link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=63 sentences=58, words=284

1. 软件生命周期

1.1 可行性分析报告和软件开发计划

- 产出可行性分析报告

1.2 需求分析阶段

- 由软件分析师来做，需要懂技术也需要懂业务
- 分析出软件需要完成什么功能
- 产出需求分析说明书和初步的用户手册

1.3 软件设计(概要设计和详细设计)

- 由架构师/项目经理来做
- 根据团队技术基础确定用什么技术栈(Java/Node/Php)
- 确定部署的操作系统
- 使用什么数据库(oracle/mysql/sqlserver)
- 设计数据库表
- 选择团队成员
- 产出软件设计文档

1.4 编码工作

- 开发人员来做
- 把设计编成代码
- 产出源代码以及清单

1.5 测试阶段

- 由测试工程师进行
- 分为白盒测试(单元测试)和黑盒测试(功能测试)
- 产出软件测试报告

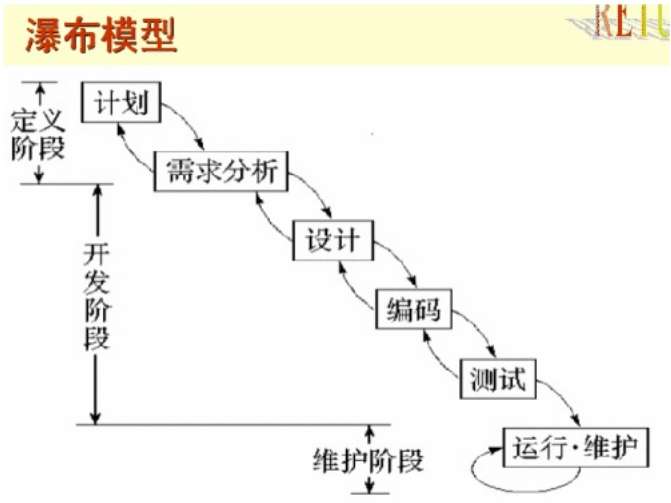
1.6 实施和维护工作

- 由实施工程师执行
- 把项目按照需要安装和配置好，让客户使用并解决简单问题
- 产出软件维护报告

2. 软件开发模型

