的复杂性。

### 常见的反规范化技术有哪些。

常见的反规范化技术包括：

(1) 增加冗余列：在多个表中保留相同的列，通过增加数据冗余减少或避免查询时的连接操作；

(2) 增加派生列：在表中增加可以由本表或其他表中数据计算生成的列，减少查询时的连接操作并避免计算或使用集合函数；

(3) 表水平分割：根据一列或多列数据的值，把数据放到多个独立的表中，主要用于表数据规模很大、表中数据相对独立或数据需要存放到多个介质上时使用；

(4) 表垂直分割：对表进行分割，将主键与部分列放到一个表中，主键与其他列放到另一个表中，在查询时减少I/O次数。

**从数据获取方式、数据交互方式和数据访问的上下文无关性三个方面对王工和李工的方案进行比较，并用500字以内的文字说明为什么没有采用王工的方案。**

（公司的架构师王工提出采用面向服务的系统架构，首先将各种待集成的第三方软件和异构数据源统一进行包装，然后将数据访问功能以标准Web服务接口的形式对外暴露，从而支持系统进行数据的分析与处理，前端则采用CSS等技术实现浏览器数据的渲染与展示。架构师李工则认为该系统的核心在于数据的定位、汇聚与转换，更适合采用面向资源的架构，即首先为每种数据元素确定地址，然后将各种数据格式统一转换为JSON格式，通过对JSON数据的组合支 持数据的分析与处理任务，处理结果经过渲染后在浏览器的环境中进行展示。在架构评估会议上，专家对这两种方案进行综合评价，最终采用了李工的方案。）

从数据获取方式看，王工的方案需要将现有的多个系统和异构的数据源包装为服务，采用Web服务暴露数据接口，客户端需要通过服务调用获取数据，这种方法工作量大，复杂度较高。李工的方案则绕开了复杂的功能封装，只需要明确数据的位置与标识， 通过特定的网络协议直接使用标识定位并获取数据，与王工的方案相比工作量小，实现简单。

从数据交互方式看，王工的方案采用远程过程调用和异步XML消息等模式实现数据交互，这种方式适合于系统之间功能调用时进行的少量数据传输，而在进行单纯的数据访问时效率不高，稳定性也较差。李工的方案则以数据资源为核心，在对数据资源进行标识的基础上，通过标识符直接对数据资源进行访问与交互，实现简单且效率较高。

从数据访问的上下文无关性看，王工的方案中数据访问是与上下文有关的，具体表现在每次客户端进行数据请求都需要附加唯一的请求标识，并且服务端需要区分不同的客户端请求，效率较低。李工的方案中数据访问是与上下文无关的，客户端通过全局唯一的统一资源标识符（URI)请求对应的数据资源，服务端不需要区分不同的客户端请求。

### 用例

**用例建模用来描述待开发系统的功能需求**，主要元素是用例和参与者。请根据题目所述需求，说明教学服务系统中有哪些参与者。

参与者是指系统以外的，需要使用系统或与系统交互的事物，包括：人或组织、设备、外部系统等。在本题中，较为容易识别的参与者包括：学生、教师、管理员，比较隐晦的参与者包括：时间、打印机。

**用例之间的关系包括：包含、扩展、泛化。**

类图主要用来描述系统的静态结构，是组件图和配置图的基础。

**类之间的关系有哪几种类型？**

类之间的关系包括：关联、聚合、组合、依赖、泛化、实现（可写可不写，因为实现是接口与类之间的关系，而接口是一种特殊的类）。

类University与类Student之间的关系是：聚合关系。

类University与类Department之间的关系是：组合关系。

类Student与类Course之间的关系是：关联关系。

依赖关系：一个事物发生变化影响另一个事物。

泛化关系：特殊/一般关系。

关联关系：描述了一组链，链是对象之间的连接。

聚合关系：整体与部分生命周期不同。

组合关系：整体与部分生命周期相同。

实现关系：接口与类之间的关系。

**在面向对象方法中通常采用用例（Use Case）来捕获系统的功能需求。用例可以按照不同的层次来进行划分，其中的Essential Use Cases 和 Real Use Cases 有哪些区别?** Essential Use Cases 可翻译为抽象用例，而Real Use Cases可翻译为基础用例。他们是区别在于：基础用例是实实在在与用户需求有对应关系的用例，是从用户需求获取的渠道得到的，而抽象用例是从基础用例中抽取的用例的公共部分，是为了避免重复工作，优化结构而提出的用例。

