希赛

系统设计

课程内容提要

希赛

- 软件架构设计（★★★）
- 人机界面设计（★★）
- 结构化设计（★★★）
- 面向对象设计（★★★★）

系统设计 – 架构设计

希赛

软件架构 = 软件体系结构

架构设计就是需求分配，即将满足需求的职责分配到组件上。

系统设计 – 架构设计

希赛

- 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。
- 架构风格定义了用于描述系统的术语表和一组指导构建系统的规则。

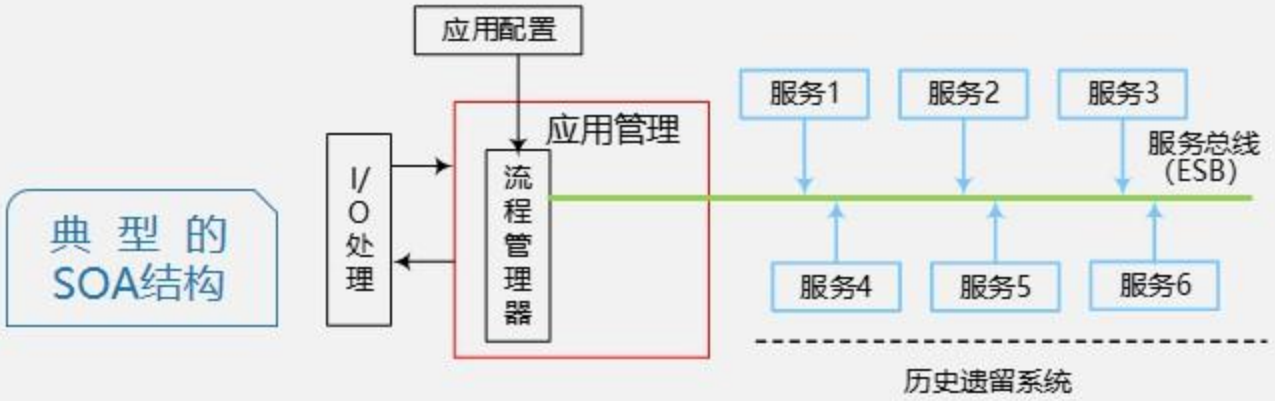
五大架构风格	子风格
数据流风格	批处理、管道-过滤器
调用/返回风格	主程序/子程序、面向对象、层次结构
独立构件风格	进程通信、事件驱动系统（隐式调用）
虚拟机风格	解释器、规则系统
仓库风格	数据库系统、黑板系统、超文本系统

系统设计 – 架构设计

希赛

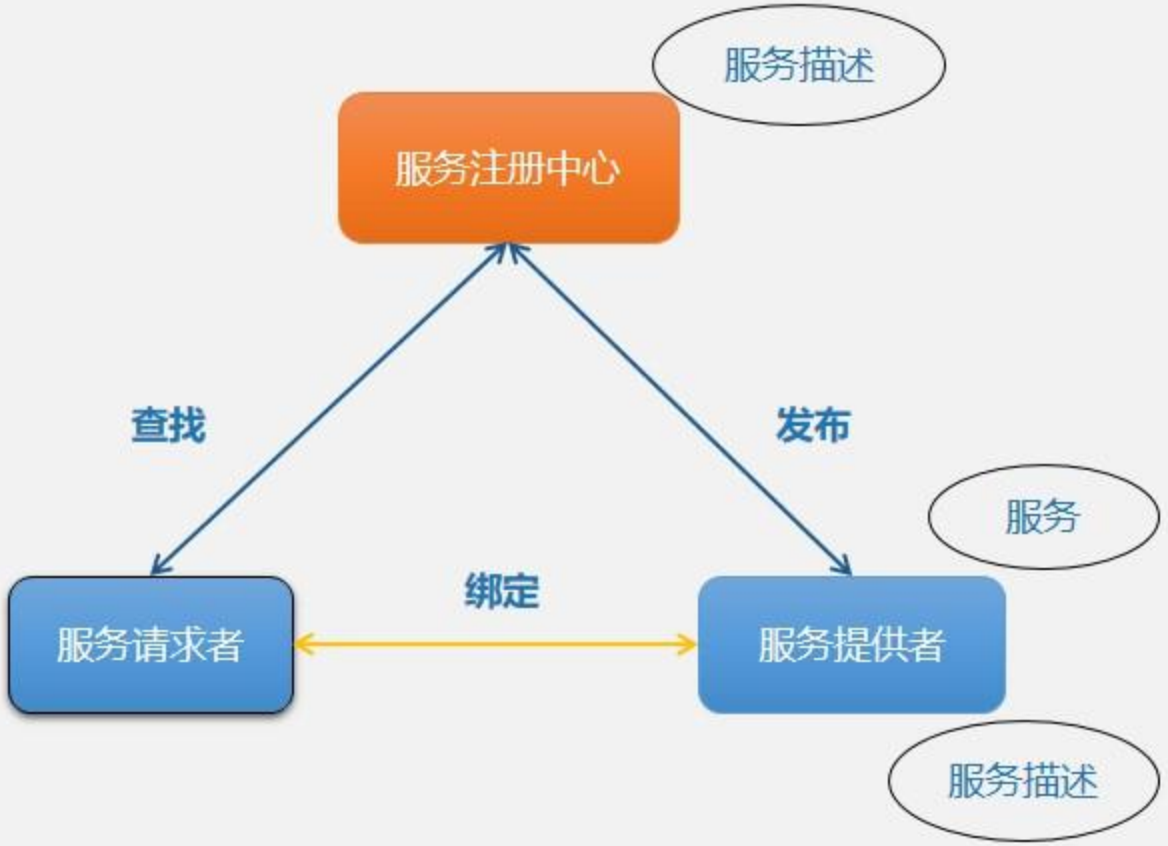
服务是一种为了满足某项业务需求的操作、规则等的逻辑组合，它包含一系列有序活动的交互，为实现用户目标提供支持。

