**目 录**

[1. 背景 4](#_Toc221417250)

[2. 需求 5](#_Toc221417251)

[3. 业务特点及开发约束 5](#_Toc221417252)

[3.1. 业务特点 5](#_Toc221417253)

[3.2. 开发约束 5](#_Toc221417254)

[4. 术语与参考资料 6](#_Toc221417255)

[4.1. 术语 6](#_Toc221417256)

[4.2. 参考资料 6](#_Toc221417257)

[5. 开发规范 7](#_Toc221417258)

[5.1. 编码规范 7](#_Toc221417259)

[5.2. 开发环境与开发工具 7](#_Toc221417260)

[5.3. 开发目录及类名、包结构命名规范 8](#_Toc221417261)

[5.3.1. 开发目录规范 8](#_Toc221417262)

[5.3.2. 包结构命名规范 9](#_Toc221417263)

[6. 双向业务架构总体设计 12](#_Toc221417264)

[6.1. 业务实体设计 12](#_Toc221417265)

[6.1.1. SSO 12](#_Toc221417266)

[6.1.2. PMC 13](#_Toc221417267)

[6.1.3. 接口服务 14](#_Toc221417268)

[6.1.4. PFS 14](#_Toc221417269)

[6.1.5. Presence 15](#_Toc221417270)

[6.2. 系统性能设计 15](#_Toc221417271)

[6.2.1. 系统扩展性 15](#_Toc221417272)

[6.2.2. 系统移植性 15](#_Toc221417273)

[6.2.3. 系统复用性 15](#_Toc221417274)

[6.2.4. 系统维护性 16](#_Toc221417275)

[6.2.5. 系统安全性 16](#_Toc221417276)

[7. 技术框架 16](#_Toc221417277)

[7.1. 技术框架图 16](#_Toc221417278)

[7.2. 技术选型 17](#_Toc221417279)

[7.2.1. J2EE技术路线 17](#_Toc221417280)

[7.2.2. Dot net技术路线 21](#_Toc221417281)

[7.2.3. 其它因素 22](#_Toc221417282)

[7.3. 使用人群及使用约束 23](#_Toc221417283)

[7.4. 框架设计思想 23](#_Toc221417284)

# 背景

在当代的经济状况下，人们不仅仅满足服饰的保温效果，对服饰的要求由保温转变成了装饰和张扬个性的平台。尤其是青少年群体，他们对于服饰的要求与之前大有不同。虽然现在服饰的款式众多，但是对现在青少年的吸引程度已经大不如前。现在的服饰单调，统一，创意很少，现在的服装需求根本满足不了现在青少年的心理需求。现在人们对于服饰追求个性，多样化。而不再是被动的接受外来的服饰创意和服饰文化。通过服饰也可以看出文化的发展潮流。

所以，应该大力发展多元化的服饰文化，不应该让服饰市场继续死板。突破传统的束缚，大力发展服饰文化来源于民众而又走入民众的战略。

通过实施本项目可以为客户带来如下收益：

* 适应未来多元化的服饰发展潮流
* 简化服饰开发商创意工作，从而降低开发商成本
* 指导服装售卖店的销量和盈利

# 需求

见《服装销售系统需求规格说明书》。

# 业务特点及开发约束

## 业务特点

1. 基于服装市场来展现业务；
2. 由于用户主要是消费者，业务主要是查询和修改。
3. 分布式部署和交互；
4. 行业内部分领域缺少或没有统一的协议和规范；

## 开发约束

无

# 参考资料

## 参考资料

|  |  |  |  |
| --- | --- | --- | --- |
| **文档** | **版本** | **作者/来源** | **备注** |
| 《服装销售系统需求规格说明书》 | V1.0 | 张宏权、李朋辉 |  |
| 《服装销售系统概要设计说明书》 | V1.0 | 方军 |  |

# 开发规范

## 编码规范

见《awm服装销售系统编程规范\_JAVA篇》。

## 开发环境与开发工具

1. 操作系统
   1. Windows10
2. 应用服务器
   1. Tomcat (8.5)
3. Java软件开发包
   1. JDK (1.8.0\_121)
4. 开发工具版本
   1. Eclipse 4.7.2
5. 数据库
   1. MySql
6. 版本控制工具
   1. TortoiseGit (2.6.0)