### 流程图与数据流图的含义及其区别

数据流图作为一种图形化工具，用来说明业务处理过程、系统边界内所包含的功能和系统中的数据流。

流程图以图形化的方式展示应用程序从数据输入开始到获得输出为止的逻辑过程，描述处理过程的控制流。

两者的区别主要包括：

(1) 数据流图中的处理过程可并行；流程图在某个时间点只能处于一个处理过程。

(2) 数据流图展现系统的数据流；流程图展现系统的控制流。

(3) 数据流图展现全局的处理过程，过程之间遵循不同的计时标准；流程图中处理过程遵循一致的计时标准。

(4) 数据流图适用于系统分析中的逻辑建模阶段；流程图适用于系统设计中的物理建模阶段。

数据流图中常见的错误分为两种类型：一类是语法错误，包括外部实体之间、数据存储之间或外部实体与数据存储之间不经过加工而存在直接数据流；另一类是逻辑错误，包括数据黑洞（只有输入没有产生输出）、灰洞（输入不足以产生输出）和无输入。

**髙质量数据流图设计时应考虑的三个原则**：

(1) 复杂性最小化原则。DFD分层结构就是把信息划分为小的且相对独立的一大批子集例子，这样就可以单独考查每一个DFD。如果要了解某个过程更加详细的信息，可以跳转到该过程的下一层；如果要知道一个DFD如何与其他DFD相关联，可以跳转到上一层的DFD进行考査。

(2) 接口最小化原则。接口最小化是复杂性最小化的一种具体规则，在设计模型时，应使得模型中各个元素之间的接口数或连接数最小化。

(3) 数据流一致性原则。一个过程和它的过程分解在数据流内容中是否有差别？是否存在有数据流出但没有相应的数据流入的加工？是否存在有数据流入但没有相应的数据流出的加工？

### 状态图和活动图的含义及其区别。

状态图主要用于描述一个对象在其生存期间的动态行为，表现一个对象所经历的状态序列，引起状态转移的事件（event），以及因状态转移而伴随的动作（action）。

活动图可以用于描述系统的工作流程和并发行为。活动图其实可看作状态图的特殊形式，活动图中一个活动结束后将立即进入下一个活动（在状态图中状态的转移可能需要事件的触发）。

两者最大的区别是：状态图侧重于描述行为的结果，而活动图侧重描述行为的动作。其次活动图可描述并发行为，而状态图不能。

### 数据流图(Data Flow Diagram)的基本元素及其作用。

四种元素：

(1) External Agent(实体/外部代理)：定义位于项目范围之外，但与正在被研发的系统有交互关系的人、部门、外部系统或组织。

(2) Process(加工/处理)：在输入数据流或条件上执行，或者对输入数据流或条件做出响应的工作。

(3) Data Store(数据存储)：静止的数据，表示系统中需要保存的数据。

(4) Data Flow(数据流)：运动中的数据，表示到一个过程的数据输入，或者来自一个过程的数据输出。

数据流图中的错误包括两类：第一类是逻辑错误，加工节点输入输出不平衡，包括黑洞、灰洞和无输入三种类型；第二类是语法错误，比如数据存储不完整、在数据存储与外部代理之间或者各自之间没有经过加工之间发生数据流等。

### 软件系统数据架构建模

数据架构定义了信息系统中文件和数据库的分布结构。数据架构建模是以数据为中心，建模业务数据类型和结构，以及设计满足应用需求的数据库系统。传统以主机为中心的信息系统开发中，利用单个的数据库系统实现数据的集中式存储，物理上所有的数据位于同一个位置，构成的是一种集中式的数据架构；现代基于网络的分布式系统开发中，很少有组织会将其全部的数据存储在单个的数据库中，通常需要多个数据库系统组成，数据在这些数据库系统之间可以传送，由多个不同的数据库管理系统控制，构成的是一种分布式的数据架构。

集中式数据架构中，一个或多个局域网中的客户共享一个单独计算机系统中的单个数据库。系统提供数据处理能力，用户可以在同样的站点上操作，也可以在地理位置隔开的其他站点上通过远程终端来操作。系统及其数据管理被某个或中心站点集中控制。单个数据库服务器结构的主要优点就是简单、易维护开发及运行成本低；但由于所有的客户直接请求服务器，容易发生性能瓶颈，如果服务失败，单个服务器不能提供备份和恢复，所有依赖的应用程序都将不能工作。

