编码规范

总页数		正文		附　录		生效日期：　　年　　月　　日
编制：			审核：			批准：

文件修改控制

修改记录编号	修改状态	修改页码及条款	修改人	审核人	批准人	修改日期

目　录

第一章 概述

1、规范制定原则

1. 方便代码的交流和维护。
2. 不影响编码的效率，不与大众习惯冲突。
3. 使代码更美观、阅读更方便。
4. 使代码的逻辑更清晰、更易于理解。

2、术语定义

2.1、Pascal 大小写

将标识符的首字母和后面连接的每个单词的首字母都大写。可以对三字符或更多字符的标识符使用 Pascal 大小写。

例：BackColor

2.2、Camel 大小写

标识符的首字母小写，而每个后面连接的单词的首字母都大写。

例：backColor

3、文件

3.1、文件命名

1. 文件名遵从 Pascal 命名法，无特殊情况，扩展名小写。

   例：MyFirst.java

2. 使用统一而又通用的文件扩展名。

   例：Java 类（.java）、XML 文件（.xml）

3.2、文件注释

1. 在每个文件头必须包含以下注释说明

```
/*
 *  Copyright (C)  福州同创志恒网络科技有限公司  版权所有。
 *  文件名：
 *  文件功能描述：
 *  创建标识：
 *
 *  修改标识：
 *  修改描述：
 *
 *  修改标识：
 *  修改描述：
 *  ……
 */
```

2. 文件功能描述只需简述，具体详情在类的注释中描述。
3. 创建标识和修改标识由创建或修改人员的拼音或英文名加日期组成。

   例：周翔 20040408
4. 一天内有多个修改的只需做一个在注释说明中做一个修改标识就够了。
5. 在所有的代码修改处加上修改标识的注释。

3.3、文件长度

文件长度 2000 行以内

第二章 代码外观

1、列宽

代码列宽控制在 80 字符左右。即每行 80 字符

2、换行

当表达式超出或即将超出规定的列宽，遵循以下规则进行换行

1. 在逗号后换行。
2. 在操作符前换行。
3. 规则 1 优先于规则 2。

当以上规则会导致代码混乱的时候自己采取更灵活的换行规则。

3、空行

空行是为了将逻辑上相关联的代码分块，以便提高代码的可阅读性。

1. 在以下情况下使用两个空行
    1) 接口和类的定义之间。
    2) 枚举和类的定义之间。
    3) 类与类的定义之间。
2. 在以下情况下使用一个空行
    1) 方法与方法、属性与属性之间。
    2) 方法中变量声明与语句之间。
    3) 方法与方法之间。
    4) 方法中不同的逻辑块之间。
    5) 方法中的返回语句与其他的语句之间。
    6) 属性与方法、属性与字段、方法与字段之间。
    7) 注释与它注释的语句间不空行，但与其他的语句间空一行。

4、空格

在以下情况中要使用到空格

1. 关键字和左括符“(”应该用空格隔开。如：while (true)

    注意：在方法名和左括符“(”之间不要使用空格，这样有助于辨认代码中的方法调用与关键字。

2. 多个参数用逗号隔开，每个逗号后都应加一个空格。
3. 除了“.”之外，所有的二元操作符都应用空格与它们的操作数隔开。

    例：

    a += c + d;

```
a = (a + b) / (c * d);
```

4. 一元操作符、++及--与操作数间不需要空格。

例：

```
while (d++ = s++) {
    n++;
}
```

5. 语句中的表达式之间用空格隔开。例：for (expr1; expr2; expr3)

5、括号-( )

1. 左括号“(”不要紧靠关键字，中间用一个空格隔开。

例：if (condition)

2. 左括号“(”与方法名之间不要添加任何空格。

例：Arrays.sort(nums)

3. 没有必要的话不要在返回语句中使用()。

例：return 1;

