# 质量属性及设计策略

1.性能：指系统的响应时间，即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能够处理事件的个数。

2.可用性：系统能够正常运行的时间比例。

3.可靠性：软件系统在应用或者错误面前，在意外或错误使用的情况下，维持系统功能特性的基本能力。

4.健壮性：处理或者环境中，系统能够承受压力或变更的能力。

5.安全性：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者拒绝服务的能力。

6.可修改性：能够快速以较高的性价比对系统变更的能力。

7.可变性：体系结构经扩充或变更成新体系结构的能力。

8.易用性：衡量用户使用一个软件产品完成指定任务的难易程度。

9.功能性：系统完成所期望工作的能力。

10.互操作性：系统与外界或者系统与系统之间相互作用的能力。

11.可测试性：软件发生故障并隔离、定位其故障的能力，以及在一定时间和成本的前提下，进行测试设计、测试执行的工作。

设计策略包括：

1. 性能：增加计算资源、改善资源需求、资源管理、资源调度
2. 安全性：攻击检测、抵御攻击（授权、认证、访问限制）、从攻击中恢复、安全审计
3. 可用性：主动冗余、被动冗余、心跳线、选举、Ping/Echo
4. 可修改性：信息隐藏、接口实现分离、抽象、模块泛化、使用中介、延迟绑定

# 2 架构分析风险点/敏感点/权衡点的定义

架构分析风险点是架构设计中，潜在的、可能出现问题的决策带来的隐患。

架构分析敏感点是为实现某个质量属性，一个或多个系统中共有的特性。

架构分析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。

# DFD中的基本元素和作用

1.数据流，数据在系统内传播的路径，因此由一定成分固定的数据组成。

2.外部实体，代表系统之外的实体，可以是人、物或者其他软件系统。

3.加工，对数据进行处理的单元，接收一定的数据输入、对其进行处理，并产生输出。

4.数据存储，表示信息的静态存储，可以是文件、文件的一部分或者数据库元素。

# 4 DFD和流程图的区别

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的数据流。

流程图以图形化的方式展示应用程序，从输入到输出的逻辑过程，描述处理过程的控制流。

两者的区别有：

1.数据流图的处理结果可以并行，流程图在某个时间点只处于一个处理过程。

2.数据流图展示系统的数据流，流程图展示系统的控制流。

3.数据流程展示全局的处理过程，遵循不一样的计时标准，流程图处理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。

# 5 高质量DFD的设计原则

1.复杂性最小化原则：DFD分层结构就是把信息划分成小的、且相对独立的一大批子集例子，这样可以单独考察每一个DFD，如果要了解某个过程更加详细的信息，可以跳转至过程的下一层，如果要知道一个DFD是如何与其他DFD关联，可以跳转至上层DFD。

2.接口最小化原则：接口最小化原则是复杂性最小化原则的一种具体规则，在设计模式时，应使得模型中各个元素间的接口数和连接数最小化。

3.数据流一致性原则，一个过程和它的过程分解应在数据流内容中保持一致。

# 6 软件架构风格的定义和常见软件架构风格的定义

软件架构风格是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构件间的组织方式，惯用模式则反映众多系统共有的结构和语义。

【一】数据流架构风格

1.管道过滤器架构风格，每一个构件都接受一组输入和输出，构件接收数据输入，经过内部处理后产生数据输出，这样的构件称为过滤器，构件间的连接件称为传输数据流的管道。

2.批处理架构风格，构件为一组固定顺序的计算单元，每一个过程都是独立的步骤，上一步骤完成后，下一步骤才能开始。构件间通过数据进行交互，数据必须是完整的，以整体的方式进行传递。

【二】调用返回架构风格

1.主程序子程序架构风格，该风格中计算构件为子程序协作工作，并由主程序顺序调用，构件间通过共享存储区交换数据。

2.面向对象架构风格，该风格的特征是将数据操作和内部表示封装在对象中，该模式的构件为对象，对象维护自身表示的完整性，对象间通过消息传输机制进行通信，对象交互时必须知道彼此的身份，通过对象间的协作完成计算处理。

3.层次架构风格，系统组织成层次架构，每一层为上一层提供服务，调用下一层的服务，每一层只能见到与自己邻接的层，上层需要知道下层的身份，而修改通常只影响上层。

【三】独立构件架构风格

1.进程间通信架构风格，构件为独立的命名过程，连接件为消息传输机制，可以为同步、异步、点多点、远程调用等多种方式。

2.基于事件的隐式调用风格，构件不接调用某个过程，而是通过触发或广播一个或多个事件。系统中其他构件的过程可以在一个或多个事件中注册，当某个事件触发时，系统将自动调用在该事件注册过的所有过程。

【四】虚拟机架构风格

1.解释器架构风格，该风格包括一套解释器引擎、一个存储被解释代码的存储区、一个记录当前解释器工作状态的数据结构、一个记录当前被解释代码进度的数据结构，基于解释器的架构风格系统包含一个虚拟机，可模拟硬件的执行过程和一些关键应用。

2.基于规则的架构风格，包含规则集、规则解释器、规则/数据选择器、以及工作内存。

【五】仓库风格

1.数据库架构风格，该风格中主要包含两大类，一类是中央数据存储单元，用于记录系统数据状态，一类是独立处理单元，用于对数据进行操作。数据库架构风格以数据格式解耦各个功能之间的关系，可灵活定义各个功能之间的逻辑顺序。

2.黑板架构风格，该风格包括黑板、知识源、控制三大部分，其中黑板是一个全局数据库，包含解域的全部状态。知识源包含若干独立的计算单元，提供解决问题的知识，响应黑板上的变化，也只修改黑板。黑板架构风格常用于解决问题没有确定性算法的系统中。

【六】其他

1.控制环路架构风格，该风格的特征是将系统输出变量的某个属性值维持在某个特定的设定值，包含操纵变量、输入变量、过程变量、设定点等构件，通过不断对比实际状态和理想状态下的系统变量状态，调整过程变量使得实际状态不断趋近理想状态。

2.MVC架构风格，将业务逻辑、数据、展示层分离的方法组织代码，将业务逻辑聚集在一个构件里，在改进和个性化定制界面及用户交互的同时，无需重新编写业务逻辑。MVC架构将整个架构分为模型、视图、控制器三大部分。模型负责维护并保存具有持久性的业务数据，实现业务处理功能并将业务数据的变化通知视图。视图负责展示模型中的业务数据，响应模型的变化，更新展示样式，同时将界面的用户动作传递给控制器。控制器将界面动作映射为模型中的处理功能，并调用，根据模型返回的结果选择新的视图。

3.C2架构风格。C2体系结构风格可以概括为:通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2风格中的系统组织规则如下:系统中的构件和连接件都有一个顶部和一个底部;构件的顶部应连接到某连接件的底部,构件的底部则应连接到某连接件的顶部,而构件与构件之间的直接连接是不允许的;一个连接件可以和任意数目的其他构件和连接件连接;当两个连接件进行直接连接时,必须由其中一个的底部到另一个的顶部。

4.C/S风格。C/S体系结构有三个主要组成部分:数据库服务器、客户应用程序和网络。

5.三层C/S结构风格。二层C/S结构是单一服务器且以局域网为中心的,所以难以扩展至大型企业广域网或Internet软、硬件的组合及集成能力有限,客户机的负荷太重,难以管理大量的客户机,系统的性能容易变坏,数据安全性不好。三层C/S体系结构是将应用功能分成表示层、功能层和数据层三个部分,削弱二层C/S结构的局限性。

6.B/S风格。B/S风格就是三层C/S结构的一种实现方式,具体结构为浏览器/服务器/数据库服务器。

# 6.1 补充：如何实现基于事件的隐式调用和解释器架构

基于事件的隐式调用架构风格的实现过程为：

1. 定义xx事件，实现事件命中后的xx函数
2. 维护一个事件注册表，将xx事件与对应的函数关联起来，成为注册表中的一个记录
3. 当事件发生时，系统查找事件注册表，调用对应的函数。