服务的特点是：**松散耦合、粗粒度、标准化接口。**



系统设计 – 架构设计

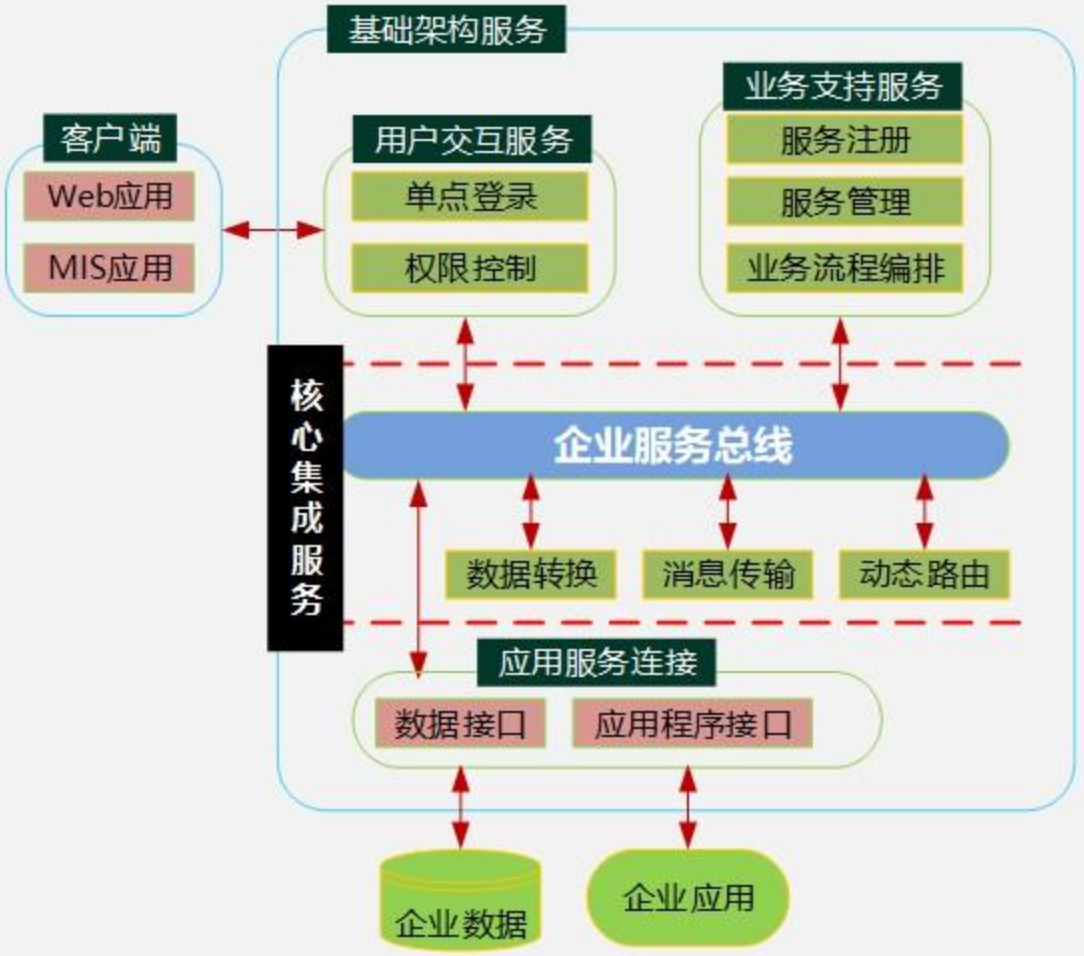
希赛



- ✓ 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
- ✓ 服务注册层

系统设计 – 架构设计

希赛



- ✓ 提供位置透明性的消息路由和寻址服务
- ✓ 提供服务注册和命名的管理功能
- ✓ 支持多种消息的传递泛型
- ✓ 支持多种可以广泛使用的传输协议
- ✓ 支持多种数据格式及其相互转换
- ✓ 提供日志和监控功能

系统设计 – 架构设计

希赛

微服务顾名思义，就是很小的服务，所以它属于面向服务架构的一种。

微服务架构是一种架构模式，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP协议的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够被独立的部署到生产环境、类生产环境等。另外，应当尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建。

特点

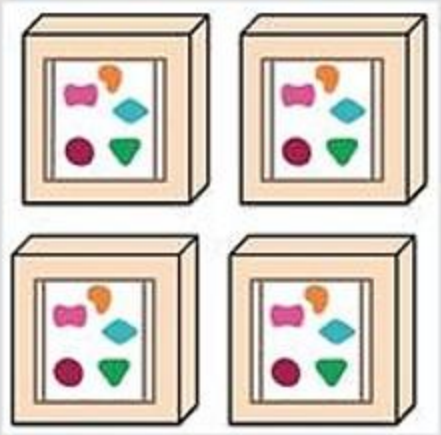
- 小，且专注于做一件事情
- 轻量级的通信机制
- 松耦合、独立部署

系统设计 – 架构设计

希赛

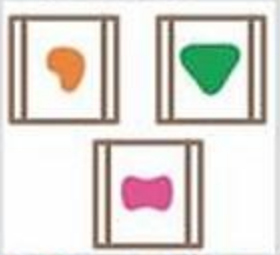
单块架构 (Monolithic)

紧耦合,所有功能都在一个进程中

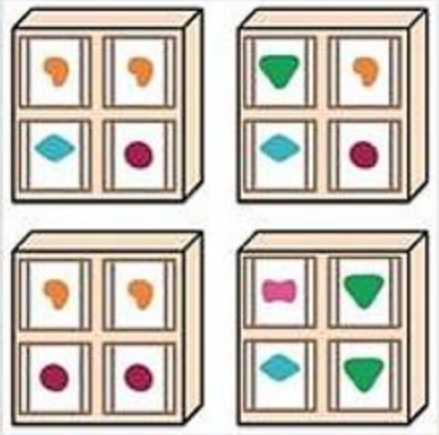


基于整个系统扩展

微服务架构 (MSA)



松耦合,功能在不同微服务的进程中



基于独立服务，按需扩展

系统设计 – 架构设计

希赛

微服务	SOA
能拆分的就拆分	是整体的，服务能放一起的都放一起
纵向业务划分	是水平分多层
由单一组织负责	按层级划分不同部门的组织负责
细粒度	粗粒度
两句话可以解释明白	几百字只相当于SOA的目录
独立的子公司	类似大公司里面划分了一些业务单元（BU）
组件小	存在较复杂的组件
业务逻辑存在于每一个服务中	业务逻辑横跨多个业务领域
使用轻量级的通信方式，如HTTP	企业服务总线（ESB）充当了服务之间通信的角色

系统设计 – 架构设计

希赛

Model Driven Architecture

– Model ?

客观事物的抽象表示

– Architecture ?

构成系统的部件、连接件及其约束的规约

– Model-Driven ?

使用模型完成软件的分析、设计、构建、部署、维护等各开发活动

– MDA起源于分离系统规约和平台实现的思想

– MDA的主要目标:

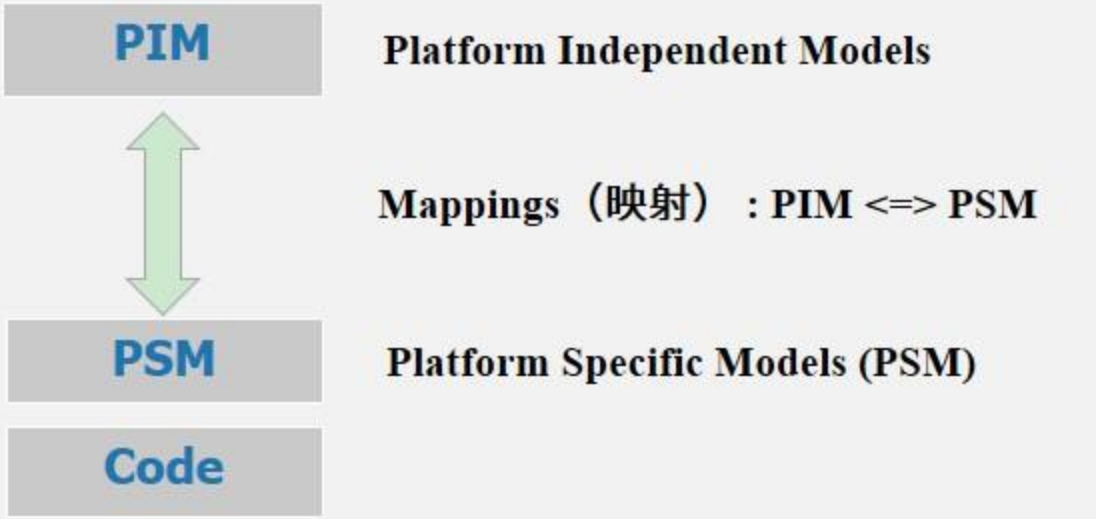
Portability（可移植性）, interoperability（互通性）,

