# 让规范使自己的代码看上去“高大上”

## 命名规范

命名时应始终采用完整的英文描述符。此外，一般应采用小写字母，但类名、接口名以及任何非初始单词的第一个字母要大写。

* 一般概念

\* 尽量使用完整的英文描述符

\* 采用适用于相关领域的术语

\* 采用大小写混合使名字可读（驼峰式）

\* 尽量少用缩写，但如果用了，要明智地使用，且在整个工程中统一

\* 避免使用长的名字（小于 15 个字母是个好主意）

\* 避免使用类似的名字，或者仅仅是大小写不同的名字

\* 避免使用下划线（除静态常量等）

* 示范

\* 包（Package） 采用完整的英文描述符，应该都是由小写字母组成。 如：

     java.awt

我们约束，包名规范，公司域名反转+项目名称+模块简称+[功能特征]，如：

     cn.sunline.rmns.param.ui.client

　\* 类（Class） 采用完整的英文描述符，所有单词的第一个字母大写，采用驼峰式单词。 如：   
       Customer   
     SavingsAccount

\* 接口（Interface） 采用完整的英文描述符说明接口封装，所有单词的第一个字母大写。

习惯上，名字后面加上后缀 able， ible 或者 er，但这不是必需的。如：   
       Contactable   
     Serializable

\* 组件/部件（Component） 使用完整的英文描述来说明组件的用途，末端应接上组件类型。

如：

      okButton   
       customerList   
       fileMenu

　　\* 异常（Exception） 通常采用字母 e 表示异常。 e

\* 类变量 字段采用完整的英文描述，第一个字母小写，任何中间单词的首字大写，禁止使用a,b等当做类变量。

如：

firstName

lastName

\* 实参/参数 同字段/属性的命名规则，约定get,set方法统一由IDE生成，禁止手敲。  
       public void setFirstName(String firstName){   
             this.firstName = firstName;   
     }

\* 局部变量 同字段/属性的命名规则

* 获取成员函数 被访问字段名的前面加上前缀 get。 getFirstName(), getLastName()
* 布尔型的获取成员函数 所有的布尔型获取函数必须用单词 is 做前缀。 isPersistent(), isString()
* 设置成员函数 被访问字段名的前面加上前缀 set。 setFirstName(), setLastName(),setWarpSpeed()
* 普通成员函数 采用完整的英文描述说明成员函数功能，第一个单词尽可能采用一个生动的动词，第一个字母小写。 openFile(), addAccount()

\* 静态常量字段（static final） 全部采用大写字母，单词之间用下划线分隔。

MIN\_BALANCE,

DEFAULT\_DATE

（一般代码中用到的常量都应该定义成static final，一般如果是数字，被称作“魔法数字”）

 \* 循环计数器 通常采用字母 i，j，k 或者 counter 都可以接受。 i, j, k, counter

 \* 数组 数组应该总是用下面的方式来命名： byte[] buffer;

规定：代码中所有数组的使用byte[] buffer方式定义，不使用byte buffer[]方式

## 注释规范

* 一般概念

\* 注释应该增加代码的清晰度

\* 保持注释的简洁

\* 在写代码之前写注释

\* 注释出为什么做了一些事，而不仅仅是做了什么

* 示范

\* 文档注释

 在紧靠接口、类、成员函数和字段声明的前面注释它们。

  /\*\*

 \*   
    \* 客户：客户是我们将服务和产品卖给的人或机构。   
     \*/   
  
　\* C 语言风格   
      采用 C 语言风格的注释去掉不再使用但你仍想保留的代码。仍想保留是因为用户万一会改变想法，或者在调试过程中想让它暂时失效。如：

 /\* 这部分代码因为已被它之前的代码取代，由 B.Gustafsson, 于 1999 年 6 月

 \* 4 日注释掉。如果两年之后还未使用，将其删除。. . .

 \*（源代码）

 \*/

　\* 单行   
      在成员函数内采用单行注释，来说明业务逻辑、代码段和暂时变量的声明。注释符"//"后必须紧跟一个空格，然后才是注释信息。 如：   
     // 遵照 Sarek 的规定，给所有   
     // 超过 $1000 的发货单   
     // 打 5% 的折扣。让利活   
     // 动于 1995年 2 月开始.   
  
   \* 注释哪些部分

类 类的目的、即类所完成的功能，注释出采用的变量。

接口 设置接口的目的、它应如何被使用以及如何不被使用。

成员函数注释 对于设置与获取成员函数，在成员变量已有说明的情况下，可以不加注释；普通成员函数要求说明完成什么功能，参数含义是什么返回什么；

普通成员函数内部注释 控制结构，代码做了些什么以及为什么这样做，处理顺序等。

实参/参数 参数含义、及其它任何约束或前提条件

字段/属性 字段描述

局部变量 无特别意义的情况下不加注释

\* 修改代码

开发人员在修改代码的时候，必须加上注释，注释需要加在类上和修改的地方，写明为什么修改，怎么修改，修改人，修改时间，针对此次修改的版本和范围。

## Java文件样式约定

所有的 Java(\*.java) 文件都必须遵守如下的样式规则：

1) 版权信息

版权信息必须在 java 文件的开头，比如：   
  
　　/\*\*   
    \* Copyright @ 2000 Shanghai XXX Co. Ltd.   
    \* All right reserved.   
    \* @author: gcgmh   
    \* date: 2008-12-22   
    \*/   
其他不需要出现在 javadoc 的信息也可以包含在这里。

2) Package/Imports   
　　package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，则应该用 \* 来处理。   
　　package hotlava.net.stats;   
      
     import java.io.\*;   
     import java.util.Observable;   
     import hotlava.util.Application;   
　　这里 java.io.\* 是用来代替InputStream and OutputStream 的。   
  
 3) Class   
　　接下来的是类的注释，一般是用来解释类的。   
　　/\*\*   
   \* A class representing a set of packet and byte counters   
   \* It is observable to allow it to be watched, but only   
   \* reports changes when the current set is complete   
   \*/   
　　接下来是类定义，包含了在不同的行的 extends 和 implements   
　 public class CounterSet extends Observable implements Cloneable{   
        ……   
         ……   
    }   
     
   4) Class Fields   
  
　　接下来是类的成员变量：   
  
　/\*\*   
    \* Packet counters   
    \*/   
    protected int[] packets;   
  
　public的成员变量必须生成文档（JavaDoc）。proceted、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。   
  
　~~5) 存取方法（类的设置与获取成员函数）   
 接下来是类变量的存取的方法。它只是简单的用来将类的变量赋值获取值的话，可以简单的写在一行上，如类的成员变量已经有注释，类变量的存取方法可以没有注释。   
  
　　public int[] getPackets() { return this.packets; }   
    public void setPackets(int[] packets) { this.packets = packets; }~~     ……   
  
　　要求说明的是，对于集合，加入成员函数来插入和删除项；另其它的方法不要写在一行上。   
  
　6) 构造函数

　　接下来是构造函数，它应该用递增的方式写（比如：参数多的写在后面）。

public CounterSet(int size){ this.size = size;}   
  
　7) 克隆方法

如果这个类是可以被克隆的，那么下一步就是 clone 方法：

　public Object clone() { try { …… }catch(CloneNotSupportedException e) { …… }}   
  
　8) 类方法 （类的普通成员函数）

下面开始写类的方法：

/\*\*   
    \* Set the packet counters   
    \* param r1 - ……   
    \* param r2 - ……   
    \* ……   
    \*/   
   protected final void setArray(int[] r1, int[] r2, int[] r3, int[] r4) throws IllegalArgumentException{   
    // Ensure the arrays are of equal size   
    ……   
   }   
  
