第一章

1. 软件架构的定义：由结构和功能各异、相互作用的构件组合，按照一定的结构方式构成的系统（三要素：构件/组件、连接、连接关系）

2. 软件系统的复杂性

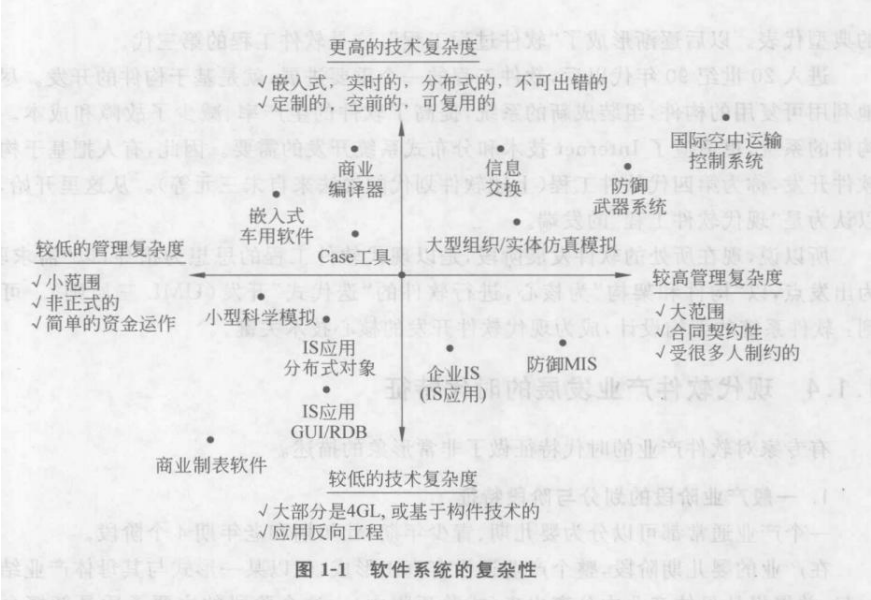


图 1-1　软件系统的复杂性

3. 在不同阶段架构发挥着不同的作用（了解）

- 在项目规划阶段：架构是项目可行性、工程复杂度、计划进度、投资规模和风险预测的依据。
- 在需求分析阶段：架构是开发团队与用户进行需求交互沟通的表达型所和结果（需求性）
- 在系统设计阶段:架构是系统设计分解和实现、验证的基础
- 在项目实施阶段：架构是工作分工、人员安排、组织协调、绩效管理的依据
- 在项目测试阶段：架构是性能测试和评审的依据
- 在维护升级阶段：架构是保证系统整体合理性、正确性、性能和可控的维护成本的前提下，软件系统修改、扩充、升级的基本模型

第二章

1. 文件传输的案例分析 P22

- 构件 $P_{24} - P_{25}$
  - 发送和接受进程
  - 各自的文件缓冲和发送接受队列
  - 数据包
  - 上层进程
- 连接件
  - 物理连接：RS-232及串口线
  - 数据连接：信息队列、文件缓冲区
  - 控制连接：进程间通信缓冲、信号灯
- 连接关系
  - 物理连接协议：RS-232串口协议
  - 数据连接协议：信息队列和文件缓存区的使用方法（头地址、指针、长度、访问方式和限制等）、数据格式、信息含义、先进先出。
  - 控制连接协议：队列、信号灯的命令含义——暂停、继续、成功、失败。