6、花括号-{ }

1. 左花括号“{”放于关键字或方法名的同行，并用一个空格间隔。

例：

```
if (condition) {
}
public int Add(int x, int y) {
}
```

2. 左花括号“{”要与相应的右花括号“}”同列对齐。

3. if、while、do 语句后一定要使用{}，即使{}号中为空或只有一条语句。

例：

```
if (somevalue == 1) {
    somevalue = 2;
}
```

4. 右花括号“}”后建议加一个注释以便于方便的找到与之相应的“{”。

例：

```
while (condition1) {
    if (condition2) {
    } // if condition2
    else {
    } // not condition2
} // end condition1
```

第三章 程序注释

1、注释概述

1. 修改代码时，总是使代码周围的注释保持最新。
2. 在每个例程的开始，提供标准的注释样本以指示例程的用途、假设和限制很有帮助。注释样本应该是解释它为什么存在和可以做什么的简短介绍。
3. 避免在代码行的末尾添加注释；行尾注释使代码更难阅读。不过在批注变量声明时，行尾注释是合适的；在这种情况下，将所有行尾注释在公共制表位处对齐。

   例：

   ```
   // 打印方法
   print(“”);
   int num;  // 设置变量
   ```

4. 避免杂乱的注释，如一整行星号。而是应该使用空白将注释同代码分开。
5. 避免在块注释的周围加上印刷框。这样看起来可能很漂亮，但是难于维护。
6. 在部署发布之前，移除所有临时或无关的注释，避免在日后的维护工作中产生混乱。
7. 如果需要用注释来解释复杂的代码节，请检查此代码以确定是否应该重写它。尽一切可能不注释难以理解的代码，而应该重写它。尽管一般不应该为了使代码更简单以便于人们使用而牺牲性能，但必须保持性能和可维护性之间的平衡。
8. 在编写注释时使用完整的句子。注释应该阐明代码，而不应该增加多义性。
9. 在编写代码时就注释，因为以后很可能没有时间这样做。另外，如果有机会复查已编写的代码，在今天看来很明显的东西六周以后或许就不明显了。
10. 避免多余的或不适当的注释，如幽默的不主要的备注。
11. 使用注释来解释代码的意图。它们不应作为代码的联机翻译。
12. 注释代码中不十分明显的任何内容。
13. 为了防止问题反复出现，对错误修复和解决方法代码总是使用注释，尤其是在团队环境中。
14. 对由循环和逻辑分支组成的代码使用注释。这些是帮助源代码读者的主要方面。
15. 在整个应用程序中，使用具有一致的标点和结构的统一样式来构造注释。
16. 用空白将注释同注释分隔符分开。在没有颜色提示的情况下查看注释时，这样做会使注释很明显且容易被找到。
17. 所有的代码修改处加上修改标识的注释。

2、文档型注释

该类注释采用 JavaDoc 来标记，在声明接口、类、方法、属性、字段都应该使用该类注

释，以便代码完成后直接生成代码文档，让别人更好的了解代码的实现和接口。

例：

```
/**
 * 功能描述：
 * 创建标识：
 *
 * 修改标识：
 * 修改描述：
 * ……
 * @param 参数名称 参数类型 参数说明
 * @return 返回值类型 返回值说明
 * @throws 异常类型
 */
public static void main(String args[]) throws Exception {
}
```

3、块级别注释

1. 包围代码块

   例：

   ```
   /*------ start:订单处理 ------*/
   //Do something here
   /*-------- end:订单处理 -------*/
   ```

2. 可以考虑使用大括号来表示注释范围

   例：

   ```
    /*----------订单处理 ------- */
   {
       //Do something here
   }
   ```

4、单行注释

该类注释用于：

1. 方法内的代码注释。如变量的声明、代码或代码段的解释。

   例：

   ```
   // 注释语句
   private int number;
   ```

2. 方法内变量的声明或花括号后的注释。

   例：

   ```
   if (1 == 1) {    // always true
       //Do something here
   }    // always true
   ```