9) toString 方法

一般情况下，每一个类都应该定义 toString 方法：

public String toString() { ……}

10) main 方法

普通类，考虑置入一个main()方法，其中包含用于测试那个类的代码，如果包含了main() 方法, 那么它应该写在类的底部。

## Java编码其他约定

n 文档化

必须用 javadoc 来为类生成文档。不仅因为它是标准，这也是被各种 java 编译器都认可的方法。~~使用 @author 标记是不被推荐的，因为代码不应该是被个人拥有的~~，我们所有的代码。   
  
　　n 缩进

缩进应该是每行2个空格。 不要在源文件中保存Tab字符， 在使用不同的源代码管理工具时Tab字符将因为用户设置的不同而扩展为不同的宽度。   
  
　　如果你使用 UltrEdit 作为你的 Java 源代码编辑器的话，你可以通过如下操作来禁止保存Tab字符， 方法是通过 UltrEdit中先设定 Tab 使用的长度室2个空格，然后用 Format|Tabs to Spaces 菜单将 Tab 转换为空格。

n 页宽

页宽应该设置为80字符。 源代码一般不会超过这个宽度， 并导致无法完整显示， 但这一设置也可以灵活调整。 在任何情况下， 超长的语句应该在一个逗号或者一个操作符后折行。 一条语句折行后， 应该比原来的语句再缩进2个字符。   
  
　　n {} 对

{} 中的语句应该单独作为一行。 例如， 下面的第1行是错误的， 第2行是正确的:

if (i>0) { i ++ }; // 错误, { 和 } 在同一行

if (i>0) {

i ++

}; // 正确, 单独作为一行   
  
　　n 括号

左括号和后一个字符之间不应该出现空格； 同样， 右括号和前一个字符之间也不应该出现空格。 下面的例子说明括号和空格的错误及正确使用:

CallProc(AParameter); // 正确

CallProc( AParameter ); // 错误

不要在语句中使用无意义的括号，括号只应该为达到某种目的而出现在源代码中。

n JSP文件命名

采用完整的英文描述说明JSP所完成的功能，尽可能包括一个生动的动词，第一个字母小写，如：viewMessage.jsp、editUser.jsp或者forumChooser.jsp等。

n Servlet类命名

一般对应于所服务的对象加后缀Service来命名，如：UserService，TradeService等。

# 从面到点式的认识系统-基础架构

项目类型分类：

…

Batch

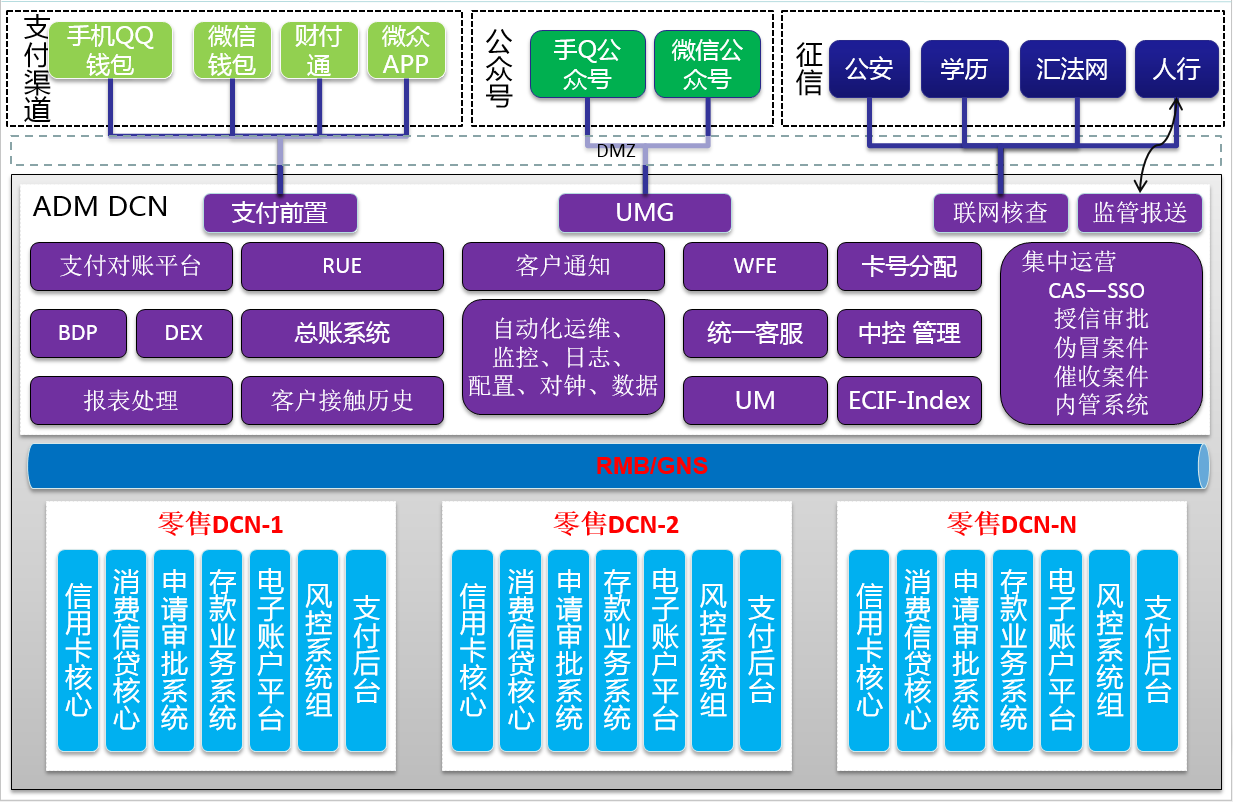
Service

WEB

## 项目结构

|  |  |
| --- | --- |
| XXX-web |  |
| XXX-ui | 页面，非参数 |
| XXX-param-ui | 参数页面 |
| XXX-service-api | 接口 |
| XXX-service-impl | 服务 |
| XXX-infrastructure | 数据模型 |
| XXX-facility | 公共业务基础服务 |
| XXX-batch | 批量 |
| XXX-frontend | 前置类 |
| XXX-adaptor | 适配器 |

