目录

[学生信息管理系统软件架构文档 3](#_Toc73105521)

[1.引言 3](#_Toc73105522)

[2.架构背景 3](#_Toc73105523)

[3.总体设计 3](#_Toc73105524)

[3.1 基本功能 3](#_Toc73105525)

[3.2 其他功能 4](#_Toc73105526)

[3.3 运行环境 4](#_Toc73105527)

[3.4 基本设计概念和处理流程 4](#_Toc73105528)

[4.各模块结构 5](#_Toc73105529)

[4.1 总体模块结构 5](#_Toc73105530)

[4.2 用户登录 5](#_Toc73105531)

[4.3系统管理员服务 6](#_Toc73105532)

[4.4老师服务 7](#_Toc73105533)

[4.5学生模块 11](#_Toc73105534)

[4.6系统辅助工具 12](#_Toc73105535)

[5.利用MySQL设计数据库 13](#_Toc73105536)

[5.1 学生信息记录表 13](#_Toc73105537)

[5.2教师信息记录表 14](#_Toc73105538)

[5.3用户信息记录表 15](#_Toc73105539)

[5.4课程信息记录表 15](#_Toc73105540)

[5.5开课信息记录表 15](#_Toc73105541)

[5.6成绩信息记录表 16](#_Toc73105542)

[5.7院系信息记录表 16](#_Toc73105543)

[5.8专业信息记录表 16](#_Toc73105544)

[6.数据库安全性 17](#_Toc73105545)

[7.数据库管理和设置 17](#_Toc73105546)

[8.为保证项目的质量属性而增加的设计（实验九） 19](#_Toc73105547)

[8.1 可修改性 19](#_Toc73105548)

[8.2性能 21](#_Toc73105549)

[8.3安全性 24](#_Toc73105550)

[8.4可复用性 26](#_Toc73105551)

[10.附录：软件架构成员学习报告（实验七） 27](#_Toc73105552)

[10.1 岳大帅 27](#_Toc73105553)

[10.2 孙颖 33](#_Toc73105554)

[10.3踪家君 36](#_Toc73105555)

[10.4刘倩 38](#_Toc73105569)

[10.5边巴曲珍 41](#_Toc73105570)

### 学生信息管理系统软件架构文档

### 1.引言

随着科学技术的发展和计算机的广泛应用，越来越多的高校开始使用计算机管理学生信息，学生管理系统是针对学校学生的大量工作处理而开发的，主要用于学校学生信息管理，实现学生各种信息管理，查询、修改、增加、删除等。

### 2.架构背景

学生信息档案的管理对于学校的管理者来说至关重要，学生信息是高等学校非常重要的一项数据资源，是一个教育单位不可缺少一部分。特别是近几年来，国家政策的调整，我国高等院校大规模的扩招，给高等院校的[教学管理](https://baike.baidu.com/item/%E6%95%99%E5%AD%A6%E7%AE%A1%E7%90%86/628913)、学生管理、[后勤管理](https://baike.baidu.com/item/%E5%90%8E%E5%8B%A4%E7%AE%A1%E7%90%86/11045247)等方面都带来不少的冲击。其包含的数据量大，涉及的人员面广，而且需要及时更新，故较为复杂，难以单纯地依靠人工管理，而且传统的人工管理方式既不易于规范化，管理效率也不高，我国各类高等院校中还有相当一部分学生档案管理还停留在纸介质的基础上，尤其是中、小学对学生档案的管理更是落后，这样的管理机制已经不能适应时代发展的要求，其管理方法将浪费许多人力和物力。随着科学技术的不断提高，[计算机科学与技术](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E4%B8%8E%E6%8A%80%E6%9C%AF)日渐成熟，计算机应用的普及已进入人类社会生活的各个领域，并发挥着越来越重要的作用。这种传统的手工管理模式必然被以计算机为物质基础的信息管理方法所取代。

作为计算机应用的一部分，使用计算机对学生档案进行管理，有着手工管理所无法比拟的优点，如：检索迅速、查找方便、可靠性高、[存储量](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E9%87%8F)大、保密性好、寿命长、成本低等。这些优点能够极大地提高学生档案管理的效率，也是学校向科学化、正规化管理发展的必要条件，更是各个高等院校与世界接轨的重要条件。

### 3.总体设计

**3.1 基本功能**

学籍管理：对学生的信息档案进行管理，主要包括学生档案信息的修改、删除、添加、档案查询及档案输出等，其中档案查询可以根据学号、姓名查询需要的信息；档案输出可以输出学生档案、成绩、课程表等。学生档案记录主要包括：学号、姓名、出生日期、年龄、性别、政治面貌、入学时间、个人简介、照片、家庭住址、邮编、所在的系别以及所学专业等字段。

成绩管理：对学生的成绩进行管理，主要就是成绩查询，成绩查询可以按学号、姓名进行查询。成绩记录主要包括：学期、学号、姓名、课程号以及成绩字段等。

课程管理：学生可以通过该模块进行课程表查看、打印课程表。

选修课选择：对学生报选修课进行管理，学生可以对自己喜爱的课程进行选择。

系统维护：教师对学生的密码进行管理，主要包括增加新用户和用户修改。其中用户修改可以实现对学生的密码修改和学生用户名的删去。

**3.2 其他功能**

系统辅助工具：通过该模块用户可以打开一些辅助工具，如记事本。

退出系统：离开本学生信息管理系统

* 1. **运行环境**

硬件环境：笔记本电脑

软件环境：

·系统要求：Windows 10

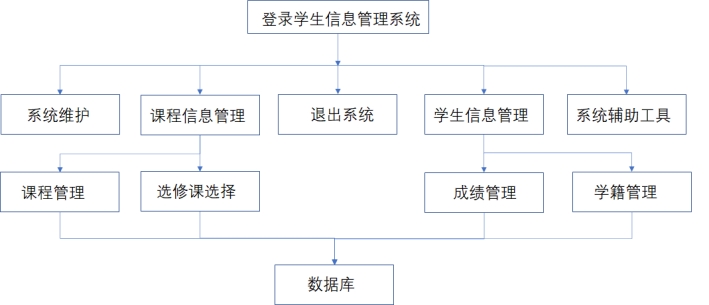
·数据库：MySQL 8.0

·开发工具：Pycharm Anaconda

**3.4 基本设计概念和处理流程**

学生管理系统包括以下几个模块：

1. 用户登录模块：用户登录
2. 学生信息模块：设置个人信息、成绩查询、课程查询、选修课选择、辅助性工具使用
3. 教师信息模块：学籍信息管理、成绩信息管理、课程信息管理
4. 教师维护模块：选修课管理、系统维护、辅助性工具



### 4.各模块结构

**4.1 总体模块结构**

整个项目结构如下图所示，共有三个子系统供三种用户使用（管理员、老师、学生）；

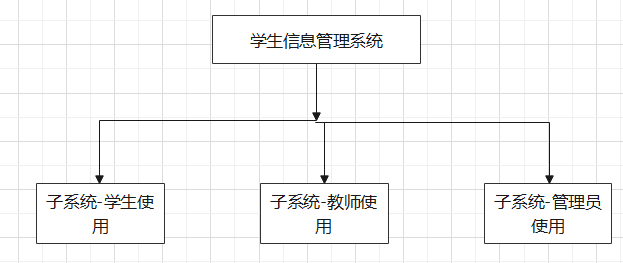


图 1-1 总体模块结构

**4.2 用户登录**

登录模块如下图所示：

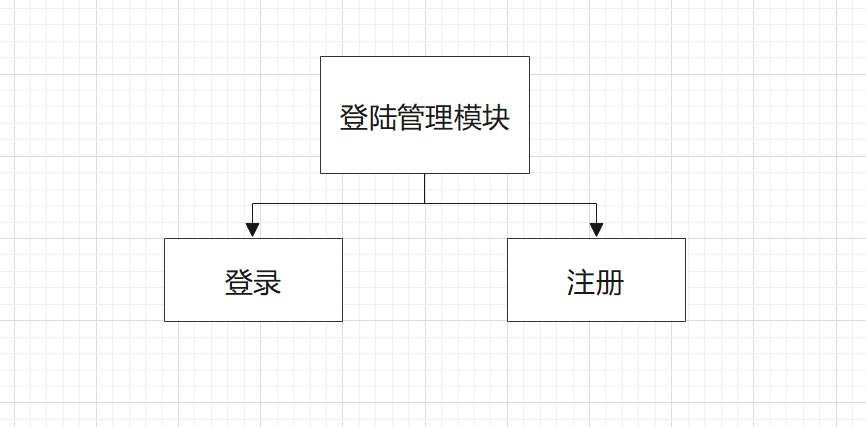
****

图2-1 用户登录模块

**4.3系统管理员服务**

**学生/老师档案的修改、删除、添加**

管理员可以根据规定对系统的老师、学生用户的档案信息进行管理，主要包括增加新档案、删除原有档案和档案信息修改。完成操作后系统对操作进行保存并更新相应老师和学生的档案信息。

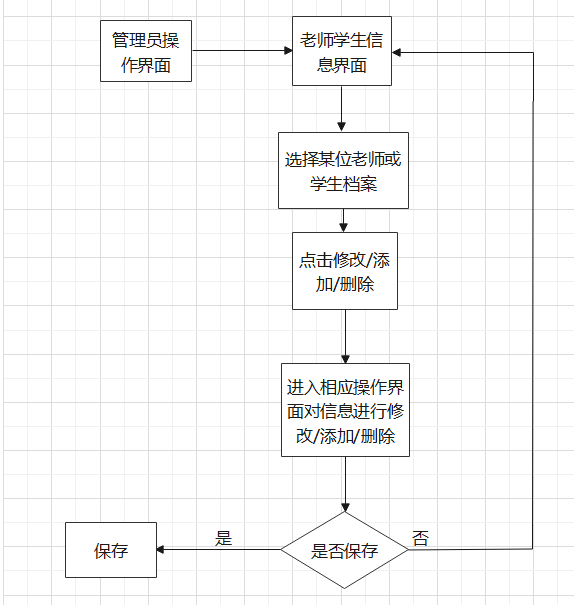


图 3-1 学生/老师档案的修改、删除、添加

**4.4老师服务**

**（1）查询所教的课的成绩：**

老师可以对学生的成绩进行查询，成绩查询可以按学号、姓名进行查询。如图4-1所示：

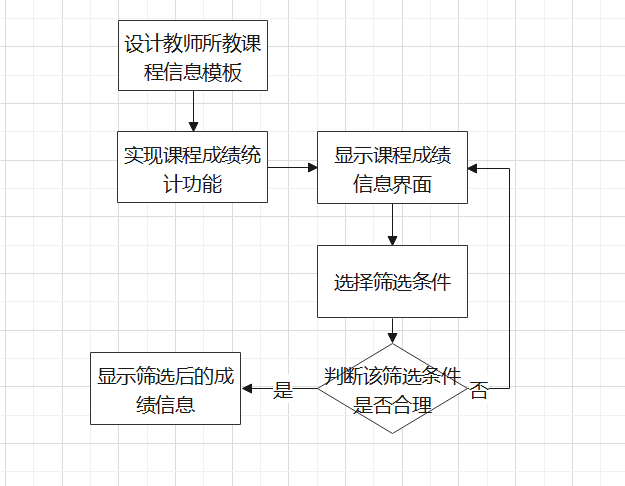
****

图4-1 成绩查询模块

**（2）所有学生档案查询**

老师可以根据规定查看所有学生的档案信息，可以根据学号姓名查询所有的学生档案信息；可以输出学生档案中的学生基本信息、成绩、课程表等。完成操作后系统对操作进行保存，老师得到相应的学生档案信息。