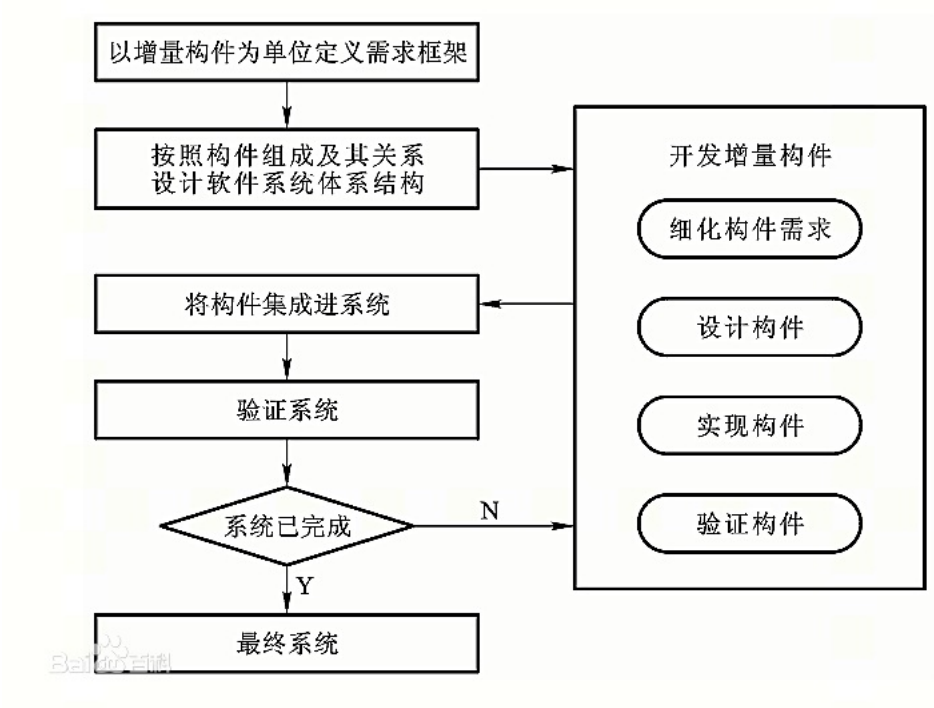
2.1 瀑布模型

- 瀑布模型（Waterfall Model） 是一个项目开发架构
- 开发过程是通过设计一系列阶段顺序展开的，从系统需求分析开始直到产品发布和维护，每个阶段都会产生循环反馈
- 如果有信息未被覆盖或者发现了问题，那么最好 "返回"上一个阶段并进行适当的修改，项目开发进程从一个阶段"流动"到下一个阶段，这也是瀑布模型名称的由来。



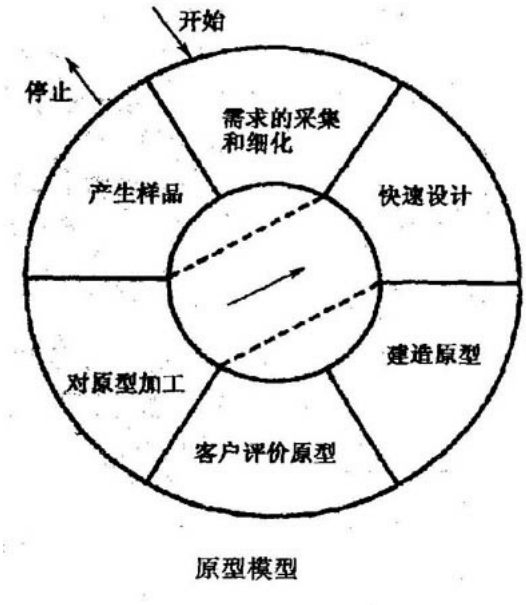
2.1 增量开发模型

- 增量模型是把待开发的软件系统模块化，将每个模块作为一个增量组件，从而分批次地分析、设计、编码和测试这些增量组件。



2.2 原型开发模型

原型模型指的是在执行实际软件的开发之前，应当建立系统的一个工作原型。



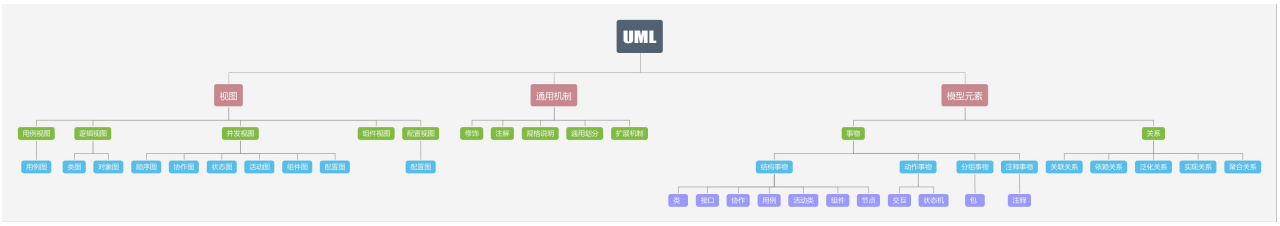
原型模型

3. 模型

- 建造一栋大楼前需要先把图纸将外观、内部结构描述清楚，这些图纸就是模型。
- Unified Modeling Language (UML)又称统一建模语言或标准建模语言**3.1 模型的三个特点#**
- 简化
- 多视角
- 通用符号

