# 15 软件工程建模

一. 数学建模.

1. 明确问题的数学框架.

2. 提炼其中的数学问题.

3. 建立数学模型.

4. 制订解决方案.

5. 检查.

6. 实施.

二. 计算机建模.

- 数学为计算机提供了理论基础
  - 现实世界的问题先转换为一个数学问题
  - 然后再用计算机解决这个数学问题
- 计算机有特有的软件和硬件实现
  - 软件框架（编程范式：命令式、函数式）
  - 硬件框架（硬件结构：冯诺依曼结构、哈佛结构）

软件框架.

- 编程范式
  - 命令式
  - 函数式
  - 逻辑式
- 层次性
  - 机器指令
  - 汇编指令
  - 高级语言

不同的软/硬件框架，解决问题是不一样的.

计算机的优势：

1) 可以节省人力.

2) 计算的快速.

3) 存储的海量.

## SSE技术：

- 1999年，Intel在其Pentium III微处理器中集成了SSE（Streaming SIMD Extensions）技术，有效增强了CPU浮点运算的能力。
- SSE兼容MMX指令，可以通过SIMD和单时钟周期并行处理多个浮点数据来有效提高浮点运算速度，对图像处理、浮点运算、3D运算、视频处理、音频处理等诸多多媒体应用起到全面强化作用。
- 具有SSE指令集支持的处理器有8个128位的寄存器，每一个寄存器可以存放4个单精度（32位）浮点数。SSE同时提供了一个指令集，其中的指令允许把浮点数加载到这些128位寄存器中，这些数就可以在这些寄存器中进行算术逻辑运算，然后把结果送回主存。也就是说，SSE中的所有计算都可以针对4个浮点数一次性完成，这种批处理带来了效率的提升。

## 三. 软件工程建模.

### 1. 软件工程框架

#### (1) 技术：

① 业务模型

② 分析模型

③ 设计模型

#### (2) 过程：软件开发过程.

计算机模型的进步：

#### (1) 高级语言编译器

#### (2) 高级的模型

### 2. 需求分析

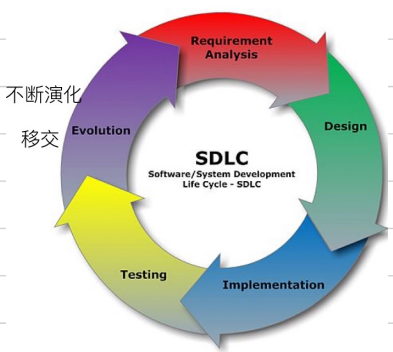
#### 分析模型 - 业务描述

- 系统管理员
  - 身份：所有系统管理员采用相同的身份和权限。
  - 操作：
    - 管理系统管理员：可以查询、添加、修改和删除系统管理员信息。
    - 管理借阅人：可以查询、添加、修改和删除借阅人信息。
    - 管理图书：可以查询、添加、修改和删除图书信息。
- 借阅人
  - 身份：分为本科生、研究生和教师三种身份，各种身份具有不同的权限。
  - 操作：
    - 查询图书：根据图书基本信息对图书进行查询。
    - 借阅图书：对选定的图书进行借阅。其中，本科生只可借阅普通图书，最多可同时借阅5本；研究生可以借阅普通图书和珍本图书，最多可同时借阅10本；教师可以借阅普通图书和珍本图书，最多可同时借阅20本。
    - 请求图书：当教师希望借阅的某种图书被借空时，可以请求图书，系统将自动通知借阅该书时间最长的本科生或研究生在7天内归还图书。
    - 查看已借图书：查看本人当前借阅图书的情况。
    - 续借图书：图书每次借阅时间为30天，本科生和研究生可以续借1次，教师可以续借2次。超期的图书和被教师请求的图书不得续借。
    - 归还图书：归还本人借阅的图书。
    - 查看消息：查看图书到期提醒、提前还书通知（本科生或研究生所借图书被请求归还时）、请求图书到馆通知（教师所请求的图书归还时）等。

3. 建立计算机模型.
4. 制订解决方案.
   构造模型：高级语言语法.
   构造结果：程序本身.
5. 软件测试
(1) 检查解决方案的有效性.
(2) 检查是否解决了问题.
6. 移交和演化
(1) 在真实环境中运行.
(2) 演化新的版本.
需求 → 设计 → 实现 → 测试 → 部署.

四. 软件开发生命周期模型.