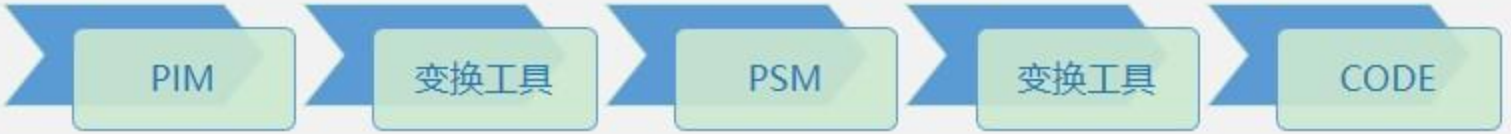
Reusability（可重用性）

系统设计 – 架构设计

希赛

- MDA的3种核心模型：
  - 平台独立模型（PIM）：具有高抽象层次、独立于任何实现技术的模型。
  - 平台相关模型（PSM）：为某种特定实现技术量身定做，让你用这种技术中可用的实现构造来描述系统的模型。PIM会被变换成一个或多个PSM。
  - 代码Code：用源代码对系统的描述（规约）。每个PSM都将被变换成代码。





系统设计 – 软件设计

希赛

软件设计包括体系结构设计、接口设计、数据设计和过程设计。

- ✓ 结构设计：定义软件系统各主要部件之间的关系。
- ✓ 数据设计：将模型转换成数据结构的定义。好的数据设计将改善程序结构和模块划分，降低过程复杂性。
- ✓ 接口设计（人机界面设计）：软件内部，软件和操作系统间以及软件和人之间如何通信。
- ✓ 过程设计：系统结构部件转换成软件的过程描述。

系统设计 – 软件设计

希赛

软件设计包括了四个既独立又相互联系的活动：高质量的（ ）将改善程序结构和模块划分，降低过程复杂性；（ ）的主要目标是开发一个模块化的程序结构，并表示出模块间的控制关系；（ ）描述了软件与用户之间的交互关系。

A 程序设计　　B 数据设计

C 算法设计　　D 过程设计

A 软件结构设计　　B 数据结构设计

C 数据流设计　　D 分布式设计

A 数据架构设计　　B 模块化设计

C 性能设计　　D 人机界面设计

系统设计 – 人机界面设计

希赛

黄金三法则

★ 置于用户控制之下

- 以不强迫用户进入不必要的或不希望的动作的方式来定义交互方式
- 提供灵活的交互
- 允许用户交互可以被中断和撤销
- 当技能级别增加时可以使交互流水化并允许定制交互
- 使用户隔离内部技术细节
- 设计应允许用户和出现在屏幕上的对象直接交互

★ 减少用户的记忆负担

- 减少对短期记忆的要求
- 建立有意义的缺省
- 定义直觉性的捷径
- 界面的视觉布局应该基于真实世界的隐喻
- 以不断进展的方式揭示信息

★ 保持界面的一致性

- 允许用户将当前任务放入有意义的语境
- 在应用系列内保持一致性
- 如过去的交互模型已建立起了用户期望，除非有迫不得已的理由，不要改变它

系统设计 – 人机界面设计

希赛

下列关于用户界面设计的叙述中，错误的是（ ）。

A 界面交互模型应经常进行修改

B 界面的视觉布局应该尽量与真实世界保持一致

C 所有可视信息的组织需要按照统一的设计标准

D 确保用户界面操作和使用的一致性

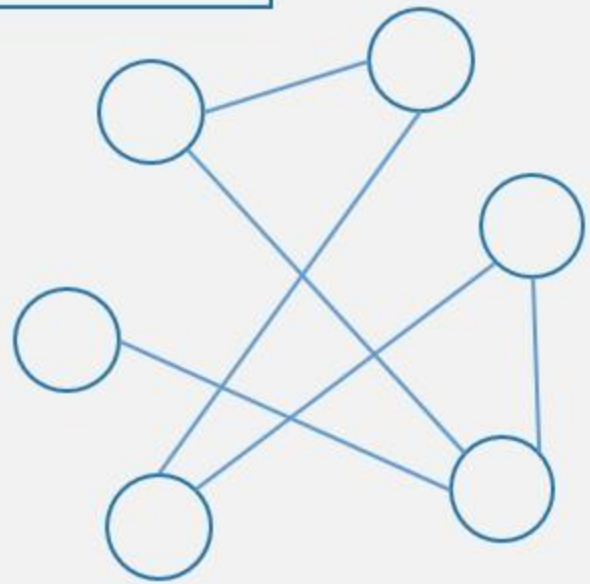
系统设计 – 结构化设计

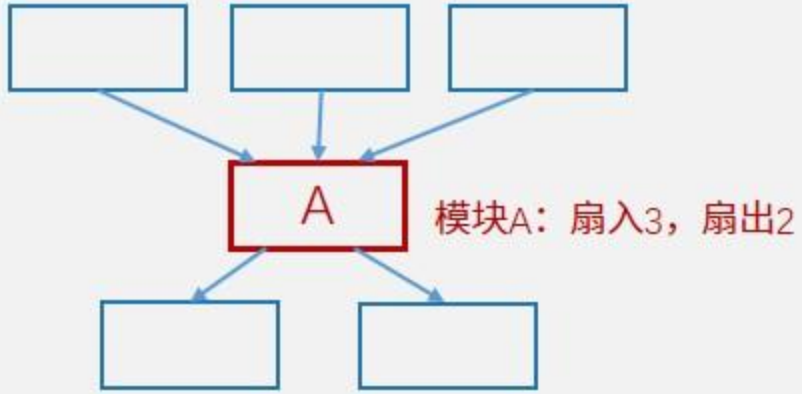
希赛

- **概要设计**【外部设计】：功能需求分配给软件模块，确定每个模块的功能和**调用关系**，形成**模块结构图**
- **详细设计**【内部设计】：为每个**具体任务**选择适当的技术手段和**处理方法**