3. 如果做过修改需加上修改者和日期
   例：
   String username; //修改者 修改日期 说明

5、备注

建议注释：

1. 循环语句和判断语句前必须注释。
2. 特殊变量声明时需要注释。
3. 如果方法允许 Null 作为参数，或者允许返回值为 Null，必须在 JavaDoc 中说明。
4. 注释中的第一个句子要以（英文）句号、问号或者感叹号结束。Javadoc 生成工具会将注释中的第一个句子放在方法汇总表和索引中。
5. 注释不是用来管理代码版本的，如果有代码不要了，直接删除，svn 会有记录的，不要注释掉，否则以后没人知道那段注释掉的代码该不该删除。

第四章 声明

1、每行声明数

一行只建议作一个声明，并按字母顺序排列。

例：

```
int level;        //推荐
int size;         //推荐
int x, y;         //不推荐
```

2、初始化

建议在变量声明时就对其做初始化。

3、位置

变量建议置于块的开始处，不要总是在第一次使用它们的地方做声明。

例：

```
void MyMethod() {
    int int1 = 0;              // 在方法块开始
    if (condition) {
        int int2 = 0;          // 在 if 语句块开始
        //Do something here
    }
}
```

不过也有一个例外

```
for (int i = 0; i < maxLoops; i++) {
    //Do something here
}
```

应避免不同层次间的变量重名，

例：

```
int count;        // 已声明
void MyMethod() {
    if (condition) {
        int count = 0;  // 重复声明，会覆盖，应避免
    }
}
```

4、类和接口的声明

1. 在方法名与其后的左括号间没有任何空格。
2. 左花括号“{”出现在声明的同行，并用一个空格间隔。
3. 方法间用一个空行隔开。

5、字段的声明

不要使用是 public 或 protected 的实例字段。如果避免将字段直接公开给开发人员，可

以更轻松地对类进行版本控制，原因是在维护二进制兼容性时字段不能被更改为属性。考虑为字段提供 get 和 set 属性访问器，而不是使它们成为公共的。get 和 set 属性访问器中可执行代码的存在使得可以进行后续改进，如在使用属性或者得到属性更改通知时根据需要创建对象。

例：

```
public class Control {
    private int handle;
    public int getHandle() {
        return this.handle;
    }
    public void setHandle(int handle) {
        this.handle = handle;
    }
}
```

6、包的导入

删除不用的导入，尽量不要使用整个包的导入。经常使用快捷键 ctrl+shift+o 修正导入。

第五章 命名规范

1、命名概述

名称应该说明“什么”而不是“如何”。通过避免使用公开基础实现（它们会发生改变）的名称，可以保留简化复杂性的抽象层。

例如，可以使用 getNextStudent()，而不是 getNextArrayElement()。

命名原则采用英语单词为主，遇到无法用英语单词的地方，可以采用拼音全拼方式，具体规则如下：

1. 选择正确名称时的困难可能表明需要进一步分析或定义项的目的。
2. 使名称足够长以便有一定的意义，并且足够短以避免冗长。
3. 唯一名称在编程上仅用于将各项区分开。
4. 表现力强的名称是为了帮助人们阅读；因此，提供人们可以理解的名称是有意义的。
5. 请确保选择的名称符合适用语言的规则和标准。

以下几点是推荐的命名方法。

1. 避免容易被主观解释的难懂的名称，如方法名 analyzeThis()，或者字段名 xxK8。这样的名称会导致多义性。
2. 在类字段的名称中包含类名是多余的，如 Book.bookTitle。而是应该使用 Book.title。
3. 只要合适，在变量名的末尾或开头加计算限定符（Avg、Sum、Min、Max、Index）。
4. 在变量名中使用互补对，如 min/max、begin/end 和 open/close。
5. 布尔变量名应该包含 is，这意味着 Yes/No 或 True/False 值，如 fileIsFound。
6. 在命名状态变量时，避免使用诸如 Flag 的术语。状态变量不同于布尔变量的地方是它可以具有两个以上的可能值。不是使用 documentFlag，而是使用更具描述性的名称，如 documentFormatType。（此项只供参考）
7. 即使对于可能仅出现在几个代码行中的生存期很短的变量，仍然使用有意义的名称。仅对于短循环索引使用单字母变量名，如 i 或 j。可能的情况下，尽量不要使用原义数字或原义字符串，