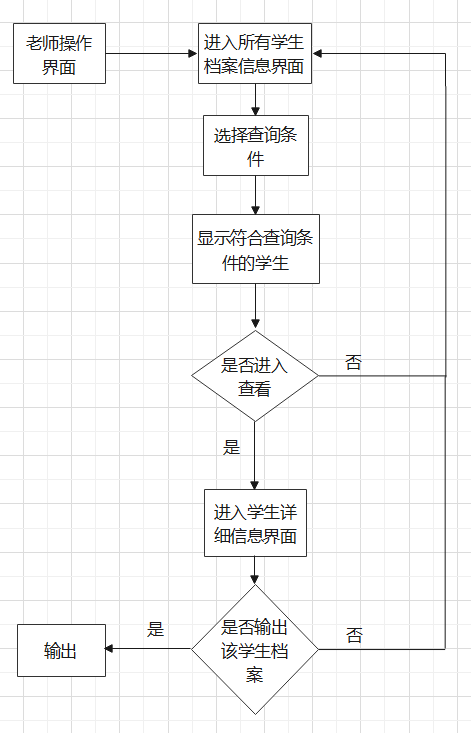


图 4-2 所有学生档案查询

**（3）学生成绩管理：**

老师可以对成绩进行记录进行添加和修改，成绩记录主要包括：学期、学号、姓名、课程号以及成绩字段等。如图4-3所示：

**![C:\Users\14774\Documents\Tencent Files\1477407083\FileRecv\MobileFile\Image\U36](VG5@MQ1GO5NWHW3[[O.png](data:image/png;base64,)**

图4-3成绩查询模块

**（4）增改学生用户信息**

老师可以根据规定对系统的学生用户进行管理，主要包括增加新用户和用户修改。其中用户修改可以实现对学生的密码修改和学生用户名的删去。完成操作后系统对操作进行保存并更新学生用户信息。

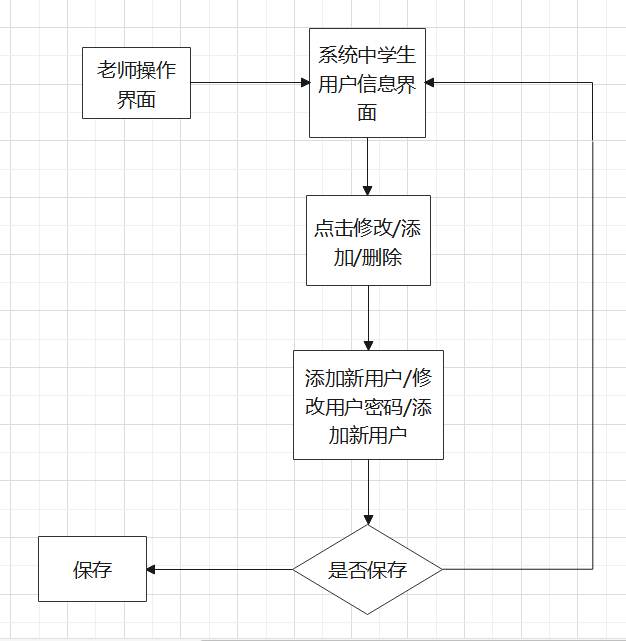


图 4-4 增改学生用户信息

**4.5学生模块**

**（1）打印课程表**

学生可以通过该模块进行课程表查看、打印课程表。如图5-1所示：

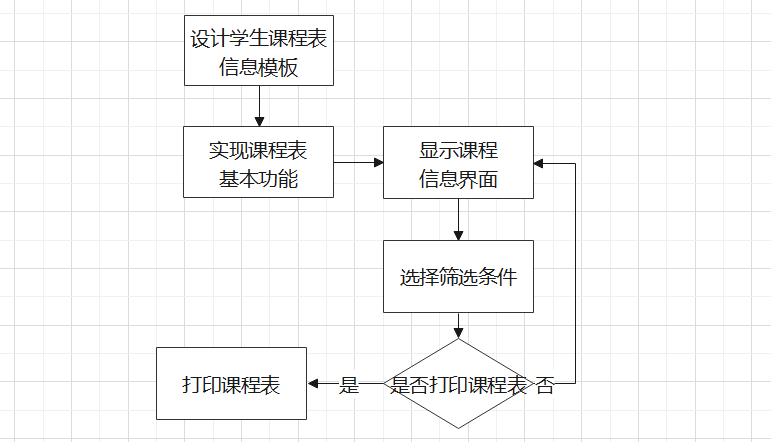


图5-1 打印课程表模块

**（2）选择选修课**

学生可以在系统中查看学校中开设的选修课有哪些，并进行选课，更新学生选课信息和课程表等。

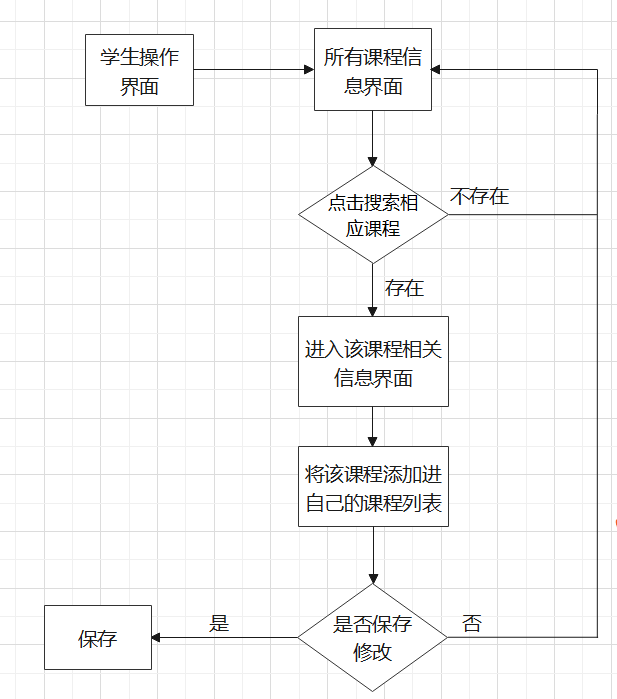


图5-2 选择选修课

**4.6系统辅助工具**

通过该模块用户可以打开一些辅助工具，如记事本和记事本，如图6-1所示：

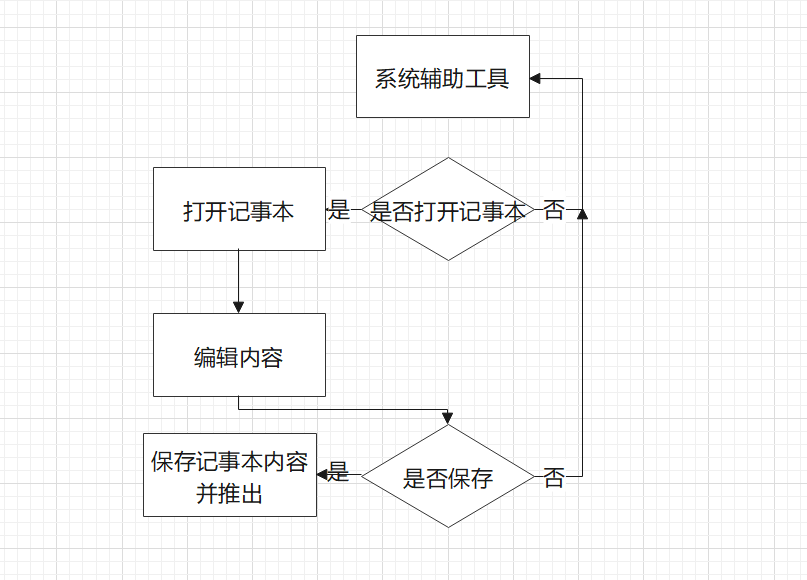


图6-1 记事本模块

### 5.利用MySQL设计数据库

**5.1 学生信息记录表**

该表记录了学生信息管理系统中所有学生的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 学号 | stu\_id | varchar | 20 | 1 | 该项一般为12位数 |
| 姓名 | stu\_name | varchar | 50 |  |  |
| 出生日期 | stu\_birth | date | yyyy-MM-dd |  |  |
| 性别 | stu\_gender | varchar | 10 |  | 男/女 |
| 政治面貌 | stu\_poli | varchar | 50 |  | 群众/共青团员/中共党员/… |
| 入学时间 | stu\_admi | date | yyyy-MM-dd |  |  |
| 个人简介 | stu\_intro | varchar | 2000 |  |  |
| 照片 | stu\_pic |  |  |  |  |
| 家庭住址 | stu\_addr | varchar | 1000 |  |  |
| 邮编 | stu\_post | varchar | 10 |  | 该项一般为6位数 |
| 所在系别号 | stu\_col | varchar | 5 |  | 该项一般为3位数 |
| 所学专业号 | stu\_profe | varchar | 8 |  | 该项一般为5位数 |

**5.2教师信息记录表**

该表记录了学生信息管理系统中所有老师的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 教工号 | tea\_id | varchar | 20 | 1 | 该项一般为12位数 |
| 姓名 | tea\_name | varchar | 50 |  |  |
| 出生日期 | tea\_birth | date | yyyy-MM-dd |  |  |
| 性别 | tea\_gend | varchar | 10 |  | 男/女 |
| 政治面貌 | tea\_poli | varchar | 50 |  | 群众/共青团员/中共党员/… |
| 照片 | tea\_pic |  |  |  |  |
| 家庭住址 | tea\_addr | varchar | 1000 |  |  |
| 邮编 | tea\_post | varchar | 10 |  | 该项一般为6位数 |
| 所在系别号 | tea\_col | varchar | 5 |  | 该项一般为3位数 |
| 所学专业号 | tea\_profe | varchar | 8 |  | 该项一般为5位数 |