## 整体架构



（图2.1.1 使用微众银行分布式架构作为示例）

注释：

RMB：微众消息总线

GNS：服务定位服务，消息路由

## Web架构

客户端

Spring MVC

服务端

DB

## Service架构

*<见图2.1.1 的零售DCN1>*

## 批量架构

*<批量分布式架构集中介绍，此文档不展开>*

# 自动代码生成

POJO对象 TM对象

解析erm生成

页面使用对象 U对象

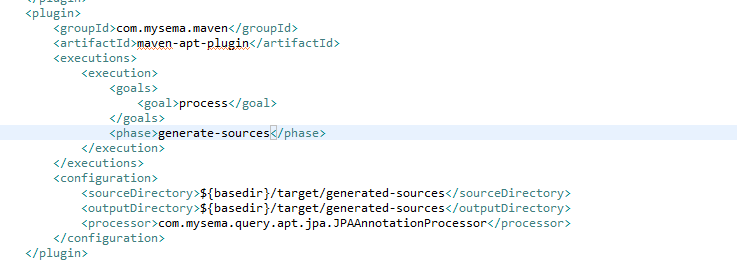
解析erm生成

Spring data数据库操作对象 R对象

通过apt插件生成

QueryDsl 对象，Q对象

通过apt插件生成



# Web开发

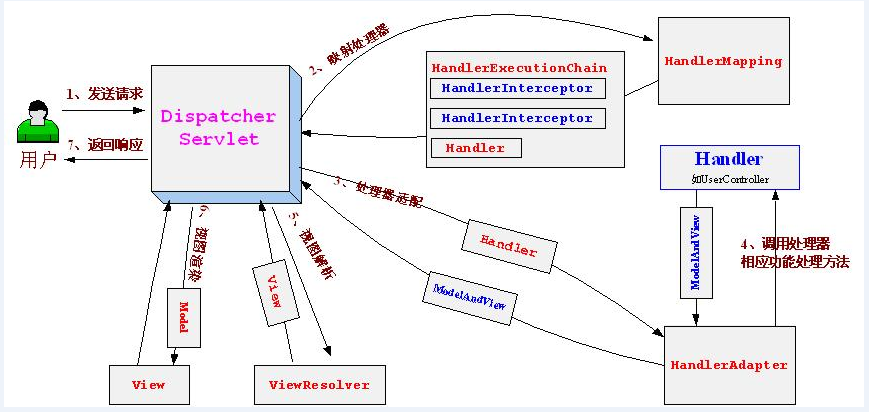
## 前端

使用GWT技术

## 后端

基于Spring的轻量级服务开发

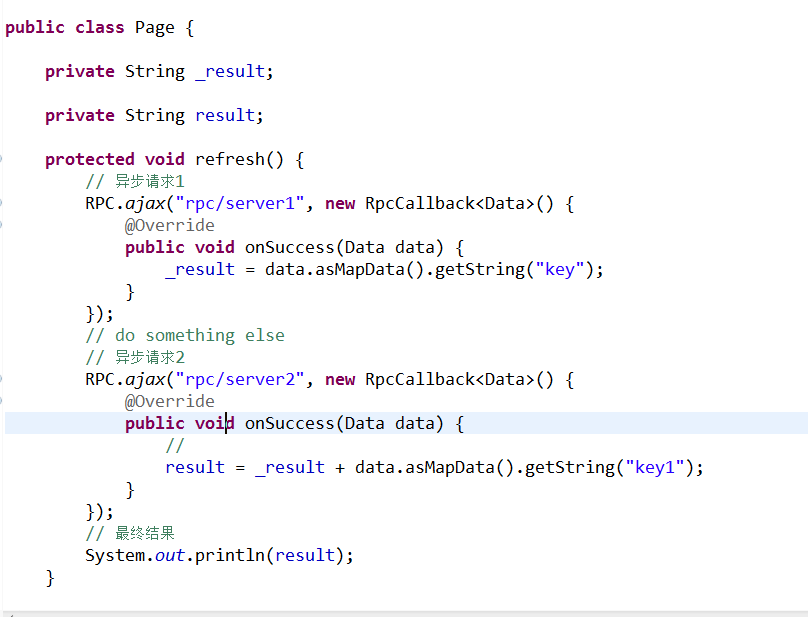
## SpringMVC



## 同步与异步

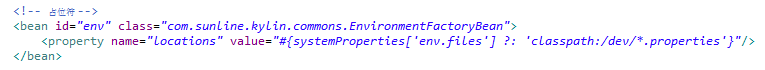
GWT的RPC技术（AJAX）：异步过程

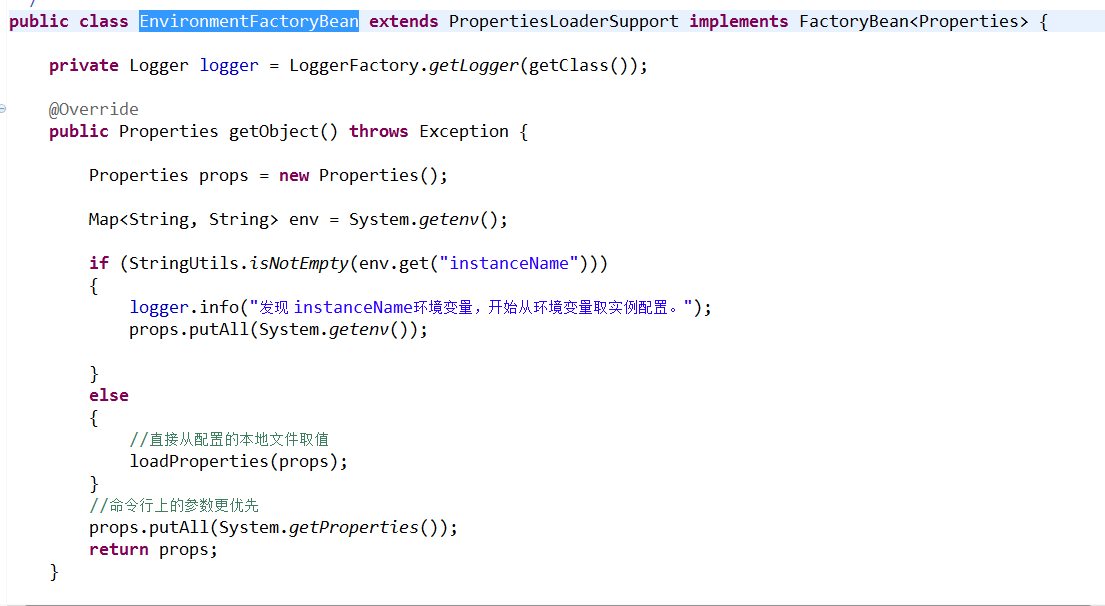
异步数据的加载：如果是异步的过程，永远不要依赖代码执行的先后顺序来决定数据返回的先后顺序。错误示例：



# Service开发

## 环境变量：env





Env包含系统环境变量，命令行参数，实例变量（数据库）怎么加载？？？

# 批量开发

*<单独介绍>*

# 持久化层（JDBC？Hibernate？还是JPA？）

## 什么是JDBC

JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用[Java语言](http://baike.baidu.com/view/229611.htm)编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序，同时，JDBC也是个商标名。

ODBC(Open Database Connectivity，开放数据库互连)是微软公司开放服务结构(WOSA，Windows Open Services Architecture)中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准API（应用程序编程接口）。这些API利用SQL来完成其大部分任务。ODBC本身也提供了对SQL语言的支持，用户可以直接将SQL语句送给ODBC

## 什么是JPA

JPA全称Java Persistence API.JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体[对象持久化](http://baike.baidu.com/view/402359.htm)到数据库中。

Sun引入新的JPA ORM规范出于两个原因：其一，简化现有Java EE和Java SE应用开发工作；其二，Sun希望整合ORM技术，实现天下归一。

总而言已，JPA是ORM（Object Relational Mapping）的标准，目的是想通过JPA将数据库访问一统天下。

## 什么是QueryDsl

是一个Java开源框架用于构建类型安全的SQL查询语句。它采用API代替拼凑字符串来构造查询语句。可跟 Hibernate 和 JPA 等框架结合使用。

QueryDsl只是一种构建SQL的工具，不和数据库打交道。

## 什么是Hibernate

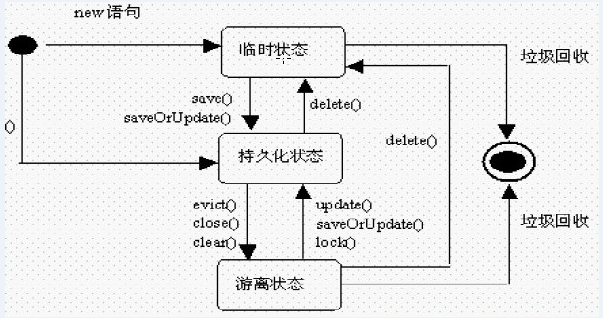
Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。

三种状态

transient(瞬时状态) 刚new出来的对象

persistent(持久化状态) 已经被保存到数据库，托管状态

detached(离线状态) 数据库中有，但是session中不存在该对象



// begin Transaction

User user = new User();

user.setUsername(“user”);

user.setPwd(“123456”);

// 以上user处于transient状态

session.save(user); // 发出一条insert语句

// 以上user处于persistent状态

user.setUsername(“user1”);

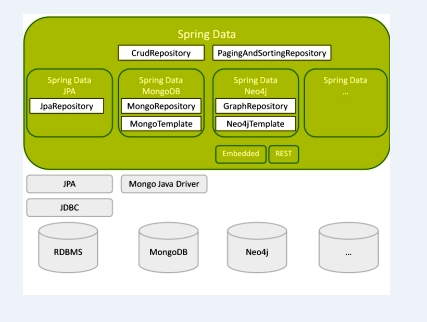
user.setUsername(“user2”); //这里会发出几条update语句？？

// Session.clear();