分布式数据架构中，使用多个计算机系统以及用户能够访问远程系统的数据，数据可以在多个不同的数据库中进行传送，由不同的数据库管理系统软件进行管理，运行在多种不同的计算机上，支持多种不同的操作系统。这些机器位于（或分布在）不同的地理位置并通过多种通信网络连接在一起。企业数据可以分布在不同的计算机上，一个应用程序可以操作位于不同地理位置的机器上的数据。多个数据库服务器结构的主要优点就是系统的容错能力和对广域网容量的需求有所降低，可以采用多种策略提升整个系统的服务质量；由于多个数据库系统分布在不同的网络节点上，位于不同位置的数据之间需要同步和协作，系统结构复杂、运行成本高并且维护困难。

**如何建立CRSS的数据库系统；对于数据的读取、添加、更改和删除操作分别如何实现。**

读写分离架构利用了数据库的复制技术，将数据的读和写分布在不同的处理节点上，从而达到提高可用性和扩展性的目的。

CRSS的分布式数据库系统需要由多个局部数据库系统、多个热备份数据库系统和多个数据缓存组成。局部数据库负责数据的写入，多个热备份数据库系统用以解决单点故障的问题，数据缓存负责为应用提供所读取的数据。

(1) 读取数据：应用访问缓存，如果命中则返回，否则从局部数据库系统中读取数据并将数据加载到缓存后返回。

(2) 添加数据：采用延迟加载策略，应用将数据直接写入局部数据库。

(3) 更改数据：应用更改局部数据库中的数据，将缓存中的数据标记为失效。

(4) 删除数据：应用删除局部数据库中的数据，将缓存中的数据标记为失效。

**说明在集中式和分布式数据架构下，可以釆用哪些方法提升系统的可扩展性。**

集中式数据架构通过向上扩展（Scale Up)提升系统的可扩展性。具体的实现方式包括硬件扩容（增加CPU数ft、内存容童、磁盘数量）和硬件升级（更换为高端主机或髙速磁盘等)。

分布式数据架构通过向外扩展（Scale Out)提升系统的可扩展性。具体的实现方式包括数据复制、数据垂直切分或/和水平切分、缓存和全文搜索。

### 信息工程方法中的“实体（entity）” 与面向对象方法中的“类（class）”之间有哪些不同之处？

实体用于数据建模，而类用于面向对象建模。实体只有属性，而类有属性和操作。

### ESB的主要功能

ESB 的主要功能包括：

(1) 应用程序的位置透明性

(2) 传输协议转换

(3) 消息格式转换

(4) 消息路由

(5) 消息增强

(6) 安全支持

(7) 监控和管理

采用ESB作为集成框架，能够实现灵活的部署结构，包括CS结构、P2P结构等。采用ESB作为集成框架，待集成系统只需要和总线进行联系，彼此之间不需要互相通信，这样就大大降低了系统的耦合程度。

采用ESB作为集成框架，在加入新的待集成系统时，只需要采用插件的方式实现传输协议和数据格式的适配即可，系统的可扩展性较强。

**需求：**

**(1) 目前使用的系统设计与开发工具的运行平台和开发语言差异较大，集成框架应无缝集成各个工具的功能；**

**(2) 目前使用的系统设计与开发工具所支持的通信协议和数据格式各不相同，集成框架应实现工具之间的灵活通信和数据格式转换：**

**(3) 集成框架需要根据实际的开发流程灵活、动态地定义系统工具之间的协作关系：**

**(4) 集成框架应能集成一些常用的第三方实用工具，如即时通信、邮件系统等。**

根据4个需求，说明每个需求应该采用何种具体的集成方式或架构风格最为合适。

对于需求（1)来说，由于需要共享系统的功能，并且系统的运行平台与语言差异较大，应该采用面向服务的方式进行功能集成，可以将工具的功能包装为服务，实现跨语言与跨平台访问。

对于需求（2)来说，工具所支持的通信协议和数据格式各不相同，并需要卖现工具之间的灵活通信协议和数据格式交换，因此应该基于消息总线，以协议及数据适配器的方式实现灵活的通信协议和数据格式转换。

对于需求（3)来说，集成框架需要根据实际的软件系统开发流程，灵活、动态地定义系统设计与开发工具之间的协作关系，因此应该引入工作流定义语言及其引擎来动态描述工具之间的协作关系。

对于需求（4)来说，应该采用界面集成的方法对第三方工具进行集成，绕过工具内部的复杂处理逻辑。

**在实现工具之间数据格式的灵活转换时，通常采用的设计模式是什么，并对实现过程进行简要描述。**

在实现工具之间数据格式的灵活转换时，通常采用适配器设计模式。即应首先定义一个统一的数据转换接口类，然后针对不同的数据格式转换需求定义对应的实际转换类，实际转换类需要继承数据转换接口类，并实现接口转换类定义的接口。