## 开发目录及类名、包结构命名规范

### 开发目录规范

1. src：存放源代码的目录，具体src中的包结构规范请参照5.3.2节的“类名、包结构命名规范”。
2. test：存放测试代码的目录，具体test中的包结构规范请参照5.3.2节的“类名、包结构命名规范”。
3. resources：存放资源文件的目录，具体为：
4. 存放各种配置文件。
5. lib：存放工程所引用的jar文件。
6. webapp\WEB-INF：存放与web工程相关的文件，具体为：
7. config: 存放各个层面的配置文件。
8. pages: 存放各种jsp文件。
9. resources/js：存放各个业务的js文件。
10. images：存放应用所需要的资源图像。

### 包结构命名规范

#### 包结构及包的命名规范

1. 公共资源包结构：edu.zut.cs.sowtfare.awm.base
2. 项目内各类型包的结构：

应用层：edu.zut.cs.sowtfare.awm.admin.web.spring.controller

业务逻辑：edu.zut.cs.sowtfare.awm.services

数据模型：edu.zut.cs.sowtfare.awm.model

数据操作：edu.zut.cs.sowtfare.awm.dao

#### 相关约束：

1. 日志：

日志模块：edu.zut.cs.sowtfare.awm.base.service.log

记录程序的运行状况，方便调试程序。

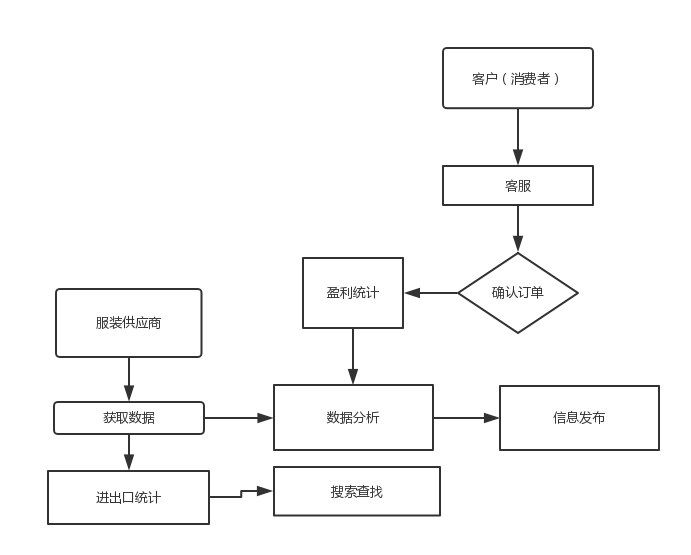
1. 其它：

开发人员可根据实际情况按照以上的规范与约束进行扩展。

#### 类的命名规范：

1. 实体定义：按照实际的意义为实体类命名，并放在模型包（domain）。如：User、Issue、Search等。
2. 实体操作：按照实体名+”Manager”方式命名，并放在数据操作包中（dao）。如：UserManager、IssueManager等。
3. 业务逻辑层的类放在service包中，接口采用以业务为中心的有意义的名字。如：CustomerServicer等。而实现类也在同一个包中，以业务接口名+”Impl”方式命名。如：CustomerServicerImpl等。

# 业务架构总体设计



业务总体架构图

## 业务实体设计

### SSO

#### 实体定义：

单点登录（Single Sign On），简称为 SSO，是目前企业级用户身份认证整合的解决方案之一。SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。

#### 设计思想：

SSO应用在这个方面：

后台管理员认证。由于一些业务的管理不是在CSS，而是通过管理员的方式进入后台查看数据，这样在每次进入各业务后台管理页面时，也要认证用户身份。通过SSO的控制，可以实现单点登录。

### PMC

#### 实体定义：

后台管理部分是为了提升运营商对于CSS的运维管理能力，提供统一的监控、管理。

#### 设计思想：

PMC主要是提供统一的管理平台。但awm服装销售系统研究小组解决方案的思想，Link方式所管理的实体本身可以脱离PMC单独管理与部署，这种实体自身提供管理平台。所以，对于这种方式的管理需要SSO认证。