// 以上user处于detached状态

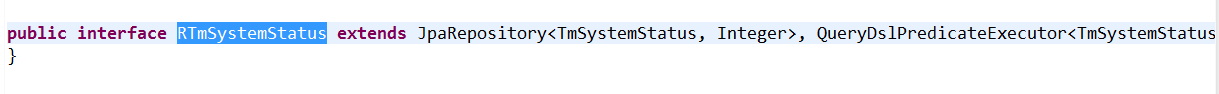
// Transaction commit

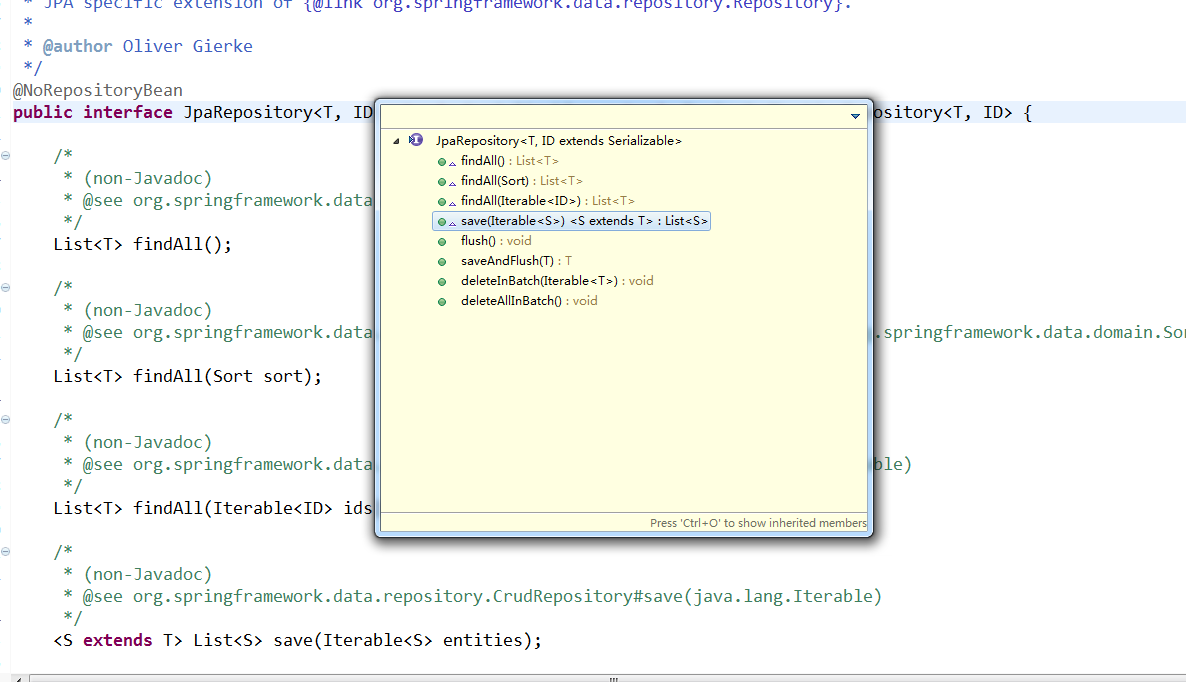
## Spring data是什么



Spring data是一个用于简化数据库访问，并支持云服务的开源框架。其主要目标是使得对数据的访问变得方便快捷，并支持map-reduce框架和云计算数据服务

Spring data也只是一个工具，为了简化你使用JPA来操作数据库





过程：

1. 生成RTmSystemStatus代理类，客户端使用该代理类访问数据库
2. 代理类根据方法名字自动生成Sql，执行数据库操作。

## JDBC，JPA，QueryDsl，SpringData，Hibernate关系图

结构关系：

Hibernate

TopLink

OpenJpa

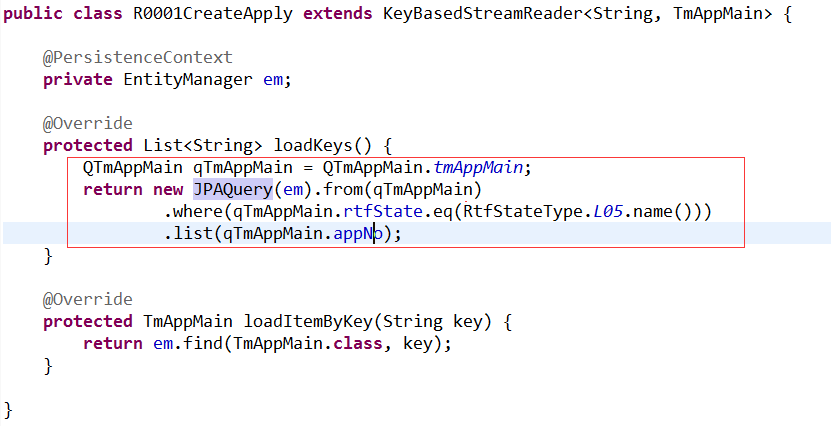
QueryDsl

Spring Data

JPA

JDBC

代码关系：





由JPA的具体实现去实现list方法。

我们项目使用JPA+Hibernate来完成数据库访问，所以list方法由Hibernate提供实现。









思考：entityManager.createNativeQuery(“SELECT XXX FROM TABLE WHERE …”).getSingleResult()使用什么来实现。

原则：尽量使用JPS接口，不要使用Hibernate接口

## 该不该选择ibatis

Ibatis更灵活，但是开发难度更大

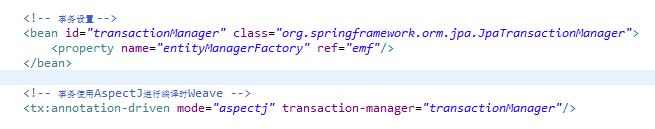
Hibernate快速开发，在大数据的时候效率较低，（内存扫描）

## 事务管理





告诉Spring识别出为其实现的持久层接口。



把事务交给JPA处理，并且由Aspectj进行编译注入（由Aspectj插件完成）。



每个Jpa的sava，update都带有一个事务。



当有@Transactional，aspectJ会对对应方法体加上事务

在了解Hibernate3种状态之后，持久化状态的对象是否需要执行update方法？？？

// begin Transaction

User user = session.load(user);

// 以上user处于persistent状态

user.setUsername(“user1”);

// 需要执行session.update(user)吗;？？？

// Transaction commit

**事务隔离机制**

**1、丢失更新**    
如果两个事务都要更新数据库一个字段X，x=100

|  |  |
| --- | --- |
| **事务A** | **事务B** |
| 读取X＝100 | 读取X＝100 |
| 写入x＝X+100 | 写入x＝X+200 |
| 事务结束x=200 | 事务结束x=300 |
|  | 最后x=300 |

**2、脏读（未提交读）**   
防止一个事务读到另一个事务还没有提交的记录。 如：

|  |  |
| --- | --- |
| **事务A** | **事务B** |
|  | 写入x＝X+100 （x=200） |
| 读取X＝200 （读取了事务B未提交的数据） |  |
|  | 事务回滚x=100 |
|  | 事务结束x=100 |
| 事务结束 |  |

**3、不可重复读**一个事务在自己没有更新数据库数据的情况，同一个查询操作执行两次或多次的结果应该是一致的；如果不一致，就说明为不可重复读。  
还是用上面的例子

|  |  |
| --- | --- |
| **事务A** | **事务B** |
| 读取X＝100 | 读取X＝100 |
| 读取X＝100 | 写入x＝X+100 |
|  | 事务结束， x=200 |
| 读取X＝200 （此时，在同一个事务A中，读取的X值发生了变化！） |  |
| 事务结束 |  |

**4 幻读（Phantom Read）**  
事务A读的时候读出了15条记录，事务B在事务A执行的过程中 增加 了1条，事务A再读的时候就变成了 16 条，这种情况就叫做幻影读。  
不可重复读说明了做数据库读操作的时候可能会出现的问题。

**二 事务隔离级别通过锁的实现机制**  
两个锁：  
排他锁（悲观锁） 被加锁的对象只能被持有锁的事务读取和修改，其他事务无法在该对象上加其他锁，也不能读取和修改该对象  
共享锁（乐观锁） 被加锁的对象可以被持锁事务读取，但是不能被修改，其他事务也可以在上面再加共享锁。