2. 软件系统的关键质量属性的定义

质量属性很难定义，但他经常可以区分产品是只完成了应该完成的任务，还是使客户感到满意。

3. 关键质量属性与系统功能的正交性

- 应用系统的“易用性”（容易使用系统）
- 应用系统的性能（关注应用系统的响应速度）
- 应用程序的可维护性（指代码易于维护）

与架构设计有关的设计：语音导航层次太宽，是否允许取消/撤销操作，是否可以重用 以前输入的数据，是否有多层次的输入支持和帮助；
无关的设计：算法、美观、操作简便。

4. 现代软件工程最典型的生命周期 p36 图 2-13

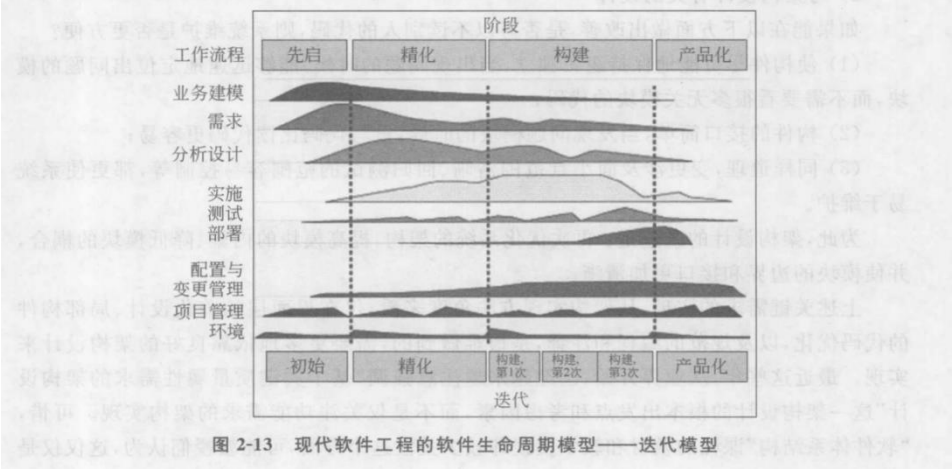


图 2-13 现代软件工程的软件生命周期模型——迭代模型

5. 软件产品线的定义

满足组特定的市场或者用户需求的，有一组公共的、可管理的产品特性的，符合上述两个条件的产品的组合。

6. 软件产品的定义

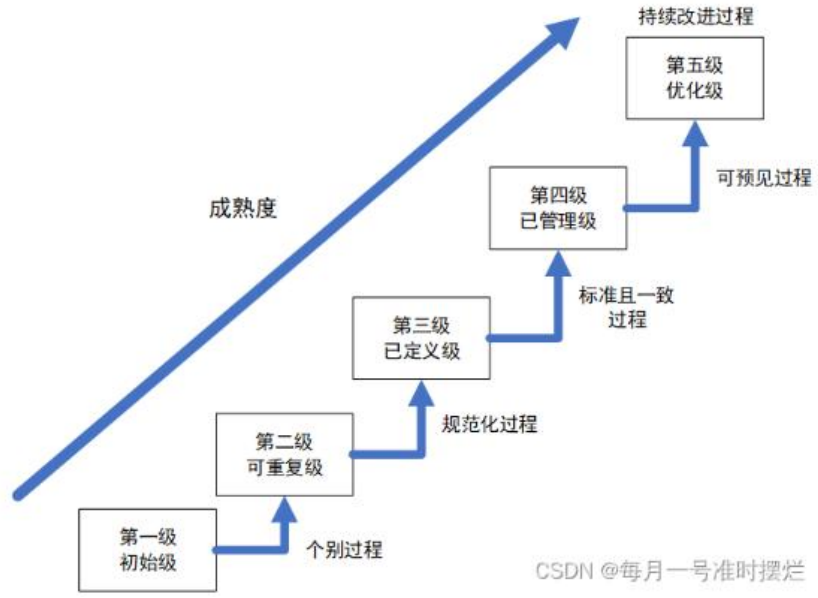
任何一款符合市场周期长、专业技术知识要求高、用户群庞大、产品功能复杂、安全和可靠性高、需要长期维护等特征的产品。

7. 软件产品线管理的三大基本活动

核心资源开发、利用核心资源的产品开发、核心资源和产品开发的技术和管理。

8. CMM 模型是什么

CMM 指的是软件过程能力成熟度模型，按软件过程的不同成熟度划分了 5 个等级，1 级被认为成熟度最低，5 级成熟度最高



9. 关键需求影响架构设计

10. 影响架构的需求

- 项目的总体范围
- 不同功能的开发的重要性和顺序优先权
- 与外部组件和系统的接口规范和标准
- 开发资源的分配
- 开发时间表的制定
- 软件个质量属性的选择和通过标准的制定
- 对软件的各种质量的测试范围
- 软件发行通过关口的检测标准

