目录

[一、引言 1](#_Toc11161861)

[二、命名规范 2](#_Toc11161862)

[2.1 包（ Package）的命名 2](#_Toc11161863)

[2.2 Class 的命名 3](#_Toc11161864)

[2.3 Class 变量的命名 3](#_Toc11161865)

[2.4 接口的命名 5](#_Toc11161866)

[2.5 参数的命名 5](#_Toc11161867)

[2.6 数组的命名 5](#_Toc11161868)

[2.7 方法的参数 5](#_Toc11161869)

[三 注释规范 6](#_Toc11161870)

[3.1 块注释 (Block Comments) 6](#_Toc11161871)

[3.2 单行注释 (Single-Line Comments) 7](#_Toc11161872)

[3.3 尾端注释 (Trailing Comments) 7](#_Toc11161873)

[3.4 行末注释 (End-Of-Line Comments) 8](#_Toc11161874)

[四 Java 文件样式 9](#_Toc11161875)

[4.1 版权信息 9](#_Toc11161876)

[4.2 Package/Imports 9](#_Toc11161877)

[4.3 Class 10](#_Toc11161878)

[4.4 Class Fields 10](#_Toc11161879)

[4.5 存取方法 10](#_Toc11161880)

[4.6 构造函数 11](#_Toc11161881)

[4.7 main 方法 11](#_Toc11161882)

[4.8 换行 (Wrapping Lines) 11](#_Toc11161883)

[4.9 行长度 (Line Length) 14](#_Toc11161884)

[4.10 语句 (Statements) 15](#_Toc11161885)

[五 排版格式 19](#_Toc11161886)

[六 JSP 编码规范 20](#_Toc11161887)

[七、文件与目录 22](#_Toc11161888)

[八、日志规范 23](#_Toc11161889)

[九、设计及技巧建议 24](#_Toc11161890)

[9.1 表现层 25](#_Toc11161891)

[9.2 业务层 25](#_Toc11161892)

[9.3 持久层 25](#_Toc11161893)

# 一、引言

编码规范对于程序员而言尤为重要，有以下几个原因：

1、 一个软件的生命周期中， 80%的花费在于维护。

2、 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。

3、 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码。

4、 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误，一如已构建的其它任何产品为了执行规范，每个软件开发人员必须一致遵守编码规范中的代码规范，代码规范提高软件代码的可读性 , 使得开发人员快速和彻底的理解新代码 。好的代码风格不仅会提高可读性，而且会使代码更健壮，更为重要的是在修改时不容易出错。现代软件开发中，维护工作会占用 80%的时间，而且开发者和维护者通常不是同一个程序员。这意味着程序员经

常要阅读和修改别人开发的程序，别人也同样可能需要阅读和修改你开发的程序，所以制定一套完善的编码规范非常重要。定义本规范的目的是使项目中所有的文档及程序编码在写作风格上具有一致性，加

可读性，减少项目组中因为换人带来的损失，从而使程序具有良好的可读性。

# 二、命名规范

## 2.1 包（ Package）的命名

包(Packages)： 一个唯一包名的前缀应全部为小写的 ASCII 字母，并且是一个顶级域名，通常为 com、edu、gov、mil、net、org 等。包名的后续部分可根据各个不同机构内部的命名规范来定义， 这类命名规范可以由特定的目录名来区分部门 (department)、项目 (project)、机器 (machine)和注册名 (login names)。

核心程序部分： com.xxxxxx.base

系统管理部分： com.xxxxxx.sys

项目管理系统程序编码规范 V1.0 4

应用维护部分： com.xxxxxx.appconfig

项目计划管理部分： com.xxxxxx.itemplan

采购管理部分： com.xxxxxx.purchase

财务管理部分： com.xxxxxx .finance

监测评价部分： com.xxxxxx.analyse

如果在开发过程中还有其他情况没有涵盖到，可以按照上述规则进行扩展。

## 2.2 Class 的命名

Class 的名字必须由一个大写字母开头其他字母为小写的单词构成。 类名要用完整的单词，避免使用缩写词 (除非该缩写词被更广泛使用像 URL，HTML) 。ActionForm 类的命名：以有意义的拼音缩写开头后加 ActionForm Action 类的命名：以有意义的拼音缩写开头后加 Action