** 3.2 开发软件 #**

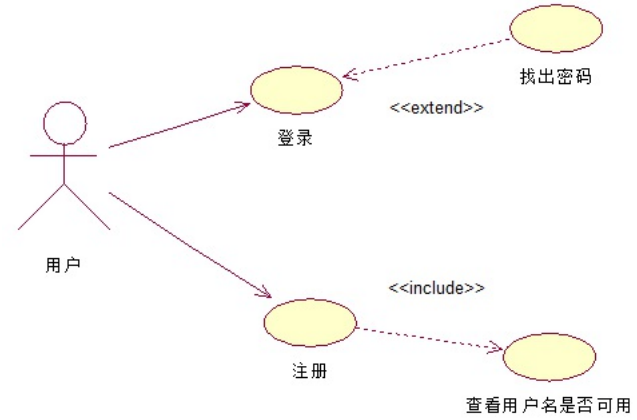
rational rose (http://www.pc0359.cn/downinfo/74454.html)
** 3.3 UML组成 #**



4. 模型图

** 4.1 用例图 #**

- 用例建模最主要的功能是表达系统的功能性需求或行为
- 参互者： 参与者指的是与系统交互的角色，可以是人，也可以是事物或其它系统
- 用例是系统为参与者提供的功能。用例名称一般是一个带有动作性的词语



用例描述

项目 内容 用例名称 登录 用例ID login 角色 用户 用例说明 描述用户的登录过程 前置条件 打开网站页面 基本事件流 1.点击登录 2. 输入用户名和秘密啊 3. 点击登录 4. 服务器会使用会话保存用户登录状态 其它事件流 1. 用户名为空提示用户名不能为空 2。密码为空提示密码不能为空 异常事件流 登录超时则返回登录页 后置条件 登录成功，进入个人中心
** 4.2 类图和对象图 #**

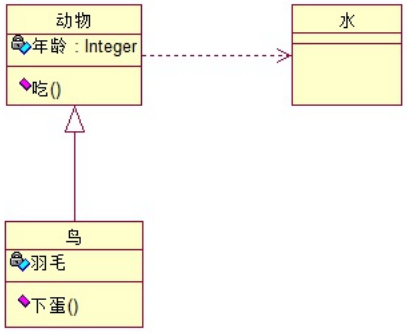
- 用于描述系统中的对象类本身的组成和对象类之间的各种静态关系
- 类之间的关系：依赖、泛化(继承)、实现、关联、聚合和组合
- 对象图描述一组对象和它们之间的关系，它是系统状态的某一时刻的快照，它的使用相当有限，它主要用于了解系统在某个特定时刻的具体状况和数据结构
- 对象图表示方法和类图大致相同，对象图中的对象属性可以有具体值，类图中的一个类可以对应成对象图中的多个对象，例如： 部门类的自关联就可以对应成多个部门之间的关联
- 一个用例图对应一个类图，一个类图包含多个类

