高级系统架构设计师下午试题(Ⅰ)模拟14

试题一

1、企业应用集成(Enterprise Application Integration，EAI)是每个企业都必须要面对的实际问题。企业服务总线(Enterprise Service Bus，ESB.是一种体系结构模式，支持通信各方间的服务交互的虚拟化和管理。它充当面向服务架构(Service-Oriented Architecture，SOA.中服务提供者和请求者之间的连接服务的中间层。与传统的EAI技术相比，ESB采用总线式的体系结构集成多个应用系统，基于开放标准实现其内部核心功能，并支持快速加入新的应用到已有的集成环境中。

请围绕"ESB模式在企业应用集成中的应用"论题，依次从以下3个方面进行论述。

1．要叙述你参与实施的企业应用集成项目(包括业务背景、组织结构、现有应用系统的分布，以及采用的技术等)，以及你所担任的主要工作。

2．详细论述ESB的核心功能和典型结构；列举目前流行的ESB产品；指出你参与的项目所选择的ESB产品，并从ESB核心功能的角度说明选择该产品的理由。

3．阐述在使用ESB技术进行应用集成过程中所遇到的问题及解决办法，简要叙述你进一步应用ESB模式的有关设想。

试题二

阅读以下关于面向服务架构的应用叙述，根据要求回答问题。

[说明]

某航空公司的主要业务系统(如订票系统、航班调度系统等)始建于20世纪七八十年代，之后随着信息化的进展，陆续积累了许多异构的遗产信息系统。这些系统部分采用了J2EE、.NET等技术进行开发，分布在不同的地理位置，采用不同的协议进行数据传输。近年来，该公司在企业集成方面也是煞费苦心，已经在几个主要的核心系统之间构建了用于信息集成的信息Hub(Information Hub)，其他业务应用之间也有不少点到点的集成。尽管这些企业集成技术在一定程度上增进了系统间的信息共享，但是面对历史异构的遗产信息系统，企业的业务整合、功能整合仍是困难重重，主要表现如下。

2由于大部分核心应用构建在主机之上，因此Information Hub是基于主机技术开发，很难被开放系统使用。

3Information Hub对事件支持不强，被集成的系统间的事件以点到点流转为主，被集成系统间耦合性强。

4牵扯到多个系统间的业务协作以硬编码为主，将业务活动自动化的成本高，周期长，被开发的业务活动模块重用性差。

某软件开发公司承接了该航空公司应用系统集成任务，项目组经过多方讨论和论证之后，决定采用以面向服务为中心的企业集成技术，一步步解决该公司所面临的企业集成问题。

2、[问题1]

项目组在讨论架构方案时，某位架构师提出采用企业服务总线(ESB.架构模式，通过ESB的事件服务(Event Service)完成订阅发布，使应用程序间的事件集成不再需要原来的点到点方式，从而解耦组件之间的依赖关系，降低软件系统互连的复杂性。结合你的系统架构设计经验，请用400字以内的文字简要说明在ESB环境中组件之间典型的交互过程，以及ESB具有的核心功能。

3、[问题2]

将彼此关联的业务活动组成自动化流程可以进一步提高该航空公司业务活动的效率。以服务为中心的企业集成通过流程服务来完成业务流程集成。结合你的系统架构设计经验，请用300字以内的文字列举出3种SOA架构中的流程服务内容，并给出简要说明。

4、[问题3]

若项目组采用Web Service作为基于SOA集成方法的实现技术，请根据该航空公司目前的实际情况，用300字以内的文字说明系统应该分为哪几个层次，并简要说明每个层次的功能。

试题三

阅读以下关于设计模式应用的叙述，根据要求回答问题。

[说明]

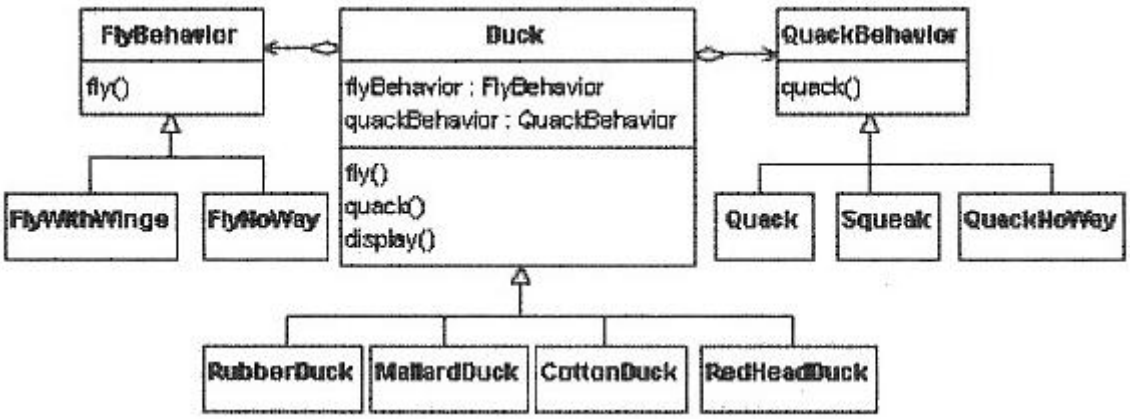
某软件公司承接了一项面向儿童的模拟游戏软件的开发任务，该游戏软件主要模拟现实世界中各种鸭子的发声特征、飞行特征和外观特征。游戏软件需要模拟的鸭子种类及其特征如表所示

鸭子种类及其特征			
鸭子种类	发声特征	飞行特征	外观特征
灰鸭(MallardDuck)	发出"嘎嘎"声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛
红头鸭(RedHeadDuck)	发出"嘎嘎"声(Quack)	用翅膀飞行(FlyWithWings)	灰色羽毛、头部红色
棉花鸭(CottonDuck)	不发声(QuackNoWay)	不能飞行(FlyNoWay)	白色
橡皮鸭(RubberDuck)	发出橡皮与空气摩擦的声音(Squeak)	不能飞行(FlyNoWay)	黑白橡皮颜色