**5.3用户信息记录表**

该表记录了使用该学生信息管理系统的所有用户的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 用户号 | user\_id | varchar | 20 | 1 |  |
| 用户密码 | user\_pass | varchar | 30 |  |  |
| 用户属性 | user\_stat | varchar | 10 |  | tea/stu |

**5.4课程信息记录表**

该表记录了学生信息管理系统中所有开设课程种类的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 课程号 | cou\_id | varchar | 15 | 1 | 该项一般为10位数 |
| 课程名称 | cou\_name | varchar | 100 |  |  |
| 开课教师号 | cou\_teach | varchar | 20 |  | 该项一般为12位数 |
| 学分 | cou\_credit | integer |  |  |  |
| 课程属性 | cou\_-chara | varchar | 10 |  | 选修/必修 |

**5.5开课信息记录表**

该表记录了学生信息管理系统中所有开设在具体某一学期的某门课的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 开课号 | les\_id | varchar | 50 | 1 | 该项一般为20位数 |
| 课程号 | cou\_id | varchar | 13 |  | 该项一般为10位数 |
| 开课年度 | les\_year | varchar | 10 |  | 该项一般为4位数 |
| 开课学期 | les\_term | varchar | 3 |  | 1/2 |
| 上课日期 | les\_date | varchar | 3 |  | 1/2/3/4/5/6/7 周几 |
| 上课时间 | les\_time | varchar | 3 |  | 1/2/3/4/5 第几节 |

**5.6成绩信息记录表**

该表记录了学生信息管理系统中所有学生参加的所有开设课程所取得的成绩的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 学号 | stu\_id | varchar | 20 | 1 | 该项一般为12位数 |
| 开课号 | les\_id | varchar | 50 | 1 | 该项一般为20位数 |
| 成绩 | sco\_score | integer |  |  | <100 |

**5.7院系信息记录表**

该表记录了学生信息管理系统中所有院系的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 院系号 | col\_id | varchar | 5 | 1 | 该项一般为3位数 |
| 院系名称 | col\_name | varchar | 50 |  |  |

**5.8专业信息记录表**

该表记录了学生信息管理系统中所有专业的信息。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 存储代码 | 类型 | 长度 | 主码 | 备注 |
| 专业号 | pro\_id | varchar | 5 | 1 | 该项一般为5位数 |
| 专业名称 | pro\_name | varchar | 50 |  |  |
| 所属院系号 | col\_id | varchar | 5 |  | 该项一般为3位数 |

### 6.数据库安全性

Web环境下，数据库的安全尤为重要，数据库安全性的保证，就是网络网站安全的护盾。在各种SQL注入，侵略，攻击的层层威胁下，保证Web数据库的安全性已成为新的课题。

**1、对用户安全管理**

首先要保证的就是数据库用户的权限的安全性，用户在访问数据库时，必须经过身份认证，对非超管用户，必须设定有限的权限和专用的密码。

比如，root是超管用户，其他用户就不能使用这个验证。

**2、定义视图**

为不同的用户定义不同的视图，可以限制用户的访问范围。通过视图机制把需要保密的数据对无权存取这些数据的用户隐藏起来，可以对数据库提供一定程度的安全保护。

**3、数据加密**

数据安全隐患无处不在，因此对数据的加密，是保护数据库安全的有效措施。

数据加密是应用最广、成本最低廉而相对最可靠的方法。数据加密是保护数据在存储和传递过程中不被窃取或修改的有效手段。

**4、启动事务管理和故障恢复**

事务管理和故障恢复主要是对付系统内发生的自然因素故障，保证数据和事务的一致性和完整性。

故障恢复的主要措施是进行日志记录和数据复制；事务运行的每一步结果都记录在系统日志文件中，并且对重要数据进行复制，发生故障时根据日志文件利用数据副本准确地完成事务的恢复。

**5、数据库备份与恢复**

数据库系统必须采取必要的措施，以保证发生故障时，可以恢复数据库。

数据备份需要仔细计划，制定出一个策略测试后再去实施，备份计划也需要不断地调整。

**6、数据库验证方式**

应用程序端连接数据库时使用的数据库验证方式为SQL Server验证。数据库连接串将使用CAPICOM技术加密存储在系统配置文件中。

### 7.数据库管理和设置

将从系统的数据库运行角度出发，对本数据库的属性设置和备份恢复策略提出指导性说明。

另外，我们在数据库的物理结构上还做了优化：数据库表设计满足 3NF，减少了冗余数据。表中的所有数据元素不但要能唯一地被主关键字所标识,而且它们之间还必须相互独立，不存在其他的函数关系接口。

1. **数据库备份恢复策略**

数据备份是容灾的基础，是指为防止系统出现操作失误或系统故障导致数据丢失，而将全部或部分数据集合从应用主机的硬盘或阵列复制到其它的存储介质的过程。传统的数据备份主要是采用内置或外置的磁带机进行冷备份。但是这种方式只能防止操作失误等人为故障，而且其恢复时间也很长。随着技术的不断发展，数据的海量增加，不少的企业开始采用网络备份。网络备份一般通过专业的数据存储管理软件结合相应的硬件和存储设备来实现。

1. 完全备份(Full Backup)

完全备份就是使用磁带对整个系统进行备份，不依赖文件的存档属性来确定备份文件。备份过程中，任何的标记都清除，都标记为已备份。这种备份的优点是：只要一盘磁带就可以恢复数据，并且大大节省了恢复时间。但因为每天都要对数据进行备份，造成备份的数据大量重复。这些重复的数据占用了大量的磁带空间，增加了成本。同时也因为备份的数据量大，对此，那些业务量大，繁忙的公司，这种备份策略是不明智的。

1. 增量备份 (Incremental Backup )

增量备份就是只对当天新的或被修改过的数据进行备份。但它的缺点在于，当灾难发生时，数据的恢复比较麻烦上一次完全备份是在周一，此时，周四产生了故障，若要将数据恢复至周三晚上的状态，则需要先使用周一的完全备份，然后使用周二的差异备份，再使用周三的差异备份，而此时如果周二的差异备份坏了就会导致数据无法恢复至周三的状态。

1. 差异备份(Differential Backup)

差异备份就是在进行了上一次备份后至今，将在这个时间段中产生差异的数据进行备份，这种备份策略的优点是节省了磁带空间，缩短了备份时间。只需要两盘磁带，即全备份磁带与灾难发生前一天的差异备份磁带，就可以将系统恢复。

在实际应用中，备份策略通常是以上三种的结合。例如每周一至周六进行一次增量备份或差异备份，每周日进行完全备份，每月底进行一次完全备份，每年底进行一次完全备份。

**目前我们的方案是使用定时脚本进行数据库的定期自动备份和事件清理。**

1. **数据库运行日常维护**

数据库的运行和维护阶段的主要工作有：  
对数据库性能的监测、分析和改善。  
数据库的转储和恢复。  
维持数据库的安全性和完整性。  
数据库的重组和重构。

### 8.为保证项目的质量属性而增加的设计（实验九）

**8.1 可修改性**

1. **引言(Introduction)**

作为软件开发人员来说频繁的更改需求是一件痛苦的事情，为什么痛苦呢？一方面来自于需求的更迭，另一方面也根我们对软件的合理架构密不可分，一个合理的软件架构足够应付需求的未知变化，这体现了软件的可修改性。在考虑软件的可修改性上作出的架构很大程度上将减少需求更迭给我们带来的“痛苦”。

1. **可修改性理解（Modifiability Understanding）**

可修改性理解可理解为：指系统或软件的能够快速地以较高的性价比对系统进行变更的能力。比如说：对于一个网站，我们要修改它某一板块的UI界面，当我们对界面进行修改时是否会引起对另一个UI模块的影响，是否会引起后台控制，业务逻辑代码的变更，是否会引起整个网站的崩溃，这体现了一个网站的整个架构的是否具备可修改性。

1. **引起修改的因素（Factors causing revision）**

为什么会引起系统的修改？笔者认为最主要（甚至是唯一）的因素是：需求，这个需求会影响到系统给我们带来的收益。它可包含两个方面：1、用户需求，2、系统内在需求。首先在这阐述下“需求，成本，修改”三者的关系：需求无处不在，时时刻刻产生，判别一个需求的重要性来自于它对系统的成本产生的影响，如果严重影响了系统带来的收益，那必须对系统进行修改，如果某一部分相对于系统收益来说微不足道，甚至不会影响系统的收益，那改不改都“无可厚非”，而且有可能你对这部分修改了还会引起系统的故障。例如：一个网站的图标有人觉得不好看，有人觉得不错。场景是：使用的人们已经习惯这个图标的表示或者惯性的认为这个图标代表什么。那么是否对这个图标进行修改？我以为没有必要修改，如果你修改了反而可能会引起使用者的错误判断，比如：这个网站是停止运维了吗？好，那我去用其他的系统吧。类似错误的判断会影响到系统的收益，所以不进行修改就挺好的。

以上说明可理解为用户需求引起的修改，那么在解释下系统内在需求引起变更。

以淘宝为说明对象，根具相关资料，初期的淘宝使用的是MySQL作为它的数据库，众所周知MySQL是一个适用于小型企业的数据库，对于起初的淘宝也是适用的，因为没有产生MySQL无法处理的大量数据及相应的实时处理业务。但是近几年来淘宝的数据爆炸产生，在2012年11月30号到达全天访问用户总人数：2亿1千3百万，占中国网民40%；高峰期：每分钟成交订单89678笔，淘宝历年交易额如图1所示。

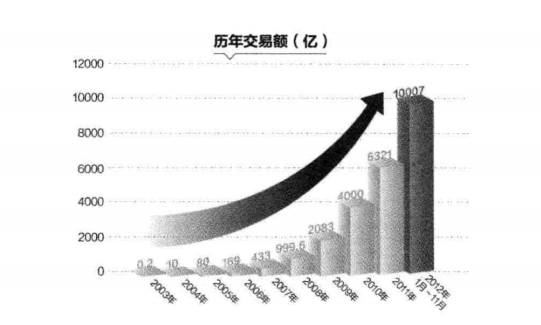


图1  淘宝历年交易额

这些大量业务数据就会引起系统“自我反馈”，如果不进行系统数据库方面的更新迭代会严重影响到淘宝未来的收益。正是在这系统本身产生的需求下，淘宝进行着对数据存储体系架构的不断升级优化。