## 2.3 Class 变量的命名

变量名应选用易于记忆、简短富于描述并能够指出其用途的单词。单词不应以下划线或美元符号开头， 除一次性临时变量外尽量避免单个字符的变量名。 除了变量名外，所有实例，包括类、类常量均采用大小写混合的方式，第一个单词的首字母小写， 其后单词的首字母大写。char c （c、d、e 常用于表示字符型） int I （i、j、k、m、n 常用于表示整型）float myWidth; 实例变量 (Instance Variables) 大小写规则和变量名相似，但前面需要一个下划线 int \_employeeId; String \_name; Customer \_customer;常量 (Constants) 类常量和 ANSI 常量的声明， 应该全部大写， 单词间用下划线隔开。 (尽量避免 ANSI 常量，容易引起错误 ) static final int MIN\_WIDTH = 4; static final int MAX\_WIDTH = 999; static final int GET\_THE\_CPU = 1; 项目管理系统程序编码规范 V1.0 5 除了以下几个特例之外，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。

1、尽量使用完整的英文描述符

2、采用适用于相关领域的术语

3、采用大小写混合使名字可读

4、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

5、避免使用长的名字（小于 15 个字母是个好主意）

6、避免使用类似的名字，或者仅仅是大小写不同的名字

7、避免使用下划线（除表态常量等）

## 2.4 接口的命名

书写规则与类名相似，如 interface RasterDelegate；interface Storing ；方法 (Methods) ：方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写，例如：

run(); runFast(); getBackground();

## 2.5 参数的命名

参数的命名必须与变量的命名规范一致。

## 2.6 数组的命名

数组应该用下面的方式来命名：

byte[] buffer; 而不是byte buffer[];

项目管理系统程序编码规范 V1.0 6

## 2.7 方法的参数

使用有意义的参数来命名，名字尽量和将要赋值的字段一致。例如：

SetCounter(int size){

this.size = size;

}

## 三 注释规范

一般情况下程序有两类注释：实现注释 (implementation comments) 和文档注释

(document comments)。实现注释是那些在 C++中见过的，使用 /\*...\*/ 和// 界定的注释。文

档注释 (称为 "doc comments") 是 Java 独有的，并由 /\*\*...\*/ 界定。文档注释可以通过

javadoc 工具转换成 HTML文件。

实现注释用来注释代码或者实现细节。文档注释从实现自由 (implementation-free)的角度描述代码的规范。 它可以使手头没有源码的开发人员读懂程序。 注释是用来对代码进行的，并提供代码自身没有提供的附加信息。 注释要包含有助于阅读和理解程序的相关信息。

例如，相应的包如何被建立或位于哪个目录下之类的信息不应包括在注释中。程序可以有 4 种实现注释的风格：块 (block) 、单行 (single-line) 、尾端 (trailing)和行末 (end-of-line) 。项目管理系统程序编码规范 V1.0 7

## 3.1 块注释 (Block Comments)

块注释通常是对文件、方法、数据结构和算法等提供描述，应置于每个文件的开始处或每个方法之前。 块注释也可以用于其他地方， 如方法的内部。 在功能和方法内部的块注释应该和它所描述的代码具有一样的缩进格式。块注释之首应该有一个空行，用于把块注释和代码分割开来，比如：

/\* \* 在这里写块注释

\*/

## 3.2 单行注释 (Single-Line Comments)

单行注释之前也应该有一个空行。若为可以显示在一行内短注释，要与其后的代码具 有一样的缩进层级。如果一个注释不能在一行内写完，就该采用块注释 (参见 "块注释 ")的方 式。以下是一个 Java代码中单行注释的例子：

if (condition) {

/\* 判断条件 \*/

...