解释器架构风格的实现过程为：

1. 为xx元素和xxx关系定义某种语言，并描述其语法及语义
2. 编写解释器对该语言解释
3. 生成对应的脚本语言程序

# 7 MVC架构风格的定义、组件、交互模式、优点

MVC架构风格是一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集在一个部件里，在改进和个性化定制界面时，不需要重新编写业务逻辑。

MVC构架把整个软件系统分为模型、视图和控制器三个部分。

模型负责维护并保存持久性的业务数据，实现业务处理功能，并将业务数据的变化及时通知给视图。

视图负责呈现模型中包含的业务数据，响应模型变化通知，更新呈现形式，并向控制器传递用户的界面动作。

控制器负责将用户的界面动作映射为模型的业务处理功能，并调用模型，根据返回结果选择新的视图。

MVC模式允许多种界面的拓展，视图的变更与增加，与模型无关

MVC模式易于维护，控制器和视图随着模型的扩展而扩展，只需要保持公共接口，旧版本可继续使用

MVC模式可支持强大功能的界面。

# 8 Rest架构风格的定义、特点、优点、缺点，将呐三者分离

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确定，客户端应用程序通过URI获取资源的表现，并通过获取资源的动作使其状态改变。

Rest风格的特点是客户端/服务器、无状态、缓存、统一接口、分层系统、按需代码。

优点：

1.实现技术成熟、简单

2.基于URI和超链接，不需要集中式的服务信息仓库

3.支持缓存、无状态、可支持大量客户端

4.轻量级Web框架

5.测试相对简单，用浏览器即可

缺点：

1.Rest要求输入参数以URI传递，对参数容量有一定限制

2.在URI中表达复杂类型的参数比较困难，且目前不存在公认编组/解组方法

3.Rest提倡的风格和实际实现仍有差距，如高层Rest用GET、POST、DELETE、PUT四种方法，而实际应用通常只采用GET和POST。

Rest将资源、资源的表现、获取资源的动作三者分离。

# 9 构件的定义、获取构件的方法、开发构件的策略、主流构件标准

构件是一个功能相对独立、且具有复用价值的软硬件单元，可实现特定的功能，符合一套接口标准并实现一组接口。

获取构件的方法：

1.从现有构件中获取符合要求的构件，直接使用或者作适用性修改，得到可重用的构件

2.通过遗留工程，将具有潜在重用价值的构件提取出来，得到可重用的构件

3.从市场中购买现成的商业构件

4.开发新的符合要求的构件

开发构件的策略：

1.分区：将问题情景的空间，分割成几乎可独立研究部分

2.抽象：对给定实践内，执行指定计算的软硬件单元的一种抽象

3.分割：将结构引入构件的行为，支持对行为性质进行排序推理。

主流构件标准：  
1.OMG的CORBA

2.Microsoft的COM、DCOM

3.SUN的J2EE/EJB

# 10中间件（分布式基础设施）的优势

1.构件管理支持：现有中间件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：现有中间件提供了高层通信协议，以屏蔽节点的物理特性及其各节点在处理器、操作系统、程序设计语言等方面的异构性。同时，基于互操作支持，开发人员开发和调用中间件时，不用自己编写底层通信代码。

3.公共服务支持：现有中间件集成了分布式软件的通用支持，公共服务的形式提供给应用程序，如命名服务、事务服务、安全服务和持久性服务。

# 11集中式数据库 VS 分布式数据库，两种如何实现拓展

集中式数据库架构是一个由处理器、与它相关的数据存储设备以及其他外围设备组成，它被物理地定义到单个位置，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操作。

分布式数据架构可以使用多个计算机系统的多个局部数据库系统组成，数据可以在多个不同的局部数据库中进行传送，由不同的数据库管理系统软件进行管理，运行在不同的计算机上，支持多种不同的操作系统，这些机器位于不同的地理位置并通过多种通信网络连接在一起，一个应用程序可以操作位于不同地理位置机器上的数据。

集中式数据库通过向上扩展提升，具体实现方式包括硬件扩容（增加CPU数量、内存容量、磁盘数量）和硬件升级（更换高端主机或高速磁盘）。

分布式数据架构通过向外扩展提升，具体的实现方式包括数据复制、数据垂直切分或水平切分、缓存和全文搜索。

# 11.1 补充“局部数据库+缓存”的读写分离 + 分布式 + 避免单点故障的建设方案

局部数据库+缓存的读写分离为：局部数据库写，缓存读

分布式：多个数据库系统

避免单点故障：热备份数据库系统

故，以上的建设方案为：

由多个局部数据库系统、多个热备份数据库系统、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，多个数据缓存负责为应用提供所读取的数据。

1. 读取操作：应用层访问缓存，若命中则返回，否则从局部数据库中读取并将数据加载至缓存后返回。
2. 添加操作：采用延迟加载策略，将数据直接写入局部数据库。
3. 修改操作：更改局部数据库中的数据，将缓存中的数据标记为无效。
4. 删除操作：删除局部数据库中的数据，将缓存中的数据标记为无效。

# 12 数据库规范化设计定义、反规范化设计定义、优点、缺点、实现技术

规范化设计：通过对数据库表的设计，降低数据库冗余程度，是数据库系统设计的重要技术。

反规范化设计：规范化设计后，数据库设计者希望牺牲部分规范化提高性能，这种从规范化设计回退的方法称为反规范化技术。

优点：

1.降低连接操作的需求

2.降低外键和索引的数目

3.提高查询效率

4.有可能减少表的数目

缺点：

1.数据重复存储，浪费磁盘空间

2.可能出现数据完整性问题

3.为保障数据一致性，增加了数据维护的复杂度

4.降低了修改速度

实现技术：

1.增加冗余列。在多个表中保留相同的列，通过增加冗余，减少或避免查询时的连接操作。

2.增加派生列。在表中增加可有本表或其他表中数据计算生成的列，减少查询的连接操作，避免计算或使用集合函数。

3.重新组表。如果许多用户需要查看两个表连接出来的结果数据，则把两个表重新组成一个表，减少连接提高性能。

4.水平切分表。根据一列或多列数据的值，把数据放在多个独立的表中，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在多个介质上时使用。

5.垂直切分表。将主键和部分列放在一个表中，主键和其他列放在另外一个表中，在查询时减少IO次数。

# 13 NoSQL有哪些潜在问题

1.现有产品不够成熟，大多数产品处于初创期

2.未形成一定标准，缺乏官方支持

3.对SQL的支持不够，学习和应用成本高

4.支持的特性不够丰富，现有产品提供的功能比较有限

# 14 ESB的定义、功能、优势

ESB是一个实现系统间集成和互联互通的重要技术架构，基于内容的路由与过滤，具备复杂数据的传输能力，可提供一系列标准接口。

主要功能有：

1.应用程序透明性

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.监控管理

优势：

1.在部署方面。能够实现灵活部署。

2.在集成系统耦合度方面。待集成系统只需和总线联系，彼此之间不需要互相通信，降低系统耦合度。

3.在可扩展性方面。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，系统可扩展性强。

# 15 企业集成方式有哪些

1.文件共享。一个应用产生提供信息的文件，由另外一个应用通过访问文件获取信息。适用于数据量大、交互少、即时性要求低的情况。

2.共享数据库。将应用程序的数据存储在一个共享数据库中，通过制定统一的数据库模式处理不同应用的集成需求。适用于实时性强、交互频繁、同步但不即时同步的情况。瓶颈在于：频繁读取修改相同数据，造成性能瓶颈；外部已封装好的应用只能采用自己的数据库模式，调整余地小。