1. **可修改性战术（Modifiable tactics）**

软件怎样具有可修改性，这里将介绍两种战术。

首先说明一下可修改性战术。可修改性战术的目标是控制实现、测试和部署变更的时间和成本。在具体的解释为根据相关战术，策略在时间和成本允许的范围内完成系统的相关变更。以下讨论两种可修改性战术：局部化修改和防止连锁反应

1. 局部化修改战术，目标是把变更限制在一定范围，在设计期间为模块分配职责，把预期变更限制在一定范围内。关键点就是“限制变更范围”防止变更的内容对其他内容（或者说程序）产生影响。
2. 防止连锁反应。我们平时编程无论是写函数还是写类，都会被其它的类或函数（方法）调用，如何实现对被调用部分的修改不会引起对调用它的部分的影响，这就是可以防止连锁反应。

无论是局部化修改还是防止连锁反应，都是基于“高内聚，低耦合”思想。那么根据软件设计模式的设计原则来分析：

1. 局部化意味着实现“模块化”思想，这里模块这样解释，一个模块只完成一个部分，这就符合设计模式中“单一职责原则”的设计原则，使每一个模块责任单一，防职责过多引起引起整体变更时的繁琐，复杂；
2. 要符合“接口隔离”设计原则，为后期的变更提供接口，当然也要规定接口调用的规范。同时根据“里氏代换”原则将接口的实例化由接口实现类完成，在此之上基于“依赖倒转原则”搭建的上层服务如果需要修改相关服务可以实现局部化修改，不用再对上层服务进行修改。在防止连锁反应的战术中“维持现有接口”就是为上层服务提供一个可使用的接口，当进行变更时对接口的实现类变更即可，无论是实现“适配器”还是“信息隐藏”都可封装在接口实现类中。
3. 降低模块之间的依赖性是防止连锁反应的必要条件。可据“迪米特法则”来实现。所谓“迪米特法则”，在C++中可理解为“友元”，java理解：在类与类间加入第三方类作为沟通，信息交流的“缓冲”，两个类间不直接相互依赖。这种第三方类可叫做“朋友”，可叫做“仲裁者”。在设计模式中可表示为“桥”“调停者”“代理”“工厂”，这些都是为了降低类间依赖。当然这又产生一个问题：大量使用第三方类使程序繁琐，复杂，这就需要架构师设计好一个明确的类图了。

以上基于软件设计模式分析了两个可修改性战术。

1. **结论(Conclusion)**

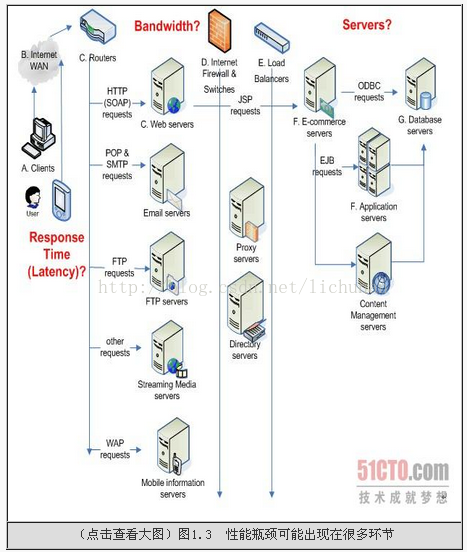
我们在做软件开发时，肯定会自觉的将需求抽象成软件的功能，这是毋庸置疑的，但是我们在开发前的架构设计阶段不单单要考虑软件的将要完成的功能，还要将软件的质量属性考虑进去，因为这将影响着一个软件的长期发展。

**8.2性能**

**8.2.1硬件性能上的改善**

目前，大部分系统都是为多用户、跨地域、多部门机构提供服务的，目前一般中小企业的员工人数也有几十号人，实际并发数量比较小，在软件性能方面影响不大，但是对于中大型企业、大型国企或跨国跨区域的集团企业，公司的员工有几千人，甚至几万人，而且一般一套系统都要运行很多年，这样存量数据也非常惊人，对于软件系统的性能表现影响巨大。

影响性能的主要因素包括：硬件设施、网络、操作系统、并发用户数、系统积累的数据量、中间件等，如图所示。如果系统中哪一个环节出现了性能瓶颈，都会严重影响系统的正常运行。



硬件是决定一套系统性能的关键因素之一，主要包括服务器的CPU、内存、磁盘I/O使用情况，如果系统并发用户数增大而内存不足，会导致CPU和磁盘使用压力。在高性能的服务器配置情况下，可以大大提高系统性能，在单实例服务器配置的硬件条件无法满足现有性能需求的情况下，可以考虑使用多台服务器以集群模式部署，配备负载均衡器进行负载均衡处理。

**8.2.2软件性能**

1. **响应时间**

响应时间是指系统对请求作出响应的时间。直观上看，这个指标与人对软件性能的主观感受是非常一致的，因为它完整地记录了整个计算机系统处理请求的时间。由于一个系统通常会提供许多功能，而不同功能的处理逻辑也千差万别，因而不同功能的响应时间也不尽相同，甚至同一功能在不同输入数据的情况下响应时间也不相同。所以，在讨论一个系统的响应时间时，人们通常是指该系统所有功能的平均时间或者所有功能的最大响应时间。当然，往往也需要对每个或每组功能讨论其平均响应时间和最大响应时间。

对于单机的没有并发操作的应用系统而言，人们普遍认为响应时间是一个合理且准确的性能指标。需要指出的是，响应时间的绝对值并不能直接反映软件的性能的高低，软件性能的高低实际上取决于用户对该响应时间的接受程度。对于一个游戏软件来说，响应时间小于100毫秒应该是不错的，响应时间在1秒左右可能属于勉强可以接受，如果响应时间达到3秒就完全难以接受了。而对于编译系统来说，完整编译一个较大规模软件的源代码可能需要几十分钟甚至更长时间，但这些响应时间对于用户来说都是可以接受的。

1. **系统响应时间和应用延迟时间**

虽然软件性能指标本身只涉及软件性能的度量，但考虑到软件性能测试的主要目的是测试和改善所开发软件的性能，对于复杂的网络化的软件而言，简单地用响应时间进行度量就不一定合适了。

考虑一个普通的网站系统。开发该网站系统时，软件开发实际上只集中在服务器端，因为客户端的软件是标准的浏览器。虽然用户看到的响应时间时使用特定客户端计算机上的特定浏览器浏览该网站的响应时间，但是在讨论软件性能时更关心所开发网站软件本身的“响应时间”。也就是说，可以把用户感受到的响应时间划分为“呈现时间”和“系统响应时间”，前者是指客户端的浏览器在接收到网站数据时呈现页面所需的时间，而后者是指客户端接收到用户请求到客户端接收到服务器发来的数据所需的时间。显然，软件性能测试更关心“系统响应时间”，因为“呈现时间”与客户端计算机和浏览器有关，而与所开发的网站软件没有太大的关系。

如果仔细分析这个例子，还可以把“系统响应时间”进一步分解为“网络传输时间”和“应用延迟时间”，其中前者是指数据（包括请求数据和响应数据）在客户端和服务器端进行传输的时间，而后者是指网站软件实际处理请求所需的时间。类似的，软件性能测试也更关心“应用延迟时间”。实际上，这种分解还可以继续下去，如果该网站系统使用了数据库，我们可以把“数据库延迟时间”分离出来，如果该网站系统使用了中间件，还可以把“中间件延迟时间”也分离出来。

以上的时间分解实际上有两方面的目的。首先，人们通常希望把与所开发软件直接相关的延迟时间和与所开发软件不相关的延迟时间分离开，因为改善前者往往需要开发人员修改程序代码，而改善后者不需要开发人员修改代码，很多时候，开发人员对后者甚至是无能为力的。其次，详细的分解有助于开发人员分析哪些部分是影响软件性能的主要因素，以便于实时性能改善方案。

1. **吞吐量**

吞吐量是指系统在单位时间内处理请求的数量。对于无并发的应用系统而言，吞吐量与响应时间成严格的反比关系，实际上此时吞吐量就是响应时间的倒数。前面已经说过，对于单用户的系统，响应时间（或者系统响应时间和应用延迟时间）可以很好地度量系统的性能，但对于并发系统，通常需要用吞吐量作为性能指标。

对于一个多用户的系统，如果只有一个用户使用时系统的平均响应时间是t，当有你n个用户使用时，每个用户看到的响应时间通常并不是n×t，而往往比n×t小很多（当然，在某些特殊情况下也可能比n×t大，甚至大很多）。这是因为处理每个请求需要用到很多资源，由于每个请求的处理过程中有许多步骤难以并发执行，这导致在具体的一个时间点，所占资源往往并不多。也就是说在处理单个请求时，在每个时间点都可能有许多资源被闲置，当处理多个请求时，如果资源配置合理，每个用户看到的平均响应时间并不随用户数的增加而线性增加。实际上，不同系统的平均响应时间随用户数增加而增长的速度也不大相同，这也是采用吞吐量来度量并发系统的性能的主要原因。一般而言，吞吐量是一个比较通用的指标，两个具有不同用户数和用户使用模式的系统，如果其最大吞吐量基本一致，则可以判断两个系统的处理能力基本一致。

1. **并发用户数**

并发用户数是指系统可以同时承载的正常使用系统功能的用户的数量。与吞吐量相比，并发用户数是一个更直观但也更笼统的性能指标。实际上，并发用户数是一个非常不准确的指标，因为用户不同的使用模式会导致不同用户在单位时间发出不同数量的请求。以网站系统为例，假设用户只有注册后才能使用，但注册用户并不是每时每刻都在使用该网站，因此具体一个时刻只有部分注册用户同时在线，在线用户就在浏览网站时会花很多时间阅读网站上的信息，因而具体一个时刻只有部分在线用户同时向系统发出请求。这样，对于网站系统我们会有三个关于用户数的统计数字：注册用户数、在线用户数和同时发请求用户数。由于注册用户可能长时间不登陆网站，使用注册用户数作为性能指标会造成很大的误差。而在线用户数和同时发请求用户数都可以作为性能指标。相比而言，以在线用户作为性能指标更直观些，而以同时发请求用户数作为性能指标更准确些。

1. **资源利用率**

资源利用率反映的是在一段时间内资源平均被占用的情况。对于数量为1的资源，资源利用率可以表示为被占用的时间与整段时间的比值；对于数量不为1的资源，资源利用率可以表示为在该段时间内平均被占用的资源数与总资源数的比值。