为支持将来能够模拟更多种类鸭子的特征，该公司架构师采用某种设计模式设计的类图如图1所示。

在图1中，类Duck描述了抽象的鸭子，方法fly5、quack5和display5分别表示不同种类的鸭子都具有飞行特征、发声特征和外观特征；类FlyBehavior与QuackBehavior分别用于表示抽象的飞行行为与发声行为。

图1



5、[问题1]

请用350字以内的文字指出该公司架构师所采用的设计模式的具体名称、设计意图及其优缺点。

6、[问题2]

请用400字以内的文字指出该公司架构师所采用的设计模式的适用性，以及图1中需要考虑哪些实现问题？

7、[问题3]

设计模式在力度和抽象层次上各不相同。按设计模式的目的划分，可分为创建型、结构型和行为型3种模式；按设计模式的范围划分，可分为类设计模式和对象设计模式两种。请将下列A～J标记的设计模式填入到下表中的(1)～(5)空缺处。(请用A～J答题)

A. Abstract Factory模式　　B. Adapter模式　　C. Chain of Responsibility模式

D. Decorator模式　　E. Factory Method模式　　F. Flyweight模式

G. Interpreter模式　　H. Iterator模式　　I. Template Method模式

J. Visitor模式

设计模式空间				
		目 的		
		创建型	结构型	行为型
范 围	类	(1)	—	(2)

	对 象	(3)	(4)	(5)

试题四

8、RUP(Rational Unified Proces)是一种软件工程过程产品，它吸取了现代软件开发中许多成功的实践。RUP把软件生存周期划分为多个循环(Cycles)，每个循环生成产品的一个新的版本。每个循环依次由4个连续的阶段(Phase)组成，每个阶段完成确定的任务。与传统的软件过程相比，基于RUP的软件过程可以降低项目的风险，规范管理和开发流程，有效地控制资源，提高软件开发的成功率和生产率。

请围绕"统一软件开发过程的应用"论题，依次从以下3个方面进行论述。

1．概要叙述你参与管理和开发的软件项目及你在其中担任的主要工作，包括角色、工作内容等。

2．论述RUP的核心工作流和典型的迭代策略模式，具体论述你所参与项目如何应用RUP，在项目实施过程中遇到了什么问题，如何解决。

3．分析与评估你在所参与项目中应用RUP裁剪的实际开发效果，以及你进一步应用RUP的有关设想。

试题五

阅读以下关于Java企业级应用系统开发架构的叙述，根据要求回答问题。

[说明]

某软件公司承担了某中小型企业应用软件开发任务，进度要求紧迫。为了按时完成任务，选择合适的企业应用系统开发架构非常重要。项目组在进行方案论证时，项目组成员提出了两种开发思路。

9刘工建议采用J2EE 5.0和EJB 3.0进行开发。理由是J2EE定义了标准的应用开发体系结构和部署环境，EJB是J2EE的基础和核心。J2EE 5.0主要目标是简化开发，相比EJB 2.1，EJB 3.0具有很多改进和提高。

10杜工建议采用Struts、Spring和Hibenate轻量级开源框架相结合的方式。理由是随着Java开源项目阵营的发展壮大，一些基于POJOs(Plan Old Java Objects)的开源框架被广泛地引入到Java企业应用开发中来，与重量级的EJB框架相比，这些轻量级的框架有很多优点。

项目组仔细比较分析了两种方案的特点、优点和不足之处。认为杜工和刘工的建议都合理，但是从结合当前项目实际情况出发，最后决定采用杜工的建议。

9、[问题1]

Java企业级应用框架一般被划分为3个层次，请用150字以内的文字说明都有哪3个层次?功能分别是什么?

10、[问题2]

请用200字以内的文字叙述Struts、Spring和Hibernate开源框架特点和结合方式。

11、[问题3]

请用200字以内的文字说明基于Struts、Spring和Hibernate的轻量级框架与基于EJB的重量级框架解决问题的侧重点有什么不同?

答案:

试题一

1、1．简要介绍你参与规划、设计、实施和管理的企业应用集成项目的基本情况(包括业务背景、组织结构、现有应用系统的分布和采用的技术等)，简要说明自己在该项目中的角色、所承担的主要任务及开展的主要工作。论文叙述自己参与管理和实施的企业应用集成项目应有一定的规模，自己在该项目中担任的主要工作应有一定的分量。

2．企业服务总线(Enterprise Service Bus，ESB)是由中间件技术实现的支持面向服务架构(SOA)的基础软件平台，支持异构环境中的服务以基于消息和事件驱动模式的交互，并且具有适当的服务质量和可管理性。ESB技术的基本思想是，提供一种标准的软件底层架构，各种程序组件能够以服务单元的方式"插入"到该平台上运行，并且组件之间能够以标准的消息通信方式来进行交互。换而言之，ESB是传统中间件技术与XML、Web服务等技术结合的产物。ESB是一个集成平台，将现有的IT设施和应用系统暴露为服务。由于ESB基于开放标准，企业的遗产系统使用的私有技术能够基于开放和现代的技术(例如Web服务和消息机制等)暴露为服务。

1)其核心功能包括位置透明性、传输协议转换、消息转换、消息路由、消息增强、安全，以及监控和管理7项内容

(1)位置透明性(Location Transparency)。位置透明性是指当一个服务消费者与一个服务提供者通过ESB进行通信时，服务消费者不需要知道服务提供者的实际位置，这意味着服务消费者与服务提供者之间是解耦合的。

(2)传输协议转换(Transport Protocol Conversion)。当服务请求者与服务提供者采用不同的传输协议时，ESB能够将基于输入传输协议格式的数据转换为不同输出传输协议格式的数据。

(3)消息转换(Message Transformation)。在服务请求者和服务提供者进行交互时，ESB基于开发标准(XLST和XPath等)提供了将消息从一种格式转换为另外一种格式的能力。

(4)消息路由(Message Router)。在实际的集成环境中，对于一个特定的输入请求消息，可能有多个应用程序参与进来作为该消息传递的目标。ESB能够决定一个消息必须发送到哪些相关的应用程序中，处理这种逻辑的核心功能称为消息路由。