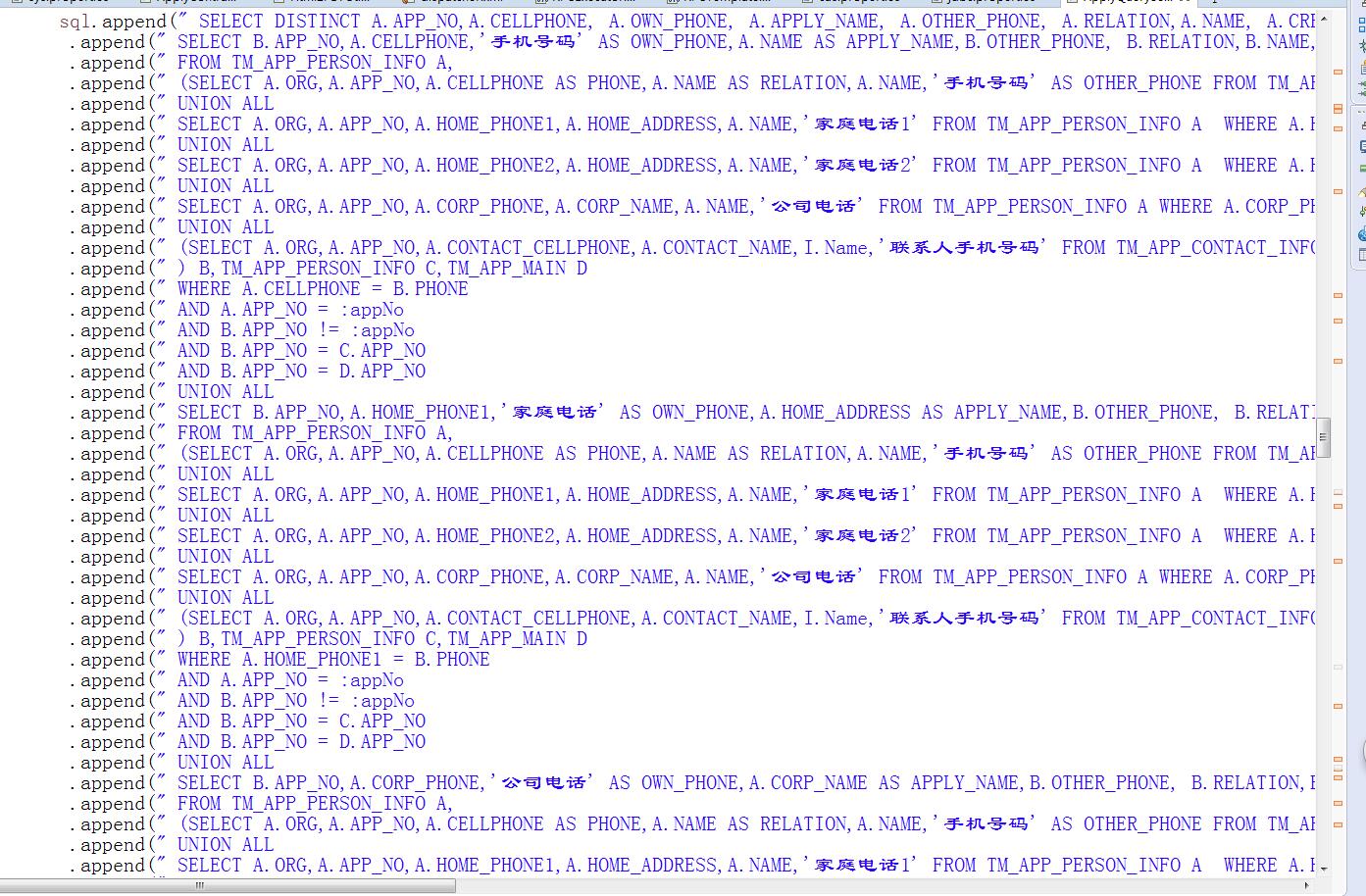
特别的，对共享锁： 如果两个事务对同一个资源上了共享锁，事务A 想更新该数据，那么它必须等待 事务B 释放其共享锁。  
  
在运用 排他锁 和 共享锁 对数据对象加锁时，还需要约定一些规则，例如何时申请 排他锁 或 共享锁、持锁时间、何时释放等。称这些规则为封锁协议（Locking Protocol）。对封锁方式规定不同的规则，就形成了各种不同的封锁协议。*<不再扩展>*

**关于我们的JpaVersion**

基于Hibernate实现的一套乐观锁机制，应用程序不应该对他赋值。

## 大SQL，慢SQL，索引

禁止使用大sql，可读性差，数据库资源利用高，性能低下，难以维护



1. 用group by 代替distinct
2. 如非必要，拒绝使用join，多表连接将产生笛卡尔积
3. 如非必要，拒绝order by
4. 如非必要，不要查询记录的所有字段，只查询需要字段
5. 如非必要，拒绝使用findAll()（内存撑爆）
6. 如非必要，拒绝使用外键，我们系统一律不使用外键，外键约束由应用程序解决
7. Where条件尽量走主键或者索引
8. 表甚建索引，索引对查询提交效率，但是对更新，插入，删除降低效率
9. Sql语句where条件的次序尽量将最大幅度过滤数据量的条件写在靠前
10. 如非必要，尽量不使用or
11. 如非必要，尽量不使用in, not in
12. 如非必要，尽量不使用like, not like
13. …

# 通讯

## RabbitMq和Spring AMQP

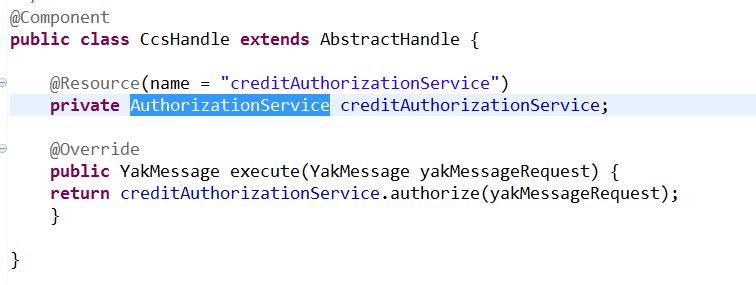
MQ全称为Message Queue, 消息队列（MQ）是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消 息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过 队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。其中较为成熟的MQ产品有IBM WEBSPHERE MQ。

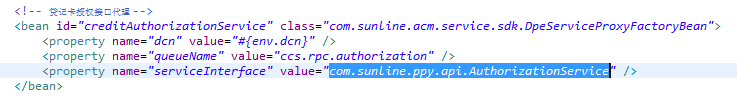
MQ是消费-生产者模型的一个典型的代表，一端往消息队列中不断写入消息，而另一端则可以读取或者订阅队列中的消息。MQ和JMS类似，但不同的是JMS是SUN JAVA消息中间件服务的一个标准和API定义，而MQ则是遵循了AMQP协议的具体实现和产品。

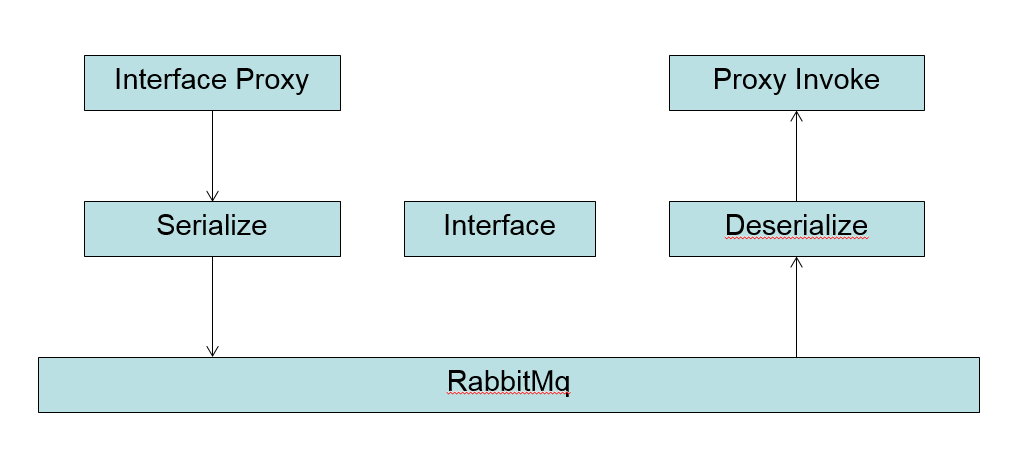
RabbitMQ是一个在AMQP基础上完整的，可复用的企业消息系统。他遵循Mozilla Public License开源协议

*<不针对RabbitMq深入展开，只介绍我们的使用>*

### 客户端







客户端对接口AuthorizationService生成代理接口，当代理接口执行authorize方法时，将方法名，方法类型，参数通过RabbitMq传到服务端，同步调用等待返回，异步调用不等待返回。

### 服务端

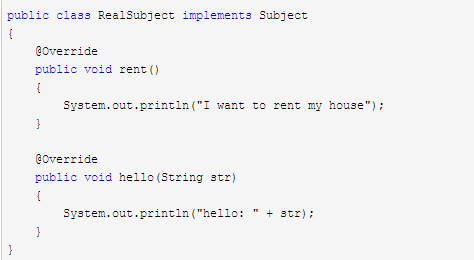
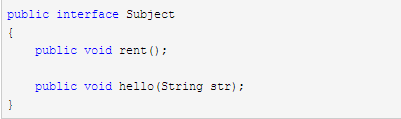