在系统可靠性中，可靠度和失效率是两个非常关键的指标，请分别解释其含义。

可靠性是指产品在规定的条件下和规定的时间内完成规定功能的能力。考虑到软件本身的复杂性，软件可靠性的定义是：在规定的条件下，在规定的时间内，软件不引起系统失效的概率。

可靠度就是系统在规定的条件下、规定的时间内不发生失效的概率。

失效率又称风险函数，也可以称为条件失效强度，是指运行至此刻系统未出现失效的情况下，单位时间系统出现失效的概率。

### PHP与JAVA比较

1、PHP只能实现简单的分布式两层或三层的架构，而JAVA在这方面就比较强大，可以实现多层的网络架构。数据库层(持久化层)、应用(业务)逻辑层、表示逻辑层彼此分开，而且现在不同的层都已经有一些成熟的开发框架的支持。

2、PHP是面向过程的语言，Java是面向对象的，面向过程语言开发的程序只要业务流程发生变化，修改工作量很大，所以可修改性差，同时可复用性也差。

3、PHP语言在可靠性方面比J2EE平台差，J2EE平台有大量增强可靠性的成熟解决方案，而PHP只是一种简单的脚本语言，在可靠性方面缺乏成熟解决方案。

4、PHP对于不同的数据库采用不同的数据库访问接口，而Java通过JDBC来访问数据库，通过不同的数据库厂商提供的数据库驱动方便地访问数据库，访问数据库的接口比较统一。所以原架构在数据库连接方面修改起来工作量也是很大的。

5、PHP适合于小型项目，所以本项目中以前采用PHP是合适的，但目前大量功能需要增加，PHP在稳定性方面也达不到要求。

6、PHP比Java的可维护性差。

7、PHP比Java的扩展性差。

8、PHP比Java的安全性差。

### 应用服务器的概念，并重点说明应用服务器如何来保障系统在大负荷和长时间运行下的稳定性以及可扩展性。

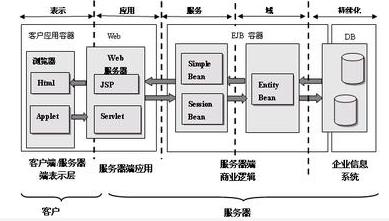
应用服务器是指通过各种协议把商业逻辑曝露给客户端的程序。

1、若系统负荷很大，可以布署多台应用服务，多台应用服务器分担任务，以达到性能要求。

2、应用服务器可以通过灵活的增加服务器完成扩展，所以可扩展性很好。

3、应用服务器可长时间稳定运行。因为当一台应用服务器出现故障时，可以将当前运行的事务转移至正常应用服务器上完成执行，不影响业务正常执行，从而保障高可靠性与稳定性。

### J2EE的N层体系结构



### EJB构件中的Bean (构件)分为哪三种类型

EJB中的Bean分三种类型：Session Bean、Entity Beans 和 Message-Driven Bean。

Session Bean的职责是：维护一个短暂的会话

Entity Beans 的职责是：维护一行持久稳固的数据

Message-Driven Bean的职责是：异步接受消息

### 什么是面向服务架构（SOA）以及 ESB在 SOA 中的作用与特点。

SOA是一个组件模型，它将应用程序的不同功能单元（称为服务）**通过这些服务之间**定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以一种统一和通用方式进行交互。