SDLC模型（Software Development Life-Cycle）




| Activity | Target | Artifact |
|---|---|---|
| Software Requirement | What | SRS |
| Software Design | How | SDD |
| Software Construction | Build | Code and executable file |
| Software Testing | Are you building the "right" thing? Are you building it "right"? | Test Report |
| Software Deliver | Install | User Document and System Document |
| Software Maintenance | Revolution | New version software |

# 五. 软件工程建模分析案例

## 1. 如何给软件需求建模

### (1)需求

- IEEE对需求的定义为[IEEE610.12-1990]:
  - (1)用户为了解决问题或达到某些目标所需要的条件或能力;
  - (2)系统或系统部件为了满足合同、标准、规范或其它正式文档所规定的要求而需要具备的条件或能力;
  - (3)对(1)或(2)中的一个条件或一种能力的一种文档化表述。

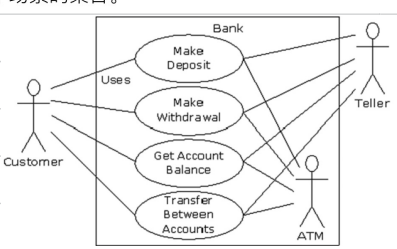
需求是一种期望.

### (2)用例 用例是组织和表达需求的方法之一.

- 用例最初由[Jacobson1992] 在 Objectory 方法中提出的,它将用例定义为"在系统(或者子系统或者类)和外部对象的交互当中所执行的行为序列的描述,包括各种不同的序列和错误的序列,它们能够联合提供一种有价值的服务"[Rumbaugh2004]。
- [Cockburn2001]认为用例描述了在不同条件下系统对 某一用户的请求的响应。根据用户的请求和请求时的系统条件,系统将执行不同的行为序列, 每一个行为序列被称为一个场景。一个用例是多个场景的集合。

用例图

四个元素: ① 参与者
② 用例
③ 参与者与用例的联系
④ 系统边界




寻找参与者:

- 谁对系统有着明确的目标和要求并且主动发出动作?
- 系统是为谁服务的?

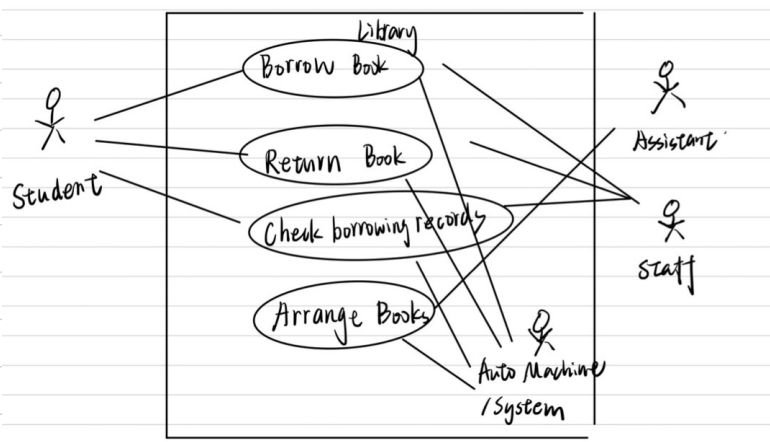
用例的特征:

- 用例是相对独立的
  - 取钱? 填写取款单?
- 用例的执行结果对参与者来说是可观测的和有意义的
  - 登陆系统? 后台进程监控?
- 这件事必须有一个参与者发起。
  - ATM吐钞票?
- 用例必须是以动宾短语形式出现的
  - 统计? 报表?
- 一个用例就是一个需求单元、分析单元、设计单元、开发单元、测试单元, 甚至部署单元

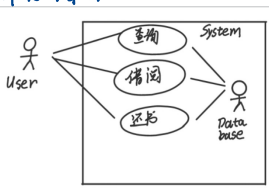
## 用例的文本描述：

| | |
|---|---|
| ID: | 用例的标识，通常会结合用例的层次结构使用 X.Y.Z 的方式 |
| 名称: | 对用例内容的精确描述，体现了用例所描述的任务，通常是“动词+名词” |
| 用例属性 | 包括创建者、创建日期、更新历史等 |
| 参与者: | 描述系统的主参与者、辅助参与者和每个参与者的目标 |
| 描述: | 简要描述用例产生的原因，大概过程和输出结果 |
| 优先级: | 用例所描述的需求的优先级 |
| 触发条件: | 标识启动用例的事件，可能是系统外部的事件，也可能是系统内部的事件，还可能是正常流程的第一个步骤 |
| 前置条件: | 用例能够正常启动和工作的系统状态条件 |
| 后置条件: | 用例执行完成后的系统状态条件 |
| 正常流程: | 在常见和符合预期的条件下，系统与外界的行为交互序列 |
| 分支流程: | 用例中可能发生的非常见的其他合理场景 |
| 异常流程: | 在非预期的错误条件发生时，系统对外界进行响应的交互行为序列 |
| 相关用例: | 记录和该用例存在关系的其他用例，关于用例之间的关系见 10.4.4 |
| 业务规则: | 可能会影响用例执行的业务规则 |
| 特殊需求: | 和用例相关的其他特殊需求，尤其是非功能性需求 |
| 假设: | 在建立用例时所做的假设 |
| 待确定问题: | 一些当前的用例描述还没有解决的问题 |

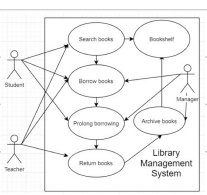
## Example：图书管理系统用例图.