3.过程调用。基于同步的方式，效率低，易失败。

4.消息传递。所有应用直接的消息通过消息队列实现，由消息队列保证数据传输的异步、稳定性。适用于数据量小、但交互频繁、可靠传输、异步传输、保证立即同步。

5.基于总线。总线是一个实现系统间集成和互联互通的重要技术架构，基于内容的路由与过滤，具备复杂数据的传输能力，可提供一系列标准接口。

6.基于工作流。引入工作流定义语言及其引擎动态，描述工具之间的协作关系。

7.面向服务的方式。把功能封装成服务，实现跨语言、跨平台访问。

8.界面集成。采用界面集成的方式和第三方工具集成，绕过工具内部的复杂处理逻辑。

9.数据集成。单表，将表包装成记录，采用主动记录的方式进行集成。多表，采用数据映射的方法完成数据集成处理。

# 16 EJB的定义、有什么元素、各自作用

EJB (Enterprise Java Beans) 是基于分布式事务处理的企业级应用程序的组件，实现业务逻辑和提供数据访问接口。

在开发分布式系统时, 采用EJB可以使得开发商业应用系统变得容易, 应用系统可以在一个支持EJB的环境中开发, 开发完之后部署在其它的EJB环境中, 随着需求的改变, 应用系统可以不加修改地迁移到其它功能更强、更复杂的服务器上。

Session Bean描述了与客户端的一个短暂的会话，是用于实现业务逻辑的，通过调用Session Bean的方法来访问部署在服务器上的应用程序，从而隐藏了内部复杂逻辑的执行过程。

Entity Bean 是持久化的数据组件，维持一行持久稳固的数据，用于实现O/R映射，负责将数据库中的表记录映射为内存中的Entity对象。

Message-Driven Bean 结合了session bean 和 JMS信监听者的功能，允许一个商业组件异步的接收JMS消息。

# 17 数据访问模式有哪些

1.在线访问。该模式是基本的数据访问模式，由业务程序直接读取数据，与后台数据源进行交互。

2.DAO模式是标准J2EE设计模式之一，该模式将底层数据访问操作与高层业务逻辑分离开。具体的DAO类包含访问特定数据源数据的逻辑。

3.Data Transfer Object。DTO是经典EJB设计模式之一。DTO本身是一组对象或是数据的容器，它需要跨越不同进程或者网络的边界来传输数据。

4.离线数据模型。是以数据为中心，数据从数据源获取后，将按照某种预定义的结构存放在系统中，成为应用的中心。其特点是离线，数据操作独立于后台数据源,与XML集成，数据可以方便地与XML格式文档相互转换。

5.对象/关系映射。ORM 是一种数据持久化技术，即在对象模型和关系型数据库之间建立起对应关系，并且提供一种机制，可通过 JavaBean 对象操作数据库表中的数据。

# 18 设计模式的定义及分类、实现方式

软件设计模式是被反复使用，大多数人知晓的软件代码设计的经验总结。设计模式提高了编码效率，增加了代码的可理解性，保证了代码的可靠型。

常用的软件设计模式包括创建型模式、结构型模式、行为型模式三大类，其中：

1. 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. Abstract Factory (抽象工厂)，提供一个创建一系列相关对象的接口，而无需指定它们具体的类。

首先定义抽象工厂接口，其中包含多种不同产品创建的抽象方法，针对不同产品定义对应的实际创建工厂，实际创建工厂继承抽象工厂接口，并实现其中方法。

1. Builder (建造者)，将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

首先定义抽象建造者类，定义各个部件的构造方法和产品建造后结果方法，针对不同的创建需求定义对应的实际建造者，实现抽象类定义的所有方法，并且返回一个组建好的对象。并定义指挥者类，负责安排已有模块的顺序，通知建造者开始建造。

1. Factory Method (工厂方法)，定义一个创建对象的接口，让其子类自己决定实例化哪一个，使其创建过程延迟到子类进行。

定义一个用于创建对象的工厂接口，声明工厂方法。子类实现工厂接口，重写工厂方法，返回实例对象，由子类决定实例化哪种实例对象。

1. Prototype (原型)，用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。

定义原型类，说明克隆自身的接口，然后针对不同的创建需求定义具体实现类，具体实现类继承原型类，并实现克隆自身的操作。

1. Singleton (单例)，保证一个类仅有一个实例，并提供一个访问它的全局访问点。
2. 结构型模式

主要用于处理类或对象的组合，为类设计以获得更大的结构提供指南。

1. Adapter (适配器)，将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

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。

1. Proxy (代理)，为其他对象提供一种代理以控制对这个对象的访问。
2. Facade (外观)，为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

定义一个外观类，包含各个子系统成员变量，为调用端提供统一的接口，并依据不同的请求调用对应子系统对象的方法。

1. Flyweight (享元)，运用共享技术有效地支持大量细粒度的对象。
2. Bridge (桥接)，将抽象部分与实现部分分离，使它们都可以独立的变化。

定义实现化角色接口，定义角色必需的行为和属性，针对不同的业务流程定义对应的实现化角色具体类，实现其方法属性。定义抽象化角色接口，定义角色的行为，同时保存一个对实现化角色的引用，定义修正抽象化角色继承抽象化角色接口，引用实现化角色对抽象化角色进行修正。

1. Composite (组合)，将对象组合成树形结构以表示"部分-整体"的层次结构，使得用户对单个对象和组合对象的使用具有一致性。
2. Decorator(装饰)，动态地给一个对象添加一些额外的职责。就增加功能来说，装饰器模式相比生成子类更为灵活。
3. 行为型模式

主要用于描述类和对象之间的交互以及职责的分配,对类之间交互及分配责任的方式提供指南。

1. Chain of Responsibility (职责链)，避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，沿着这条链传递请求，直到有对象处理它。
2. Command (命令)，将请求封装成对象，允许使用不同的请求对客户端参数化。
3. Interpreter (解释器)，给定一个语言，定义它的文法表示，并定义一个解释器解释语言中的句子。
4. Iterator (迭代器)，提供一种方法顺序访问一个聚合对象中各个元素, 而又无须暴露该对象的内部表示。
5. Mediator (中介者)，用一个中介对象来封装一系列的对象交互，使得各对象不需要显式地相互引用。
6. Memento (备忘录)，在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。

首先定义发起人Originator类，记录内部状态信息，提供创建备忘录和恢复备忘录数据的方法。声明备忘录Memeto类，存储发起人类的内部状态。声明管理者CareTaker类，提供对备忘录管理、保存和获取的方法。

1. Observer (观察者)，定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。
2. State (状态)，允许对象在内部状态发生改变时改变它的行为。
3. Strategy (策略)，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。
4. Template Method (模板方法)，定义一个算法的骨架，将一些步骤延迟到子类中。
5. Visitor (访问者)，主要将数据结构与数据操作分离。

# 19 SOA的定义和实现方式

SOA是一个组件模型，将应用程序不同功能单元封装为服务，通过定义好良好的接口和契约联系起来，接口采用中立的方式定义，不涉及底层编程接口和通信模型，使得系统中的服务可以以一种统一、通用的方式交互。

面向服务架构的常用实现方式包括webservice、ESB。

webservice包含服务提供者、服务请求者和服务注册中心

1、服务提供者通过发布的Web服务描述语言（WSDL），定义了服务的接口、入参和出参等信息，描述对外提供的服务功能；

服务提供者主要完成服务的设计、描述、定义和发布工作。在服务设计阶段，如何高效地对服务进行设计和定义是一个难题。为解决该问题，制定出基本的设计原则：需要对业务流程分析梳理，并综合考虑粗粒度、松耦合、自包含等特点，同时为了避免服务间通信量过大、交互频繁，需尽量减少了服务的数量。

2、服务请求者通过简单对象访问协议(SOAP)访问服务提供者，SOAP主要定义了服务之间如何交互以及消息格式；

服务的请求者即是服务的消费者，通过webservice技术调用服务。服务请求者通过服务注册中心可获取对应的服务接口、参数和返回值，之后利用webservice技术传递服务所需要的参数，进而调用该服务相关的运算、处理和分析。

3、服务注册中心一般采用UDDI实现，UDDI用于描述、发现与集成服务，服务提供者向UDDI发布服务，服务调用者查询UDDI上已发布的服务。

服务注册中心是连接服务提供者和服务请求者的纽带，服务提供者在此发布服务描述，服务请求者在此查找需要的服务。虽然在某些情况下服务注册中心是可选角色，但注册中心的存在可使服务提供者和服务消费者进一步解耦。