结构化设计原则：

- **模块独立（高内聚、低耦合）**
- 保持模块的大小适中
- 多扇入，少扇出
- 深度和宽度均不宜过高





系统设计 – 结构化设计

希赛

高内聚

低内聚

内聚类型	描　述
功能内聚	完成一个单一功能，各个部分协同工作，缺一不可
顺序内聚	处理元素相关，而且必须顺序执行
通信内聚	所有处理元素集中在一个数据结构的区域上
过程内聚	处理元素相关，而且必须按特定的次序执行
瞬时内聚（时间内聚）	所包含的任务必须在同一时间间隔内执行
逻辑内聚	完成逻辑上相关的一组任务
偶然内聚（巧合内聚）	完成一组没有关系或松散关系的任务

低耦合

高耦合

耦合类型	描　述
非直接耦合	两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的
数据耦合	一组模块借助参数表传递简单数据
标记耦合	一组模块通过参数表传递记录信息（数据结构）
控制耦合	模块之间传递的信息中包含用于控制模块内部逻辑的信息
外部耦合	一组模块都访问同一全局简单变量，而且不是通过参数表传递该全局变量的信息
公共耦合	多个模块都访问同一个公共数据环境
内部耦合	一个模块直接访问另一个模块的内部数据；一个模块不通过正常入口转到另一个模块的内部；两个模块有一部分程序代码重叠；一个模块有多个入口

系统设计 – 结构化设计

希赛

模块的四个要素

输入和输出：模块的输入来源和输出去向都是同一个调用者，即一个模块从调用者那儿取得输入，进行加工后再把输出返回调用者。

处理功能：指模块把输入转换成输出所做的工作。

内部数据：指仅供该模块本身引用的数据。

程序代码：指用来实现模块功能的程序。

系统设计 – 结构化设计

希赛

以下关于软件系统模块结构设计的叙述中，正确的是（ ）。

A 当模块扇出过大时，应把下级模块进一步分解为若干个子模块

B 当模块扇出过小时，应适当增加中间的控制模块

C 模块的扇入大，表示模块的复杂度较高

D 模块的扇入大，表示模块的复用程度高

系统设计 – 结构化设计

希赛

内聚表示模块内部各部件之间的联系程度，（ ）是系统内聚度从高到低的排序。

A 通信内聚、瞬时内聚、过程内聚、逻辑内聚

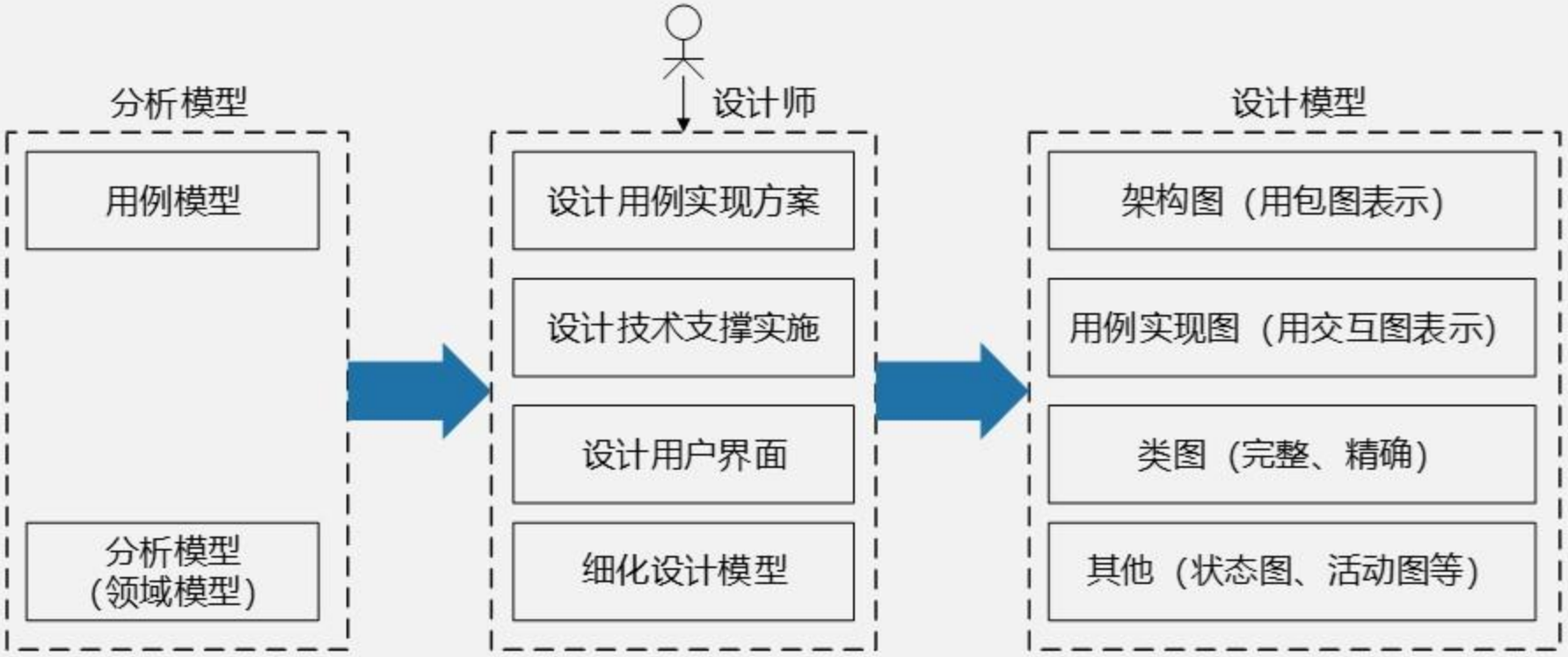
B 功能内聚、瞬时内聚、顺序内聚、逻辑内聚

C 功能内聚、顺序内聚、瞬时内聚、逻辑内聚

D 功能内聚、瞬时内聚、过程内聚、逻辑内聚

系统设计 – 面向对象设计 – 基本过程

希赛



系统设计 – 面向对象设计 – 设计原则

希赛

- ★ 单一职责原则：设计目的单一的类
- ★ 接口隔离原则：使用多个专门的接口比使用单一的总接口要好
- ★ 开放-封闭原则：对扩展开放，对修改封闭
- ★ 李氏（Liskov）替换原则：子类可以替换父类
- ★ 依赖倒置原则：要依赖于抽象，而不是具体实现；针对接口编程，不要针对实现编程
- ★ 组合重用原则：要尽量使用组合，而不是继承关系达到重用目的
- ★ 迪米特（Demeter）原则（最少知识原则）：一个对象应当对其他对象有尽可能少的了解

系统设计 – 面向对象设计 – 设计原则

希赛

某网站系统在用户登录时使用数字校验码。为了增强安全性，现在要求在登录校验码中增加字母或图片。如果直接修改原有的生成登录校验码的程序代码，则违反了面向对象设计原则中的（ ）。

A 开闭原则

B 里氏替换原则

C 最少知识原则

D 组合复用原则

系统设计 – 面向对象设计 – 设计模式

★ 架构模式：软件设计中的高层决策，例如C/S结构就属于架构模式，架构模式反映了开发软件系统过程中所作的基本设计决策

★ 设计模式：主要关注软件系统的设计，与具体的实现语言无关

★ 惯用法：是最低层的模式，关注软件系统的设计与实现，实现时通过某种特定的程序设计语言来描述构件与构件之间的关系。每种编程语言都有它自己特定的模式，即语言的惯用法。例如引用-计数就是C++语言中的一种惯用法