实体组成：

1. 资源管理：主要是对资源的监控，并可进行添加、删除资源和资源的启动与停用功能。

资源：这里的资源不是广义的，而是指硬件资源。

1. 配置管理：对PMC所管理的资源、业务等的配置。
2. 日志管理：按条件查看系统日志和应用日志。
3. 统计分析：按时间地域来统计在线人数和栏目点击量。
4. License管理：规定可以注册的最大用户数量。
5. 帮助：对各种操作提供帮助性说明。

### 接口服务

#### 实体定义：

对生产厂商需要定义统一接口，方便生产厂商提供的数据导入到数据库。也方便CSS的操作和管理。

#### 设计思想：

接口服务有自己独立的后台管理，脱离PMC仍然可以支持其它的应用。接口服务同PMC的配置管理同样可以对外挂系统信息进行配置，写入到数据库的同一个地方。在接口服务启动的时候先去读取这些信息，然后开放相应的接口服务。

其中包括两类接口：一个是获取内容的接口。例如：元数据信息，产品信息等。另一个是业务要素信息。

### PFS

#### 实体定义：

Portal作为终端用户的门户系统，为终端用户提供展现各种业务的统一平台。

#### 设计思想：

Portal就是展示各种业务，为终端用户提供各种操作平台的门户。Portal的功能设计一定要简化，能拆分出来的尽量不要放在portal中。

### Presence

#### 实体定义：

Presence是数字电视双向业务的基础应用，负责收集从终端STB处定时上报的用户行为数据包，通过“代理——服务器”方式维护管理全部在线终端（STB）的状态。

#### 设计思想：

由于广电用户群数量庞大，如果系统要获得实时性高的数据，与用户的通讯频率就会高，因此需要的网络带宽和处理资源将非常多。但是因为用户使用模式决定（电视用户大都不会很频繁的切换节目），其中大部分数据都是冗余数据。根据网络负责能力和处理能力，用一批前端代理服务器来分别与一部分用户通讯，分析、筛选有效数据。将筛出的数据发送到一台中心服务器汇总，由中心服务器通过接口提交给外部系统使用数据。

由于网络状况和机器负载能力的不同，需要能可配置机顶盒发送数据包的参数，比如发送频率、超时确认时长等。

## 系统性能设计

### 系统精确度

|  |  |
| --- | --- |
| **性能指标** | **精确度说明** |
| 时间特性 | 响应时间：用户操作，最多5秒内得到响应。若页面很复杂，或逻辑很复杂，至少能先有部分结果，或者友好的提示。 |
| 性能特性 | 在页面响应时间3秒内，支持不低于100并发用户 |
| 规模特性 | 能够支持不低于1千用户的使用 |

### 系统扩展性

#### 在数据持久层方面

采用开源的Hibernate 持久层组件，该组件是有名的OR-Mapping工具，使用该组件可以对性能做灵活的配置和性能优化，可以对一级缓存进行管理，可以开启二级缓存、控制事务策略，有灵活的1对1 关系、1对多关系的映射。

#### 在数据库设计方面

对于海量数据的存储采用分区的策略，可以是范围分区、列表分区、散列分区、组合范围散列分区等。

为经常查询的列建立索引以此来加快访问数据库表中特定信息的速度，避免或简化排序操作，避免相关子查询，使用存储过程加快数据库逻辑的牏速度。

针对特定的数据库做特殊优化。

#### 在代码方面

遵循J2EE相关规范进行开发，使用Spring 等AOP框架，灵活的集成或扩展应用。

使用缓存组件，对经常被访问的数据进行缓存，加快查询及展示的速度。

### 系统移植性

本项目使用Java语言，采用Spring + Hibernate + SpringMvc的开源框架，遵循J2EE框架开发。

由于Java的跨平台特性使得本项目所开发的系统可以运行在Window系列、Linux系列操作系统之上。

开源的Hibernate 持久层组件来支持跨数据库的操作，Hibernate 组件特有的 HQL语言在执行时被解释成特定的数据库SQL语言，保证了跨数据库的特性。

遵循J2EE规范进行开发，使得开发后的B/S架构Web 系统可以运行在任何遵循Sun公司（关于Web 应用服务器的）J2EE规范所开发实现的应用服务器之上，这些服务器可以是开源的也可以是商用的应用服务器，如：Apache Tomcat。