ESB是一个实现系统间集成和互联互通的重要技术架构，基于内容的路由与过滤，具备复杂数据的传输能力，可提供一系列标准接口。

# 20 SOA包含哪些服务类别

1. 交互服务。交互服务实现人与服务之间的交互功能。人可以是服务的消费者,也可以是服务的提供者。人员需要请求服务时，向连通服务发送消息请求，由连通服务查找服务，并将请求消息传递给服务提供者。

2.连接服务。连接服务又称连通服务，在完成服务的接入、服务间的通信和交互基础上,还提供安全性、可靠性、高性能的服务能力保障。

3. 业务服务。业务服务指为新建服务提供的特定运行支持环境。

4. 业务流程服务。流程服务是业务流程的运行环境，提供流程驱动、服务调用、事务管理等功能。

5. 信息服务。信息服务特指为上层应用系统、同层的其他服务等提供数据访问及资源访问服务。其目标是使应用系统能够统一、透明、高效地访问和操纵位于网络环境中的各种分布、异构的数据资源。

6. 协作服务。协作服务既可以实现组织之间的交互通信,也可以实现组织内部之间的交互通信。

顺口“交连业业，信息协作“

# 21 常见架构评估方法

常用的架构评估方法有：基于问卷调查的评估方式、基于度量的评估方式和基于场景的评估方式。

1. 基于问卷调查的评估方式是由多个评估专家通过调查问卷的方式回答问卷中的问题，对多个评估结果进行综合，最终得到最终结果。其评价的具有很大的主观性。
2. 基于度量的评估方式虽然评价比较客观，但是需要评估者对系统的架构有精确的了解。
3. 而基于场景的评估要求评估者对系统中等了解，评价比较主观，基于场景的评估方式又分为架构权衡分析法ATAM，软件架构分析法SAAM和成本效益分析法CBAM。

# 22 可靠度、可靠性、失效率的定义

可靠度：系统在规定条件下、规定的时间内不发生失效的概率。

失效率：系统运行至此刻未出现失效的情况下，单位时间系统出现失效的概率。

可靠性：软件系统在应用或者错误面前，在意外或错误使用的情况下，维持系统功能特性的基本能力。

可靠性的子特性有：成熟性、容错性、易恢复性、可靠性的依从性。

顺口：“可-容-成-恢“

# 23 提高可靠性的技术和方式

1.N版本程序设计

N版本程序设计是一种静态的故障屏蔽技术，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。

2.恢复快方法

恢复块方法是一种动态的故障屏蔽技术，采用后向恢复策略，它提供具有相同功能的主块和几个后备块，一个块就是一个执行完整的程序段，主块首先投入运行，结束后进行验证测试，如果没有通过，系统经现场恢复后由一个后备块运行。

3.冗余设计

动态冗余，通过故障检测、故障定位、故障恢复达到容错的目的，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代替它重新运行。备用模块在其待机时，可与主模块一样工作或不工作，前者为热备份，后者为冷备份。

4.双机热备或集群系统

5..防卫式程序设计

防卫式程序设计通过在程序中包含错误检查代码和错误恢复代码，使得一旦错误发生，程序能撤销错误状态，恢复到一个已知的正确状态中去。其实现策略包括错误检测、破坏估计和错误恢复3个方法。

# 24检错技术的优缺点、实现方式、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。

检错技术的缺点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：

1、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

2、计算运行时间，如果时间超出预期时间，则进行异常处理

3、状态标志位法，根据实际情况使用

检错技术的处理方式，大多采用“查处故障-停止软件运行-报警”的处理方式。但根据故障的不同，也可采用不停止或部分停止系统运行的情况，一般由是否需要实时处理决定。

# 25 软件和硬件可靠度的各自特点

硬件可靠度：

1.失效率服从浴缸曲线，老化状态类似于软件调试状态

2.即使不使用，材料劣化也会导致失效

3.硬件维修会恢复原始状态

4.硬件失效之前会报警

软件可靠性：

1.不考虑软件演化时，失效率在软件上是非增的

2.如果不使用该软件，永远不会发生失效

3.软件维护会创建新的代码

4.软件失效前很少报警

顺口“使用维修报警+浴缸曲线“

# 26 uml图中类图与用例图都有什么关系

用例图关系有三种：

1、包含(使用)关系：include，用例可以简单地包含其他用例具有的行为。

2、扩展关系：extend，指在一定条件下，把新的行为加入到已有的用例中，其中获得的新用例称为扩展用例。

3、泛化关系：强调父子关系

类图关系有六种：

1、依赖关系。如果类A用到类B，但是和B的关系不是太明显的时候，就可以把这种关系看作是依赖关系。

2、关联关系。指类和类之间的连接，它使一个类知道另一个类的属性和方法。

3、聚合关系。聚合关系体现的是整体与部分的拥有关系。整体与部分之间是可分离的，它们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享，

4、组合关系。组合关系体现整体与部分间的包含关系，整体与部分是不可分的，部分也不能给其它整体共享，作为整体的对象负责部分的对象的生命周期。

5、泛化关系。泛化关系指类与类之间的继承关系。

6、实现关系。实现关系是指接口及其实现类之间的关系。

# 27 uml图的定义、动态图、静态图、各自分类和定义

UML是Unified Model Language的缩写，中文是统一建模语言，是由一整套图表组成的标准化建模语言，由构造块、公共机制、架构三部分组成，其中构造快由建模元素、关系、图组成。

动态图（7种）。动态图描述系统的动态模型和组成对象间的交互关系。

1、用例图。由参与者、用例、边界及它们之间的关系构成描述系统功能的视图，用于对系统的功能行为建模。

2、顺序图。序列图显示具体用例的详细流程，显示了流程中中不同对象之间的调用关系。

3、通信图。描述了接发消息的对象的组织关系，强调对象之间的合作关系。

4、时序图。时序图用来显示随时间变化，一个或多个元素的值或状态的更改。

5、状态图。用于描述一个对象在其生存周期间的动态行为，表现一个对象所经历的状态序列、引起状态转移的事件，以及状态转移伴随的动作。

6、活动图。用于描述系统的工作流程和并发行为，活动图中一个活动结束后立刻进入下一个活动。

特殊的，泳道活动图：将一个活动图分组，每个组表示一个特定的类或对象，它们负责完成组内的活动。每个活动都明确属于一个泳道，不可跨越泳道，而转移可以跨越泳道。

7、交互概览图。一个交互概览图是活动图的一种形式，它的节点代表交互图。

静态图（7种），静态图显示了系统的静态结构，特别是存在事物的种类的内部结构、相互之间的联系。

1、类图。展示模型的静态结构，描述类、接口、协作以及他们之间关系。

2、对象图。对象图表示在某一时刻一组对象以及它们之间关系。

3、包图。包图由包和包之间的关系构成，它是维护和控制系统总体结构的重要建模工具。

4、构件图。描绘了系统中的构件接口，以及它们之间的关系。

5、部署图。部署图表示该软件系统如何部署到硬件环境中。

6、制品图。展示了一组制品以及其间依赖关系。利用制品图可以对系统的静态实现视图建模。

7、组合结构图。描述了一个"组合结构"的内部结构，以及他们之间的关系。

# 28“4+1”视图定义，项目中如何实现的视图

“4+1”视图模型从5个不同的视角来描述软件架构，每个视图只关心系统的一个侧面，5个视图结合在一起才能反映系统的软件结构的全部内容。

这5个不同的视角包括逻辑视图、开发视图、进程视图、物理视图和场景视图。

1.逻辑视图。逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。使用UML中的类图描述逻辑视图。——>系统有哪些功能，拆分成哪些类

2. 实现视图。开发视图也称为模块视图、开发视图，侧重于软件模块的组织和管理，考虑软件内部的需求。使用UML中的实现图来来建模。

3. 进程视图。进程视图侧重于系统的运行特性以及逻辑视图中的功能抽象如何适应进程结构等，关注一些非功能性需求。

4. 物理视图。物理视图考虑如何把软件映射到硬件上，解决系统拓扑结构、系统安装和通信等问题。使用UML中的部署图来建模。

5. 场景。场景可以看作是系统中重要系统活动的抽象，它使4个视图有机联系起来。场景使用UML中的用例图建模。

部署视图

1 Docker部署：

    系统微服务采用分布式的部署方法，每颗卫星的处理服务单独编译、打包，连同运行环境一起封装成Docker容器，根据实际情况可在同一台物理机器或多台物理机器上同时部署多个实例。