** 4.3 类图的关系 #**** 4.3.1 依赖关系(Dependence) #**

只要在类中用到了对方，那么它们之间就存在依赖关系，如果没有对方，连编译都通过不了

** 4.3.2 泛化关系(Generalization) #**

泛化关系实际上就是继承关系，他就是依赖关系的特例

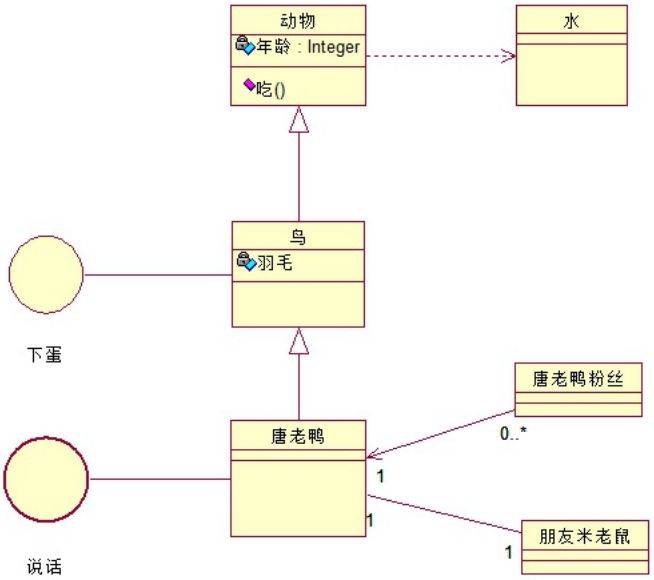


** 4.3.3 实现关系(Implementation) #**

实现关系实际上就是A类实现B类，他就是依赖关系的特例

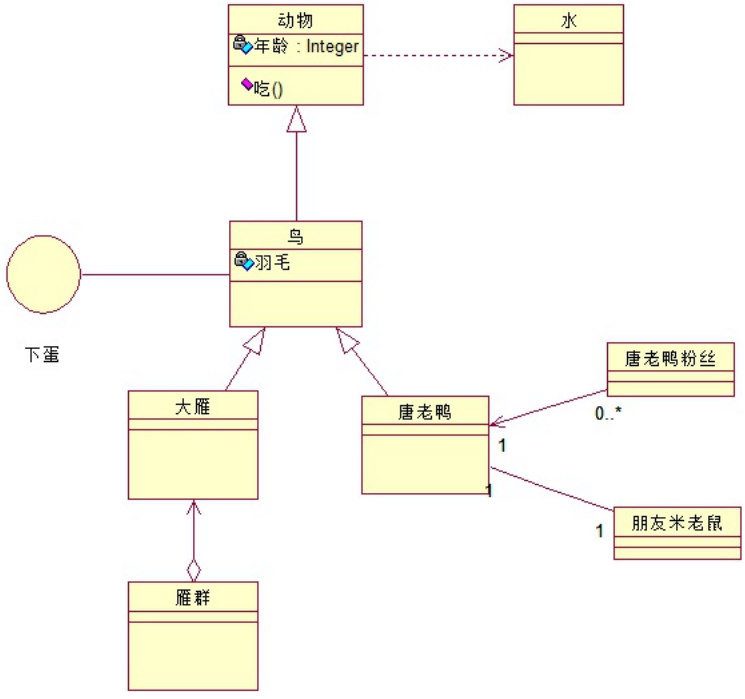
** 4.3.4 关联关系 #**

- 关联关系实际上就是类与类之间的联系，他是依赖关系的特例。
- 关联关系比依赖的关系更强
- 关联具有导航性，即双向关系或单向关系，表示关系在那一方维护
- 关联具有多重性，如
    - 1 表示有且仅有一个
    - 0... 表示0或者多个
    - 0,1 表示0个或者一个
    - n..m 表示n到m个都可以
    - m... 表示至少m个