(5)消息增强(Message Enhancement)。在某些情况下，可能需要为请求数据添加额外的数据或转换已有的数据，在这种情况下，ESB应该提供对外部数据的访问能力，支持用户编写客户端代码对数据进行访问和处理。

(6)安全(Security)。ESB必须支持对消息的授权和认证能力，如果输入数据可能被恶意解析，还要提供加密能力。ESB的安全包括消息的机密性、完整性和可用性等，支持不同的安全策略与方法。

(7)监控和管理(Monitor and Management)。关注ESB的维护和管理能力。监控与管理功能包含多个方面，例如对于消息层来说，其管理主要包括管理消息队列，监控消息大小和消息队列的吞吐率等。对于Web服务，主要包括监控每个服务是否启动和运行，在每分钟有多少调用请求，对于一个Web服务，有多少服务实例在运行等。

(论文中只要给出以上7个核心功能中的5个即可)

2)ESB提供了一个基于标准的松散应用耦合模式，在层次化的技术结构中，ESB至少包含以下3层

(1)总线接入层：通过这一层可以使用户各种应用接入ESB，使用ESB的各种服务。在这一层提供对多种主流应用的接入协议支持，如HTTP、JCA/J2C、NET和IBM/CICS等。同时考虑到一些客户自己定制的应用与ESB的连接，在总线接入层提供了适配器服务。

(2)核心层：提供多种企业服务总线所需的必要服务支持，在这一层除了提供总线基本服务(如分发/订阅、队列、安全服务和仲裁服务等)外，还提供了QoS的支持(如高可用性、确保消息传输等)。

(3)微流程组合/拆分或定制路由层：这一层是侧重在业务支持上。通过通用和标准的对象和服务模型，可以在这一层上定义可重用和基于业界标准的业务流程。

3)目前流行的ESB产品包括商业产品和开源产品两类

(1)商业产品IBM的WebSphere ESB、Oracle的Oracle Service Bus(前身是BEA的AquaLogic ServiceBus)和微软的BizTalk Server等。

(2)开源产品：Mule、Apache ServiceMix、JBossESB、OpenESB和WSO2等。

(论文中只要给出以上产品中的4个即可)

4)结合项目实践经验，说明你参与管理和实施的工程项目所采用的ESB产品，然后围绕7个核心功能，并结合企业应用集成项目的实际特点，论述选择该ESB产品的原因，原因的描述要具有一定的广度和深度，要客观、适当。

3．具体说明你参与管理和开发的项目中，使用ESB技术进行应用集成时所遇到的问题。这些问题包含但不限于以下内容。

(1)如何根据企业应用集成的需求选择合适的ESB产品？

(2)如何根据企业的具体组织结构确定集成系统的体系结构，并据此设计系统的功能分布与物理拓扑结构？

(3)相关子系统之间的数据格式转换问题。

(4)针对具体业务编写合适的处理逻辑并确定消息路由问题等。

论述解决以上问题所采取的策略、具体办法和步骤，以及它们对该工程项目后期的工作产生了哪些积极(或消极)的影响(效果和存在的问题)。论文最后可以进一步讨论你在该工程项目中获得的与ESB应用相关的几点体会，以及在今后的工作过程中，如果碰到类似的开发项目你将如何应用这些经验或教训。对需要进一步改进的地方，应有具体的着眼点，不能泛泛而谈。

4．论文写作过程中值得关注的一些要点如下(全书同)。

(1)整篇论文要结构合理、切中要害、陈述完整、言简意赅、语言流畅、字迹清楚，切忌对知识点的堆积、长篇大论、言之无物。

(2)选择自己参与过的工程项目进行分析论述，所述项目切题真实，介绍清楚。

(3)下午试卷Ⅱ是论述题目，问题中提到的中心内容在题目的说明中都有所涉及。在答题时首先要冷静并认真阅读题目，找出和问题相关的知识点，确定考题的关键考点，这是答题的前提。

(4)摘要是全文概括，千万不要写成引言。

(5)围绕论文主题，对所参与的项目进行科学叙述与评价，要有具体的着眼点，不能泛泛而谈，尽可能从字里行间让阅卷者体会到你的实际工作能力、业务水平和项目实践经验。

(6)在考试过程中应注意技巧，让答题的思路最大限度地符合出题的思路，避免跑题，这样容易得到阅卷老师的共鸣。

(7)根据考生对所参与的项目中针对本论文主题的相关叙述与评价，可确定他(她)有无参与信息系统项目开发过程的实践经验。

试题二

2、企业服务总线(Enterprise Service Bus，ESB)是由中间件技术实现的支持面向服务架构(SOA)的基础软件平台，支持异构环境中的服务以基于消息和事件驱动模式的交互，并且具有适当的服务质量和可管理性。ESB技术的基本思想是，提供一种标准的软件底层架构，各种程序组件能够以服务单元的方式"插入"到该平台上运行，并且组件之间能够以标准的消息通信方式来进行交互。

一个在ESB环境中组件之间典型的交互过程是：首先由服务请求者触发一次交互过程，产生一个服务请求消息，并将该消息按照．ESB的要求标准化，然后标准化的消息被发送给服务总线。ESB根据请求消息中的服务名或者接口名进行目的组件查找，将消息转发至目的组件，并最终将处理结果逆向返回给服务请求者。

这种交互过程不再是点对点的直接交互模式，而是由事件驱动的消息交互模式。通过这种方式，ESB最大限度上解耦了组件之间的依赖关系，降低了软件系统互连的复杂性。连接在总线上的组件无须了解其他组件和应用系统的位置及交互协议，只需要向服务总线发出请求，消息即可获得所需服务。服务总线事实上实现了组件和应用系统的位置透明和协议透明。技术人员可以通过开发符合ESB标准的组件(适配器)将外部应用连接至服务总线，实现与其他系统的互操作。同时，ESB以中间件的方式，提供服务容错、负载均衡、QoS保障和可管理功能。