2 服务注册：

    服务启动后会将自身信息注册到已部署好的分布式服务注册中心，所有请求服务通过服务注册中心来进行服务发现，然后通过反向代理的方式调用具体卫星的处理服务。

3 网络隔离：

    系统内部网络与外部网络分属于不同的逻辑网络，以实现系统内部与外部网络的隔离。

4 负载均衡：

    在分布式部署多个服务的基础上，通过负载均衡服务器做统一代理访问，每当请求来临时，任务分发单元会将任务平滑分配给相对空闲的微服务节点，这样的部署方法以廉价且透明的方式扩展了服务器和网络的带宽，可以大大提升系统的并发量，同时保证系统整体的稳定性和可靠性。

# 29 信息系统面临的安全威胁

1.物理安全威胁：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、设备被盗等造成数据丢失或信息泄漏。

2.通信链路安全威胁：指在传输路线上安装窃听器装置或对通信链路进行干扰。

3.网络安全威胁：指由于因特网的开放性、国际性、无安全管理性，对内部网络形成的严重安全威胁。

4.操作系统安全威胁：指操作系统本身的后门或安全陷阱，如木马等。

5.应用系统安全威胁：指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。

6.管理安全威胁：指人员管理和各自安全管理制度。

# 30五大类安全服务和实现方式

1.认证与鉴别服务。鉴别参与通信的对等实体和数据源的合法性，认证服务提供了实体声称其身份的保证，只有在主体和验证者的关系背景下，认证才有意义。认证服务的基本目的，是为了防止其他实体占用和独立操作被认证实体的身份。

2.访问控制服务。访问控制服务决定开放环境中允许使用哪些资源、在什么地方适合被授权访问的过程。在访问控制实例中，访问可以是对一个系统或对系统内部进行。

3.数据保密性服务，对数据提供保护，防止数据未经授权而被泄漏，防止在系统之间交换数据时被截取。主要通过禁止访问和加密机制两种方式实现。

4.数据完整性服务。防止系统之间交换数据，非法修改数据或丢失数据。常见实现方式包括阻止对数据传输媒介访问的机制和探测对数据非授权修改的机制。

5.抗抵赖服务。阻止通信双方否认发送和接收数据的行为，实现方式主要包括数字签名、用户认证、操作日志等技术。

# 31安全属性有什么

1.保密性：指系统中的信息必须按照信息拥有者的要求保证一定的秘密性，不会被未经许可的第三方非法窃取。

2.完整性：指系统中的信息应当安全、准确、有效，要求数据不能被非法改动或删除。

3.真实性：指对信息发送者身份的确认，保证信息可信度。

4.不可抵赖性：指保护信息赖以存储的节点、介质、载体等不被盗用或窃取。

# 32认证服务有哪些

1.口令和用户名认证，通过客户端和服务器共知的口令进行验证。

2.令牌认证：进行验证的密钥存放在令牌中，常用令牌有安全证书、智能卡等方式。

3.生物识别认证：根据认证者的图像、指纹、气味和声音作为认证依据。

4.相信可靠的第三方建立的认证。

5.环境，如主机地址等

# 33访问控制方式有哪些

1.自主访问控制（DAC)。一个拥有一定权限范围的主体可以直接或者间接地把权限授予其他的主体。

2.访问控制列表(ACL)，从资源角度出发，允许某些主体可访问。与自主访问的区别在于：自主访问是给主体赋予权限，访问控制列表为某个资源赋予哪些主体可访问的列表。

3.强制访问控制（MAC)，强制访问控制是为主体和客体划分等级，而且强制所有主体都必须服从这种访问权限的分配。 MAC 一般用在安全级别层次比较多的军事、安全等特殊应用领域中。

4.基于角色的访问控制(RBAC)。用户通过角色与权限进行关联，将用户和权限解耦，一个用户可有多种角色，一个角色可有若干权限，减少了授权管理的复杂性，是面向企业的系统中一种十分有效的访问控制策略。

5.基于任务的访问控制（TBAC）。参与某个任务的相关人员可拥有相应权限，任务灵活配置。常用于OA系统。

# 34基于口令的认证方式VS基于公钥的认证方式

1.基于口令的认证方式实现简单，但是由于口令复杂度及管理方面的原因，容易受到认证攻击；而基于公钥的认证方式，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比更加安全。

2.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私密信息；而基于公钥的认证方式中，可以基于用户私钥对私有数据加密保护。

# 35数据库加密方式

加解密API：数据库管理系统提供在SQL语句中调用加解密API，应用可利用这些API构建自己的基础架构，对数据进行加密保护。----------> 灵活性强，但构建和管理复杂

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要调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。----------> 管理简单、应用程序无负担，但灵活性差

# 36访问控制-基于角色的访问控制策略和可拓展访问控制标记语言(XACML)

基于角色的访问控制策略，指用户通过角色与权限进行关联，将用户和权限解耦，一个用户可有多种角色，一个角色可有若干权限，减少了授权管理的复杂性，是面向企业的系统中一种十分有效的访问控制策略。

1.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和细粒度的访问控制需求，适合企业复杂业务。

2.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分布管理访问控制权限，相比之下，需对访问控制权限集中管理的MAC，并不适用于分布式系统。

# 37常用安全手段及实现过程

1、对称加密，加密速度快，但密钥强度不高、密钥分发困难，适用于大内容传输；DES、3DES、AES、RC-5、IDEA

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。

2、非对称加密 - 接收方公钥加密，接收方私钥解密；RSA、ECC

发送者利用接收者的公钥对要发送的数据进行加密，只有拥有对称私钥的接收者才能将数据正确解密。

3、数字信封 - 对称加密和非对称加密的混合使用，发送方将原文用对称密钥加密，将对称密钥以接收方公钥加密的方式发送。接收方接收到数字信封，用自己的私钥解密，取出对称密钥解密。

4、信息摘要。

发送者根据要发送的数据生成信息摘要，利用对称密钥对消息摘要加密并附到数据发送。接收者使用相同密钥解密，根据收到的数据重新生成信息摘要，比较两个摘要是否相同，以验证数据完整性。

发送者根据要发送的数据生成信息摘要，利用自己的私钥对信息摘要加密，并附到数据上发送。接收者利用对方公钥将信息摘要解密，并根据收到的数据重新生成信息摘要，比较两个摘要是否相同，以验证数据完整性。

常用的信息摘要算法：MD5、SHA等。

5、数字签名。

发送者根据要发送的数据和签名算法生成数字签名，利用对称密钥对数字签名加密并附到数据发送。接收者使用相同密钥解密，对其中的数字签名进行验证，确认其合法性。

发送者根据要发送的数据和签名算法生成数字签名，利用自己的私钥对数字签名加密，并附到数据上发送。接收者利用对方公钥将数字签名解密，对其中的数字签名进行验证，确认其合法性。

# 38授权侵犯的定义、防止授权侵犯和保留用户痕迹的技术手段

授权侵犯：也称为内部攻击，指被授权以某一目的使用某一资源的某人，将此权限用于其他非授权的目的。

防止授权侵犯和保留用户痕迹的技术是抗抵赖框架。

抗抵赖框架的内容：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。

抗抵赖服务目的：提供有关特定事件或行为的证据。

抗抵赖服务可保护的行为：发送X.400消息、数据库插入记录、请求远程操作等，事件或行为本身以外的其他实体可请求抗抵赖服务。

抗抵赖框架包括四个阶段：

1.证据的生成：证据生成请求者请求证据生成者为事件生成证据。卷入事件的实体称为证据实体，其卷入关系由证据建立。根据抗抵赖服务的性质，证据可由证据实体、可信第三方服务一起生成；或单独由第三方服务生成。