**8.2.3证据案例**

<https://blog.csdn.net/zhaoenweiex/article/details/107436992>

<https://blog.csdn.net/zhaoenweiex/article/details/107526638>

如上图两个链接，在将外部系统的数据导入到本系统中并在地图上显示出来的时候，在原始思路的基础上批量进行优化、删除不必要的操作，加快最终模型性能，一些经验为：

·不要过早优化性能

·识别关键业务指标和影响因素

·建立有效的测量基准和方法

·充分利用20/80原则，解决核心瓶颈

·了解不同路线的极限

·以业务为导向，适当容许技术债务

**8.3安全性**

1. **软件特征化：**

学生信息管理系统具有以下功能：

a.学籍管理：对学生的信息档案进行管理，主要包括学生档案信息的修改、删除、添加、档案查询及档案输出等，其中档案查询可以根据学号、姓名查询需要的信息；档案输出可以输出学生档案、成绩、课程表等。学生档案记录主要包括：学号、姓名、出生日期、年龄、性别、政治面貌、入学时间、个人简介、照片、家庭住址、邮编、所在系别以及所学专业等字段。

b.成绩管理：对学生的成绩进行管理，主要就是成绩查询，成绩查询可以按学号、姓名进行查询。成绩记录主要包括：学期、学号、姓名、课程号以及成绩字段等。

c.课程管理：学生可以通过该模块进行课程表查看、打印课程表。

d.选修课选择：对学生报选修课进行管理，学生可以对自己喜爱的课程进行选择。

e.系统维护：教师对学生的密码进行管理，主要包括增加新用户和用户修改。其中用户修改可以实现对学生的密码修改和学生用户名的删去。

f.系统辅助工具：通过该模块用户可以打开一些辅助工具，如记事本和记事本。

g.退出系统：离开本学生信息管理系统

1. **威胁分析:**

可能出现的威胁：

a.觊觎学校学生及教师信息的不法分子可能攻击网站；

b.管理员不小心错误地改动了账号信息；

c.得到了错误信息的管理员修改了错误的信息；

d. 不法分子获得了管理员向泄露的账号信息；

1. **漏洞评估：**

只要登陆管理员账号即可修改指定信息，没有在修改信息时再次确认信息；

1. **风险可能性决策：**

a.管理员错误地修改了信息，会导致被修改的信息产生错误，是局部性的，只在特定情况下产生影响；

b.管理员账号信息泄露较为严重，可能导致大批信息的丢失，极大的影响了信息的可靠性；

c.被不法分子攻击最为严重，可能导致网站崩溃，信息无法使用，甚至设备损坏；

1. **风险影响决策：**

a.业务终止性的：

被不法分子攻击后，可能导致网站崩溃，信息无法使用和恢复，设备损坏

b.损害性的：

管理员信息泄露后，被修改的信息可以重新统计进行恢复，但是会耗费巨大的人力和时间。

c.可恢复的：

学籍等信息可以直接从教务处导入，无需耗费巨大人力和时间；

d.妨碍：

本来就是错误的信息被删除，或者是即将被删除的信息丢失；

1. **风险缓解计划：**

a.对于网站被攻击：

网站建设完毕后及时封闭端口，并设定严格的操作权限，防止被不法分子钻空子，及时完善，不断升级网络安全加密技术等。

b.对于局部信息错误修改：

增设修改信息时的身份核验，避免无心导致的错误。

**8.4可复用性**

**8.4.1软件可复用性含义**

软件复用是使用现有软件组件实现或更新软件系统的过程。

**8.4.2软件可复用性流程**

设计出可复用性的软件->利用设计的可复用的软件进行应用开发.

**8.4.3两种基于软件可复用性实现**

1. 面向复用编程：开发出可复用的软件。构建可复用的软件,实现复杂；分析各个实现的基本共性操作,实现共性操作。
2. 基于复用编程：利用已有的可复用软件搭建应用系统。根据具体实现利用已实现的可复用软件实现自己的具体实现。

**8.4.4面向复用的软件构造技术**

1. 继承与重写( 行为子类型Liskov替换原则)：继承就是子类继承父类。继承要满足行为子类型和 Liskov替换原则。行为子类型: 就是子类继承父类,父类的行为子类一定要实现。在可以使用父类的场景，都可以用子类代替而不会有任何问题；
2. 重载：方法名相同,参数不同；
3. 参数多态与泛型编程：因List类型擦除，引出了泛型List—这是一个未知类型的列表接收A的父类接收A的子类；
4. 组合与委托：将功能实现利用接口抛出去,利用别的类实现具体功能。不同的接口定义不同的功能。通过接口组合完成各种操作实现。将接口作为该类的一个属性,使用接口属性使用功能（委托）。

**8.4.5面向复用的设计模式**

1. 创建型模式：

a.工厂方法模式：当client不知道要创建哪个具体类的实例，或者不想在client代码中指明要具体创建的实例时，用工厂方法。定义一个用于创建对象的接口，让其子类来决定实例化哪一个类，从而使一个类的实例化延迟到其子类。相当于工厂和产品，产品不直接使用产品的构造方法实例，通过工厂提供的方法实例化；

b.抽象工厂模式：提供接口以创建一组相关/相互依赖的对象， 但不需要指明其具体类。创建的不是一个完整产品，而是“产品族”（遵循 固定搭配规则的多类产品的实例），得到的结果是：多个不同产品的object，各产品创建过程对client可见，但“搭配”不能改变。本质上，Abstract Factory是把多类产品的factory method组合在一起；

1. 结构型模式：

a.适配器模式（委托实现）：将某个类/接口转换为client期望的其他形式。通过增加一个接口，将已存在的子类封装起来，client面向接口编程，从而隐藏了具体子类。将已实现的功能通过接口满足client实现；

b.装饰器模式：抽象接口,共性操作。用每个子类实现不同的特性。如果需要特性的任意组合,对每一个特性构造子类，通过委派机制增加到对象上；

c.外观模式：提供一个统一的接口来取代一系列小接口调用，相当于对复杂统做了一个封装，简化客户端使用。各个小接口抛出，具体功能子类实现，外观模式提供统一的接口，根据参数选择不同接口的操作；

1. 行为类模式：

a.策略模式（委托实现）：为不同的实现算法构造抽象接口，利用delegation，运行时动态传入client倾向的算法类实例；

b.模板模式：两种实现：（a）黑盒框架：用户不知道内部实现，总体框架已经实现，将需要实现的方法通过接口抛出，子类具体实现，再通过委托传给总体框架；（b）白盒框架：直接继承父类，继承重写需要实现的功能；

c.迭代器模式：客户端希望遍历被放入容器/集合类的一组ADT对象，无需关心容器的具体类型。Iterator pattern：让自己的集合类实现Iterable接口，并实现自己的独特Iterator迭代器(hasNext, next, remove)，允许客户端利用这个迭代器进行显式或隐式的迭代遍历；

**8.4.6实验中的可复用性实例**

在学生信息管理系统的实验中，我们几乎将所有函数模块化，经过层层调用来实现某些功能。

一旦要修改某些基础功能时，仅通过修改最终调用的函数就可以改变与这个函数相关的所有功能。

当我们相信加一个新的功能时，可以通过调用已有函数和新加少量函数，并将它们组织起来，就可以实现我们想要的功能。

### 10.附录：软件架构成员学习报告（实验七）

**10.1 岳大帅**

**1．体系结构的应用场景**

首先需要承认的是，在软件设计的不同阶段需要不同的设计方法。面对不同粒度，不同复杂度的系统，你需要使用不同的思考框架。

体系结构设计与算法设计就需要不同的思考框架。

在算法设计中，动态规划、贪心算法等这些算法思想是非常重要的思考方式/思考框架，他们在一定程度上能够保证快速得到计算的结果。它们面临的场景更多类似于“快速在某个字符串中快速寻找某个子串”，这样一些小问题。

算法虽然能够攻克粒度较小的问题，但是在构建系统的时候，往往面临的问题是：软件系统规模大且复杂性高。这时对系统的全局结构设计和规划更加重要。比如说，给你一个需求，设计一个图书管理系统，这时候动态规划这些思考框架就显得有些力不从心了。这时候，体系结构设计就应运而生了，体系结构设计是一种构建复杂系统时的思考框架，注意它服务的对象是粗粒度的复杂系统，所以你也不要奢望它能对你的算法设计有太大的帮助。

**2．理解软件体系结构**

软件体系结构（Software Architecture）包括构成系统的设计元素的描述、 设计元素

之间的交互、 设计元素的组合模式以及在这些模式中的约束。

理解上述抽象概念并不是一件容易的事情，需要一定的工程经验。这里简单谈谈自己的理解。

理解的角度1：（集合角度）

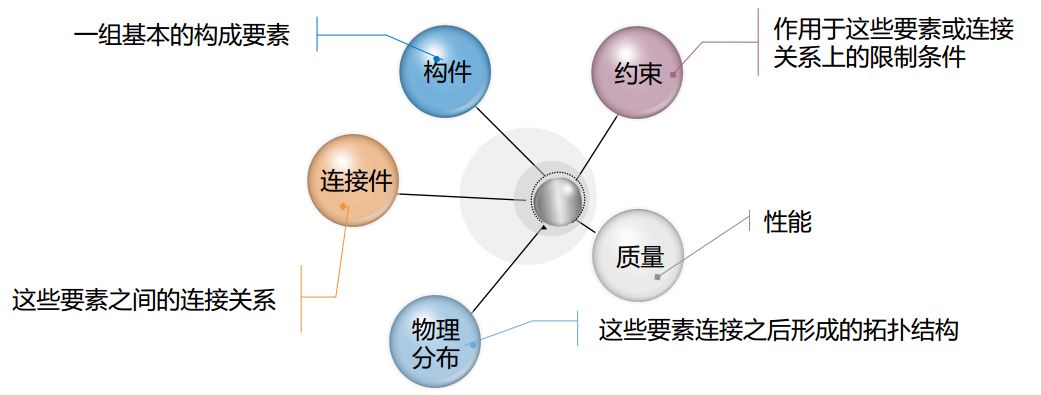
在计算机中，集合是我们一个有利的工具，我们定义事物，理解事物都可以从中获得启发。在集合中，你需要规定基本的元素（这里指构件这些要素），同时还需要制定这些元素的联系方式（这里是指元素的组合模式，及模式中的约束），适当的时候你还需要对具有某些特殊性质的元素进行分类（这里根据元素的作用，将元素分为了两大类：构件 & 连接件）

接下来，你可以从利用常规的集合理论理解该思考框架了。

理解的角度2：（实际意义）

我们人类解决复杂问题的方式，基本都是采用分解的方式，即分而治之。所以给定一个系统的时候，我们尝试将其分解成不同的部分，按照一定的划分标准（比如，功能作用），我们可将其分出实际进行处理的模块（构件、连接件），以及负责连接（负责通信）的要素（分布，约束）等。