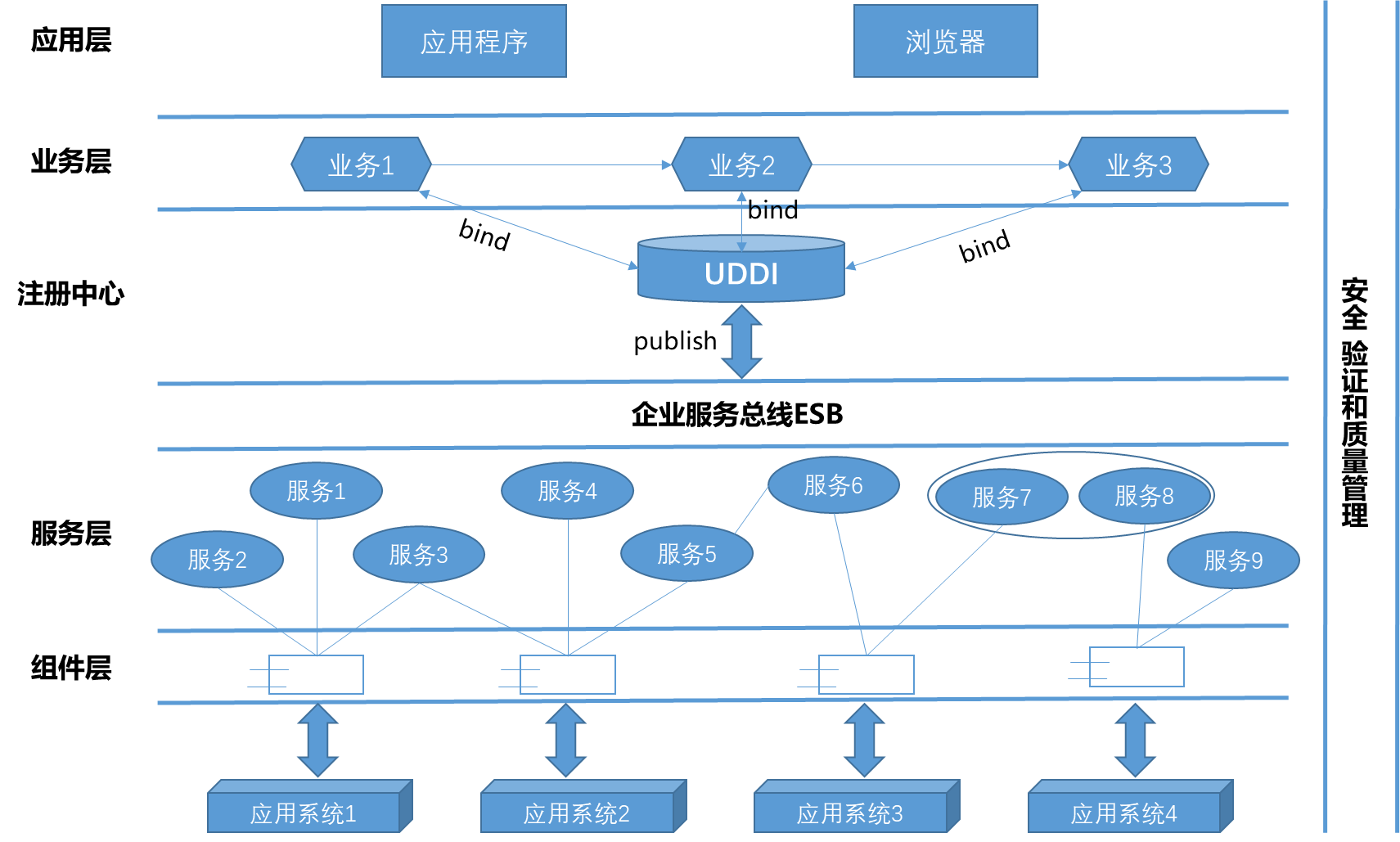
ESB作用于特点：

1、SOA的一种实现方式，ESB在面向服务的架构中起到的是总线作用，将各种服务进行连接与整合；

2、描述服务的元数据和服务注册管理；

3、在服务请求者和提供者之间传递数据，以及对这些数据进行转换的能力，并支持由实践中总结出来的一些模式如同步模式、异步模式等；

4、发现、路由、匹配和选择的能力，以支持服务之间的动态交互，解耦服务请求者和服务提供者。高级一些的能力，包括对安全的支持、服务质量保证、可管理性和负载平衡等。



**基于SOA的信息系统架构设计**

### 什么是REST,并指出在REST中将哪三种关注点进行分离。

REST从资源的角度来定义整个网络系统结构，分布在各处的资源由统一资源标识符（URI)确定，客户端应用程序通过URI获取资源的表现，并通过获得资源表现使得其状态发生改变。