2.证据传输、存储和恢复：证据在实体间传输或从存储器取出来或传到存储器。

3.证据验证：证据在证据使用者的请求下被证据验证者验证。本阶段的目的是出现纠纷的事件中，让证据使用者确信被提供的证据是充分的，可信第三方也可参与，以提供验证该证据的信息。

4.解决纠纷：仲裁者有解决双方纠纷的责任。

# 39 SQL设计原则

1.查询尽量不返回不需要的行、列

2.多表查询时，使用连接查询，避免使用子查询

3.避免使用NOT IN、LIKE、NOT LIKE等

4.避免使用DISTINCT关键字

# 40数据持久层定义及优点，设计时应该考虑的问题

数据持久层是一组软件服务，将应用程序与该程序所使用的数据源分离，为整个项目提供一个统一的、安全的、并发的数据持久机制。

数据持久层的优点：

1.程序代码重用性强，即便更改数据库，只需修改配置文件，不必重写代码程序

2.业务逻辑代码可读性强，不会有大量SQL

3.持久化技术可自动优化，减少对数据库访问，提高效率

4.通过OR映射，向业务逻辑提供面向对象的访问

5.简化开发工作，关注于业务逻辑开发

在持久层设计阶段需要考虑的问题包括：

1.网络流量问题；

2.返回结果集的问题；

3.查询或锁定超时的问题；

4.应用程序开发工具的问题；

5.使用游标的问题；

6.应用层设计的问题等。

# 41数据访问层定义、为什么要增加数据访问层

数据访问层可以使用户便捷访问持久性数据，向用户隐藏了操作数据的复杂性。

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，此时需要单独的数据访问层屏蔽差异性。

2.数据访问复杂性增加，不宜与业务逻辑混合在一起

3.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。

# 42数据在线访问的优缺点

优点：

1、性能比直接SQL好

2、可以处理复杂查询语句

缺点：

1、要求程序员懂SQL语句

2、修改与维护相对困难

# 43数据库事务的实现方式

1、原子性。数据库中实现原子性和持久性最简单的策略是“影子拷贝”，该策略假定某个时刻只有一个活动的事务，首先对数据库做副本，所有的写操作都在数据库副本上执行，而保持原始数据库不变，若任一时刻操作终止，只需要删除副本即可，原始数据不受影响。

2、一致性。常见完整性约束检查保持一致性。

3、隔离性。两段锁协议是实现隔离性的常见方案。通过定义锁的增长、收缩两个阶段约束事务的加锁、解锁过程，保证事务串行化执行，但由于事务不能一次性获得所有的锁，有可能死锁。

4、持久性。所有的数据库修改操作均在日志中记录，事务的写操作等到事务提交后才会执行。日中中记录事务开始、事务结束时间、以及事务修改的新值。

# 44应用服务器定义及如何保证系统稳定性

应用服务器是指通过各种协议把商业逻辑暴露给客户端的程序。

1.若系统负荷很大，可部署多台应用服务，多台应用服务分担任务，以达到性能要求。

2.应用服务器可通过灵活增加服务器完成扩展

3.应用服务器可长时间稳定运行。当一台应用服务器出现故障时，可将当前运行的事务转移到正常应用服务器上完成执行，不影响业务正常运行，从而保证高可靠和稳定性。

# 45 ORM和SQLMapping定义及优缺点

ORM对象关系映射，在对象模型和关系型数据库之间建立起对应关系，并且提供一种机制，可通过 JavaBean 对象操作数据库表中的数据。使得程序员在开发过程中仅仅面对一个对象的概念，降低了对程序员数据库知识的要求，简化了数据库相关的开发工作。

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(不写SQL导致性能低，不能处理复杂语句)

1、不太容易处理复杂查询语句。

2、性能较直接用SQL差。

SqlMapping，致力于POJO和SQL语句之间的映射，将接口和 Java对象映射成数据库中的记录。

优点：

1、入门简单，即学即用，提供了数据库查询的自动对象绑定功能，而且延续了很好的SQL使用经验，对于没有那么高的对象模型要求的项目来说，相当完美。

2、可以进行更为细致的SQL优化，可以减少查询字段。

缺点

1、框架还是比较简陋，功能尚有缺失，不支持数据复杂、数据对象聚合、继承等关系

2、仍需要使用SQL，不利于移植数据库

# 46 非功能性需求包括什么

1.操作性需求。操作性需求指定了系统完成任务所需的操作环境及其可能的改变。操作性需求包括**技术环境需求、系统集成需求、可移植性需求、可维护性**需求。

2.性能需求。性能需求的核心是性能问题,如响应时间、容量和可靠性。每一个需求必须是可测量的,才能进行基准比较,进而检验性能需求的成果。性能需求包括**速度需求、容量需求、可用性与可靠性**需求。

3.安全需求。安全性是防止信息系统崩溃和数据丢失的能力。新系统的开发人员必须保证系统的安全需求,提供合理的预防措施以防止问题发生。安全性需求包括访问控制需求、加密与验证需求、病毒控制需求。

4.文化与政治需求。文化与政治需求是指针对使用系统的不同国家所特有的需求。文化与政治需求包括**多语种需求、用户定制需求**、未申明的术语、法律需求。

# 47 常用的软件设计方法

1. 模型驱动设计。

模型驱动设计是一种系统设计方法，强调通过绘制图形化系统模型描述系统的技术和实现。通常从模型驱动中-分析逻辑模型-导出系统设计模型，最终系统设计模型将作为构造和实现新系统的蓝图。

2.原型设计。

原型法是一种反复迭代过程，它需要设计人员和用户之间保持紧密的工作关系，通过构造一个预期系统的小规模的、不完整的但可以工作的示例，来与用户交互设计的结果。

3.快速应用开发。

快速应用开发是一种各种结构化技术与原型化技术联合的应用开发技术，用以加速系统的开发。快速应用开发要求反复的使用结构化技术和原型技术来定义用户的需求并设计最终系统。

4. 结构化设计。

它是一种面向过程的系统设计技术，它将系统过程分解成一个容易实现和维护的计算机程序模块。一个模块就是一组指令，模块需要满足高内聚松散耦合的特征。基本方法是跟踪系统的数据流,研究问题域中数据如何流动以及在各个环节上进行何种处理,从而将问题域映射为数据流、加工以及数据存储等元素并组成数据流图。结构化建模方法所绘制的模型称为数据流图，可用于分析一个现有的系统以及定义新系统的业务需求，适合流程较为稳定的系统。

5. 信息工程法。

信息工程是一种用来计划，分析和设计信息系统的模型驱动、以数据为中心的但对过程敏感的技术，其核心是构识别实体及其关系。实体用于描述问题域中的一个事物,它包含一组描述事物数据信息的属性:关系描述问题域中的各个事物之间在数据方面的联系,它可以带有自己的属性。信息工程建模方法所创建的模型被称为实体联系图，强调在分析和研究过程需求之前，首先研究和分析数据需求。

6. 面向对象分析方法。

从面向对象设计领域发展而来,它通过对象对问题域进行完整的映射,对象包括了事物的数据属性和行为特征;用结构和连接如实反映问题域中事物之间的关系,通过封装、继承和消息机制等使问题域的复杂性得到控制。面向对象建模方法所创建的模型被称为对象模型。随着面向对象技术不断的发展和应用，形成了面向对象建模的标准，即UML。UML定义了几种不同类型的模型图，这些模型图以对象的形式共建一个信息系统或应用系统。

# 48 软件系统中常用的建模方法

1. 功能分解法

功能分解法以系统需要提供的功能为中心来组织系统。首先定义各种大的功能,然后把功能分解为子功能,同时定义功能间的接口。比较大的子功能还可以被进一步分解,直到我们可以对它进行明确的定义。

2. 结构化分析建模方法

它是一种面向过程的系统设计技术，它将系统过程分解成一个容易实现和维护的计算机程序模块。一个模块就是一组指令，模块需要满足高内聚松散耦合的特征。基本方法是跟踪系统的数据流,研究问题域中数据如何流动以及在各个环节上进行何种处理,从而将问题域映射为数据流、加工以及数据存储等元素并组成数据流图。结构化建模方法所绘制的模型称为数据流图，可用于分析一个现有的系统以及定义新系统的业务需求，适合流程较为稳定的系统。