   例：

   ```
   for (int i = 0; i < 7; i++) {
   }
   ```

   应该使用命名常数，以便于维护

   ```
   for (int i = 0; i < NUM_DAYS_IN_WEEK; i++) {
   }
   ```

2、大小写规则

下表汇总了大小写规则，并提供了不同类型的标识符的示例。

标识符	大小写	示例
类	Pascal	AppDomain
枚举类型	Pascal	ErrorLevel
枚举值	Pascal	FatalError
异常类	Pascal	WebException 注意：总是以 Exception 后缀结尾。
接口	Pascal	IDisposable 注意：总是以 I 前缀开始。
常量		RED_VALUE 注意：全部大写，多个单词间用下划线分割
包		org.tcedu.entity 注意：用域名的颠倒作为开头，且全部小写
公共实例字段	Camel	redValue
受保护的实例字段	Camel	redValue
私有的实例字段	Camel	redValue
属性	Camel	backColor
方法	Camel	toString()
参数	Camel	typeName
方法内的变量	Camel	backColor

3、缩写

为了避免混淆和保证跨语言交互操作，请遵循有关区缩写的使用的下列规则：

1. 不要将缩写或缩略形式用作标识符名称的组成部分。例如，使用 GetWindow，而不要使用 GetWin。
2. 不要使用计算机领域中未被普遍接受的缩写。
3. 在适当的时候，使用众所周知的缩写替换冗长的词组名称。

   例：用 UI 作为 User Interface 缩写，用 OLAP 作为 On-line Analytical Processing 的缩写。
4. 使用缩写时，对于超过两个字符长度的缩写请使用 Pascal 大小写或 Camel 大小写。

   例：使用 HtmlButton 或 HTMLButton。

但是，应当大写仅有两个字符的缩写，

例：UI，而不是 Ui。

5. 不要在标识符或参数名称中使用缩写。如果必须使用缩写，对于由多于两个字符所组成的缩写请使用 Camel 大小写，虽然这和单词的标准缩写相冲突。

4、包（package）

1. 命名包时的一般性规则是业务领域名.子系统名.层名

   例：
   org.tcedu.dao.weibo

2. 包名全部小写，用“.”分隔开。
3. technology 指的是该项目的英文缩写，或软件名。
4. 包和类不能使用同样的名字。

5、类（class）

1. 用名词或名词短语命名类。使用 Pascal 大小写。
2. 使用全称避免缩写，除非缩写已是一种公认的约定，如 URL、HTML
3. 不要使用类型前缀，如在类名称上对类使用 C 前缀。

   例：使用类名称 FileStream，而不是 CFileStream。

4. 不要使用下划线字符“_”。
5. 有时候需要提供以字母 I 开始的类名称，虽然该类不是接口。只要 I 是作为类名称组成部分的整个单词的第一个字母，这便是适当的。例如，类名称 IdentityStore 是适当的。
6. 在适当的地方，使用复合单词命名派生的类。派生类名称的第二个部分应当是基类的名称。

   例：ApplicationException 对于从名为 Exception 的类派生的类是适当的名称，原因 ApplicationException 是一种 Exception。

7. 匿名内部类 20 行以内 ，太长的匿名内部类影响代码可读性，建议重构为命名的（普通）内部类。
8. 类名往往用不同的后缀表达额外的意思

   例：

后缀名	意义	举例
Service	表明这个类是个服务类，里面包含了给其他类提同业务服务的方法	PaymentOrderService
Impl	这个类是一个实现类，而不是接口	PaymentOrderServiceImpl

后缀名	意义	举例
Dao	这个类封装了数据访问方法	PaymentOrderDao
Action	直接处理页面请求，管理页面逻辑了类	UpdateOrderListAction
Listener	响应某种事件的类	PaymentSuccessListener
Event	这个类代表了某种事件	PaymentSuccessEvent
Servlet	一个 Servlet	PaymentCallbackServlet
Factory	生成某种对象工厂的类	PaymentOrderFactory
Adapter	用来连接某种以前不被支持的对象的类	DatabaseLogAdapter
Job	某种按时间运行的任务	PaymentOrderCancelJob
Wrapper	这是一个包装类，为了给某个类提供没有的能力	SelectableOrderListWrapper
Bean	这是一个 POJO	MenuStateBean