}

## 3.3 尾端注释 (Trailing Comments)

极短的注释可以与它们所要描述的代码位于同一行，但是应该有足够的空白将代码和注释分开。若有多个短注释出现在大段代码中，它们应该具有相同的缩进。以下是一个 Java 代码中尾端注释的例子：

if (a == 2) {

return TRUE; /\* special case \*/

} else {

return isPrime(a); /\* works only for odd a \*/

项目管理系统程序编码规范 V1.0 8

}

## 3.4 行末注释 (End-Of-Line Comments)

利用注释界定符 "//" ，可以注释掉整行或者一行中的一部分。 它一般不用于连续多行的注释文本中，常用来注释掉连续多行的代码段。以下是所有三种风格的例子：

if (foo > 1) {

// Do a double-flip.

...

}

else {

return false; // Explain why here.

}

//if (bar > 1) {

//

// // Do a triple-flip.

// ...

//}

//else {

// return false;

//}

## 四 Java 文件样式

所有的 Java(\*.java) 文件都必须遵守如下的样式规则

## 4.1 版权信息

版权信息必须在 java 文件的开头，比如：

/\*\* 版权所有 ? \*\*\*\* 年-\*\*\*\* 年 ×××××科技有限公司，

\*保留所有权利。

项目管理系统程序编码规范 V1.0 9 \*/

其他不需要出现在 javadoc 的信息也可以包含在这里。

## 4.2 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 com.aoch.ecity. ⋯⋯ ;

import java.io.\*;

import java.util.Observable;

import hotlava.util.Application;

这里 java.io.\* 使用来代替 InputStream and OutputStream 的。

## 4.3 Class

类的注释一般是用来解释类功能的，需要按照如下格式编写：

/\*\* \* <p> 类名 : Jpc</p> \* <p> 描述 : Jpc 服务的用户接口类，调用了 JpcService 类的实现。

\* </p> \* <p> 版权声明 : Copyright (c) 2003-2004 ， \* <a href="http://www. 公司网址 ">××××科技有限公司 </a> </p> \* \* @author <a href=" 作者邮箱 ">作者姓名 </a>

\* @version 1.5.6

\*/

项目管理系统程序编码规范 V1.0

10

## 4.4 Class Fields

类的成员变量必须通过注释说明其含义， 不管成员变量是 public、protected 还是 private

的都要说明。

## 4.5 存取方法

若类的存取方法只是用于对类的变量赋值、取值的话，在类的成员变量有注释的情况下，类变量的存取方法可以没有注释。

/\*\* \* 功能说明： ⋯ \* @param opcData 要设置的 Opc 数据 \* @return true 成功 false 失败 \* @throws ServiceUnavailableException 服务不可用异常

\*/

public static boolean setDataToOpc(OpcData opcData) throws ServiceUnavailableException {

return true;

}

public int[] getPackets(String s) { return copyArray(packets, offset); }

public int[] getBytes() { return copyArray(bytes, offset); }

public int[] getPackets() { return packets; }

public void setPackets(int[] packets) { this.packets = packets; }

## 4.6 构造函数

构造函数，要用递增的方式书写（参数多的写在后面） ，这样可以使方法和参数更易读。项目管理系统程序编码规范 V1.0 11

## 4.7 main 方法

如果已经定义了 main(String[]) 方法，那么它就应该写在类的底部。

## 4.8 换行 (Wrapping Lines)

当一个表达式无法容纳在一行内时，可以依据如下一般规则断开：

在一个逗号后面断开在一个操作符前面断开宁可选择较高级别 (higher-level) 的断开，而非较低级别 (lower-level) 的断开新的一行应该与上一行同一级别表达式的开头处对齐 - 如果以上规则导致你的代码混乱或者使代码都堆挤在右边，那就代之以缩进 8 个空格。