## 常见错误：




Database 包含实现，不应该出现




1. 用例之间没有连接
2. 整理书是否为用例，其中是否含有用例

## 2.性能的分析 - String类的使用

- 不可改变的字符串具有一个很大的优点：编译器可以把字符串设置为共享。

- JAVA为了提高效率，所以对于String类型进行了特别的处理－－－为String类型提供了串池
  定义一个string类型的变量有两种方式：
  - String name= "tom ";
  - String name =new String( "tom ")
  - 使用第一种方式的时候，就使用了串池；使用第二种方式的时候，就是一种普通的声明对象的方式 。如果你使用了第一种方式，那么当你再声明一个内容也是 "tom "的String时，它将使用串池里原来的那个内存，而不会重新分配内存，也就是说，String name= "tom ", 将会指向同一块内存

- 另外关于String类型是不可改变的问题：
  String类型是不可改变的，也就是说，当你想改变一个String对象的时候，比如
  - name= "madding "
  - 那么虚拟机不会改变原来的对象，而是生成一个新的String对象，然后让name去指向它，如果原来的那个 "tom "没有任何对象去引用它，虚拟机的垃圾回收机制将接收它。

## Example. String类追加

### 1) + 重载

```
String tempstr = "abcdefghijklmnopqrstuvwxyz";
int times = 5000;
long lstart1 = System.currentTimeMillis();
String str = "";
for (int i = 0; i < times; i++) {
    str += tempstr;
}
long lend1 = System.currentTimeMillis();
long time = (lend1 - lstart1);
System.out.println(time);
```

每一次都需要重新复制

### 2) StringBuffer 默认构造

```
String tempstr = "abcdefghijklmnopqrstuvwxyz";
int times = 5000;
long lstart2 = System.currentTimeMillis();
StringBuffer sb = new StringBuffer();
for (int i = 0; i < times; i++) {
    sb.append(tempstr);
}
long lend2 = System.currentTimeMillis();
long time2 = (lend2 - lstart2);
System.out.println(time2);
```

默认为16字节 溢出后需要复制

### 3) StringBuffer 带参构造

```
String tempstr = "abcdefghijklmnopqrstuvwxyz";
int times = 5000;
long lstart2 = System.currentTimeMillis();
StringBuffer sb = new StringBuffer(tempstr.length()*times);
for (int i = 0; i < times; i++) {
    sb.append(tempstr);
}
long lend2 = System.currentTimeMillis();
long time2 = (lend2 - lstart2);
System.out.println(time2);
```

性能最好

# Java对象在JVM中的存储

- 一般而言，Java 对象在虚拟机的结构如下：
  - 对象头（object header）：8 个字节
  - Java 原始类型数据：如 int, float, char 等类型的数据，各类型数据占内存如 表 1. Java 各数据类型所占内存.
  - 引用（reference）：4 个字节
  - 填充符（padding）

# String在JVM中的存储

- 然而，一个 Java 对象实际还会占用些额外的空间，如：对象的 class 信息、ID、在虚拟机中的状态。在 Oracle JDK 的 Hotspot 虚拟机中，一个普通的对象需要额外 8 个字节。
- 如果对于 String（JDK 6）的成员变量声明如下：
  - private final char value[];
  - private final int offset;
  - private final int count;
  - private int hash;
- 那么因该如何计算该 String 所占的空间?
- 首先计算一个空的 char 数组所占空间，在 Java 里数组也是对象，因而数组也有对象头，故一个数组所占的空间为对象头所占的空间加上数组长度，即 8 + 4 = 12 字节，经过填充后为 16 字节。
- 那么一个空 String 所占空间为：
- 对象头（8 字节）+ char 数组（16 字节）+ 3 个 int（3 × 4 = 12 字节）+1 个 char 数组的引用 (4 字节 ) = 40 字节。
- 因此一个实际的 String 所占空间的计算公式如下：
  - 8*( ( 8+12+2*n+4+12)+7 ) / 8 = 8*(int) ( ( ( (n) *2 )+43) /8 )
- 其中，n 为字符串长度。char数组中一个char占用两个字节。默认使用unicode作为编码。

# String构造的方法选择

- 常见的创建一个 String 可以用赋值操作符"=" 或用 new 和相应的构造函数。初学者一定会想这两种有何区别，举例如下：
- String a1 = “Hello”;　a1 存放于代码区
- String a2 = new String(“Hello”);　a2 存放于堆区
  代码区.
- 第一种方法创建字符串时 JVM 会查看内部的缓存池是否已有相同的字符串存在：如果有，则不再使用构造函数构造一个新的字符串，直接返回已有的字符串实例；若不存在，则分配新的内存给新创建的字符串。
- 第二种方法直接调用构造函数来创建字符串，如果所创建的字符串在字符串缓存池中不存在则调用构造函数创建全新的字符串，如果所创建的字符串在字符串缓存池中已有则再拷贝一份到 Java 堆中。