系统设计 – 面向对象设计 – 设计模式

希赛

（ ）的选择是开发一个软件系统时的基本设计决策；（ ）是最低层的模式，关注软件系统的设计与实现，描述了如何实现构件及构件之间的关系。引用-计数是C++管理动态资源时常用的一种（ ）。

A 架构模式　　B 惯用法

C 设计模式　　D 分析模式

A 架构模式　　B 惯用法

C 设计模式　　D 分析模式

A 架构模式　　B 惯用法

C 设计模式　　D 分析模式

系统设计 – 面向对象设计 – 设计模式

希赛

考查点：

1、设计模式三种类型的定位

2、设计模式分类

3、设计模式应用场景及特点

系统设计 – 面向对象设计 – 设计模式的分类

希赛

创建型模式：创建对象

- 工厂方法（Factory Method）模式
- 抽象工厂（Abstract Factory）模式
- 原型（Prototype）模式
- 单例（Singleton）模式
- 构建器（Builder）模式

结构型模式：更大的结构

- 适配器（Adapter）模式
- 桥接（Bridge）模式
- 组合（Composite）模式
- 装饰（Decorator）模式
- 外观（Facade）模式
- 享元（Flyweight）模式
- 代理（Proxy）模式

行为型模式：交互及职责分配

- 职责链（Chain of Responsibility）模式
- 命令（Command）模式
- 解释器（Interpreter）模式
- 迭代器（Iterator）模式
- 中介者（Mediator）模式
- 备忘录（Memento）模式
- 观察者（Observer）模式
- 状态（State）模式
- 策略（Strategy）模式
- 模板方法（Template Method）模式
- 访问者（Visitor）模式

有下划线的表示既可以是类模式，也可以是对象模式；
无下划线的表示只是对象模式

系统设计 – 面向对象设计 – 创建型模式

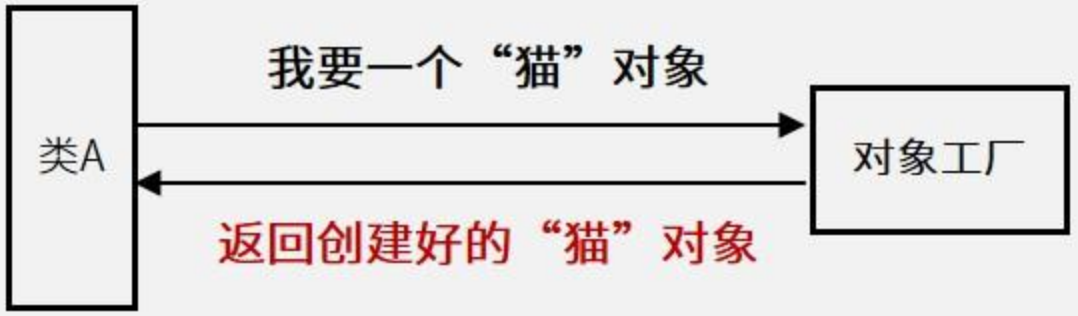
希赛

设计模式名称	简要说明	速记关键字
Factory Method 工厂方法模式	定义了创建对象的接口，它允许子类决定实例化哪个类	动态生产对象
Abstract Factory 抽象工厂模式	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	生产成系列对象
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	复杂对象构造
Prototype 原型模式	允许对象在不了解要创建对象的确切类以及如何创建等细节的情况下创建自定义对象。通过拷贝原型对象来创建新的对象	克隆对象
Singleton 单例模式	确保一个类只有一个实例，并且提供了对该类的全局访问入口	单实例

系统设计 – 面向对象设计 – 创建型模式

希赛

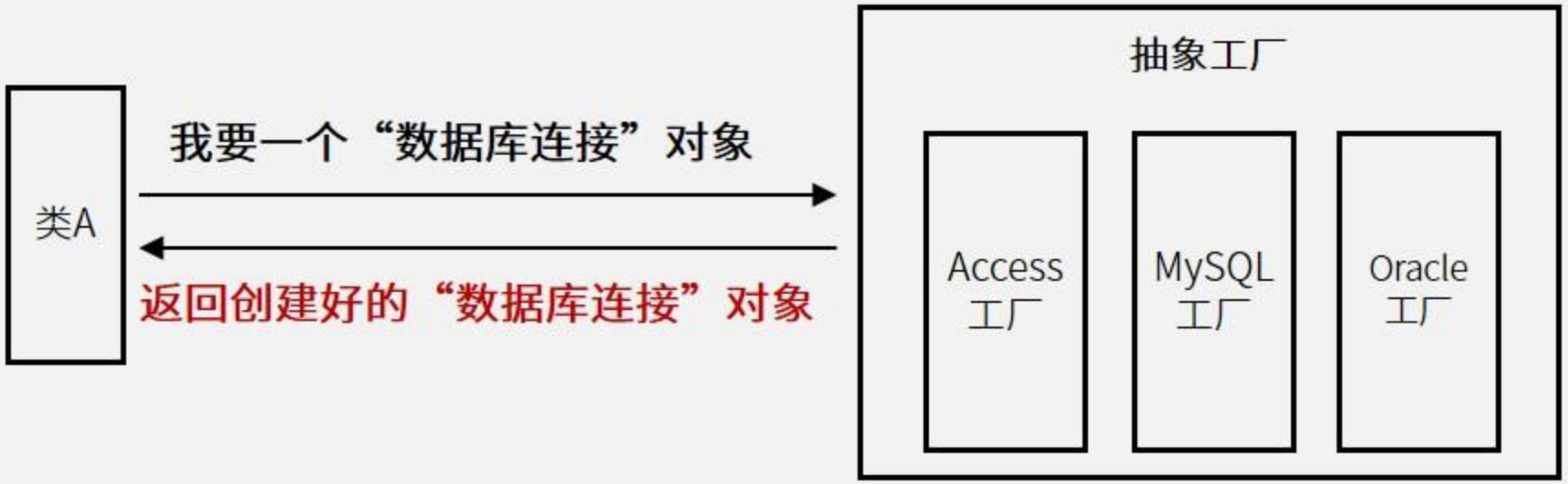
设计模式名称	简要说明	速记关键字
Factory Method 工厂方法模式	定义了创建对象的接口，它允许子类决定实例化哪个类	动态生产对象



系统设计 – 面向对象设计 – 创建型模式

希赛

设计模式名称	简要说明	速记关键字
Abstract Factory 抽象工厂模式	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	生产成**系列对象**