ESB的基本核心功能归纳如下。

(1)提供位置透明性的消息路由和寻址服务。

(2)提供服务注册和命名的管理功能。

(3)支持多种消息传递范型(如请求/响应、发布/订阅等)。

(4)支持多种可以广泛使用的传输协议(即传输协议转换)。

(5)支持多种数据格式及其相互转换(即多种平台下多种集成方式的支持)。

(6)提供日志和监控功能。

3、企业部门内部的IT系统通过将业务活动自动化来提高业务活动的效率。但是这些部门的业务活动并不是独立的，而是和其他部门的活动彼此关联的。将彼此关联的业务活动组成自动化流程可以进一步提高业务活动的效率。以服务为中心的企业集成通过流程服务来完成业务流程集成。在业务流程集成中，粒度的业务逻辑被组合成业务流程，流程服务提供自动执行这些业务流程的能力。在参考架构中，流程服务包括以下内容。

(1)编排服务(Choreography Service)：通过预定义的流程逻辑控制流程中业务活动的执行，并帮助业务流程从错误中恢复。

(2)事务服务(Transaction Service)：用于保证流程执行中的事务特性(ACID)。对于短流程，通常采用传统的两阶段提交技术；对于长流程，一般采用补偿的方法。

(3)人工服务(Staff Service)：用于将人工的活动集成到流程中。一方面，它通过关联的交互服务使得人工可以参与到流程执行中；另一方面，它需要管理由于人工参与带来的管理任务，如任务分派、授权和监管等。

4、在采用Web Service作为SOA的实现技术时，根据该航空公司目前的实际情况，从功能角度考虑，该系统应该至少分为6个层次，分别为底层传输层、服务通信协议层、服务描述层、服务层、业务流程层和服务注册层，如下表所示。

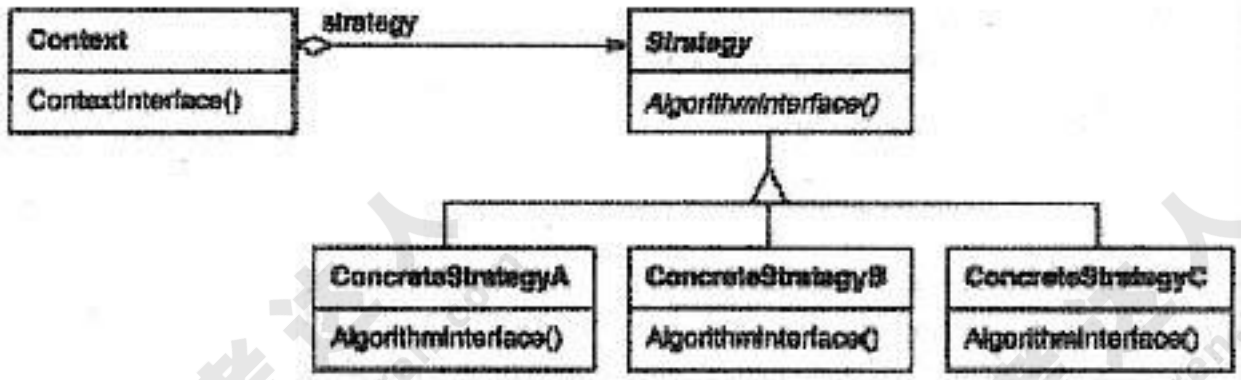
Web Service各层次功能及相关标准		
层 次	功 能	相关标准
底层传输层(Transport)	主要负责消息的传输机制	HTTP、SMTP、FTP和JMS等
服务通信协议层(Service Communication Protocol)	描述并定义服务之间进行消息传递所需的技术标准	SOAP和REST协议
服务描述层(Service Description)	主要以一种统一的方式描述服务的接口与消息交换方式	WSDL
服务层(Service)	将遗产系统进行包装，并通过发布的WSDL接口描述被定位和调用	
业务流程层(Business Process)	支持服务发现、服务调用和点到点的服务调用，并将业务流程从Web Service的底层调用抽象出来	WS-BPEL(BPEL4WS)
服务注册层(Service Registry)	使服务提供者能够通过WSDL发布服务定义，并支持服务请求者查找所需的服务信息	UDDI

试题三

5、依题意，在图1中，Duck为抽象类，描述了抽象的鸭子，方法fly()、quack()和display()分别表示不同种类的鸭子都具有飞行特征、发声特征和外观特征；而类RubberDuck、MallardDuck、CottonDuck和RedHeadDuck分别描述具体的鸭子种类；类Fly Behavior与QuackBehavior为抽象类，分别用于表示抽象的飞行行为与发声行为；类Fly NoWay与Fly WithWings分别描述不能飞行的行为和用翅膀飞行的行为；类Quack、Squeak与QuackNoWay分别描述发出“嘎嘎”声的行为、发出橡皮与空气摩擦声的行为和不发声的行为。鉴于不同的鸭子种类只是在行为方面有所区别，且为支持将来能够模拟更多种类鸭子的特征，该公司架构师最有可能采用策略(Strategy)设计模式来设计如图1所示的模拟鸭子游戏软件。

Strategy模式定义了一组能够用来表示可能行为集合的类。这些行为可以在应用程序中使用，来修改应用程序功能。Strategy(策略)模式的设计意图是，定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换，使得算法可独立于使用它的客户而变化。具体而言，该模式是一种定义一系列算法的方法，从概念上看，所有这些算法完成的都是相同的工作，只是实现不同，它可以以相同的方式调用所有的算法，减少了各种算法类与使用算法类之间的耦合。Strategy模式的一般结构如图2所示。

图2



Strategy 模式的一般结构图

Strategy模式具有以下一些优点和缺点。

(1)另一种子类化方法。Strategy类层次为Context(上下文)定义了一系列的可供重用的算法或行为。继承有助于析取出这些算法中的公共功能。可以直接生成一个Context类的子类，从而给它以不同的行为。但这会将行为强制编制到Context中，而将算法的实现与Context的实现混合起来，从而使Context难以理解、难以维护和难以扩展，而且还不能动态地改变算法。最后得到一堆相关的类，它们之间的唯一差别是它们所使用的算法或行为。将算法封装在独立的Strategy类中使得架构师可以独立于Context而改变它，使它易于切换、理解和扩展。