REST中将资源、资源的表现和获取资源的动作三者进行分离。

### 什么是"响应式 Web 设计"，响应式Web设计的实现方式。

响应式web设计是指我们设计与开发的页面可以根据用户的行为和不同的设备环境做出相应的响应来调整页面的布局，以提供用户可感知的、流畅的阅读和操作体验。

实现方式：

（1）流式布局

（2）弹性布局加媒体查询

### 列举 3 种可实现信息系统安全保障的措施。

1、引入https协议或采用加密技术对数据先加密再传输

2、采用信息摘要技术对重要信息进行完整性验证

3、交易类敏感信息采用数字签名机制

### 信息系统面临哪些方面的安全威胁

信息系统面临的安全威胁来自于物理环境、通信链路、网络系统、操作系统、应用系统以及管理等多个方面。

物理安全威胁是指对系统所用设备的威胁，如自然灾害、电源故障、数据库故障和设备被盗等造成数据丢失或信息泄漏。

通信链路安全威胁是指在传输线路上安装窃听装置或对通信链路进行干扰。

网络安全威胁当前主要是指由于因特网的开放性、国际性与无安全管理性，对内部网络形成的严重安全威胁。

操作系统安全威胁指的是操作系统本身的后门或安全缺陷，如“木马”和“陷阱 门”等。

应用系统安全威胁是指对于网络服务或用户业务系统安全的威胁，包括应用系统自身漏洞，也受到“木马”的威胁。

管理系统安全威胁指的是人员管理和各种安全管理制度。

### 信息系统主要的认证方式

目前主要的认证方式有三类：

(1) 用户名和口令认证：主要是通过一个客户端与服务器知的口令（或与口令相关的数据）进行验证。根据处理形式的不同，分为验证数据的明文传送、利用单向散列函数处理验证数据、利用单向散列函数和随机数处理验证数据。

(2) 使用令牌认证：该方式中，进行验证的密钥存储于令牌中，目前的令牌包括安全证书和智能卡等方式。

(3)生物识别认证：主要是根据认证者的图像、指纹、气味和声音等作为认证数据。根据该企业的业务特征，采用令牌认证较为合适。

请解释授权侵犯的具体含义；针对王工的意见给出相应的解决方案，说明该解决方案的名称、内容和目标。

（李工认为，原业务系统只针对企业内部员工，采用了用户名/密码方式是可以的，但扩展为基于互联网的B2C业务系统后，认证方式过于简单，很可能造成用户身份被盗取；**王工认为，防止授权侵犯和保留用户痕迹的要求在方案中没有体现**。而刘工则认为，即使是在原有业务系统上的扩展与改造，也必须全面考虑信息系统面临的各种威胁，设计完整的系统安全架构，而不是修修补补。）

授权侵犯指的是被授权以某一目的使用某一系统或资源的某个人，将此权限用于其他非授权的目的，也称作“内部攻击”。

针对王工的建议，从系统安全架构设计的角度需要提供抗抵赖框架。

抗抵赖服务包括证据的生成、验证和记录，以及在解决纠纷时随即进行的证据恢复和再次验证。

框架中抗抵赖服务的目的是提供有关特定事件或行为的证据。例如，必须确认数据原发者和接收者的身份和数据完整性，在某些情况下，可能需要涉及上下文关系（如曰期、时间、原发者/接收者的地点等）的证据，等等。

### 分别针对采用对称加密策略与公钥加密策略，说明如何利用加密技术为在网络中传输的数据提供机密性与完整性保障。

对称加密策略：

机密性：发送者利用对称密钥对要发送的数据进行加密，只有拥有正确相同密钥的接收者才能将数据正确解密，从而提供机密性。

完整性：发送者根据要发送的数据生成消息认证码（或消息摘要），利用对称密钥对消息认证码进行加密并附加到数据上发送；接收者使用相同密钥将对方发送的消息认证码解密，并根据接收到的数据重新生成消息认证码，比较两个认证码是否相同以验证数据的完整性。

公钥加密策略：

机密性：发送者利用接收者的公钥对要发送的数据进行加密，只有拥有对应私钥的接收者才能将数据正确解密，从而提供机密性。

完整性：发送者根据要发送的数据生成消息认证码（或消息摘要），利用自己的私钥对消息认证码进行加密并附加到数据上发送；接收者利用对方的公钥将对方发送的消息认证码解密，并根据接收到的数据重新生成消息认证码，比较两个认证码是否相同以验证数据的完整性。

**从授权的可管理性、细粒度访问控制的支持和对分布式环境的支持三个方面指出项目组采用王工方案的原因。**

（经过分析和讨论，项目组决定采用加密技术为网络中传输的数据提供机密性与完整性保障。但在确定具体访问控制机制时，张工认为应该采用传统的强制访问控制 (Mandatory Access Control)机制，而王工则建议采用基于角色的访问控制（Role-Based Access Control)与可扩展访问控制标记语言(extensible Access Control Markup Language, XACML)相结合的机制。项目组经过集体讨论，最终采用了王工的方案。）

采用王工方案是因为基于角色的访问控制与XACML相结合的机制具有以下优势： 授权的可管理性：RBAC将用户与权限分离，相比MAC减小了授权管理的复杂性, 更适合于大型企业级系统的安全管理：

细粒度访问控制的支持：XACML提供了统一的访问控制策略描述语言，策略表达能力强，可用来描述各种复杂的和细粒度的访问控制安全需求，更适合企业复杂业务功能的访问控制要求；