举个简单的例子，以 “人体”这个系统为例，根据人体各部分的功能，我们可将人体分为不同的构件：头、手、中间部位身体、腿。但是只具备这些要素还是不够的，他们必须连起来，相互配合才能形成真正的人体。



简而言之，软件体系结构 = 构件 + 连接件 + 约束 。

**3．体系结构的目标**

该体系结构的主要目标是确定影响应用程序结构的需求。布局合理的体系结构可减少与构建技术解决方案相关的业务风险，并在业务和技术要求之间架起一座桥梁。

其他一些目标如下-

公开系统的结构，但隐藏其实现细节。

实现所有用例和方案。

尝试解决各种利益相关者的要求。

处理功能和质量要求。

降低所有权目标并改善组织的市场地位。

改善系统提供的质量和功能。

提高外部对组织或系统的信心。

但是软件体系结构仍然是软件工程中的新兴学科。它具有以下限制-

缺乏用于表示架构的工具和标准化方法。

缺乏分析方法来预测体系结构是否会导致实现符合要求的分析方法。

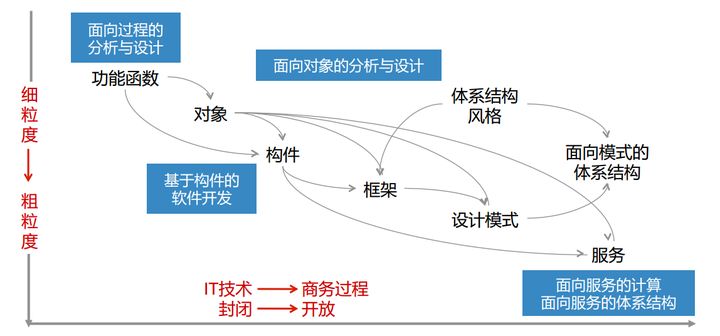
缺乏对体系结构设计对软件开发的重要性的认识。

缺乏对软件架构师角色的了解，以及利益相关者之间的沟通不畅。

缺乏对设计过程的了解，设计经验和设计评估。

**4．体系结构的发展**

现在软件的复杂性及多变性，导致了软件粒度越来越粗，越来越开放。



**5．软件体系结构的建模研究**

研究软件体系结构的首要问题是如何表示软件体系结构，即如何对软件体系结构建模。根据建模的侧重点的不同，可以将软件体系结构的模型分为5种：结构模型、框架模型、动态模型、过程模型和功能模型。在这5个模型中，最常用的是结构模型和动态模型。

结构模型

这是一个最直观、最普遍的建模方法。这种方法以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质。研究结构模型的核心是体系结构描述语言。

框架模型

框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

动态模型

动态模型是对结构或框架模型的补充，研究系统的"大颗粒"的行为性质。例如，描述系统的重新配置或演化。动态可能指系统总体结构的配置、建立或拆除通信通道或计算的过程。这类系统常是激励型的。

过程模型

过程模型研究构造系统的步骤和过程。因而结构是遵循某些过程脚本的结果。

功能模型

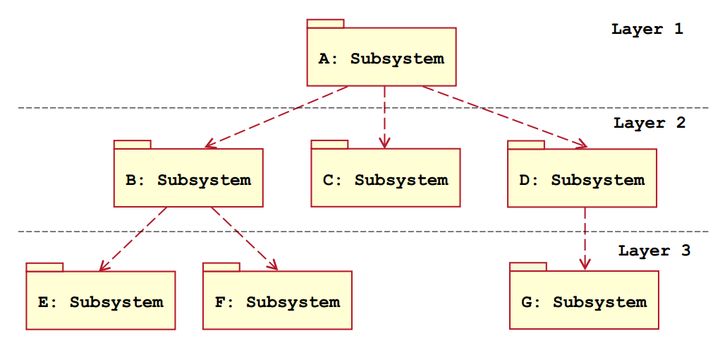
该模型认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。它可以看作是一种特殊的框架模型。

这5种模型各有所长，也许将5种模型有机地统一在一起，形成一个完整的模型来刻画软件体系结构更合适。例如，Kruchten在1995年提出了一个“4+1”的视角模型。“4+1”模型从5个不同的视角包括逻辑视角、过程视角、物理视角、开发视角和场景视角来描述软件体系结构。每一个视角只关心系统的一个侧面，5个视角结合在一起才能够反映系统的软件体系结构的全部内容。

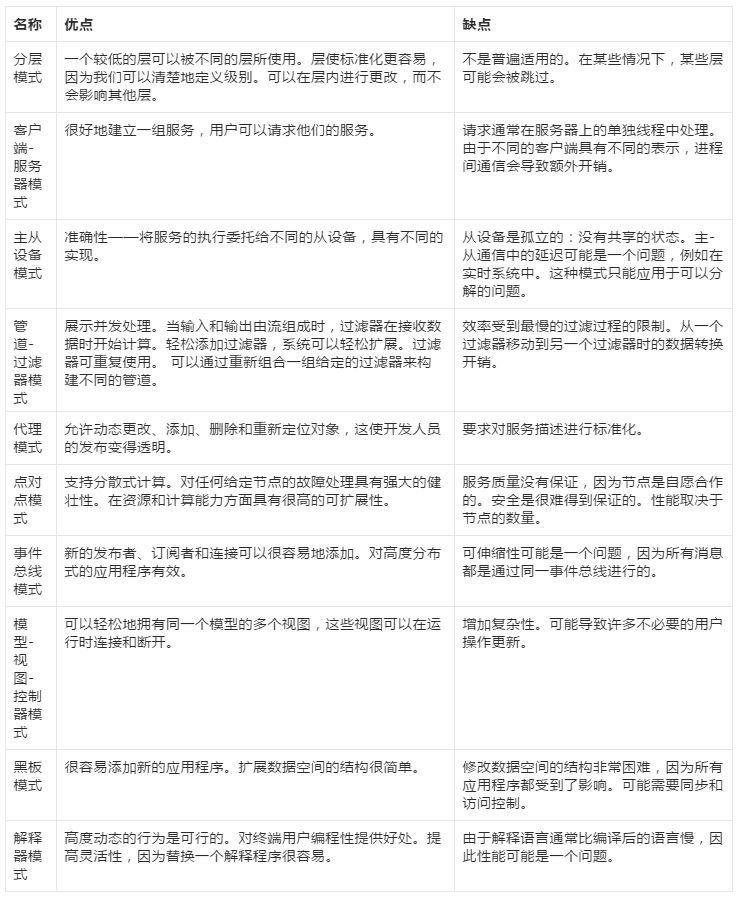
层次化

先给出层次化的表现形式：

层次化是模块化的一种特例。在模块化的基础上，加入了一些限制。在层次化结构中，layer可以作为一种特殊的模块（包含特定的子系统集合），在这类模块（layer）的联系中，规定了layer模块只能被设计成类似于双向链表的顺序结构，而不能出现树状结构或者图状结构。



10种常见的体系架构模式优缺点



**6．软件体系结构仍是研究热点**

当前，体系结构仍是一个非常新的研究领域，其概念还相当模糊。但软件体系结构作为软件工程领域中的一个组成部分，已经取得了长足的发展，受到大多数软件系统设计和研究人员的重视。

软件体系结构目前较活跃的研究方向包括：（1）软件体系结构形式基础的研究；（2）针对软件体系结构描述中特有的问题研究新的专门的高级语言；（3）建立用于度量和评价软件体系结构的模型和方法；（4）建立面向专门领域的软件体系结构范型库。（5）把软件体系结构从目前的直觉和经验状态过渡到理论。

**10.2 孙颖**

**1．什么是软件架构？**

答：程序或计算系统的软件架构是该系统的结构，包括软件组件、那些组件的外部可见的属性，以及那些组件之间的关系。

**2．什么是SAD？**

答：SAD即软件架构文档（Software Architecture Document）。

软件体系结构文档的内容应该提供整个软件体系结构文档的概述。

它应该包括软件体系结构文档的目的、范围、定义、缩略语、缩写、引用和概述。

**3．为什么需要编写SAD？**

答：

a.从软件架构实现本身来看：

软件架构可以递归地使用。考虑一个属于某个系统的软件架构组成部分的软件组件 (C1)。软件架构师将该组件及其应该公开的属性、功能和非功能特性及其与其他软件组件的关系交给系统设计人员。设计人员在分析软件组件 C1 之后，决定将该组件分解为更细粒度的组件（C11、C12 和 C13），其中每个组件提供可重用的功能，这些功能将用于实现 C1 的要求属性。设计人员详细设计了 C11、C12、C13 及其接口。

此时，对设计人员来说，C11、C12 和 C13 是体系结构构造（或组件）；其中每个构造具有显式定义的外部接口。对设计人员来说，C11、C12 和 C13 是软件组件 C1 的体系结构，并且这些构造需要进一步的改进和设计，以处理它们的内部实现。通过将大型、复杂的系统划分为小型的构成部分并集中于每个部分，可以递归地使用体系结构。

体系结构使用共同满足行为和质量目标的体系结构构建块将系统绑定在一起。参与者必须能够理解体系结构。因此必须为体系结构编写足够的文档说明，包括SAD。

b.从软件架构的参与者来看：

软件架构的参与者包括：体系结构的下游设计和实现用户。为体系结构的定义、维护和增强功能进行投资的人。

而向参与者传达正在构建的系统蓝图的关键是为系统体系结构编写文档说明。软件架构通过不同的视图进行表示——功能、操作、决策等等。没有任何单一视图能够表示整个体系结构。并非所有视图都需要表示特定企业或问题领域的系统体系结构。架构师将确定足以表示所需软件架构范畴的视图集。

综上，SAD提供了系统的全面架构概述，使用许多不同的架构视图来描述系统的不同方面。其目的是捕获和传递在系统上做出的重要架构决策，且同时确定文档的特定受众，并指出各类使用者将如何使用文档。

**4．SAD的实用性？**

答：

1. 可以向开发团队和业务及 IT 参与者传达有关该不断发展的系统的信息;
2. 有利于架构师弥补用白板描述解决方案（使用框线图方法）与以对下游设计和实现团队有意义的方式表示解决方案之间众所周知的差距；
3. 促进创建切合实际并且可以系统开发（例如遵循标准模板）的体系结构构件；
4. 帮助新成员快速熟悉工程内容，更快成为高效参与者。