服务端收到消息后，拿到穿过来的方法名，参数，通过反射找到实现类的具体方法，然后拿着收到的参数去调用该方法得到结果，同步返回结果，异步不返回结果。

SpringAMQP是对AMQP协议的封装，Spring Rabbit负责与RabbitMq通讯。

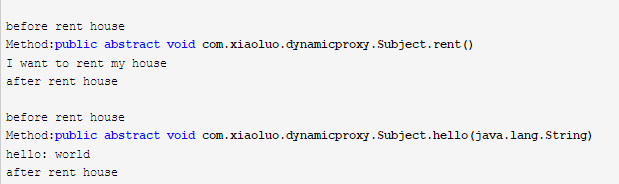
### 设计模式：动态代理

基于接口的动态代理：Proxy，InvocationHandle









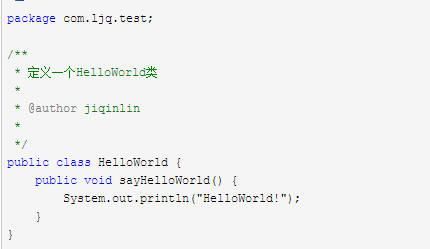
代理对象（接口）

代理前逻辑

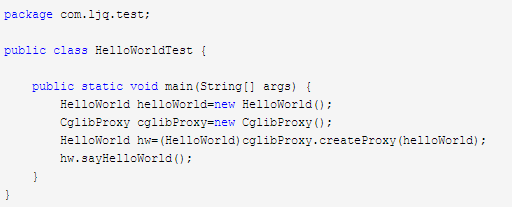
代理后逻辑

被代理对象（实现）

没有接口的动态代理：cglib







代理对象（自动生成，Classload自动缓存被代理对象类型）

代理前逻辑

代理后逻辑

被代理对象（实现）

## Socket通讯

按连接方式分：

长连接

单工长连接

双工长连接

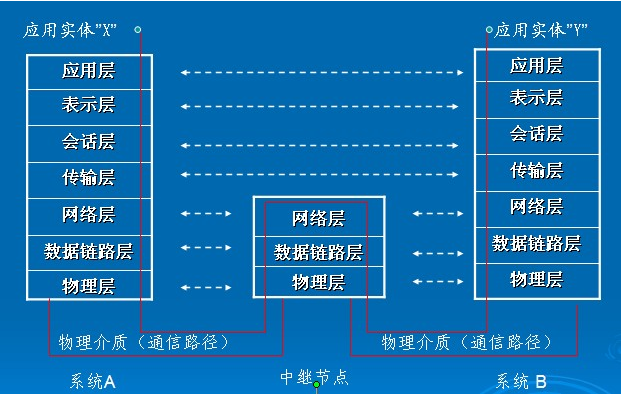
短连接

按协议分：

TCP/IP 传输层协议/网络层协议

UDP/IP 传输层协议/网络层协议

HTTP，FTP，SMTP…（属于TCP，更高层） 应用层协议



我们的系统：

### MINA（Apache Mina）

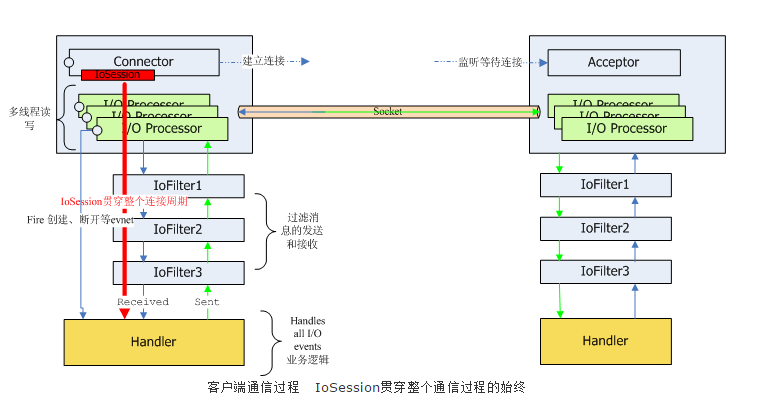
ApacheMINA是一个网络应用程序框架，用来帮助用户简单地开发高性能和高可扩展性的网络应用程序。它提供了一个通过Java NIO在不同的传输例如TCP/IP和UDP/IP上抽象的事件驱动的异步API。

特点：

基于NIO非阻塞网络通讯框架

基于过滤器的面向业务的网络通讯框架

基于线程池的并发网络通讯框架



*<详见代码>*

其他通讯框架：netty，dubbo，Grizzly

# 什么是Spring？为何使用Spring？

Spring：Spring框架是由于软件开发的复杂性而创建的。Spring使用的是基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅仅限于服务器端的开发。从简单性、可测试性和松耦合性的角度而言，绝大部分Java应用都可以从Spring中受益。

◆目的：解决企业应用开发的复杂性

◆功能：使用基本的JavaBean代替EJB，并提供了更多的企业应用功能

◆范围：任何Java应用

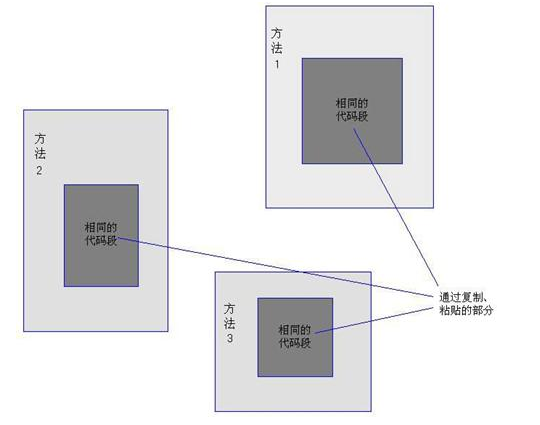
Spring是一个轻量级控制反转(IoC)和面向切面(AOP)的容器框架。

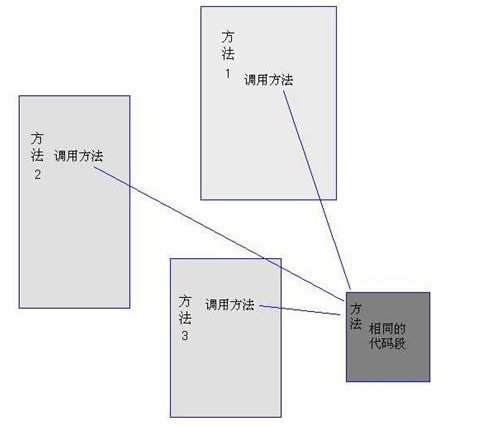
## Spring IOC和Spring AOP

Spring IOC：Inversion of Control，控制反转

Spring AOP: Aspect Orient Programming，面向切面编程（Cglib技术）

主要解决多个地方包含相同代码的代码冗余问题





1.通知(Advice):

通知定义了切面是什么以及何时使用。描述了切面要完成的工作和何时需要执行这个工作。

2.连接点(Joinpoint):

程序能够应用通知的一个“时机”，这些“时机”就是连接点，例如方法被调用时、异常被抛出时等等。

3.切入点(Pointcut)

通知定义了切面要发生的“故事”和时间，那么切入点就定义了“故事”发生的地点，例如某个类或方法的名称，Spring中允许我们方便的用正则表达式来指定

4.切面(Aspect)

通知和切入点共同组成了切面：时间、地点和要发生的“故事”

5.引入(Introduction)

引入允许我们向现有的类添加新的方法和属性(Spring提供了一个方法注入的功能）

6.目标(Target)

即被通知的对象，如果没有AOP,那么它的逻辑将要交叉别的事务逻辑，有了AOP之后它可以只关注自己要做的事（AOP让他做爱做的事）

7.代理(proxy)

应用通知的对象，详细内容参见设计模式里面的代理模式

8.织入(Weaving)

把切面应用到目标对象来创建新的代理对象的过程，织入一般发生在如下几个时机:

(1)编译时：当一个类文件被编译时进行织入，这需要特殊的编译器才可以做的到，例如AspectJ的织入编译器

(2)类加载时：使用特殊的ClassLoader在目标类被加载到程序之前增强类的字节代码

(3)运行时：切面在运行的某个时刻被织入,SpringAOP就是以这种方式织入切面的，原理应该是使用了JDK的动态代理技术

代理前织入逻辑

切点

目标

代理

代理后织入逻辑

切面

AOP典型应用：Spring事务管理

# Maven

Maven是基于项目对象模型(POM)，可以通过一小段描述信息来管理项目的构建，报告和文档的软件项目管理工具。

常用命令：

mvn archetype：create 创建Maven项目

mvn compile 编译源代码

mvn deploy 发布项目

mvn test-compile 编译测试源代码

mvn test 运行应用程序中的单元测试

mvn site 生成项目相关信息的网站

mvn clean 清除项目目录中的生成结果

mvn package 根据项目生成的jar

mvn install 在本地Repository中安装jar

mvn eclipse:eclipse 生成eclipse项目文件

mvnjetty:run 启动jetty服务

mvntomcat:run 启动tomcat服务

mvn clean package -Dmaven.test.skip=true:清除以前的包后重新打包，跳过测试类

补充：

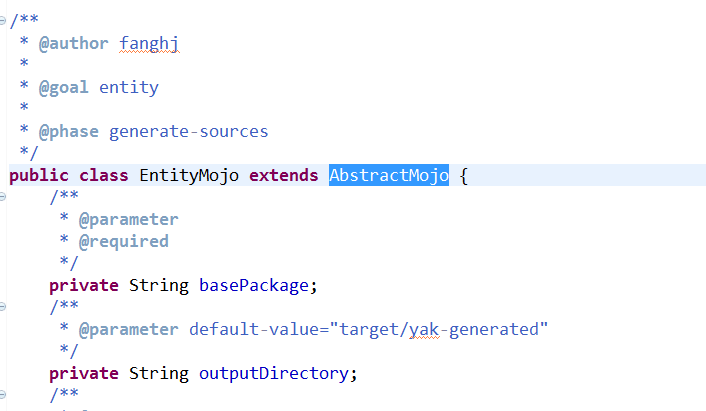
mvn xmlbeans:xmlbeans

## maven插件开发



Goal:entity,class-def 目标执行插件

Phase:生命周期，生成源码阶段





 1 clean生命周期

1. pre-clean    ：执行清理前的工作；
2. clean    ：清理上一次构建生成的所有文件；
3. post-clean    ：执行清理后的工作

2 default生命周期

default生命周期是最核心的，它包含了构建项目时真正需要执行的所有步骤。

1. validate
2. initialize
3. generate-sources
4. process-sources
5. generate-resources
6. process-resources    ：复制和处理资源文件到target目录，准备打包；
7. compile    ：编译项目的源代码；
8. process-classes
9. generate-test-sources
10. process-test-sources
11. generate-test-resources
12. process-test-resources
13. test-compile    ：编译测试源代码；
14. process-test-classes
15. test    ：运行测试代码；
16. prepare-package
17. package    ：打包成jar或者war或者其他格式的分发包；
18. pre-integration-test
19. integration-test
20. post-integration-test
21. verify
22. install    ：将打好的包安装到本地仓库，供其他项目使用；
23. deploy    ：将打好的包安装到远程仓库，供其他项目使用；

1 . 3 site生命周期

1. pre-site
2. site    ：生成项目的站点文档；
3. post-site
4. site-deploy    ：发布生成的站点文档

# 多线程怎么玩（如何让你的程序飞起来）

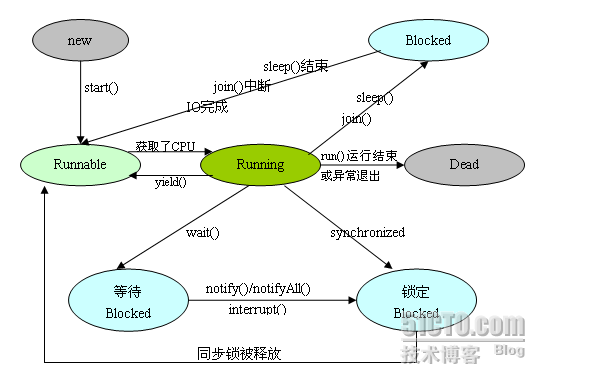
当有一个5个独立的处理逻辑

|  |  |
| --- | --- |
| 逻辑 | 耗时 |
| 逻辑1 | 100ms |
| 逻辑2 | 50ms |
| 逻辑3 | 500ms |
| 逻辑4 | 600ms |
| 逻辑5 | 800ms |
|  |  |

如果使用单线程串行运行，需要时间 （100+50+500+600+800）ms = 2.05s

如果并行执行，需要时间为最耗时逻辑的耗时时间（在硬件资源充足的情况下），需要时间为800ms。

操作系统线程模型：



1、新建状态（New）：新创建了一个线程对象。

2、就绪状态（Runnable）：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态（Running）：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。

## 信号量与临界区（线程间通讯）

临界区：指的是一个访问共用资源（例如：共用设备或是共用存储器）的程序片段，而这些共用资源又无法同时被多个[线程](http://baike.baidu.com/view/1053.htm)访问的特性。当有线程进入临界区段时，其他线程或是[进程](http://baike.baidu.com/view/19746.htm)必须等待（例如：bounded waiting 等待法），有一些同步的机制必须在临界区段的进入点与离开点实现，以确保这些共用资源是被互斥获得使用，例如：[semaphore](http://baike.baidu.com/view/1499210.htm)。只能被单一线程访问的设备，例如：[打印机](http://baike.baidu.com/view/7836.htm)。[1]

信号量：有时被称为信号灯，是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被[并发](http://baike.baidu.com/view/684757.htm)调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。为了完成这个过程，需要创建一个信号量VI，然后将Acquire Semaphore VI以及Release Semaphore VI分别放置在每个关键代码段的首末端。确认这些信号量VI引用的是初始创建的信号量。直接理解为资源即可。

信号量

占用资源，线程1执行

wait

wait

wait

线程4

线程3

线程2

线程1



运行程序输出1 0后就进入死锁状态，该程序永远也不会停止，因为两个线程同时处于等待状态。线程t1锁住了o1对象，等待o2对象，而线程t2锁住o2等待o2对象，谁也不让谁，这就进入了一个循环占有等待的情况了，死锁也就出现了。