11. 影响架构的系统设计

- 灵活、具有可伸缩性
- 考虑全面并可扩展
- 思路简单明了、直接可以理解
- 结构划分和关系定义清楚
- 模块职责明确和分布合理
- 效益和技术综合平衡

12. 架构师的任务和责任

- 领导并负责架构设计
- 实际参与架构原型的开发实现
- 讲解架构，指导开发，协调冲突
- 支持项目经理，如技术可行性研究、任务划分、人员招聘
- 了解所在组织的业务目标，令架构更好地支持组织的业务目标
- 评估新技术，并提出采用建议。

第三章

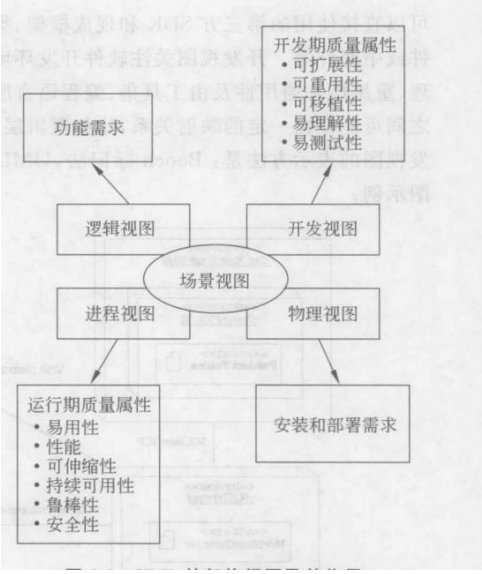
1. 架构视图的定义

是对于从某一视角或某一点上看到的系统所做的简化描述，它涵盖了系统的某一特定方面，而省略了与此无关的实体。

2. 五个架构的基本描述方向（架构描述的顺序从上到下）

- 开发架构：反映的是开发期的质量需求。
- 物理架构：反映安装和部署需求。
- 运行架构：反映的是运行期的质量需求。
- 逻辑架构：反映的是功能需求是如何被分解和协同实现的。
- 数据架构：反映的是数据需求。