以下是断开方法调用的一些例子：

someMethod(longExpression1,longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

someMethod2(longExpression2,

longExpression3));

以下是两个断开算术表达式的例子。前者更好，因为断开处位于括号表达式的外边，这是个较高级别的断开。

longName1 = longName2 \* (longName3 + longName4 - longName5)

+ 4 \* longname6; //PREFFER

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6; //A VOID

以下是两个缩进方法声明的例子。前者是常规情形。后者若使用常规的缩进方式将会使第二行和第三行移得很靠右，所以代之以缩进 8 个空格

//CONVENTIONAL INDENTATION

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

项目管理系统程序编码规范 V1.0 12

}

//INDENT 8 SPACES TO A VOID VERY DEEP INDENTS

private static synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

if 语句的换行通常使用 8 个空格的规则， 因为常规缩进 (4 个空格 )会使语句体看起来比较费劲。比如：

//DON ’T USE THIS INDENTATION

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) { //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

}

//USE THIS INDENTA TION INSTEAD

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

//OR USE THIS

if ((condition1 && condition2) || (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

这里有三种可行的方法用于处理三元运算表达式：

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta : gamma;

项目管理系统程序编码规范 V1.0 13

alpha = (aLongBooleanExpression)

? beta

: gamma;

## 4.9 行长度 (Line Length)

尽量避免一行的长度超过 80 个字符，因为很多终端和工具不能很好处理之。

注意：用于文档中的例子应该使用更短的行长，长度一般不超过 70 个字符。

## 4.10 语句 (Statements)

简单语句 (Simple Statements)

每行至多包含一条语句，例如：

argv++; // Correct

argc--; // Correct

argv++; argc--; // A VOID!

复合语句 (Compound Statements)

复合语句是包含在大括号中的语句序列，形如 "{ 语句 }" ，例如下面各段：

- 被括其中的语句应该较之复合语句缩进一个层次

- 左大括号 "{" 应位于复合语句起始行的行尾； 右大括号 "}" 应另起一行并与复合语句首行对齐。

- 大括号可以被用于所有语句，包括单个语句，只要这些语句是诸如 if-else 或 for控制结构的一部分。这样便于添加语句而无需担心由于忘了加括号而引入 bug。项目管理系统程序编码规范 V1.0 14 返回语句 (return Statements) 一个带返回值的 return 语句不使用小括号 "()" ，除非它们以某种方式使返回值更为显见。例如：

return;

return myDisk.size();

return (size ? size : defaultSize);

if，if-else，if else-if else语句 (if, if-else, if else-if else Statements)

if-else 语句应该具有如下格式：

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

项目管理系统程序编码规范 V1.0 15

statements;

} else{

statements;

}

注意： if 语句总是用 "{" 和"}" 括起来，避免使用如下容易引起错误的格式：

if (condition) //AVOID! THIS OMITS THE BRACES {}!

statement;

for 语句 (for Statements)

一个 for 语句应该具有如下格式：

for (initialization; condition; update) {

statements;

}

一个空的 for 语句 (所有工作都在初始化，条件判断，更新子句中完成）应该具有如下

格式：

for (initialization; condition; update);

当在 for 语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在 for 循环之前 (为初始化子句 )或 for 循环末尾 (为更新子句 )使用单独的语句。

while 语句 (while Statements)

一个 while 语句应该具有如下格式，项目管理系统程序编码规范 V1.0 16

while (condition) {

statements;

}

一个空的 while 语句应该具有如下格式：

while (condition);

do-while 语句 (do-while Statements)

一个 do-while 语句应该具有如下格式：

do {

statements;

} while (condition);

switch 语句 (switch Statements)

一个 switch 语句应该具有如下格式：

switch (condition) {

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

项目管理系统程序编码规范 V1.0 17

case XYZ:

statements;

break;

default:

statements;

break;