### 系统复用性

从设计的角度，纵向方面，系统采用组件化的方式的进行设计，合理的将公告、电视回看EPG、VOD EPG等业务组件化，保证开发后的系统可以“有选择性”的复用到未来相类似的Portal系统中。

横向方面，系统采用Persistence + Business (两层business，使用business delegate) + Presentation 多层设计，针对业务需求科学的决策，开发可复用的核心技术框架，这样对于新的应用可直接复用并基于核心技术框架进行开发，即使要对核心技术框架修改也不会对应用产生大的影响。Business层基于Persistence之上根据业务进行开发不需要关心采用何种展现层技术进行展现，使得设计出的系统灵活、可复用。展现层采用MVC模式的Struts组件，使得视图与业务很好的分离开，视图中的action提交给定义好的 action name，无需关心该action的具体实现，使得“功能代码”与“视图（界面）”很好的分离。

### 系统维护性

系统采用组件化的方式的进行设计，将业务组件（套件）化，采用Spring作为集成框架，针对具体需求可以灵活的装配业务组件，达到灵活、可配置。

系统提供PMC(Portal Center Manager)对系统的后台进行参数配置、管理，PCM提供配置管理，应用管理，管理员管理，操作日志，统计分析，减少后台管理员的维护工作量，让系统的配置、管理变得容易简单、可视化。

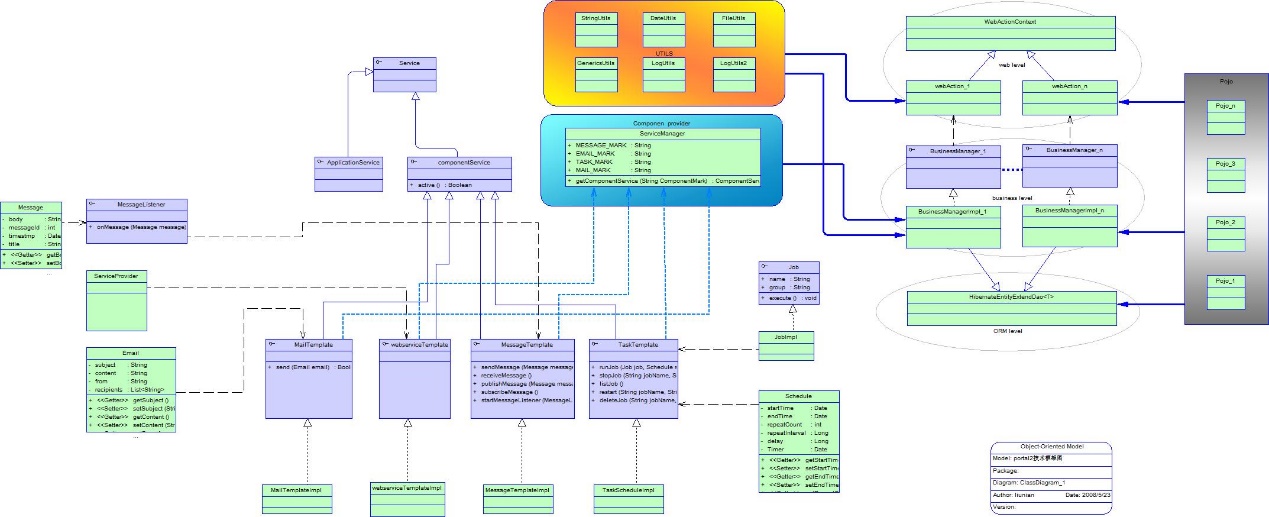
### 系统安全性

系统支持HTTPS协议，使用随机生成的token作为摘要，然后MD5算法对登录的STB机顶盒用户的密码和摘要进行加密，保证身份信息的传递安全。

采用SSO单点登录对所有的登录使用进行身份验证，身份验证的机制可以是数据库验证，也可以是LDAP验证。用户身份验证的全过程使用凭证进行操作，包括服务凭证（ST）、代理授权凭证（PGT）、代理凭证（PT），构成完整的凭证链，保证了系统的安全。