**我们系统：**





## Asyncload简介

异步并行加载框架，自行了解

# 缓存怎么玩

**按存储位置分类：**

客户端缓存

服务端缓存

缓存服务器

CDN缓存

缓存算法：

LFU: 最不经常使用算法

LRU: 最近最少使用算法

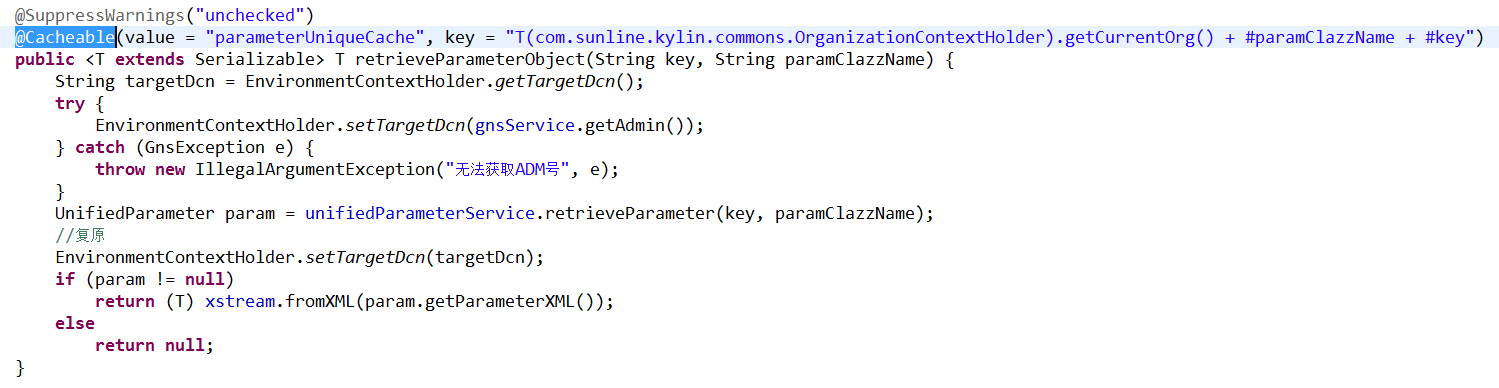
ARC: 自适应缓存替换算法

MRU: 最近最常使用算法

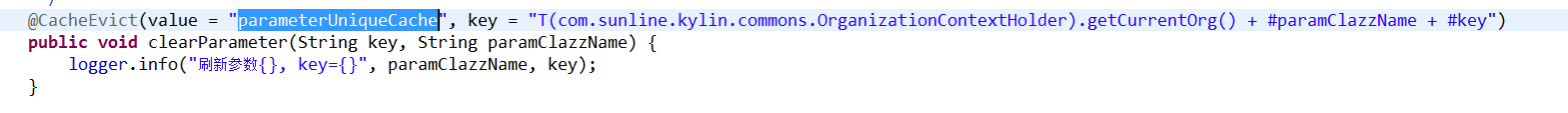
我们系统：

## Spring缓存怎么玩

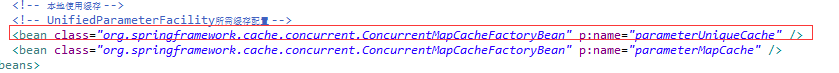
缓存管理器+具体缓存



定义一个名字为parameterUniqueCache



用于刷新缓存



缓存定义

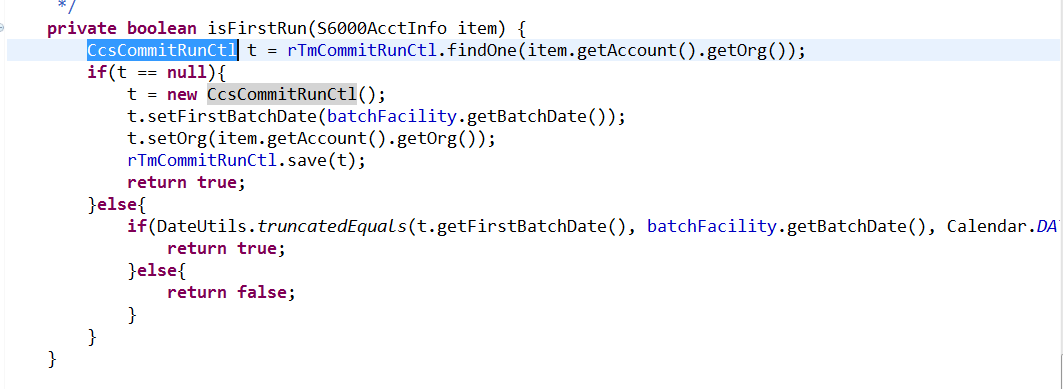


缓存管理器



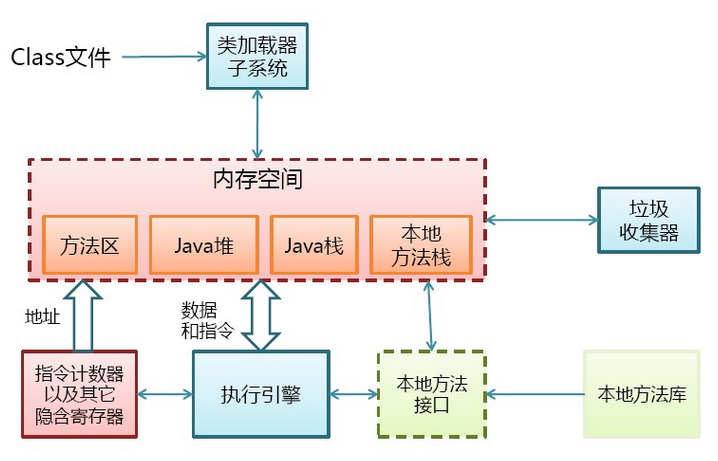
# 线程安全

永远要考虑到你的代码在并发情况下会不会有问题。



# 内存泄露

Jvm内存模型与垃圾回收机制（SUN公司出品的JVM）



垃圾回收主要回收Java堆内存，java堆内存也成为“GC堆”

Java堆细分：

维度一：

新生代

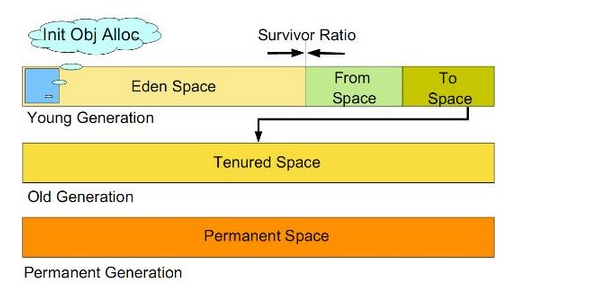
老年代

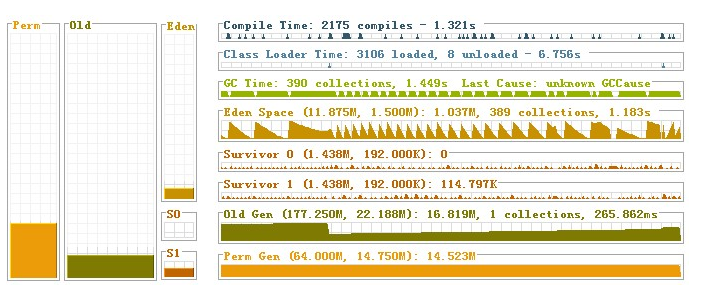
维度二：

Eden空间 8

From Survivor空间 1

To Survivor空间 1





新生代通常存活时间较短，因此基于Copying算法来进行回收，所谓Copying算法就是扫描出存活的对象，并复制到一块新的完全未使用的空间中，对应于新生代，就是在Eden和From Space或To Space之间copy。新生代采用空闲指针的方式来控制GC触发，指针保持最后一个分配的对象在新生代区间的位置，当有新的对象要分配内存时，用于检查空间是否足够，不够就触发GC。当连续分配对象时，对象会逐渐从eden到survivor，最后到旧生代

每次使用Eden和其中一块Survivor空间，将Eden和其中一块Survivor空间里面存活的对象一次性copy到另一块Survivor空间，并清除Eden和刚刚使用过的Survivor空间。（复制算法）

其他算法：

标记删除算法

标记整理算法

分代搜集算法

一般98%的对象处于新生代，朝生夕死，程序中应避免出现不断的new Object往新生代装入对象

-Xmx，-Xms：堆内存最大最小值

-Xmn：新生代大小

-XX:SurvivorRatio：设置eden和survivor比例

-XX:PermSize，-XX:MaxPermSize：持久代大小的最大值最小值

# 设计模式

学会看别人代码（非垃圾代码，能看懂垃圾代码的人必须佩服，因为他肯定很牛逼），其实牛人就在我们身边，等着一双眼睛去发现。多看看诸如Spring出品，Apache出品的源代码，学习他们的编码风格，设计思想。

1. Factory Method（工厂方法）

2. Abstract Factory（抽象工厂）

3. Builder（建造者）

4. Prototype（原型）

5. Singleton（单例）

结构型

6. Adapter Class/Object（适配器）

7. Bridge（桥接）

8. Composite（组合）

9. Decorator（装饰）

10. Facade（外观）

11. Flyweight（享元）

12. Proxy（代理）

行为型

13. Interpreter（解释器）

14. Template Method（模板方法）

15. Chain of Responsibility（责任链）

16. Command（命令）

17. Iterator（迭代器）

18. Mediator（中介者）

19. Memento（备忘录）

20. Observer（观察者）

21. State（状态）

22. Strategy（策略）

23. Visitor（访问者）

# 看懂异常

*<Exception>*

# 编程的思想

为了实现功能而写代码？

你的追求是什么？完美，优雅，谨慎

你考虑过系统的性能吗？

合理的创建类，合理的封装方法，是否你的代码有大量的冗余，方法的相互调用异常复杂，导致最后自己看着都累。

# 互联网时代，千万不要忘了最原始的知识积累途径 - 书

永远不要相信百度查出来的结果，真正的权威是不会在那么低端的地方发表任何言论的，除非是论文。

解决问题只是目标，让你成长的是解决目的的方法，途径，过程。