3. 信息工程建模法

信息工程是一种用来计划，分析和设计信息系统的模型驱动、以数据为中心的但对过程敏感的技术，其核心是构识别实体及其关系。实体用于描述问题域中的一个事物,它包含一组描述事物数据信息的属性:关系描述问题域中的各个事物之间在数据方面的联系,它可以带有自己的属性。信息工程建模方法所创建的模型被称为实体联系图，强调在分析和研究过程需求之前，首先研究和分析数据需求。

4. 面向对象建模法

从面向对象设计领域发展而来,它通过对象对问题域进行完整的映射,对象包括了事物的数据属性和行为特征;用结构和连接如实反映问题域中事物之间的关系,通过封装、继承和消息机制等使问题域的复杂性得到控制。面向对象建模方法所创建的模型被称为对象模型。随着面向对象技术不断的发展和应用，形成了面向对象建模的标准，即UML。UML定义了几种不同类型的模型图，这些模型图以对象的形式共建一个信息系统或应用系统。

# 49 ATAM架构评估步骤

ATAM是一种基于场景的软件架构评估方法，在 SAAM 的基础上发展起来的，主要针对性能、实用性、安全性和可修改性，对系统的多个质量属性基于场景进行评估，对这些质量属性进行评价和折中。

在使用ATAM进行架构评估时，我方根据项目需要成立了项目评估小组。其主要成员包括：评估小组负责人、项目决策者、架构设计师、系统运维人员、开发人员、测试人员、系统部署人员等项目干系人。架构的评估经历了描述和介绍阶段、调查和分析阶段、测试阶段和报告阶段四个阶段，下面将详细论述这四个阶段的具体实施过程。

1、描述和介绍阶段（介绍ATAM、业务驱动因素、介绍体系架构）

在描述和介绍阶段，由于项目评估成员有部分人员对ATAM并不熟悉，由我首先介绍ATAM的方法。ATAM是一种基于场景的软件架构评估方法，在 SAAM 的基础上发展起来的，主要针对性能、实用性、安全性和可修改性，对系统的多个质量属性基于场景进行评估，对这些质量属性进行评价和折中。单位领导作为项目决策者，再次重申地面处理系统的目的是确保地面处理一系列流程能够高效自动化运行，满足多星大规模计算任务的并行调度。而后，我作为系统架构设计师，介绍项目采用的整体架构，系统基于J2EE，采用微服务架构，系统主要分为三大模块：数据处理模块用以模拟卫星数据生产模型，自动化运行处理；数据归档模块提供数据归档存储及生命周期管理工作；数据分发模块用于交付用户订单，以专线形式推送。

2、调查和分析阶段（确定架构方法、生成质量属性效用树、分析体系结构方法）

在调查分析阶段，不同的需求方基于各自的场景都提出了各自的要求。系统运维人员表示系统必须是高可用的，且各个卫星间不能干扰，一颗卫星不工作时不可影响其他卫星。系统开发人员提出卫星载荷等信息是市场变动的，且要能横向增加新的卫星处理流程，系统安全人员提出系统整体部署环境已用网闸与外界隔绝，且系统使用人员均为单位内部人员，安全性要求较低。针对这些场景我们创建了质量属性效用树，并分析了项目开发过程中的风险点、敏感点和权衡点。

3、测试阶段（头脑风暴优先场景、分析架构方法）

经过评估小组集体讨论，确定了不同场景的优先级如下：系统的可用性最高，性能次之，可修改性再次，安全性优先级较低。为保证系统可用性，以及各星部署的独立性，系统的各个模块采用容器集群独立部署，各个星皆有一套容器集群，保证各星流程及操作的独立隔绝性。

4、报告阶段

最终，在报告阶段形成了评估报告，经过对架构的评估，确定了系统的风险点、敏感点、权衡点和非风险点，最后以文档的形式表现。其包括的内容有：架构分析方法文档、架构的不同场景及各自的优先级、质量效应树、风险点决策、非风险点决策及每次的评估会议记录。

# 50 ABSD步骤

需要经历架构需求、架构设计、架构文档化、架构复审、架构实现和架构演化六个阶段：

1、架构需求阶段。需要明确用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。其主要活动包括需求获取、标识构件和架构评审。

2、架构设计阶段。利用架构需求生成并调整架构决策。主要活动包括提出架构模型、将已标识的构件映射到架构中、分析构件之间的相互作用、产生系统架构和架构设计评审。

3、架构文档化。 架构文档化的主要活动是对架构设计进行分析与整理，生成架构规格说明书和测试架构需求的质量设计说明书。

4、架构复审。经过以上架构分析后，项目主版本软件架构分析已经确定，邀请外部专家参加架构复审。

5、架构实现。架构实现主要是对架构进行实现的过程。主要活动包括架构分析与设计、构件实现、构件组装和系统测试。

6、架构演化阶段。架构演化阶段主要解决用户在系统开发过程中发生的需求变更问题。体系结构演化是使用系统演化步骤去修改应用，以满足新的需求。主要包括以下六个步骤：

a.需求变动归类。首先必须对用户需求的变化进行归类，使变化的需求与已有构件对应。对找不到的构件的变动，也要做好标记，在后续工作中，将创建新的构件，以对应这部分变化的需求。

b.制定体系结构演化计划。在改变原有结构之前，开发组织必须制订一个周密的体系结构演化计划，作为后续演化开发工作的指南。

c.修改增加或删除构件。在演化计划的基础上，开发人员可根据在第1步得到的需求变动的归类情况，决定是否修改或删除存在的构件、增加新构件。最后，对修改和增加的构件进行功能性测试。

d.更新构件的相互作用。 随着构件的增加、删除和修改，构件之间的控制流必须得到更新

e.构件组装与测试。通过组装支持工具把这些构件的实现体组装起来，完成整个软件系统的连接与合成，形成新的体系结构。然后对组装后的系统整体功能和性能进行测试。

f.技术评审。对以上步骤进行确认，进行技术评审。评审组装后的体系结构式否反映需求变动，符合用户需求。如果不符合，则需要在第b到第f步之间进行迭代。

# 51 DSSA步骤

领域工程分为3个主要阶段，领域分析、领域设计和领域实现。通常这三个阶段是顺序进行的，同时领域工程是一个迭代的、逐渐精化的过程，在实现领域工程的每个阶段时，都有可能返回以前的步骤，对以前步骤得到的结果进行修改和完善，再回到当前步骤，在新的基础上进行本阶段的行为。

1、领域分析是领域工程的第一个阶段，这个阶段的主要目标是获得对于目标领域的问题域和系统责任的认识，并将这种认识显式地表现出来，获得领域分析模型。领域分析模型描述了领域中系统之间的共同需求。

2、领域设计的目标是获得特定领域的软件架构DSSA，它针对领域分析模型中表示的需求给出解决方案，它不是单个系统的表示，而是能够适应领域中多个系统需求的高层次设计。由于领域分析模型的领域需求具有一定的变化性，DSSA也要相应地变化。设计领域包括初步领域设计、结合设计模式实现变化性、建立设计模型与分析模型间的追踪性，其中在设计层实现领域中的变化性是领域设计的重点。

3、领域设计完成后，我们得到了领域的DSSA，其中包含了若干构件的规约。领域实现的主要工作是根据领域分析模型和DSSA，实现领域中的架构和可复用构件。在领域实现阶段，首先对DSSA和构件进行了详细设计，然后进行DSSA和构件的实现。在实现时，采用的主要方法是从现有系统中提取出一些系统成分，依据详细设计进行修改，作为实现级DSSA，同时另外一部分构件依据详细设计重新开发。

# 52 需求获取技术有哪些

1.抽样技术：对现有文档、表和文件进行抽样；

2.调研和实地访问；

3.观察工作环境：通过观察用户的活动、现场及工作习惯来了解项目需求；

4.问卷调查：通过使用调查表收集信息。调查表有两种格式：自由格式和固定格式;