# 技术框架

## 技术框架图



## 技术选型

### J2EE技术路线

#### 开发语言特性

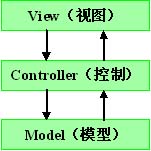
1. 技术来自于AWM服装系统研发小组；
2. 开源产品众多，免费框架居多；
3. 成果众多，相应的最佳实践设计模式层出不穷；
4. 平台移植性好，支持所有操作系统，这一方面成本降低；
5. 开源社区活跃，可以快速解决开发过程中遇到的问题。

#### 开发框架

1. MVC展现层开发框架

整个项目团队对于MVC框架较为熟悉的就是Struts，一方面Struts作为业务的通过选择，另一方面可以较少开发周期并直接带来成本的下降。

模型-视图-控制器（Model-View-Controller）是80年代出现的一种软件设计模式，现在已经被广泛的使用。它强制的把应用程序的输入、处理、输出分开，三个核心模块M-V-C分别负担不同的任务。



模型（Model），模型是应用程序的主体部分。模型表示业务数据和业务逻辑，一个模型可以为多个视图提供数据，提高了应用的可重用性。

视图（View），视图是应用程序中用户界面相关的部分，视图向用户显示数据，并能接收用户的输入数据，但它并不进行任何实际的业务处理。

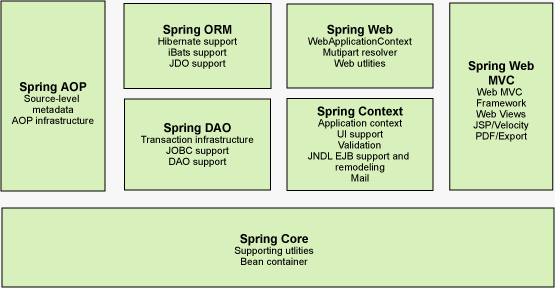
控制器（controller) ，控制器工作就是根据用户请求，调用相应的模型组件处理请求，然后调用相应的视图显示模型返回的数据。

选择这个框架的原因是程序员都很熟悉struts，而且网上相关的文档也很多，框架的的后续支持高，使用人群大，问题解决很容易。

1. Spring 集成框架

Spring 是一个开源框架，是为了解决企业应用程序开发复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许您选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。

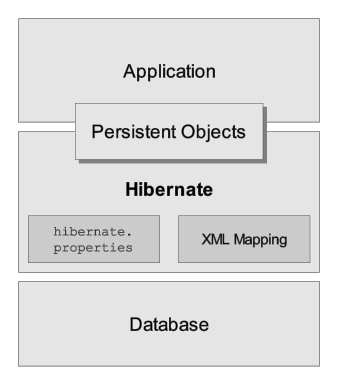
Spring作为AOP, IOC框架， 其采用分层设计可以方便的集成各组件。



使用Spring主要是为了发展自身的核心组件。Spring已经集成了众多优秀的开源组件，而且大部分开源框架都会集成Spring。使用人数众多，问题解决容易。

1. Hibernate 持久层框架

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了轻量级的对象封装，使 Java 程序员可以随心所欲的使用对象编程思维来操纵数据库。它不仅提供了从 Java 类到数据表之间的映射，也提供了数据查询和恢复机制。



相对于使用 JDBC 和 SQL 来手工操作数据库，Hibernate 可以大大减少操作数据库的工作量。 另外 Hibernate 可以利用代理模式来简化载入类的过程，这将大大减少利用 Hibernate QL 从数据库提取数据的代码的编写量，从而节约开发时间和开发成本 Hibernate 可以和多种Web 服务器或者应用服务器良好集成。

#### 数据库

由于Oracle需要收费，所以在这里优先考虑选用Mysql的数据库。由于自身良好的性能、稳定性以及对于程序开发人员较好的熟悉程序成为首选。

#### 操作系统

在企业级的应用中只能选择Window Server 系列产品，如：Window Server 2003，Window Server 2008等，但Window系列操作系统的安全性较Unix，Linux系列操作系统低一些。

#### 应用服务器

可以采用开源的Tomcat作为开发和部署应用服务器，如果用户对Web 应用服务器要特殊的要求，可以按照客户的需要进行选择。Tomcat是遵循Sun关于应用服务器的相关规范开发的开源软件，开源社区比较活跃，在使用的过程中如果有任何问题都可以很容易的找到解决方案。