分布式环境的支持：XACML的标准性便于各子系统的协作交互，各子系统或企业业务部门可以分布管理访问控制权限，而MAC则通常需要对访问控制权限集中管理， 不太适合企业基于SOA集成后的分布式系统。

**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。**

*(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曳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。*

**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。

**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。**

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。

**从构件管理支持、互操作支持以及公共服务支持三个方面说明现有分布式基础设施为构建分布式系统所提供的基本支撑。**

(1) 构件管理支持：现有分布式基础设施一般通过构件容器为构件提供基本的运行环境；具体功能一般包括管理构件的实例及其生命周期、管理构件的元信息等。

(2) 互操作支持：现有分布式基础设施均提供了高层通信协议以屏蔽节点的物理特性以及各节点在处理器、操作系统、程序设计语言等方面的异构性；基于互操作支持， 开发人员在开发与调用分布式对象时，均不需自己编写处理底层通信的代码。

(3) 公共服务支持：现有分布式基础设施通常将针对分布式软件的通用支持集成于一身，以公共服务的形式提供给应用程序；其提供的常见公共服务包括命名服务、事务 服务、安全服务、持久性服务等。

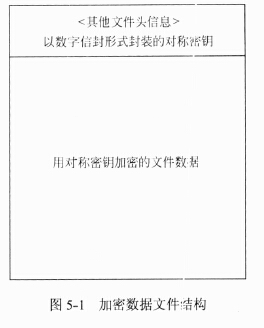
**在确定该支撑平台所采用的用户身份鉴别机制时，王工提出采用基于口令的简单认证机制，而李工则提出采用基于公钥体系的认证机制。项目组经过讨论，确定采用基于公钥体系的机制，请结合上述需求具体分析采用李工方案的原因。**

(1) 基于口令的认证方式实现简单，但由于口令复杂度及管理方面的原因，易受到认证攻击；而在基于公钥体系的认证方式中，由于其密钥机制的复杂性，同时在认证过程中私钥不在网络上传输，因此可以有效防止认证攻击，与基于口令的认证方式相比更为安全。

(2) 按照需求描述，在完成用户身份鉴别后，需依据用户身份进一步对业务数据进行安全保护，且受保护数据中包含用户私有的终端机数据文件，在基于口令的认证方式中，用户口令为用户和认证服务器共享，没有用户独有的直接秘密信息，而在基于公钥的认证方式中，可基于用户私钥对私有数据进行加密保护，实现更加简便。

(3) 基于公钥体系的认证方式协议和计算更加复杂，因此其计算复杂度要高？基于口令的认证方式，但业务环境的总用户数在100人以内，用户规模不大，运行环境又为局域网环境，因此基于公钥体系的认证方式可满足平台效率要求。

**基于数字信封的加密方式**



使用流加密方式可以获得更高的加解密效率。

数据加密与解密过程如下：

其加密过程为：首先生成一个对称密钥，使用用户公钥加密这个对称密钥后存储在文件头，然后用生成的对称密钥加密文件数据存储；

其解密过程为：用户首先使用自己的私钥解密被加密的对称密钥，再用该对称密钥解密出数据原文。

目前**数据库管理系统提供的基本数据加密支持**主要有以下两种：

(1) 加解密API:数据库管理系统提供可在SQL语句中调用的加解密API,应用可以利用这些API构建自己的基础架构，对数据进行加密保护。

(2) 透明加密：安全管理员为数据库敏感字段选择加密方式及密钥强度，应用访问受保护数据时只需使用口令打开或关闭密钥表，对数据的加密和解密由数据库管理系统自动完成。

加解密API方式的灵活性强，但构建和管理复杂；而透明加密方式管理简单，应用程序负担轻，但灵活性较差。用户要求尽可能减少安全管理与应用程序的负担，因此应选择透明加密方式。

### 开放架构应具有以下4个基本特点：（嵌入式）

①可移植性。各种计算机应用系统可在具有开放架构特性的各种计算机系统间进行移植，不论这些计算机是否同种型号、同种机型。

②可互操作性。如计算机网络中的各结点机都具有开放架构的特性，则该网上各结点机间可相互操作和资源共享。

③可剪裁性。如某个计算机系统是具有开放架构特性的，则在该系统的低档机上运行的应用系统应能在高档机上运行，原在高档机上运行的应用系统经剪裁后也可在低档机上运行。

④易获得性。在具有开放架构特性的机器上所运行的软件环境易于从多方获得，不受某个来源所控制。

### 系统常见问题出现原因

(1) **用户响应时间慢：**大型社交网络系统要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强可以， 但是应付上万次SQL写数据请求，硬盘I/O就已经无法承受了。特别是涉及到多表连接 操作，会导致响应变慢。