**5．SAD的基本构成？**

答：

**1. 介绍**

[软件体系结构文档的介绍提供整个软件体系结构文档的概述。其包括软件体系结构文档的目的、范围、定义、缩略语、缩写、引用和概述。]

1.1目的

本文档提供系统的全面架构概述，使用许多不同的架构视图来描述系统的不同方面。其目的是捕获和传递在系统上做出的重要架构决策。同时应该确定文档的特定受众，并指出他们将如何使用文档。

1.2范围

a.软件体系结构文档应用于什么的简要描述;

b.受本文档影响的内容。

1.3定义、缩写和缩略语

提供正确解释软件体系结构文档所需的所有术语、缩写词和缩写词的定义。这些信息可以通过参考术语表项目来提供。

1.4参考资料

提供软件体系结构文档中其他地方引用的所有文档的完整列表。每个文件都应该通过标题、报告编号(如果适用)、日期和出版机构来标识。指定可以获得引用的来源。该信息可以通过参考附录或其他文件来提供。

1.5概述

描述软件体系结构文档的其余部分包含什么，并解释软件体系结构文档是如何组织的。

**2. 架构表示**

描述当前系统的软件架构是什么，以及它是如何表示的。在用例视图、逻辑视图、流程视图、部署视图和实现视图中，枚举必要的视图，并且对于每个视图，解释它包含的模型元素的类型。

**3.架构目标和约束**

描述对体系结构有重大影响的软件需求和目标，例如，安全性、安全性、隐私性、成品产品的使用、可移植性、分发和重用。捕获可能应用的特殊约束、设计和实现策略、开发工具、团队结构、进度、遗留代码，等等。

**4. 用例视图**

列出用例模型中的用例或场景，如果它们代表最终系统的一些重要的、中心的功能，或者如果它们有一个很大的体系结构覆盖范围——它们执行许多体系结构元素，或者如果它们强调或说明体系结构的一个特定的、微妙的点。

4.1用例实现

通过给出一些选定的用例(或场景)实现来说明软件实际上是如何工作的，同时解释各种设计模型元素是如何贡献它们的功能的。

**5. 逻辑视图**

描述设计模型在体系结构上的重要部分，例如分解为子系统和包。而对于每个重要的包，将其分解为类和类实用程序。引入体系结构上重要的类，并描述它们的职责，以及一些非常重要的关系、操作和属性。

5.1概述

根据包的层次结构和层描述了设计模型的总体分解。

5.2架构上重要的设计包

对于每个重要的包，包括一个带有它的名称的分段，它的简要描述，以及包含在包中的所有重要的类和包的图表。

对于包中的每个重要类，包括它的名称、简要描述，以及一些主要职责、操作和属性的描述(可选)。

**6. 流程视图**

描述系统分解为轻量级进程(单线程控制)和重量级进程(轻量级进程分组)。通过通信或交互的过程组来组织部分。描述进程之间通信的主要模式，如消息传递、中断和集合。

**7. 部署视图**

介绍一种或多种部署并运行软件的物理网络(硬件)配置。它是部署模型的视图。对于每个配置，它至少应该指出执行软件的物理节点(计算机、cpu)和它们的互连(总线、局域网、点对点等等)。还包括一个流程视图到物理节点的流程映射。

**8. 实现视图**

描述实现模型的整体结构，将软件分解为实现模型中的层和子系统，以及任何体系结构上重要的组件。

8.1概述

命名并定义各个层及其内容，管理包含到给定层的规则，以及层之间的边界。包括一个显示层之间关系的组件图。

8.2层

对于每一层，包括一个带有其名称的分段，位于该层中的子系统的枚举，以及一个组件图。

**9. 数据视图(可选)**

系统的持久化数据存储视图的描述。如果只有很少或没有持久数据，或者设计模型和数据模型之间的转换很简单，则此部分是可选的。

**10. 尺寸和性能**

对影响体系结构的软件的主要尺寸特征的描述，以及目标性能约束。

**11. 质量**

对软件体系结构如何对系统的所有功能(除功能外)作出贡献的描述:可扩展性、可靠性、可移植性，等等。如果这些特征具有特殊意义，例如安全、安全或隐私影响，则应明确描述。

**10.3踪家君**

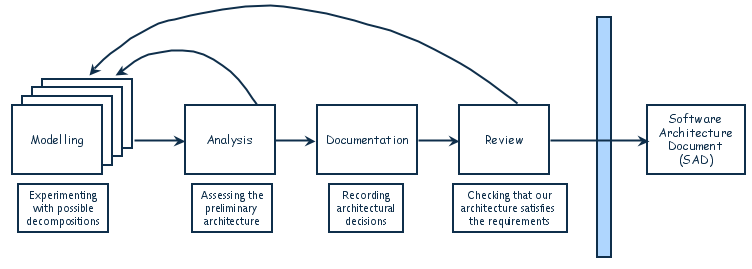
**1.概述**

设计是一种创造性的过程，它考虑如何实现所有客户的需求，设计所产生的计划也称为设计，早期的设计专注于系统的体系结构，后续的设计专注于如何实现单个的单元，设计软件是种具有智力挑战性的任务。通过学习优秀的设计例子可以提高设计技巧。

大部分设计工作是例程设计 (routine design), 通过对相似问题的解决方案进行复用和调整来解决当前问题.

2.设计原则:

软件系统设计是一个迭代的过程,最终结果是软件体系结构文档 (SAD).



流行的设计方法:面向功能的分解、面向特征的分解、面向数据的分解、面向进程的分解、面向事件的分解、面向对象的设计。

## 2.1面向功能的分解

将功能或要求划分为模块

从要求规范中列出的功能开始

较低级别的设计将这些功能划分为子功能，然后分配给较小的模块

描述哪些模块（子功能）相互调用

## 2.2面向特征的分解

将功能分配给模块

高级设计从服务和功能集合的角度描述了该系统

低级别设计描述每个功能如何增强服务并识别功能之间的交互

## 2.3面向进程的分解

将系统划分为并发过程

高级设计：识别系统的主要任务、将任务分配给运行时间过程、解释任务如何相互协调

低级设计：更详细地描述了流程

## 2.4面向事件的分解

专注于系统必须处理的事件，并将事件责任分配给不同的模块

高级设计编目系统的预期输入事件

低级别设计将系统分解为状态，并描述事件如何触发状态转换

## 2.5面向对象的设计

将对象分配给模块

高级设计可识别系统的对象类型，并解释对象彼此的关系

低层设计详细说明了对象的属性和操作

**3.常见体系结构视图**

## 3.1分解视图

分解视图将系统描述为可编程单元

此视图可能是分层的

可能由多个模型表示

## 3.2依赖视图

依赖视图显示软件单元之间的依赖关系

此视图在项目规划中很有用

也可用于评估对某些软件单元进行设计更改的影响

## 3.3泛化视图

泛化视图显示的软件单元是相互概括或专业化的

此视图在设计抽象或可扩展的软件单元时很有用

## 3.4执行视图

执行视图是软件架构师绘制的传统箱箭图，从组件和连接器上显示系统的运行时间结构

每组件都是一个独特的执行实体，可能带有自己的程序堆栈

连接器是某些内部通信机制，例如通信通道、共享数据存储库或远程程序呼叫

## 3.5实现视图

实现查看地图代码单位到包含其实施的源文件

帮助程序员在源代码文件迷宫中找到软件单元的实现

## 3.6部署视图

部署将运行时间实体（如组件和连接器）映射到计算机资源（如处理器、数据存储和通信网络）上

它帮助架构师分析设计的质量属性，如性能、可靠性和安全性

## 3.7工作分配视图

工作分配视图将系统设计分解为可分配给项目团队的工作任务

帮助项目经理规划和分配项目资源，并跟踪每个团队的进度

**10.4刘倩**

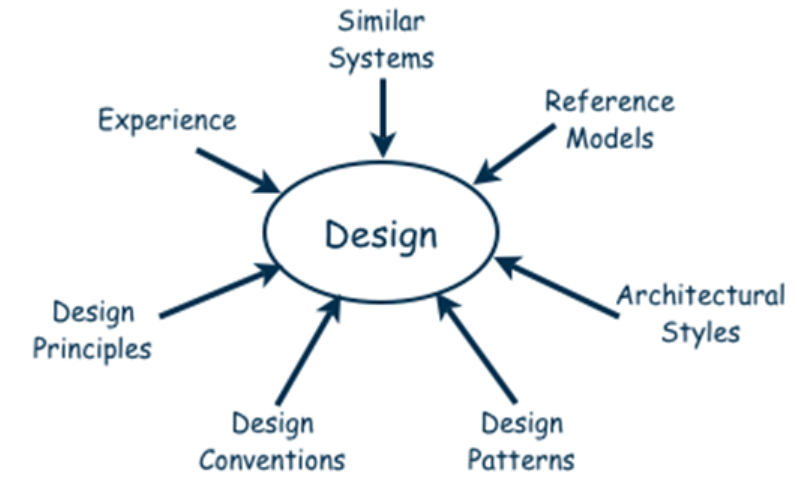
**1.设计过程**

设计是一种创造性的过程，它考虑如何实现所有客户的需求； 设计所产生的计划也称为设计。

早期的设计专注于系统的体系结构，后续的设计专注于如何实现单个的单元。

设计软件是一种具有智力挑战性的任务。

通过学习优秀的设计例子可以提高设计技巧。大部分设计工作是例程设计 (routine design), 通过对相似问题的解决方案进行复用和调整来解决当前问题。多种利用现有解决方案的办法（克隆、参考模型）。



如果我们需要解决的问题没有参考模型，软件体系结构也有一般性的解决方案，称为体系结构风格。专注于一种体系结构风格会产生问题，好的体系结构设计应选择、改进以及集成多种体系结构风格，以产生符合期望的产品。

多种理解决策以及评估选择的体系结构的工具（设计模式、设计公约、创新设计、设计原则）。

软件系统设计是一个迭代的过程，最终结果是软件体系结构文档 (SAD)。

**2.分解和视图**

创建逐层细化的层次式信息。

一些设计问题没有现成的解决方案，设计人员必须利用分解隔离出系统的关键问题。

流行的设计方法a.面向功能的分解b.面向特征的分解c.面向数据的分解d.面向进程的分解e.面向事件的分解f.面向对象的设计

当系统的每个活动都仅由对应的软件单元实现，并且每个软件单元的输入和输出都已经明确地被定义时，设计才可以说是模块化的。如果一个软件单元的接口能够准确无误地指定该单元的外部可见行为，则称该软件单元是定义明确的。