}

每当一个 case顺着往下执行时 (因为没有 break 语句)，通常应在 break 语句的位置添加注释。

上面的示例代码中就包含注释 /\* falls through \*/ 。try-catch 语句 (try-catch Statements) 一个 try-catch 语句应该具有如下格式：

try {

statements;

} catch (ExceptionClass e) {

statements;

}

一个 try-catch 语句后面也可能跟着一个 finally 语句，不论 try 代码块是否顺利执行完，它都会被执行。

try {

项目管理系统程序编码规范 V1.0 18

statements;

} catch (ExceptionClass e) {

statements;

} finally {

statements;

}

# 五 排版格式

1. 关键词和操作符之间加适当的空格。

2. 相对独立的程序块与块之间加空行。

3. 较长的语句、表达式等要分成多行书写。

4. 划分出的新行要进行适应的缩进，使排版整齐，语句可读。

5. 长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

6. 循环、判断等语句中若有较长的表达式或语句，则要进行适应的划分。

7. 若函数或过程中的参数较长，则要进行适当的划分。

8. 不允许把多个短语句写在一行中，即一行只写一条语句。

9. 函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格。

10. 用大括号‘ {’和‘ }’界定一段程序块的，编写程序块时‘ {’和 ‘}’应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及 if、for、do、while、 switch、case 语句中的程序都要采用如上的缩进方式。项目管理系统程序编码规范 V1.0 19

# 六 JSP 编码规范

1. 整个 jsp/jsp bean 表示层应当尽可能的瘦和简单化。

2. 牢记大多数的 JSP都应当是只读的视图，而由页面 bean 来提供模型。

3. 应当一起设计 JSP和 JSP bean

4. 在尽可能合理的情况下， 把业务逻辑从 JSP中移走。具体于 HTTP 的逻辑（如，对Cookie 的处理）属于 bean 或支持类中，而不是 JSP中。

5. 尽量把条件逻辑放在控制器中而不是放在视图中。

6. 为 JSP、包含的文件、 JSP Bean和实现扩展标记的类使用遵循标准的命名惯例。 如：jsp 控制器 xxxxController.jsp 被包含的 : jsp \_descriptiveNameOfFragment.jsp ，jsp会话bena: xxxxSessionBean 标记类 : xxxxTag,xxxxTagExtraInfo

7. 应当避免设计既显示表单又处理结果的页面。

8. 在 jsp 中避免代码重复。把要重复的功能放在一个包含的 jsp bean或标记扩展中，使得它能够被重用。

9. jsp bean 应当永远不要去产生 HTML

10. 在 jsp 中应该避免使用 out.pringln() 发放来产生页面内容。

11. jsp 层不应该直接访问数据，这包括 JDBC 数据库访问和 EJB 访问。

12. 在长度上，代码片的代码最好不要超过 10 行。

13. 除了 jsp bean 之外， jsp 不应当去实例化复杂的可读写的对象。如果这样的话，就有可能在 jsp 中去执行不适当的业务逻辑。

14. jsp bean 中不应当包含大量的数据。

15. 如果使用了 <jsp:forward> 和<jsp:include 标记 >，并且必须使用简单类型的值来与外部页面进行通讯的话，就应当使用一个或多个 <jsp:param>元素。

16. 定制标记应当用在适合把逻辑从 jsp 中移走的地方。

17. 应当谨慎地使用 <jsp:forward> 标记，在 jsp 中它是一个等价的 goto。

18. 应当使用隐藏的注释来阻止输出的 HTML 过大。

19. 在 jsp 中避免进行异常处理。

项目管理系统程序编码规范 V1.0 20

20. 每个 jsp 文件中都应当使用一个错误页面来处理不能够从中恢复的异常。

21. 只有在能够获得性能上的好处时，才使用 jspInit() 方法和 jspDestroy() 方法。获取和放弃资源是 jsp beans和标记处理器的事，而不是由 jsp 来负责的。