(2) **数据格式变化：**大型社交网络系统随着用户的使用，会不断地增加新的功能， 导致原有数据格式发生变化，甚至出现新的数据格式。但关系数据库中采用元组方式组织数据，难以使用新型数据格式，难以维护。

(3) **数据容量超过设计上限：**对于大型社交网络系统，往往会在很短时间内产生海量数据。关系数据库多采用中央数据存储，使得数据容量受限于前期设计的上限，很难实现数据容量的横向扩展。

(4) **系统可用性差：**关系数据库采用中央数据存储，容易成为系统的性能瓶颈，单点故障很容易导致系统崩溃，负载过高往往导致系统出现宕机现象。

针对问题（1)，NoSQL数据库支持高并发数据访问，性能较高。

针对问题（2），NoSQL数据库的数据存储结构松散，能够灵活支持多种类型的数据格式。

针对问题（3)，NoSQL数据库能够支持海量数据的存储，且易于横向扩展。

针对问题（4），NoSQL数据库基于分布式数据存储，不存在单点故障和性能瓶颈，系统可用性高。

**采用NoSQL数据库时可能存在的问题有：**

(1) NoSQL数据库的现有产品不够成熟，大多数产品处于初创期。

(2) NoSQL数据库并未形成一定的标准，产品种类繁多，缺乏官方支持。

(3) NoSQL数据库不提供对SQL的支持，学习和应用迁移成本较高。

(4) NoSQL数据库支持的特性不够丰富，现有产品提供的功能比较有限。

### 项目计划包括：

(1) 项目总计划（包括范围计划、工作范围定义、活动定义、资源需求、资源计划、活动排序、费用估算、进度计划及费用计划）》

(2) 项目辅助计划（质量计划、沟通计划、人力资源计划、风险计划、采购计划）。

### 基于构件的软件开发中，获取构件的4种方法：

(1) 从现有构件中获得符合要求的构件，直接使用或做适应性修改，得到可复用的构件；

(2) 通过遗留工程（Legacy Engineering)，将具有潜在复用价值的软件提取出来，得到可复用的构件；

(3) 从市场上购买现成的商业构件，BPCOTS(Commercial Off-The-Shell)构件；

(4) 开发新的符合要求的构件。

### 开发构件通常采取3种策略：

(1) 分区（partitioning)：指的是将问题情景的空间分割成几乎可以独立研究的部分；

(2) 抽象（abstraction)：是对在给定实践内执行指定计算的软/硬件申.元的一种抽象；

(3) 分割（segmentation)：是将结构引入构件的行为，支持对行为性质进行时序推理。

### 当前主流构件标准有：

(1) CORBA：由OMG(对象管理集团）制定；

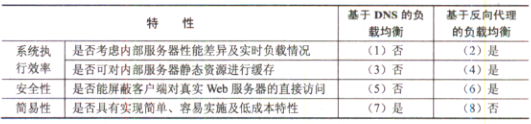
(2) COM/DCOM：由Microsoft制定；

(3) EJB：由SUN的Java企业Bean制定。

### 负载均衡机制的基本原理：

基于DNS的负载均衡机制通过DNS服务器实现，通常通过循环复用具有同一域名的多个主机地址的服务器实现负载均衡。

反向代理负载均衡则是将来自Internet的连接请求以反向代理的方式动态转发给内部网络上的多台服务器进行处理，从而达到负载均衡的目的。



从系统执行效率方面讲，基于DNS的负载均衡机制实现简单，但其通常不能区分服务器的差异，也不能反映服务器的当前运行状态。基于反向代理的则可以根据内部服务器的性能差异及实时负载情况进行动态负载均衡，当系统多个Web服务器性能存在明显差异或内部Web服务器出现故障时，负载均衡器可以更快做出响应，从而保证客户端的访问效率。采用基于反向代理的负载均衡机制，可在代理服务器中引入调速缓存机制，对Web服务器返回的静态页面或图片等静态资源进行缓存，由代理服务器承担对原始服务器的静态资源访问请求，从而进一步降低原始Web服务器的负载。

从安全性方面讲，采用基于反向代理的负载均衡机制，代理服务器屏蔽了客户端对真实Web服务器的直接访问，恶意用户无法对真实Web服务器进行攻击，且可以通过代理服务器为原本不安全的客户端与Web服务器之间的连接建立安全通道。因此采用基于反向代理的负载均衡机制可为系统提供更好的安全性保障。