常见体系结构视图:a.分解视图b.依赖视图c.泛化视图d.执行视图e.实现视图f.部署视图g.工作分配视图

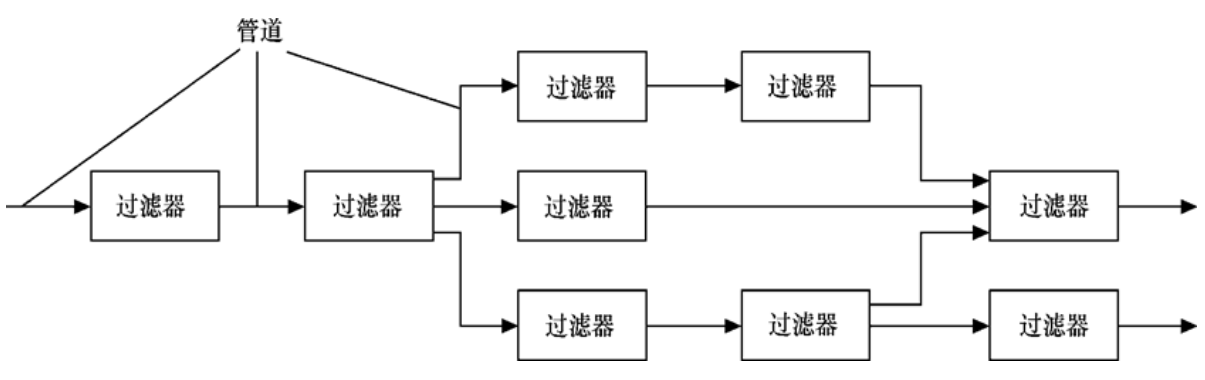
**3.体系结构风格**

体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。对体系结构风格的研究和实践为大粒度的软件复用提供了可能。

经典的体系结构风格：

a.数据流风格: 批处理序列; 管道/过滤器。

当输入数据经过一系列的计算和操作构件的变换形成输出数据时，可以应用这种体系结构。管道/过滤器、批处理序列都属于数据流风格。管道/过滤器结构如下图所示。



管道-过滤器风格：把系统任务分成为几个独立的功能模块。这些功能模块采用通过系统的数据流连接。在管道-过滤器风格下，每个功能模块都有一组输入和输出。功能模块称作过滤器（filters）；功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作管道（pipes）。管道-过滤器风格的特性之一在于过滤器的相对独立性，即过滤器独立完成自身功能，相互之间无需进行状态交互

b.调用/返回风格：

主程序/子程序；面向对象风格；层次结构。

层次结构风格：整个系统被组织成一个分层结构，每一层为上层提供服务，并作为下一层的客户

c.独立构件风格：进程通讯；事件系统。

d.虚拟机风格：解释器；基于规则的系统。

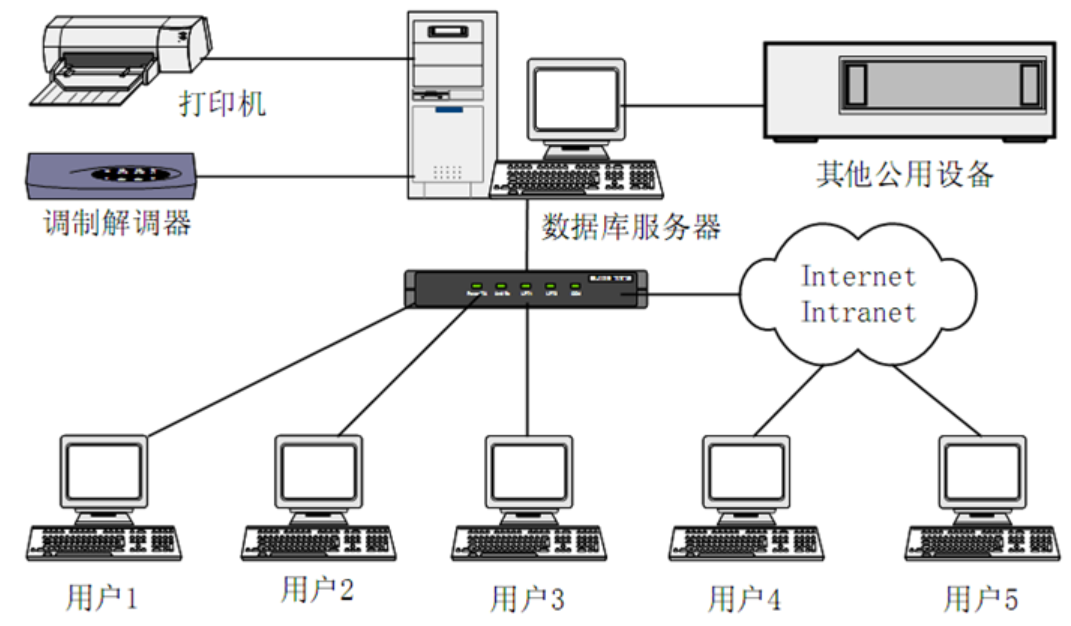
e.仓库风格：数据库系统；超文本系统；黑板系统。

f.过程控制环路

g.C/S风格

客户机/服务器（C/S）体系结构是基于资源不对等，且为实现共享而提出来的。

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。



h.B/S风格

B/S体系结构主要利用不断成熟的WWW浏览器技术，特别是浏览器嵌入的多种脚本语言B/S体系结构有三个主要组成部分：浏览器、 Web服务器、数据库服务器。

**10.5边巴曲珍**

软件系统设计是一个迭代的过程,最终结果是软件体系结构文档 (SAD)

设计是一种创造性过程 (continued)多种利用现有解决方案的方法

克隆: 借鉴现有的整个设计，只做少许的调整

参考模型: 用于特定领域的分解系统的标准的、一般性的体系结构

软件体系结构也有一般性的解决方案, 称为体系结构风格

专注于一种体系结构风格会产生问题

好的体系结构设计应选择、改进以及集成多种体系结构风格，以产生符合期望的产品

体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

对体系结构风格的研究和实践为大粒度的软件复用提供了可能。

经典的体系结构风格:

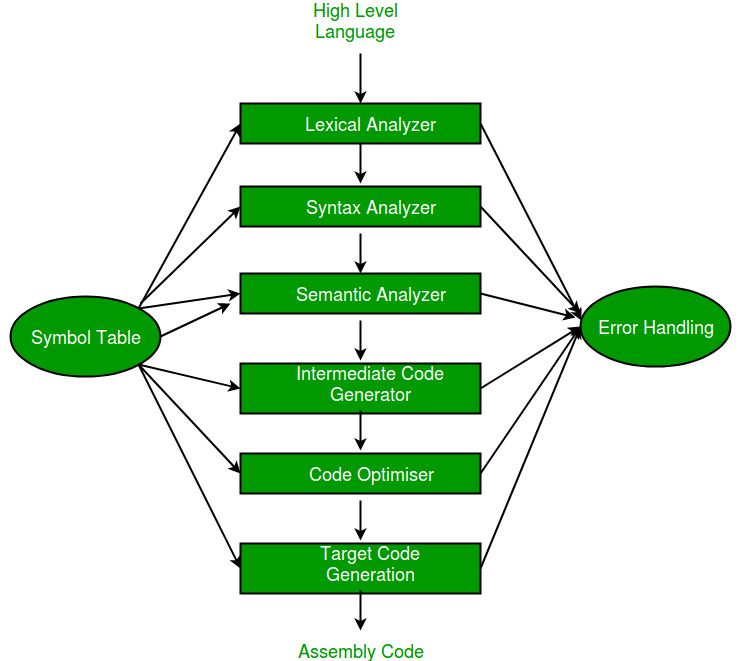
1．数据流风格: 批处理序列; 管道/过滤器。

把系统任务分成为几个独立的功能模块。这些功能模块采用通过系统的数据流连接。

在管道-过滤器风格下，每个功能模块都有一组输入和输出。功能模块称作过滤器（filters）；功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作管道（pipes）。

管道-过滤器风格的特性之一在于过滤器的相对独立性，即过滤器独立完成自身功能，相互之间无需进行状态交互。管道-过滤器风格优点设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成。 基于管道-过滤器风格的系统具有较强的可维护性和可扩展性支持一些特定的分析，如吞吐量计算和死锁检测等,管道-过滤器风格具有并发性 管道-过滤器风格不足 交互式处理能力弱 具体实现比较复杂 数据流同步问题 数据加密与解析

编译器的管道-过滤器风格;



2.调用/返回风格：主程序/子程序；面向对象风格；层次结构。

层次结构风格

整个系统被组织成一个分层结构，每一层为上层提供服务，并作为下一层的客户。

层次结构的基本结构如下图所示

层次结构风格优点和缺点:

结构风格支持系统设计过程中的逐级抽象

基于层次结构风格的系统具有较好的可扩展性

层次结构风格支持软件复用

缺点：

并不是每个系统都可以很容易地划分为分层的模式

很难找到一个合适的、正确的层次抽象方法。

3.独立构件风格：进程通讯；事件系统。

独立构件风格的优势在于，由于构件之间的耦合度低，因此构件的可扩展性和可替换性要比调用返回风格高很多。但也是由于构件的独立特性，使得需要专门的IT系统来维护这些构件的运行。

虚拟机风格：解释器；基于规则的系统。解

4.仓库风格：数据库系统；超文本系统；黑板系统。

虚拟机风格包含

解释器

基于规则的系统

解释器

侧重于自定义，灵活的功能特性。这些功能特性通过解释器将其转换为目标平台上的可执行代码。

基于规则的系统

侧重于规则集，规则解释器，常见于人工智能领域和DSS（Decision Support Design）中。

相较于“数据流风格，调用返回风格，独立构件风格“，虚拟机风格搭建的是运行规则/语言/自定义描述的平台，而非实现具体的业务逻辑。

5.过程控制环路

是将过程输出的指定属性维护在一个特定的设定值，将事务处理看成输入、加工、输出、反馈、在输入的一个持续的过程模型。

6.C/S风格

客户机/服务器（C/S）体系结构是基于资源不对等，且为实现共享而提出来的。

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。

C/S风格—优点和缺点

优点：

界面和操作可以很丰富

安全性高

响应速度快

缺点：

适用面窄

用户群固定

维护成本高

7.B/S风格

B/S体系结构主要利用不断成熟的WWW浏览器技术，特别是浏览器嵌入的多种脚本语言

B/S体系结构有三个主要组成部分：浏览器、 Web服务器、数据库服务器。

B/S风格—优点和缺点

优点：

维护和升级方式简单

交互性较强

缺点：

在速度和安全性上需要花费巨大的设计成本

通常需要刷新页面（Ajax等可以一定程度缓解该问题）

通信开销大