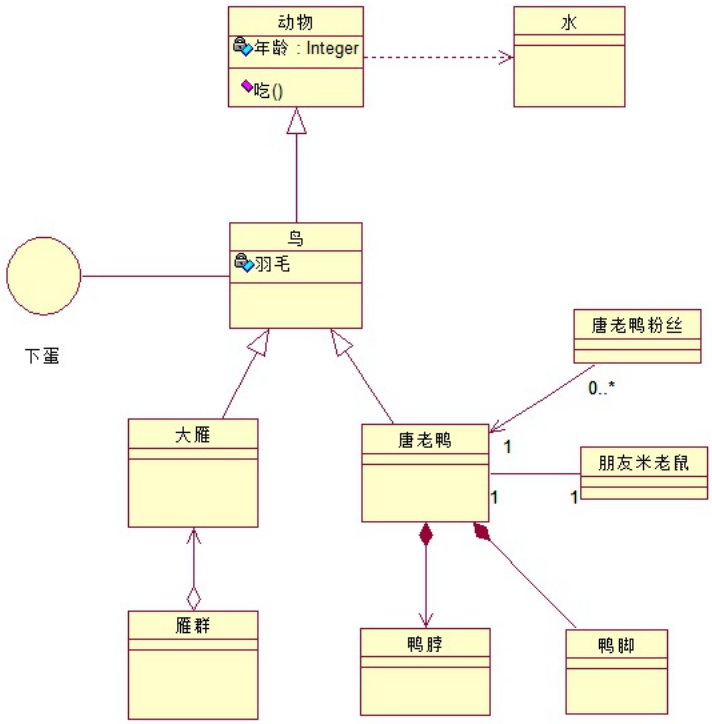
** 4.3.4 聚合关系 #**

- 聚合关系表示的是整体和部分的关系，整体与部分可以分开
- 聚合关系是关联关系的特例，所有它具有关联的导向性和多重性
- 聚合的双方生命周期是独立的



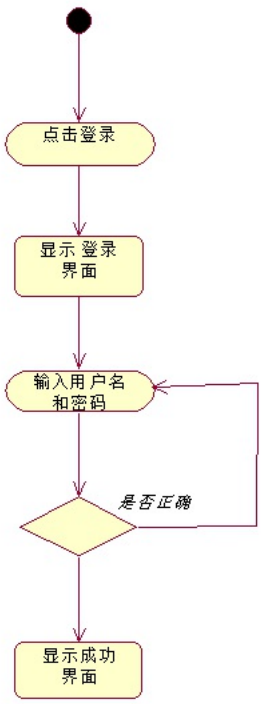
** 4.3.5 组合关系 #**

- 也是整体和部分的关系，但是整理和部分不可分开
- 整体和部分生命周期一致



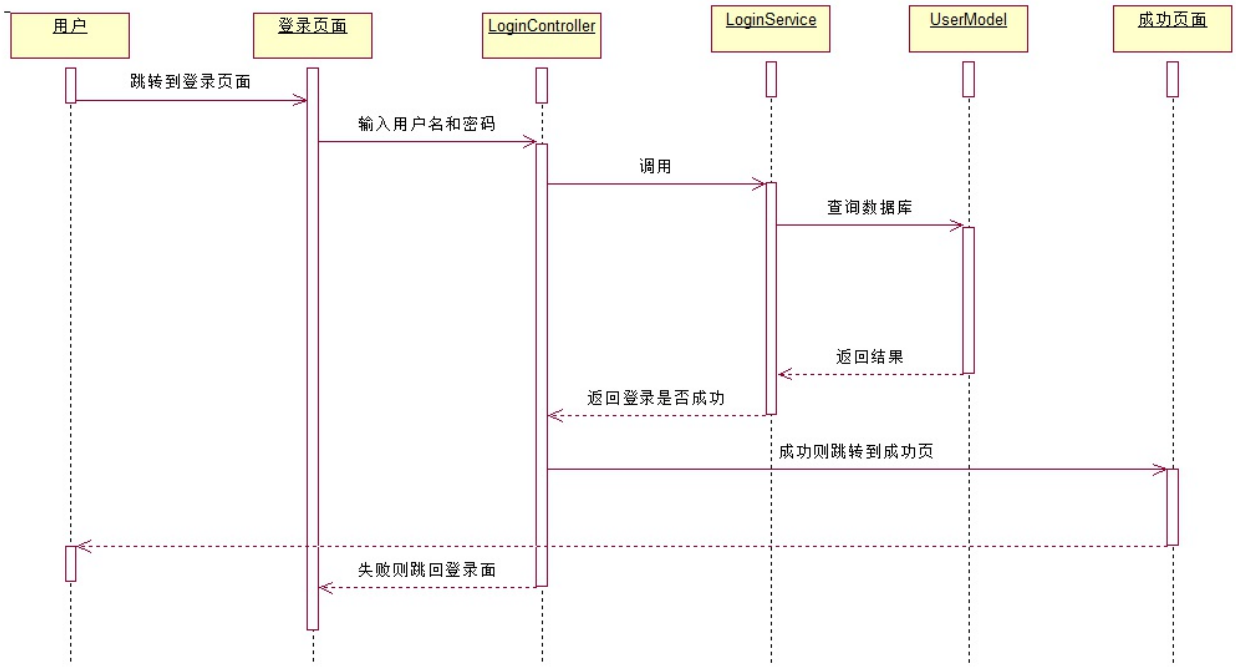
4.4 活动图

- 在UML里，活动图本质上就是流程图，
- 它描述系统的活动，判断系统的活动，判断点和分支等。



4.5 时序图

- 时序图强调消息时间顺序的交互图
- 时序图描述类系统中类与类之间的交互，它讲这些交互建模换成消息交换
- 时序图用于描述对象之间如何随着时间进行协作
- 时序图由活动者(Actor)、对象(Object)、消息(Message)、生命线(Lifeline)和控制焦点(Focus Of Control)组成
- 不同元素有不同表示
    - 对象是一个矩形，对象名称下有下划线
    - 消息用由方向的箭头表示，调用是实线，返回消息是虚线