3. 基于 UML4+1 的架构视图（会画）



第四章

1. 现代软件工程的需求工程

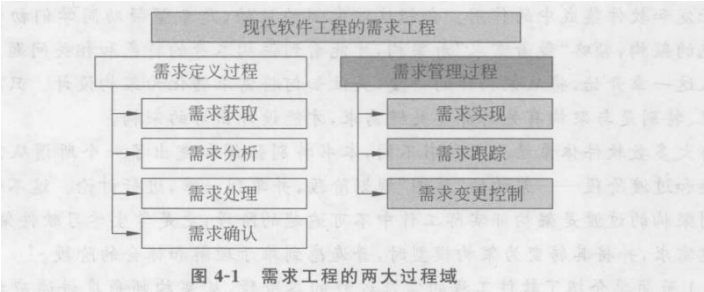


图 4-1　需求工程的两大过程域

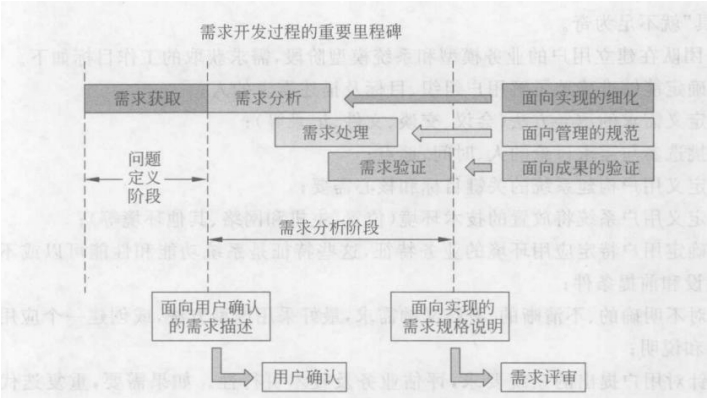


图 4-2　需求开发过程的 4 个过程域

2. 需求获取产生的主要交付物成果

- 参与需求诱导活动的客户、用户和其他干系人名单
- 系统目标与核心关注
- 系统技术环境的描述
- 功能点列表及相应的假设和限制

3. 需求分析模型与架构设计模型的对应关系

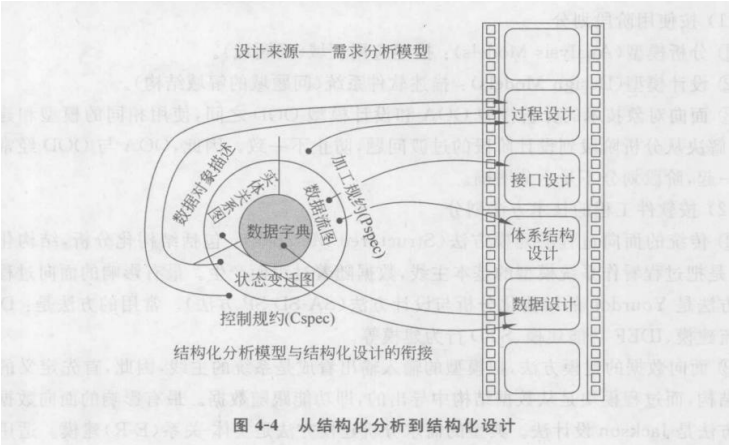


图 4-4　从结构化分析到结构化设计

4. U/C 矩阵的概念（Use/Create）

功能/数据分析是在业务流程、数据流程及数据分析的基础上，为了整体地考虑新系统的功能子系统和数据资源的合理分布，而进行的系统化的分析方法。

采用 U/C 矩阵方法进行的数据正确性分析是基于以下的所谓数据守恒原理，即：
(1) 数据必定有一个产生的源，而且必定有一个或多个用途。
(2) 具体在 U/C 矩阵中，就是：
① U/C 矩阵的每一个列只能有一个 C(来源唯一)；
② U/C 矩阵的每一个列至少有一个 U(至少有一个用途)；
③ 不能出现空行或空列(没有来源或没有用途)。

5. P150 4.4.1 简单了解

6. 以下四个为去年重点，今年并未提及

1. 需求超集是什么？

   通常软件架构的设计应当基于并有可能超出当前的软件需求定义的边界之外。

2. 需求基线

   需求的基线决定了开发阶段的里程碑，项目团队再某个时间点上应达到的状态，和应完成的交付成功，以及变更管理等。它是软件需求和过程跟踪管理的计划与目标

3. 需求模型：在需求的获取和分析阶段，获取并记录整理的需求描述，可以称为需求模型。
4. 架构模型：在架构设计阶段产生的设计思想和架构方案，被记录下来，称为架构模型。

第五章

1. OSI 模型引出的三个概念

- 服务：每一层为它的上一层提供服务
  对架构设计的参考意义：
  以架构格局来看，每一层提供了一个相对上一层而言，更“基础“的服务功能。架构师对于服务层次的设计，应该看到的是”逐级抽象“。
- 接口：接口告诉自己的上层，应该如何访问自己，定义了访问的参数和预期的结果，也和该层如何实现无关。
  对架构设计的参考意义：
  接口不单是模块之间访问的通道，而且成为基于架构的系统开发标准。
- 协议：协议是端到端的对等协议，是该层的内部事物
  对架构设计的参考意义：
  在架构师看来，协议是模块内部的功能，所以不关心

2. C/S(Client 客户端 /Server 服务器)两层架构

C/S 结构是一种软件系统体系结构，这种结构是将需要处理的业务合理地分配到客户端和服务器端，这样可以大大降低通信成本，但是升级维护相对困难。比如我们手机中安装的微信、qq、王者荣耀等应用程序就是 C/S 结构。

3. B/S(Browser 游览器 /Server 服务器)三层架构

B/S 结构是随着互联网的发展，web 出现后兴起的一种网络结构模式。这种模式统一了客户端，让核心的业务处理在服务端完成。你只需要在自己电脑或手机上安装一个浏览器，就可以通过 web Server 与数据库进行数据交互。在手机或电脑上用浏览器上百度搜索、看新闻等就是在使用“B/S”结构进行数据交互。这种“B/S”结构的好处：维护和升级方式更简单，客户端是浏览器，基本不需要维护，只需要维护升级服务器端就可以。

第七章

1. 接口的概念

- 接口（硬件类接口）是指同一计算机不同功能层之间的通信规则称为接口。例如通过 usb 接口把鼠标与计算机连接起来。
- 接口（软件类接口）是软件接口是指对协定进行定义的引用类型。简单的说就是实现不同软件或软件的不同部分之间的数据交互的定义，可以是个函数，类又或者是个定义。