(2)在类自身中定义了每一个行为，从而减少了一些条件语句；Strategy模式提供了用条件语句选择所需行为以外的另一种选择。当不同的行为堆砌在一个类中时，很难避免使用条件语句来选择合适的行为。将行为封装在一个个独立的Strategy类中消除了这些条件语句。

(3)更容易扩展模型，即在不对应用程序进行代码修改的情况下，使该模式具有新的行为。

(4)客户必须了解不同的Strategy。该模式有一个潜在的缺点，就是一个客户要选择一个合适的Strategy就必须知道这些Strategy到底有何不同。此时可能不得不向客户暴露具体的实现问题。因此仅当这些不同行为变成与客户相关的行为时，才需要使用Strategy模式。

(5)Strategy和Context之间的通信开销。无论各个ConcreteStrategy(具体策略)实现的算法是简单还是复杂，它们都共享Strategy定义的接口。因此很可能某些ConcreteStrategy不会都用到所有通过这个接口传递给它们的信息；简单的ConcreteStrategy可能不使用其中的任何信息。这就意味着有时Context．会创建和初始化一些永远不会用到的参数。如果存在这样问题，那么将需要在Strategy和Context之间进行更加紧密的耦合。

(6)增加了对象的数目。Strategy增加了一个应用中的对象的数目。有时可以将Strategy实现为可供各Context共享的无状态的对象来减少这一开销。任何其余的状态都由Context维护。Context在每一次对Strategy对象的请求中都将这个状态传递过去。共享的Strategy不应在各次调用之间维护状态。

6、在以下情况中，应该使用Strategy模式。

(1)许多相关类只是在行为方面有所区别。“策略”提供了一种用多个行为中的一个行为来配置一个类的方法。

(2)需要使用一个算法的不同变体。例如，定义了一些反映不同的空间或时间权衡的算法，当这些变体实现为一个算法的类层次时，可以使用策略模式。

(3)算法使用客户端未知的数据，可使用策略模式以避免暴露复杂的、与算法相关的数据结构。

(4)一个类定义了多种行为，并且这些行为在这个类的操作中以多个条件语句的形式出现。将相关的条件分支移入它们各自的Strategy类中以代替这些条件语句。

依题意，在图1中，需要考虑以下一些Strategy模式实现问题。

(1)定义类Duck和类FlyBehavior(或类QuackBehavior)接口。这些接口必须使得类FlyWithwings、类FlyNoWay、类Quack、类Squeak和类QuackNoWay等能够有效地访问它所需要的类Duck中的任何数据，反之亦然。一种解决办法是让类Duck将数据放在参数中传递给类FlyBehavior(或类QuackBehavior)操作，也就是说，将数据发送给类FlyBehavior(或类QuackBehavior)。这使得类FlyBehavior(或类QuackBehavior)和类Duck解耦。但从另一个

角度考虑，类Duck也可能发送一些类FlyBehavior(或类QuackBehavior)不需要的数据。

另一种解决办法是让类Duck将自身作为一个参数传递给类FlyBehavior(或类QuackBehavior)，该类FlyBehavior(或类QuackBehavior)再显式地向类Duck请求数据；或者类FlyBehavior(或类QuackBehavior)可以存储对它的类Duck的一个引用，这样根本不再需要传递任何东西。这两种情况下，类FlyBehavior(或类QuackBehavior)都可以请求到它所需要的数据。但要求类Duck必须对它的数据定义一个更为精细的接口，这将使得类FlyBehavior(或类QuackBehavior)和类Duck更加紧密地耦合在一起。

(2)将类FlyBehavior(或类QuackBehavior)作为模板参数。例如，在C++中，可利用模板机制用一个Strategy来配置一个类。然而这种技术仅当下面条件满足时才可以使用：可以在编译时选择Strategy；它无须在运行时改变。在这种情况下，要被配置的类(如类Duck)被定义为以一个Strategy类作为一个参数的模板类。使用模板不再需要定义给类FlyBehavior(或类QuackBehavior)定义接口的抽象类。把类FlyBehavior(或类QuackBehavior)作为一个模板参数也使得可以将一个类FlyBehavior(或类QuackBehavior)和它的类Duck静态地绑定在一起，从而提高效率。

(3)尽量使类FlyBehavior(或类QuackBehavior)成为可选的对象。即使在不使用额外的FlyBehavior(或类OuackBehayior)对象的情况下，类Duck也还有意义，那么它还可以被简化。类Duck在访问类FlyBehavior(或类QuackBehavior)前先检查它是否存在，如果有，那么就使用它；如果没有，那么类Duck执行默认的行为。这种方法的好处是客户根本不需要处理FlyBehavior(或类QuackBehavior)对象，除非它们不喜欢默认的行为。　　7、设计模式主要用于得到简洁灵活的系统设计，GoF的书中共有23个设计模式，这些模式可以按两个准则来分类：一是按设计模式的目的划分，可分为创建型、结构型和行为型3种模式；二是按设计模式的范围划分，即根据设计模式是作用于类还是作用于对象来划分，可分为类设计模式和对象设计模式，如表所示。

设计模式空间				
		目的		
		创建型	结构型	行为型
范围	类	Factory Method	Adapter(类)	Interpreter Template Method
	对象	Abstract Factory Builder Prototype Singleton	Adapter(对象) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

创建型模式是对对象实例化过程的抽象，它通过采用抽象类所定义的接口，封装了系统中对象如何创建及组合等信息。该模式允许在系统中创建对象，而不需要在代码中标识特定类的类型，这样用户就不需要编写大量复杂的代码来初始化对象。它是通过该类的子类来创建对象的。但是，这可能会限制在系统内创建对象的类型或数目。创建型模式主要有Factory Method(工厂方法)、Abstract Factory(抽象工厂)、Builder(构建器)、Prototype(原型)和Singleton(单独)等模式。