5.面谈：通过直接、面对面的交互获取需求。

6.JRP （联合需求计划），是一个通过高度组织的群体会议来分析企业内的问题并获取需求的过程，它是联合应用开发（JAD）的一部分。

# 53软件可靠性设计的定义、设计原则、影响可靠性的因素有哪些

可靠性设计是在常规的软件设计中，应用各种方法和技术使程序设计在兼顾用户功能和性能需求的同时，全面满足软件的可靠性要求。软件可靠性设计技术就是以提高和保障软件的可靠性为目的，在软件设计阶段运用的一种特殊的设计技术。

影响软件可靠性的主要因素有：

1.运行环境

2.软件规模

3.软件内部结构

4.软件的开发方法和开发环境

5.软件的可靠性投入等

可靠性设计需要遵循的原则有：

1.软件可靠性设计是软件设计的一部分，必须在软件的总体设计框架中使用，并且不能与其他设计原则相冲突。

2.软件可靠性设计在满足提高软件质量要求的前提下，以提高和保障软件可靠性为最终目标。

3.软件可靠性设计应确定软件的可靠性，不能无限扩大，并且排在功能、用户需求、开发费用之后考虑。

# 54可靠性分析方法

在软件可靠性设计之前和软件可靠性设计过程中,都需要采用软件可靠性分析方法,来确定当前系统中的主要可靠性因素和目标。

常见的软件可靠性分析方法包故障树分析方法、失效模式与效应分析方法等。

故障树分析方法：一种自顶向下的软件可靠性分析方法,特别是对人员和设备的安全及可靠性产生重大影响的事件开始,向下逐步追查导致顶事件发生的原因,直至基本事件,从而确定软件故障原因。基本的步骤是软件故障树的建立、定性分析和定量分析。

失效模式与效应分析方法：在软件开发阶段的早期，通过识别软件失效模式，研究分析各种失效模式产生的原因,寻找消除和减少其有害后果的方法，以便尽早发现潜在的问题，并采取相应的措施，从而提高软件的可靠性和安全性。

# 55数据库引入主从复制机制的优势

1、提升性能

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，避免单点故障，主服务器实时、异步复制数据到从服务器，当主服务器宕机时，可以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。

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。

# 56为什么要用Memcached缓存代替数据库查询缓存

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，可拓展性更好。

2.缓存有效性：数据库查询缓存只要发生写操作就会失效，即使更新的是数据库中的其他行。而Memcached可通过键值对将数据散列缓存，有效降低缓存的更新频率，从而提高缓存有效性。

3.缓存数据类型：数据库查询缓存只能缓存数据库行，而Memcached理论上可以缓存任何内容，可将分散在数据库的关系或列表组合后缓存，提高缓存针对性和效率。

# 57 关系型数据库VS文件系统

关系型数据库

1、数据结构需要符合关系模式，设计难度大

2、遵守数据库范式，数据冗余少

3、以数据库为中心，组织管理数据

4、数据独立于应用系统，很容易在不同应用系统之间共享数据

文件系统

1、针对特定应用系统设计，难度小

2、可能在多个文件中复制相同的数据属性，数据冗余较大

3、以应用系统为中心，组织管理数据

4、符合特定应用系统需求的文件数据源，在不同应用系统之间较难共享

# 58内存数据库VS关系数据库

内存数据库

1、键值对模式

2、内存直接读写，性能相对较高

3、基于内存存储，存储容量受限

4、恢复机制复杂，可靠性低

关系数据库

1、关系模式

2、外存读写，性能相对较低

3、基于磁盘存储，存储容量大

4、内建恢复机制，可靠性较高

# 59画出J2EEN层架构模型

# 60 PHP VS Java

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，持久化层、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。

5、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。

# 61 什么是响应式web设计，及实现方式

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调整页面的布局，以提供用户可感知、流畅的阅读和操作体验。

实现方式有：流式布局、弹性布局、媒体查询。

# 62工厂设计模式优点、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，数据访问层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。

优点是可以非常方便的创建一系列的对象。

使用场景是创建系列对象的情况。

在数据访问层中，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。

# 63 分布式缓存定义

分布式缓存指的是在高并发环境下，为减轻数据库压力和提高系统响应时间，在数据库系统和应用系统之间增加的独立缓存系统。

# 64 Redis分布式存储的两种方案、Redis集群切片的常见方式

Redis分布式存储常见方案：

1、主从模式

2、哨兵模式

3、集群模式

Redis集群切片常见方式：

1、客户端切片，客户端就key的哈希值对应到不同的服务器

2、中间件切片，在应用软件和Redis之间，由中间件实现服务到后台Redis节点的路由分派。

3、客户端服务端协作分片。客户端采用一致性哈希，服务端提供错误节点的重定向服务，对应到不同的服务器。

# 65Memcached缓存 VS Redis缓存，Memcached缓存为什么会发生可靠性和一致性问题

Memcache:

1、数据类型：key-value

2、不支持持久化

3、分布式存储方式为客户端哈希分片、一致性哈希

4、支持多线程

5、支持内存管理，有私有内存池

6、不支持事务

Redis：

1、数据类型有key-value，list、set、hash等多种数据结构

2、支持持久化

3、多种分布式存储方式，主从、哨兵、集群

4、不支持多线程

5、无内存管理

6、有限支持事务

Memcache缓存可靠性和一致性问题：

1、MemCache没有持久化功能，掉电后数据全部丢失，且无法恢复，存在可靠性问题。

2、MemCache不支持事务，操作过程中存在数据不一致问题。

# 66如何实现数据交互安全性需求

1、引入https协议或采用加密技术对数据加密后传输

2、采用信息摘要技术对重要信息进行完整性验证

3、防火墙系统

4、安全检测

5、网络扫描

# 67 信息工程方法中的“实体”和面向对象方法中的“类”之间不同

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。

# 68 面向对象方法中的“用例”可分为什么层次

用例按照层次，可以分为“Essential Use Cases”抽象用例、“Real Use Cases”基础用例。

抽象用例描述用例的本质属性，与如何实现这个用例无关，独立于实现该用例的软硬件技术。用于分析阶段。

基础用例描述用例的实现方式，表达设计和实现用例时采用的方法技术，用于设计阶段。

# 69 什么是SQL注入攻击，常见抵御SQL注入攻击方式。

SQL注入攻击是通过把SQL命令插入到web表单提交，或者输入请求的查询字符串，最终欺骗服务器执行恶意的SQL命令。

常见抵御SQL注入攻击方式有：

1、使用正则表达式

2、使用参数化的过滤性语句

3、检查用户合法输入

4、用户相关数据加密处理

5、存储过程执行查询（不会暴露表结构，安全）

6、使用专业的漏洞扫描工具

（正则-过滤-加密-工具-存储过程-合法输入）

# 70 构件分类方法

1、关键字分类法：根据领域分析的结果，将应用领域的概念按照从抽象到具体的顺序，逐次分解为树状、有向、无向图结构，每个概念用一个描述性的关键字表示。

2、刻面分类法：定义若干用于刻画构件特征的面（facet），每个面包含若干概念，刻画可以描述构件执行的功能、被操作的数据、构件应用的语境或任意其他特征。

3、超文本组织方法：基于全文搜索技术，所有构件必须辅以详尽的功能说明文档。

# 71 构件组装方法

1、基于功能的组装技术

2、基于数据的组装技术

3、面向对象的组装技术

# 72需求管理包含的活动

需求管理过程中主要包含变更控制、版本控制、需求跟踪和需求状态跟踪四项活动。

1、变更控制活动包括问题分析和变更描述、变更分析和成本计算、变更实现。

2、版本控制活动主要包括定义需求文档的版本格式、制订需求文档的修改模式和确定需求文档版本等三项工作。

3、需求跟踪活动主要包括定义对其他需求的跟踪能力链和编制每个需求同系统元素之间的联系文档等两项工作。

4、需求状态跟踪活动主要包括定义需求状态和跟踪需求每一个状态等两项工作。