22. 如果没有充分的理由，就不要在 jsp 中定义方法和内部类。

# 七、文件与目录

1. 所有编译好的 java 类要打包成 jar 格式，统一放在应用目录的 WEB-INF/lib 目录下，

而不能把 class文件直接放在 WEB-INF/classes 目录下。

2. 系统中需要采用的第三方的类库也应放在 WEB-INF/lib 目录下，统一打包。

3. 所有图片应该放在应用目录的 images 目录下， 项目和产品的图片件放在 images 下

的一级子目录中，如 CMS 的图片文件可以放在 images/CMS 。

4. Struts 的配置文件必须放在 WEB-INF/config 目录下，每个产品应该有一个或多个

struts 的配置文件。

5. 产品自定义的 struts 配置文件的命名应该以 “struts-config ”+产品的名称 +某块名称

+“.xml”，struts-config-CMS-pub.xml

6. Spirng 的配置文件必须放在 WEB-INF/ 目录下。

7. 产品自定义的 sping 配置文件的命名应该以“ applicationContext ”+产品的名称 +某块名称 +“.xml”，applicationContext-CMS-pub.xml

# 八、日志规范

采 用 log4j 作 为 日 志 模 块 的 标 准 框 架 ， 版 本 采 用 1.2.8 ， log4j 的 配 置 文 件 为WEB-INF/classes/ log4j.properties ，配置方式采用 log4j 的标准配置文件方式，如：

# For JBoss: Avoid to setup log4j outside $JBOSS\_HOME/server/default/deploy/log4j.xml 项目管理系统程序编码规范 V1.0 21

# For all other servers: Comment out the Log4J listener in web.xml too

#log4j.rootCategory=INFO,logfile,console

log4j.rootCategory=INFO,console

#log4j.rootCategory=ERROR

log4j.appender.logfile=org.apache.log4j.RollingFileAppender

log4j.appender.logfile.File=c:/ezframework.log

log4j.appender.logfile.MaxFileSize=512KB

# Keep three backup files

log4j.appender.logfile.MaxBackupIndex=3

log4j.appender.logfile.layout=org.apache.log4j.PatternLayout

#Pattern to output: date priority [category] - <message>line\_separator

log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - <%m>%n

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d %p [%c] - <%m>%n

log4j 的参数配置参考 log4j 官方文档。

# 九、设计及技巧建议

系统主要的层次是表现层、 业务层和持久层， 整体应采用 taglib+struts+spring+hibernate架构，各层：项目管理系统程序编码规范 V1.0 22

## 9.1 表现层

表现层主要是用 taglib 来实现的， 数据来源是业务层， 不要直接使用持久层获取数据，查询数据需要组装查询 VO(\*\*\*QueryVO) 按模块来命名包名例如 com.xxxxxx.xxx.repair, 每个模块内的包分别是：actions，按功能分别存放 Action forms, 存放各个功能分别用到的 formbean models 存放各种类型的 list 所用到的 DataModel

## 9.2 业务层

业务层为表现层提供接口（用统一的工厂来管理这些接口） ，并且对常用数据做缓存。总的业务层包名是： com. xxxxxx.xxx. business, 按模块分别命名也存放接口和实现

# 9.3 持久层

持久层主要功能是为业务层提供数据，从数据库获取数据，也要承担少部分的业务逻辑，并且对实体和 VO 之间进行转换（不要把实体带到业务层甚至表现层）;尽量不直接使用各种数据库的功能，例如存储过程、约束等，用代码和配置文件来完成这些功能，防止跨数据库出现兼容问题持久层包名是： com.××××××.xxx.persistence，下面是 Dao(存放 Dao 接口和实现 ),entity（按不同功能分别存放要用到的数据库实体和影射文件） ,services（持久层服务接口和实现，以及管理这些接口的工厂） ，vo(根据业务组装 VO，VO 要尽量简单，例如针对有父子关系的实体可以只保存父类实体的 id)