从部署架构上讲，应用服务器可以与Web服务器相连，共同为客户端服务，或应用服务器独立为客户端服务。

### Dot net技术路线

#### 开发语言特性

1. 技术来自于一家公司；
2. 多数设计模式最佳实践灵感来源于J2EE阵营；
3. 仅支持Windows系列操作系统；
4. 无开源社区支持，是以框架开发者为主导的设计；
5. 门槛很低，使用方便，学习成本较低；
6. 新技术更新较慢。

#### 开发框架

数据持久层Hibernate和SpringData这两个都是非常优秀的数据持久层，。SpringData用于支持非常好的面向对象的设计的模型，SpringData用于支持应用程序的移植，这样就具有更大的弹性。

IOC容器Spring.Net和Castle，这是两个dotnet非常优秀的IOC容器。Spring.Net同样是Java的Spring 的移植版本，目前的版本是0.6。

Castle则是dotnet下出现新的IOC容器,它的功能，成熟度方面比Spring.Net好得多，框架中准备采用Castle.最后的平台的技术架构就是Nhibernate/IbatisNet + Castle + ASP.NET。

#### 数据库

这个选用的是MySql数据库系统，可以充分利用该数据库许多特性和让数据库的优势发挥到最大化。

#### 操作系统

在此系统中只能选择Window Server 系列产品，如：Window Server 2003，Window Server 2008等，但Window系列操作系统的安全性较Unix，Linux系列操作系统低一些。

#### 消息中间件

Microsoft BizTalk Server 用来连接内部和跨组织的系统，包括交换数据和梳理跨多系统的业务流程，集成商业规则与人力工作流服务。Microsoft BizTalk Server的实时、端到端的供应链管理涉及企业内部或跨企业的每个客户，系统，人和流程，使用户能够根据来自于地理上分布的集成系统的实时数据产生可靠的商务决策。

#### Web 服务器

只能使用Microsoft 公司的IIS(Internet Information Server)。

### 其它因素

1. 本项目复用“Portal二期项目”的工作成果，“Portal二期项目”采用J2EE技术路线；
2. 项目组成员对J2EE路线相关技术较熟悉；
3. 项目组外其他技术人员也都对J2EE路线较熟悉，这样，在人力资源紧缺的时候可以很快的得到补充；
4. 客户指定的硬件设备不适于安装和使用Windows系列操作系统，也就不能很好的支持.NET 语言；
5. Dot net 技术路线所需的软件产品通常依赖于商业的操作系统、数据库、消息中间件、开发框架等软件产品。

## 使用人群及使用约束

1、框架开发人员

框架开发人员可以对整个框架进行修改。

2、程序开发人员

程序开发人员只能利用暴露的接口。

## 框架设计思想

在整体框架设计上采用spring的IOC思想，使得各层解耦。通过动态注入，可以随时切换底层的实现技术。但程序开发人员根本无法感觉到差异的存在。框架设计的原则是：简单、实用、透明、规范。

1. ServiceManager

ServiceManager为静态类。通过ServiceManager可以直接获得各种组件。此静态类事直接暴露给程序开发人员使用的。

1. componentService

组件服务层是通过ServiceManager获得的。其中包括：MailTemplate、webserviceTemplate、MessageTemplate、TaskTemplate。此层组件是可以互相独立的，并且直接提供给业务逻辑层使用的，不与其它层有交互的。这四个接口是直接提供给程序开发人员使用的，并且为了每个接口提供了配套使用的DTO类或接口。用户不知道底层实现的技术和方式。

1. Utils

公用组件。是提供给每一个层次使用的组件，无需ServiceManager管理。这一层包括transform、log、configuration等组件，可直接提供给各层开发人员。

1. DAO层

DAO层是直接服务于业务逻辑层的供开发人员直接使用的。由于springside已经对hibernate做了三层封装，并且在对数据库的操作上是具备一定的复杂性和不可预见性的，所以我们不打算在此基础上进行进一步的封装，而是直接提供给程序开发人员使用。但是，由于springside对hibernate的封装有三层方式，使得开发方式有多种选择。为了使得程序员在开发的时候能够统一，我们提出了相关约束来规范：

1. 抽象出每个模块所使用的数据对象（POJO）
2. 采用POJO与POJO manager一一对应的原则
3. 采用POJO manager继承GenericManager<T，Long>的方式。
4. 如果有功能扩展，在每个manager上进行