使用抽象工厂前，需要先指定工厂

系统设计 – 面向对象设计 – 创建型模式

希赛

设计模式名称	简要说明	速记关键字
Builder 构建器模式	将一个复杂类的表示与其构造相分离，使得相同的构建过程能够得出不同的表示	复杂对象构造

以创建一个游戏人物为例：

构建器

第一步 初始化游戏人物（）

第二步 创建武器（）

第三步 创建战袍（）

第四步 创建鞋子（）

…

系统设计 – 面向对象设计 – 结构型模式

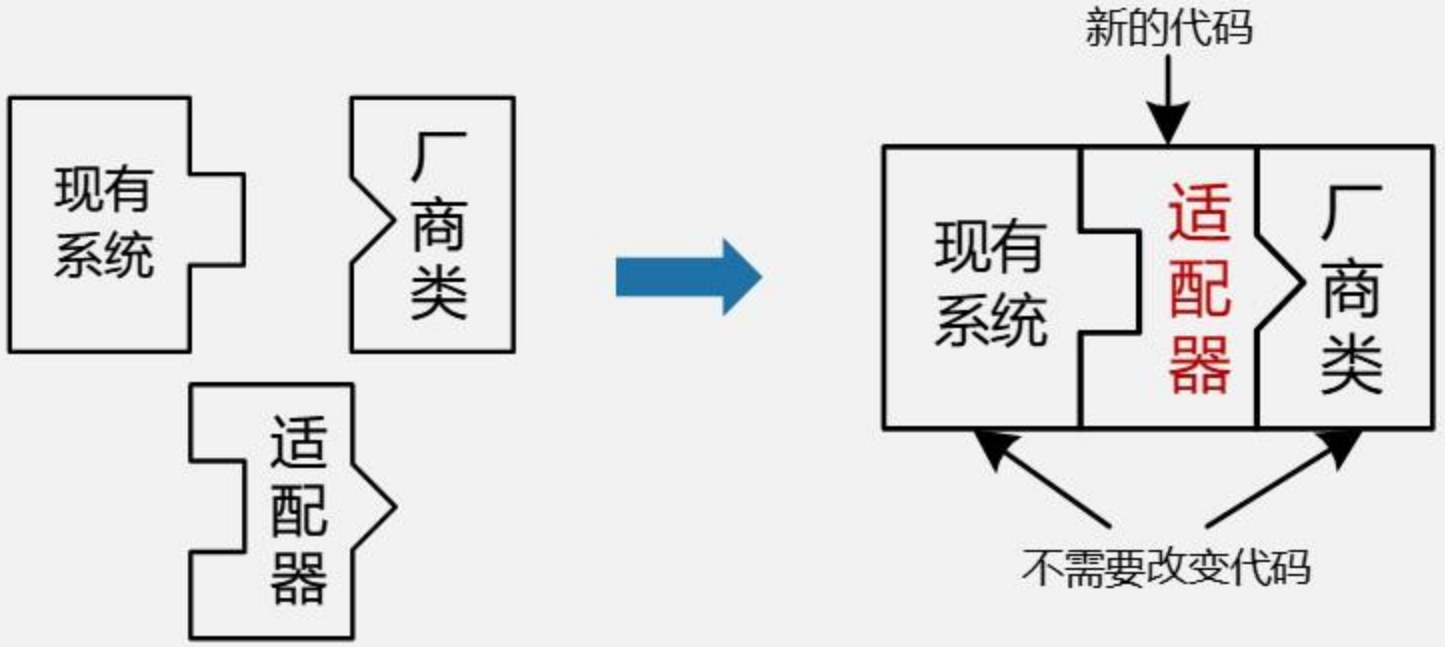
希赛

设计模式名称	简要说明	速记关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换接口
Bridge 桥接模式	将一个复杂的组件分成两个独立的但又相关的继承层次结构 将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化	继承树拆分
Composite 组合模式	创建树型层次结构来改变复杂性，同时允许结构中的每一个元素操作同一个接口。用于表示“整体-部分”的层次结构	树形目录结构
Decorator 装饰模式	在不修改对象外观和功能的情况下添加或者删除对象功能 即动态地给一个对象添加一些额外的职责	动态附加职责
Facade 外观模式	子系统中的一组接口提供了一个统一的接口	对外统一接口
Flyweight 享元模式	可以通过共享对象减少系统中低等级的、详细的对象数目 提供支持大量细粒度对象共享的有效方法	汉字编码
Proxy 代理模式	为控制对初始对象的访问提供了一个代理或者占位符对象	快捷方式

系统设计 – 面向对象设计 – 结构型模式

希赛

设计模式名称	简要说明	速记关键字
Adapter 适配器模式	将一个类的接口转换成用户希望得到的另一种接口。它使原本不相容的接口得以协同工作	转换接口

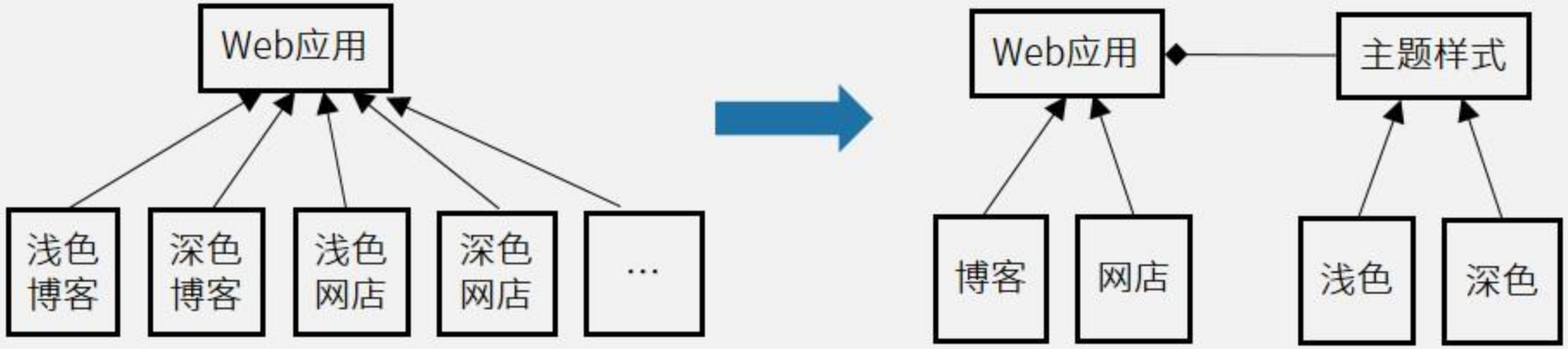


系统设计 – 面向对象设计 – 结构型模式

希赛

设计模式名称	简要说明	速记关键字
Bridge 桥接模式	将一个复杂的组件分成两个独立的但又相关的继承层次结构 将类的抽象部分和它的实现部分分离开来，使它们可以独立地变化	继承树拆分