2. 接口的定义和实现

```
public interface MyInterface {
    public void add(int x,int y);//接口中定义的方法 1
    public void volume(int x,int y,int z);//接口中定义的方法 2
}
```

```
public class Demo implements MyInterface {
    public static void main(String[] args) {
        Demo d=new Demo();//创建类 Demo 的实例 d
        d.add(1,2);
        d.volume(1,2,3);
    }
    public void add(int x, int y) {//实现接口中定义的方法 1
        System.out.println("x+y="+(x+y));
    }
    public void volume(int x, int y, int z) {//实现接口中定义的方法 2
        System.out.println("x*y*z="+(x*y*z));
    }
}
```

3. 用接口实现多继承 P259

4. 设计模式的概念

一种从软件设计过程中总结出来的、广泛应用和成熟的结构和结构关系。

5. 单实例模式/单件模式/单例模式 记住 PPT 对应的代码

单例模式 | 菜鸟教程 (runoob.com)

- 意图：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。
- 动机

> 有很多环境要求只能有一个实例。例如，系统只能定义一个默认打印机，如果有两个就不叫默认了。类负责保存它的唯一实例，保证没有其他实例可以被创建(通过截获创建新实例的请求），并且它可以提供一个访向该实例的方法，这就是单例模式。

- 适用性

> 当类只能有一个实例并且客户从一个众所周知的访同点访问它时(无需选择)。当这个唯一的实例应该是通过子类化可扩展的，并且客户应该无须重改代码就能使用一个扩展的实例时(只有一条通道走到底)!

- 注意

> - 单例类只能有一个实例。
> - 单例类必须自己创建自己的唯一实例。
> - 单例类必须给所有其他对象提供这一实例。

6. 适配器模式

适配器模式 | 菜鸟教程 (runoob.com)

- 定义

> 作为两个不兼容的接口之间的桥梁，这种类型的设计模式属于**结构型模式**，它结合了两个独立接口的功能。
>
> 适配器模式分为类结构型模式和对象结构型模式两种，前者类之间的耦合度比后者高，且要求程序员了解现有组件库中的相关组件的内部结构，所以应用相对较少些。

- **意图**：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。
- **动机**：有时，某些被复用的工具仅仅因为接口不匹配而不能被复用
- **主要解决**：主要解决在软件系统中，常常要将一些"现存的对象"放到新的环境中，而新环境要求的接口是现对象不能满足的。
- **使用场景**：有动机地修改一个正常运行的系统的接口，这时应该考虑使用适配器模式。
- **适用性**

> 你想使用一个已经存在的类，而它的接口不符合你的需要
> 你想创建一个可以复用的类，该类可以与其他不相关的类或不可预见的类（接口可能不兼容）一起工作
> 你想使用一些已经存在的子类，但是不可能对每个都进行子类化以匹配它们的接口。

- **注意事项**：适配器不是在详细设计时添加的，而是解决正在服役的项目的问题。
- 适配器继承或依赖已有的对象，实现想要的目标接口

7. 设计模式的四个要素

- 模式名
- 问题：描述应该在合适使用模式，它针对哪些特定的问题，以及使用的先决条件

- 解决方案：是模式本体，描述模式是如何构成的，它们之间的相互关系和责任。模式是一个问题类的解决方案的抽象模板，因此，它不描述具体的设计或实现，只提供设计问题的抽象思路。
- 效果：描述模式的应用效果和使用模式应权衡的问题。这是在设计阶段进行设计选择时，不可避免要考虑的问题。包括：好处和代价、时间和空间、复用和灵活性、可扩展和可移植性等的平衡。

第八章

1. MVC 模式的核心思想

MVC 的核心思想是将一个应用分成三个基本部分：*Model*（模型）、*View*（视图）和 *Controller*（控制器），这三个部分以最少的耦合协同工作，从而提高应用的可扩展性及可维护性。

2. Struts 应用框架的概念

Struts 是一个基于 Java 的开源 Web 应用框架，主要用于构建企业级的 Java Web 应用程序。它是最早的 Java EE 平台上的 MVC（Model-View-Controller）设计模式的实现之一。Struts 提供了一套标准化的解决方案来分离应用程序的不同层，以便于开发、维护和扩展。

3. IoC 控制反转

- Spring中的应用

  > Spring 通过一种称作控制反转(Inversion of Conttol,IoC)的技术促进了松耦合。当应用了 IoC，一个对象依赖的其他对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。可以认为 IoC 与JNDI 相反——不是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求,就主动将依赖传递给它——依赖注入(Dependence Injection, DI).