结构型模式主要用于如何组合已有的类和对象以获得更大的结构，一般借鉴封装、代理或继承等概念将一个或多个类或对象进行组合和封装，以提供统一的外部视图或新的功能。该模式允许在不重写代码或自定义代码的情况下创建系统，从而使系统具有增强的重复使用性和应用性能。该模式控制了应用程序大部分之间的关系，将以不同的方式影响应用程序。结构型模式主要有Adapter(适配器)、Bridge(桥接)、Composite(组成)、Decorator(装饰)、Fagade(外观)、Flyweight(享

元)和Proxy(代理)等。

行为型模式主要用于对象之间的职责及其提供的服务的分配，它不仅描述对象或类的模式，还描述它们之间的通信模式，特别是描述一组对等的对象怎样相互协作以完成其中任意一个对象都无法单独完成的任务。该模式可以影响一个系统的状态和行为流。通过优化状态和行为流转换及修改的方式，可以简化、优化并且提高应用程序的可维护性。行为型模式主要有Interpreter(解释器)、TemplateMethod(模板方法)、Chain of Responsibility(职责链)、Command(命令)、Iterator(迭代器)、Mediator(中介者)、Memento(备忘录)、Observer(观察者)、State(状态)、Strategy(策略)和Visitor(访问者)等。

试题四

8、简要介绍你参与管理和开发的大中型信息系统软件工程项目的基本情况，简要说明自己在该项目中的角色、所承担的主要任务及开展的主要工作。论文叙述自己参与设计和实施的软件项目应有一定的规模，自己在该项目中担任的主要工作应有一定的分量。

2．统一软件开发过程(RUP)是一种用例驱动的，以体系结构为中心、迭代和增量的软件开发过程。可以采用二维模型来描述RUP—时间和内容。从时间维来看，软件生存周期被划分为不同的循环(Cycles)。每个循环又被划分为4个连续的阶段(Phase)，每个阶段都包含一个妥善定义的里程碑(Milestone)；每个阶段还可以被进一步划分为若干轮迭代(Iterations)。一次迭代是一次完整的开发过程，每次迭代结束时会发布一个可执行的产品，这个产品是正在开发的软件系统的一个子集，它会逐渐扩展为最终系统。内容结构指的是一些将活动(Activities)组织在一起的、天然存在的规则。

1)RUP的生命周期

RUP把生命周期模型划分为初始阶段(Inception)、细化阶段(Elaboration)、构造阶段(Construction)和交付阶段(Transition)4个阶段，如下表所示。每个阶段结束于一个主要的里程碑(MaiorMilestones)。每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足。如果评估结果令人满意的话，可以允许项目进入下一个阶段。

RUP各阶段说明				
阶段	目标	目标说明	里程碑	里程碑说明
初始阶段	确定项目开发的目标和范围，即确定项目的边界	为了达到该目的必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。对于建立在原有系统基础上的开发项目来讲，初始阶段可能很短	生命周期目标(Lifecycle Objective)	评价项目基本的生存能力
精化阶段	确定系统架构和明确需求。分析问题领域，建立健全的体系结构基础，编制项目计划，淘汰项目中最高风险的元素	为了达到该目的，必须在理解整个系统的基础上，对体系结构作出决策，包括其范围、主要功能和诸如性能等非功能需求。同时为项目建立支持环境，包括创建开发案例，创建模板、准则及准备工具	生命周期结构(Lifecycle Architecture)	为系统的结构建立了管理基准并使项目小组能够在构建阶段中进行衡量。此刻，要检验详细的系统目标和范围、结构的选择，以及主要风险的解决方案
构建阶段	实现剩余的系统功能，所有的功能被详细测试	所有剩余的构件和应用程序功能被开发并集成为产品，所有的功能被详细测试。从某种意义上说，构建阶段是一个制造过程，其重点放在管理资源及控制运作以优化成本、进度和质量	初始功能(Initial Operational)	决定了产品是否可以在测试环境中进行部署。此刻，要确定软件、环境及用户是否可以开始系统的运作。此时的产品版本也常被称为"Beta"版
移交阶段	完成软件的交付工作，将系统移交给客户	重点是确保软件对最终用户是可用的，该阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。用户反馈应主要集中在产品调整、设置、安装和可用性问题，所有主要的结构问题应该已经在项目生命周期的早期阶段解决了	产品发布(Product Release)	此时要确定目标是否实现，是否应该开始另一个开发周期。在一些情况下这个里程碑可能与下一个周期的初始阶段的结束重合

2)RUP中有9个核心工作流，分为6个核心过程工作流(Core Process Workflows)和3个核心支持工作流(Core Supporting Workflows)。9个核心工作流在项目中轮流被迭代使用，在每一次迭代中以不同的重点和强度重复

(1)业务建模(Business Modeling)工作流：描述了如何为新的目标组织开发一个构想，并基于这个构想在商业用例模型和商业对象模型中定义组织的过程、角色和责任。

(2)需求(Requirements)工作流：描述系统应该做什么，并使开发人员和用户就这一描述达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织和文档化；最重要的是理解系统所解决问题的定义和范围。

(3)分析和设计(Analysis或Design)工作流：将需求转化成未来系统的设计，为系统开发一个健壮的结构并调整设计使其与实现环境相匹配，优化其性能。分析设计的结果是一个设计模型和一个可选的分析模型。

(4)实现(Implementation)工作流：以层次化的子系统形式定义代码的组织结构，以组件的形式(源文件、二进制文件或可执行文件)实现类和对象，将开发出的组件作为单元进行测试，并集成由单个开发者(或小组)所产生的结果，使其成为可执行的系统。

(5)测试(Test)工作流：检验对象间的交互作用，验证软件中所有组件的正确集成，检验所有的需求已被正确的实现，识别并确认缺陷在软件部署之前被提出并处理。

(6)部署(Deployment)工作流：软件打包、生成软件本身以外的产品、安装软件，以及为用户提供帮助等。成功的生成版本并将软件分发给最终用户。该工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动。

(7)配置和变更管理(Configuration或Change Management)工作流：描述了如何管理并行开发、分布式开发、如何自动化创建工程，以及对产品修改原因、时间和人员保持审计记录。该工作流提供了准则来管理演化系统中的多个变体，跟踪软件创建过程中的版本。