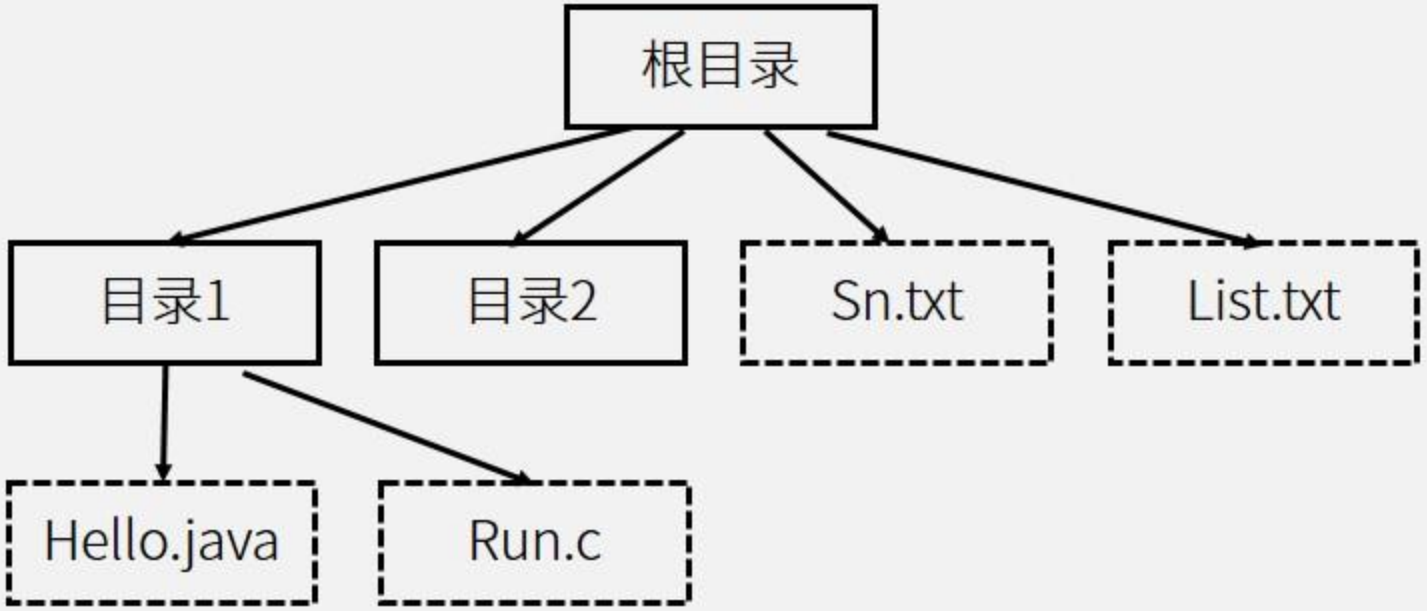
现在要创建一个Web应用框架，此框架能创建不同Web应用，比如博客、新闻网站和网上商店等；并可以为每个Web应用创建不同的主题样式，如浅色或深色等。



系统设计 – 面向对象设计 – 结构型模式

希赛

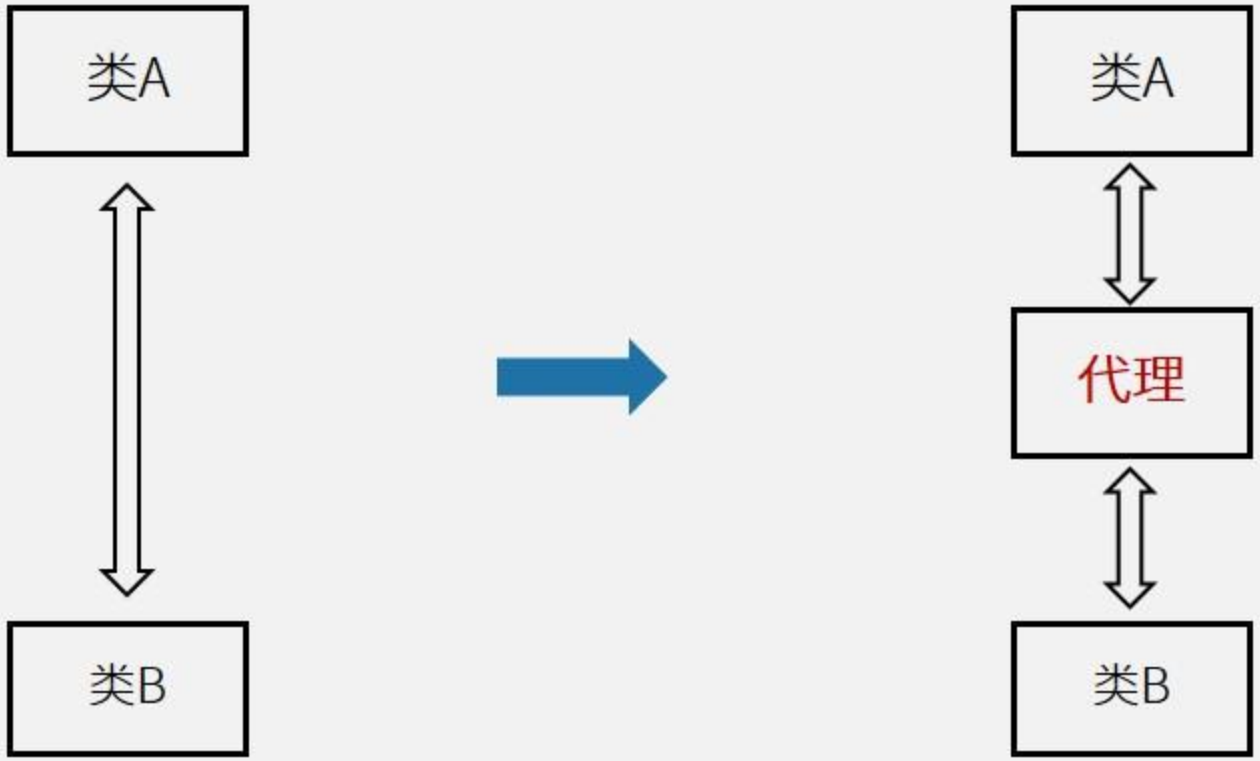
设计模式名称	简要说明	速记关键字
Composite 组合模式	创建树型层次结构来改变复杂性，同时允许结构中的每一个元素操作同一个接口。用于表示“整体-部分”的层次结构	树形目录结构



系统设计 – 面向对象设计 – 结构型模式

希赛

设计模式名称	简要说明	速记关键字
Proxy 代理模式	为控制对初始对象的访问提供了一个代理或者占位符对象	快捷方式



系统设计 – 面向对象设计 – 行为型模式(1)

希赛

设计模式名称	简要说明	速记关键字
Chain of Responsibility 职责链模式	可以在系统中建立一个链，这样消息可以在首先接收到它的级别处被处理，或者可以定位到可以处理它的对象	传递职责
Command 命令模式	在对象中封装了请求，这样就可以保存命令，将该命令传递给方法以及像任何其他对象一样返回该命令	日志记录，可撤销
Interpreter 解释器模式	可以解释定义其语法表示的语言，还提供了用表示来解释语言中的语句的解释器	虚拟机的机制
Iterator 迭代器模式	为集合中的有序访问提供了一致的方法，而该集合是独立于基础集合，并与之相分离的	数据集
Mediator 中介者模式	通过引入一个能够管理对象间消息分布的对象，简化了系统中对象间的通信	不直接引用

系统设计 – 面向对象设计 – 行为型模式(1)