- 与MVC 的核心是变更—传播机制一样，Spring 框架的核心，就是控制翻转 IoC/依赖注入DI机制。
- IoC 是指由容器中控制组件之间的关系，而非传统实现中由程序代码直接操控，这种将控制权由程序代码到外部容器的转移，称为"翻转"

4. ORM 模块

使得常见工具 Hibernate,iBATIS,JDO 很好的集成到 Spring 应用中

- ORM(Object-Relational Mapping) P_{391}

  > 即对象关系映射，是指以 O/R 原理设计的持久化框架(Framework），包括 0/R机制、SQL 自生成、事务处理和 Cache 管理等。

- ORM 的实现思想

  > 将关系数据库表中的数据记录，映射成为对象，以对象的形式展现，这样开发人员就可以把对数据库的操作转化为对这些对象的操作。因此它的目的是为了方便开发人员以面向对象的思想来实现对数据库的操作。

5. 容器的概念

为组件提供特定服务和技术支持的一个标准化运行时的环境

UserLognAction 类校验输入

```
/**
     * Method execute
     * @param mapping 用于映射操作结果到相应的页面
     * @param form 封装了用户登录表单的数据。
     * @param request 包含客户端的请求数据
     * @param response 用于向客户端发送响应
     * @return ActionForward 指向前向页面
     */
public ActionForward execute(ActionMapping mapping, ActionForm form,
                             HttpServletRequest request, HttpServletResponse
response) {
    // 将 ActionForm 对象转换为 UserLognForm 对象，用于处理特定的登录逻辑
    UserLognForm userLognForm = (UserLognForm) form;

if(userLognForm.getUsername().equals("zzz")&&userLognForm.getPassword().equ
als("123")) {
        //如果登录成功，将用户名保存到请求属性中，并转发到成功页面。
        request.setAttribute("username", userLognForm.getUsername());
        return mapping.findForward("success");
    }
    {
        //// 如果登录失败，同样将用户名保存到请求属性中，并转发到失败页面。
        request.setAttribute("username", userLognForm.getUsername());
        return mapping.findForward("failure");
    }

}
```

ActionMapping

在一个 Web 应用中，每个资源都必须通过 URI 来进行引用。资源包括 HTML 页面，JSP 页面和定制动作。为了给定制动作一个 URI 或者说路径，Struts 框架提供了一个 ActionMapping 对象。像 ActionForward 和 ActionForm 一样，ActionMapping 通常也在 XML 配置文件中定义。

Struts-config. xml 文件片断：

```
<!-- 定义动作映射，用于处理用户登录请求 -->
<action-mappings >
   <!-- attribute="userLognForm"指定用于表单验证的 JavaBean 属性名 -->
    <!-- input="/form/userLogn.jsp"指定表单验证失败时的转向页面 -->
    <!-- name="userLognForm"指定表单验证的 JavaBean 名称 -->
```

```
    <!-- path="/userLogn"指定处理请求的 Action 类 -->
    <!-- scope="request"指定 Action 类处理请求时使用的作用域 -->
    <!-- type="com.yourcompany.struts.action.UserLognAction"指定处理请求的
Action 类 -->
   <action
     attribute="userLognForm"
     input="/form/userLogn.jsp"
     name="userLognForm"
     path="/userLogn"
     scope="request"
     type="com.yourcompany.struts.action.UserLognAction">
      <!-- 设置属性，表示此动作可以被取消 -->
     <set-property property="cancellable" value="true" />
      <!-- 定义一个转发，用于登录失败的情况，跳转到特定页面 -->
     <forward name="failure" path="/UserLoginFailure.jsp" />
      <!-- 定义一个转发，用于登录成功的情况，跳转到特定页面 -->
     <forward name="success" path="/UserLoginSuccess.jsp" />
   </action>
 </action-mappings>
```

6. 实验 4 对应的技术要理解，代码要会写！！！