(8)项目管理(Project Management)工作流：为项目的管理提供框架，为计划、人员配备、执行和监控项目提供实用的准则，为管理风险提供框架等。软件项目管理平衡各种可能产生冲突的目标，管理风险，克服各种约束并成功交付使用户满意的产品。

(9)环境(Environment)工作流：向软件开发组织提供软件开发环境，包括过程和工具。该工作流集中于配置项目过程中所需要的活动，同样也支持开发项目规范的活动，提供了逐步的指导手册并介绍了如何在组织中实现过程。

3)关于RUP迭代计划的安排，通常有以下4种典型的策略模式

(1)增量式(Incremental)。该模式的特点是项目架构的风险较小(往往是开发一些重复性的项目)，所以精化阶段只需要一个迭代。但项目的开发工作量较大，构建阶段需要有多次迭代来实现，每次迭代都在上一次迭代的基础上增加实现一部分的系统功能。

(2)演进式(Evolutionary)。当项目架构的风险较大时(从未开发过类似项目)，需要在精化阶段通过多次迭代来建立系统的架构，架构是通过多次迭代的探索，逐步演化而来的。当架构建立时，往往系统的功能也已经基本实现，所以构建阶段只需要一次迭代。

(3)增量提交(Incremental Deliveq)。该模式的特点是产品化阶段的迭代较多，比较常见的例子是项目的难度并不大，但业务需求在不断地发生变化，所以需要通过迭代来不断地部署完成的系统；但同时又要不断地收集用户的反馈来完善系统需求，并通过后续的迭代来补充实现这些需求。

(4)单次迭代(Grand Design)。传统的瀑布模型可以看做是迭代化开发的一个特例，整个开发流程只有一次迭代。但这种模式有一个固有的弱点，由于它对风险的控制能力较差，往往会在产品化阶段产生一些额外的迭代，造成项目的延误。

4)结合项目实践经验，说明在你参与管理和开发的软件项目中如何应用RUP，在项目实施过程中遇到了什么问题，采用了哪些技术、方法和步骤来解决相关的问题，以及它们对该工程项目后期的工作产生了哪些积极(或消极)的影响(效果和存在的问题)。对所提出的问题，应有具体的着眼点，不能泛泛而谈。

3. RUP是一个通用的过程模板，包含了很多开发指南、制品，以及开发过程所涉及的各种角色说明。RUP非常庞大，没有一个项目会使用RUP中的所有东西，针对具体的开发机构和项目，应用RUP时还要做裁剪，即要对RUP进行配置。RUP就像一个元过程(meta-process)，通过对RUP进行裁剪可以得到很多不同的开发过程，这些软件开发过程可以看做RUP的具体实例，以适应于不同的开发机构和项目需求。针对一个具体的软件项目，RUP裁剪可以分为以下几个步骤。

(1)确定本项目需要哪些工作流。RUP的9个核心工作流并不总是需要的，可以根据项目的规模、类型等对核心工作流做一些取舍。例如，嵌入式软件系统项目通常不需要业务建模这一工作流。

(2)确定每个工作流要产出哪些制品。例如，规定某个工作流应产出哪些类型的文档。

(3)确定生命周期的4个阶段之间如何演进。确定阶段间演进要以风险控制为原则，决定每个阶段要执行哪些工作流，每个工作流执行到什么程度，产出的制品有哪些，每个制品完成到什么程度等。

(4)确定每个阶段内的迭代计划。规划RUP的4个阶段中每次迭代开发的具体内容有哪些。

(5)规划工作流内部结构。工作流不是活动的简单堆积，它涉及角色、活动及制品，其复杂程度与项目规模及角色多少等因素有关。通常用活动图的形式给出所规划的工作流的内部结构。这一步决定裁剪后的RUP要设立哪些角色，是对RUP进行裁剪的难点。

结合RUP裁剪步骤的相关知识，分析与评估你在所参与项目中应用RUP裁剪的实际开发效果。注意要给出具体的评价依据，评价要客观、适当。论文最后可以进一步讨论你在该工程项目中获得的、应用RUP方面的几点经验体会，以及在今后的工作过程中，如果碰到类似的开发项目你将如何应用这些经验或教训。

试题五

9、这是一道要求读者掌握Java企业应用框架层次结构及其各层功能的简答题。本题所涉及的知识点如下。

(1)Java企业应用框架一般被划分为表现层、业务逻辑组件层和持久层等3个逻辑层次。

(2)其中，表现层用来建立应用系统的界面，对应视图(View)。该层集中于为从客户端发来的请求服务的对象及其行为，用于展现数据；负责View组件实现模式、组件在View显示粒度、页面跳转，以及事件触发等功能。例如，表现层采用JSF(Java Server Face)，JSF的开发流程的核心是事件驱动，组件和标签的封装程度非常高，很多典型应用已经不需要开发者去处理HTTP，整个过程是通过IoC(依赖注入)来实现的，即可以帮助对客户端请求进行先期及后期的处理等。

(3)业务逻辑组件层用来开发应用逻辑，对应控制器(Controller)。例如，业务逻辑组件层采用EJB3.0的Session Bean。EJB 3.0允许开发者使用耦合松散的组件来开发应用，实现一个EJB所有使用的类和接口都减少了。

该层集中于支持由表现层发起的(某些情况下也可能由持久层直接发起)业务数据的逻辑处理。例如，隐藏业务对象的复杂性，集中工作流的处理；分离表现层与持久层，并为服务提供外观和代理接口等。

(4)持久层是实现持久化存储，对应模型(Model)。例如，采用EJB 3.0实体Bean持久化模型，吸收了Hibernate的一些思想采用O/R MappinR模式。

该层集中于支持外部资源通信。例如，与数据库交互数据；抽象数据源，提供透明的数据访问；帮助进行EJB组件中的异步处理等。

10、这是一道要求读者掌握Struts，Spring和Hibenate轻量级开源框架特点和结合方式的简答题。本题所涉及的知识点如下。

(1)由于EJB容器能够很好的处理系统性能、事务机制、安全访问权限，以及分布式运算等问题，基于EJB框架进行开发能保证企业应用平滑发展，而不是发展到一种规模就重新更换一套软件系统，且可以保证开发人员将大部分精力集中在业务逻辑的开发上。采用EJB框架开发的企业应用具有必须继承或依赖EJB容器的特点。EJB充分考虑到了顶级大型项目的需求，使用它几乎能解决企业级应用涉及的所有问题，相应的基于EJB框架也是一个功能复杂的重量级框架。