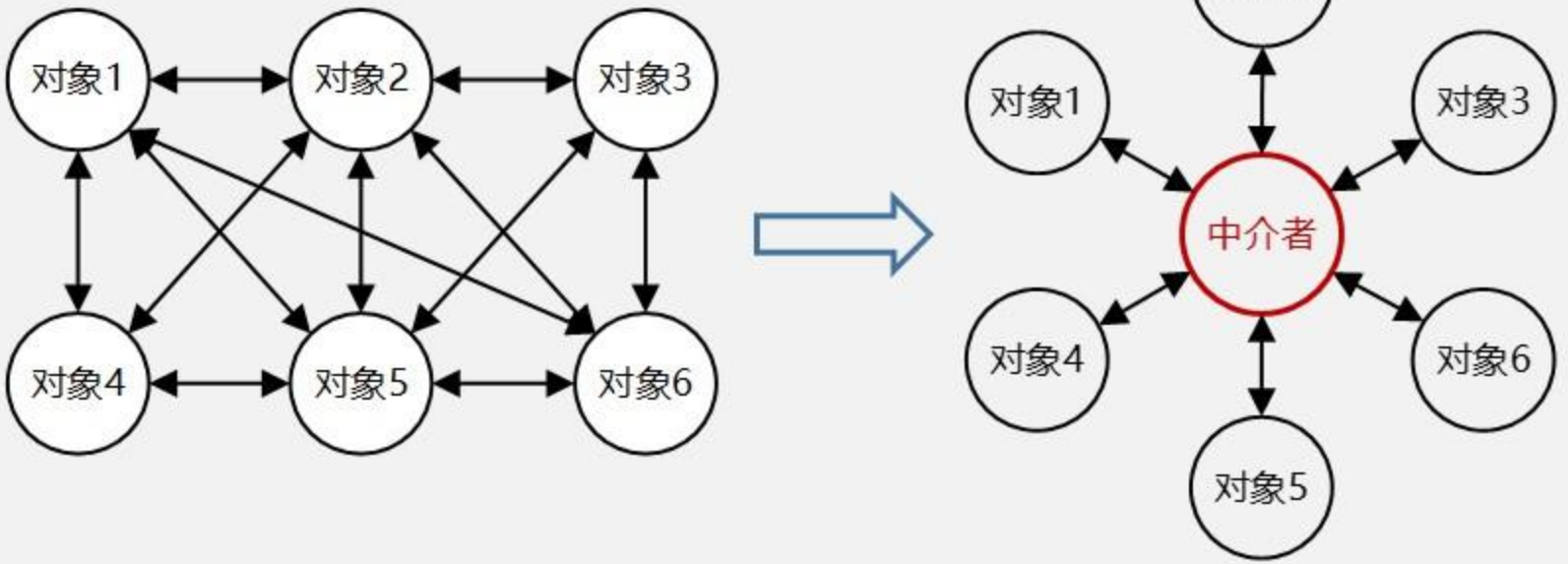
希赛

设计模式名称	简要说明	速记关键字
Chain of Responsibility 职责链模式	可以在系统中建立一个链，这样消息可以在首先接收到它的级别处被处理，或者可以定位到可以处理它的对象	传递职责

系统设计 – 面向对象设计 – 行为型模式(1)

希赛

设计模式名称	简要说明	速记关键字
Mediator 中介者模式	通过引入一个能够管理对象间消息分布的对象，简化了系统中对象间的通信	不直接引用



系统设计 – 面向对象设计 – 行为型模式(2)

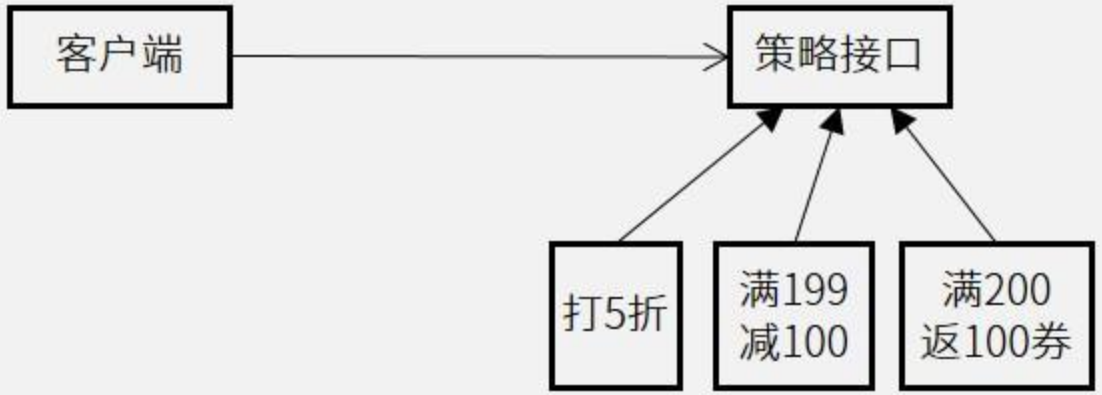
希赛

设计模式名称	简要说明	速记关键字
Memento 备忘录模式	保持对象状态的“快照”(snapshot)，这样对象可以在不向外界公开其内容的情况下返回到它的最初状态	游戏存档
Observer 观察者模式	为组件向相关接收方广播消息提供了灵活的方法 定义对象间的一种一对多的依赖关系	订阅、广播、联动
State 状态模式	允许一个对象在其内部状态改变时改变它的行为	状态变成类
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	多方案切换
Template Method 模板方法模式	提供了在不重写方法的前提下允许子类重载部分方法的方法	框架
Visitor 访问者模式	提供了一种方便的、可维护的方法来表示在对象结构元素上要进行的操作	数据与操作分离

系统设计 – 面向对象设计 – 行为型模式(2)

希赛

设计模式名称	简要说明	速记关键字
Strategy 策略模式	定义一系列算法，把它们一个个封装起来，并且使它们之间可互相替换，从而让算法可以独立于使用它的用户而变化	多方案切换



系统设计 – 面向对象设计 – 设计模式

设计模式按照目的可以划分为三类，其中，（ ）模式是对对象实例化过程的抽象。例如（ ）模式确保一个类只有一个实例，并提供了全局访问入口；（ ）模式允许对象在不了解要创建对象的确切类以及如何创建等细节的情况下创建自定义对象；（ ）模式将复杂对象的构建与其表示分离。

A 创建型	B 结构型	C 行为型	D 功能型
A Facade	B Builder	C Prototype	D Singleton
A Facade	B Builder	C Prototype	D Singleton
A Facade	B Builder	C Prototype	D Singleton

系统设计 – 面向对象设计 – 设计模式

希赛

某互联网公司正在设计一套网络聊天系统，为了限制用户在使用该系统时发表不恰当言论，需要对聊天内容进行特定敏感词的过滤。针对上述功能需求，采用（ ）能够灵活配置敏感词的过滤过程。

A 责任链模式　　B 工厂模式　　C 组合模式　　D 装饰模式

某软件公司正在设计一个图像处理软件，该软件需要支持用户在图像处理过程中的撤销和重做等动作，为了实现该功能，采用（ ）最为合适。

A 单例模式　　B 命令模式　　C 访问者模式　　D 适配器模式