    - 生命线由纵向的虚线表示
    - 控制焦点是纵向的举行，也就是活动条(Activiation Bar)

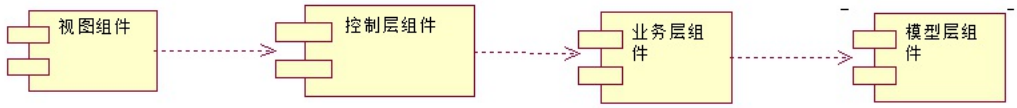


4.6 协作图

- 协作图是时序图的一种变种
- 协作图强调的是发送和接受消息的对象之间的组织结构
- 协作图显示了一系列对象和在这些对象之间的联系以及对象间发送和接受的消息
- 时序图主要侧重与对象间消息传递在时间上的先后关系，而协作图则侧重与对象间以及对象和角色交互的静态关系

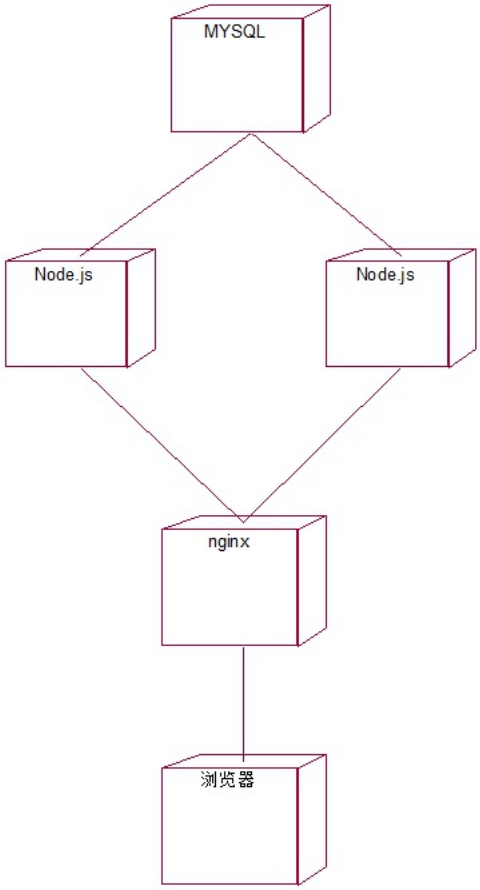
4.7 组件图

- 组件图用例建立系统的各个组件之间的关系，它们是通过软件或者文件组织在一起，使用组件图可以帮助读者了解某个功能位于软件包中的哪一个位置，以及各个版本的软件包含那些功能。
- 组件图可以用来帮助设计系统的整体架构



4.8 部署图

- 部署图是来帮助读者了解软件中的各个组件运行硬件什么位置，以及这些硬件之间的交互关系
- 节点： 用来表示一种硬件，它可以是服务器，计算机等。节点的符号是一个三位盒子，在左上角包含节点的名称
- 通信关联： 节点通过通信关联建立彼此的关系，采用从节点到节点绘制实线来表示关联


MYSQL
Node.js
Node.js
nginx
浏览器