(2)在基于POJOs轻量级框架上开发的应用程序无须依赖于EJB容器可独立运行，对应于Java企业应用3个层次的轻量级框架技术分别都得到了一定的发展。Struts框架+Spring框架+Hiberhate框架实现了表现层、业务逻辑组件层和持久层的结合。

(3)目前比较流行的开源表现层框架主要有Struts和Tapestry。其中，Struts是基于模型—视图—控制器(MVC)模式的开源框架，主要用于企业应用中表示层的实现。借助于Struts，开发人员可以把主要精力集中在业务处理上，简化遵循MVC设计模式的Web应用开发工作，很好地实现代码重用，提高开发效率。

Straats框架包括：①模型(Model)。在Straats中模型是一个Action类，开发者通过其实现商业逻辑，同时用户请求通过控制器向Action的转发过程是基于由Struts-config. xml文件描述的配置信息的。②视图(View)。视图是由与控制器配合工作的一整套JSP定制标签库构成，利用它们可以快速建立应用系统的界面。③控制器(Contoller)，本质上是一个Servlet，将客户端请求转发到相应的Action类。④一堆用来做XML文件解析的工具包。

Struts应用框架由于出现时间早，因此使用相对广泛，很容易找到很多现成的开源功能标签以供使用，以及样例程序可供参考。但是它的组件在页面中显示的粗粒度，以及框架类的限制在很多情况下会表现得过于死板，给表示层的开发会带来一些额外的代码开销。

Tapestry与之不同的是，它是基于组件，而不是面向脚本语言(比如JSP和Velocity)的，组件是由一个定义文件(以XML的格式)、一个HTML模板和一个Java类构成的。

(4)Spring是业务组件层轻量级框架。Spring框架是一个基于IoC(依赖注入)和AOP(面向方面编程)的构架。用户可以通过Spring来利用普通Java对象(POJO)编程，使用依赖注入解析POJO间的依赖性，然后使用面向方面编程(AOP)将服务与它们相关联。采用依赖注入使得它可以很容易地实现Bean的装配，提供了简洁的AOP并据此实现事务管理等，但是它不具备处理应用分布式的能力。Spring的核心要点是支持不绑定到特定J2EE服务的可重用业务和数据访问对象。这样的对象可以在不同的J2EE环境(Web或EJB)、独立应用程序和测试环境之间重用。

Spring框架处于应用服务器和服务库的上方，服务整合的代码属于框架，并暴露于应用开发者。

它与应用服务器整合的能力相对EJB 3.0要弱。但是Spting框架模块的可分离配置体现了它优于EJB 3.0的灵活性。

(5)持久层框架主要有Hibernate和各种JDO产品，以及iBATIS等。其中，Hiberhate是一个开源的O/R Mapping框架，它对JDBC进行了非常轻量级的对象封装，可以应用在任何使用JDBC的场合，可以在应用EJB的J2EE框架中取代CMP，完成数据持久化的重任。相对而言，Hibernate基本优势表现在：使用Java反射机制而不是字节码增强程序来实现透明性；使用简单；映射的灵活性很出色，它支持各种关系数据库，从一对一(1:1)到多对多(m:n)的各种复杂关系。其缺点是限制所使用的对象模型(例如，一个持久性类不能映射到多个表)。

iBATIS是一个简易的SQL Map工具，它是将手工编写的在XML。配置文件中的SQL语句映射成Java对象。使用iBATIS提供的O/R Mapping机制，对业务逻辑实现人员而言，面对的是纯粹的Java对象，这一层与通过Hibernate实现O/R. Mapping而言基本一致，而对于具体的数据操作，HiberTlate会自动生成SOL语句，而iBATIS则要求开发者编写具体的SQL语句。相对Hibernate等"全自动"O/R Mapping机制而言，iBATIS以SQL开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。作为"全自动"ORM实现的一种有益补充，iBATIS的出现显得别具意义。

11、这是一道要求读者对比轻量级框架与重量级框架解决问题的侧重点的综合理解题。本题所涉及的知识点如下。

(1)设计与性能是实际框架选择的两个基本点，善于平衡才是框架选择的主要宗旨。轻量级框架和重量级框架解决问题的侧重点是不同的。

(2)轻量级框架侧重于减小开发的复杂度，相应的它的处理能力便有所减弱(如事务功能弱、不具备分布式处理能力)，比较适用于开发中小型企业应用。采用轻量级框架后，一方面因为采用基于POJOs的方法进行开发，使应用不依赖于任何容器，这可以提高开发调试效率；另一方面轻量级框架多数是开源项目，开源社区提供了良好的设计和许多快速构建工具，以及大量现成可供参考的开源代码，这有利于项目的快速开发。例如，目前Tomcat+Spring+Hibernate已经成为许多开发者开发J2EE中小型企业应用偏爱的一种架构选择。

(3)作为重量级框架的EJB框架则强调高可伸缩性，适用于开发大型企业应用。在EJB体系结构中，一切与基础结构服务相关的问题和底层分配问题都由应用程序容器或服务器来处理，且EJB容器通过减少数据库访问次数及分布式处理等方式提供了专门的系统性能解决方案，能够充分解决系统性能问题。

(4)轻量级框架的产生并非是对重量级框架的否定，在某种程度上可以说二者是互补的。轻量级框架旨在开发具有更强大、功能更完备的企业应用；而EJB3.0规范则在努力简化J2EE的使用，以使得EJB不仅仅是擅长处理大型企业系统，也利用开发中小型系统，这也是EJB轻量化的一种努力。对于大型企业的应用及将来可能涉及能力扩展的中小型应用，结合使用轻量级框架和重量级框架也不失为一种较好的解决方案。