6、接口（interface）

以下规则概述接口的命名指南：

1. 用名词或名词短语，或者描述行为的形容词命名接口。使用 Pascal 大小写。

   例：

   - 接口名称 IComponent 使用描述性名词。
   - 接口名称 ICustomAttributeProvider 使用名词短语。
   - 接口名称 IPersistable 使用形容词。

2. 少用缩写。
3. 给接口名称加上字母 I 前缀，以指示该类型为接口。在定义类/接口对（其中类是接口的标准实现）时使用相似的名称。两个名称的区别应该只是接口名称上有字母 I 前缀。
4. 不要使用下划线字符“_”。
5. 当类是接口的标准执行时，定义这一对类/接口组合就要使用相似的名称。两个名称的不同之处只是接口名前有一个 I 前缀。

   例：
   public interface IServiceProvider
   public interface IFormatable

7、枚举（enum）

以下规则概述枚举的命名指南：

1. 对于 enum 类型，使用 Pascal 大小写。
2. 少用缩写。

3. 不要在 enum 类型名称上使用 Enum 后缀。
4. 对大多数 Enum 类型使用单数名称，但是对作为位域的 enum 类型使用复数名称。
5. 全大写，用下划线分割

例：

```
public enum Events {
    ORDER_PAID,
    ORDER_CREATED
}
```

8、常量（const）

以下规则概述常量的命名指南：

所有单词大写，多个单词之间用“_”隔开。

例： public static final string PAGE_TITLE = "Welcome";

9、变量（variate）

以下规则概述变量的命名指南

1. 使用 Camel 大小写。
2. 定义局部变量尽量在那段代码的开始处，如方法的开始处。

例：

```
public void show() {
    String username;
    //Do something here
}
```

3. 如果是 if，for，while 代码块，尽量在左花括号“{”的下一行处定义要使用的局部变量。

例：

```
if (condition) {
    String username;
    //Do something here
}
```

例：

序号	变量名称	注　释
1	strFileName	“文件名”字符串类型
2	intFileCount	“文件总数”整型
3	strFames	多个“文件名”的集合
4	gMemory	全局变量

10、字段（field）

以下规则概述字段的命名指南：

1. 使用 Camel 大小写。
2. 拼写出字段名称中使用的所有单词。仅在开发人员一般都能理解时使用缩写。
3. 不要对字段名使用匈牙利语表示法。好的名称描述语义，而非类型。
4. 在类定义的开始，按照 public，protected，package，private 顺序放置。
5. 不要对字段名或静态字段名应用前缀。具体说来，不要对字段名称应用前缀来区分静态和非静态字段。

11、静态字段（static field）

以下规则概述静态字段的命名指南：

1. 使用名词、名词短语或者名词的缩写命名静态字段。 使用 Camel 大小写。
2. 对静态字段名称使用匈牙利语表示法前缀。
3. 建议尽可能使用静态属性而不是公共静态字段。

12、属性（sproperty）

以下规则概述属性的命名指南：

1. 使用名词或名词短语命名属性。 使用 Camel 大小写。
2. 考虑用与属性的基础类型相同的名称创建属性。

   例：如果声明名为 Color 的属性，则属性的类型同样应该是 Color。以下代码示例阐释正确的属性命名。

```
public class SampleClass {
    public Color getBackColor() {
        // Code for Get and Set accessors goes here.
    }
}
```

以下代码示例阐释提供其名称与类型相同的属性。

```
public enum Color {
    // Insert code for Enum here.
}
public class Control {
    public Color getColor() {
    }
    public void setColor(Color color) {
    }
}
```

以下代码示例不正确，原因是 Color 属性是 Integer 类型的。

```
public class Control {
    public int getColor() {
    }
}
```

14、方法（method）

以下规则概述方法的命名指南：

1. 使用动词或动词短语命名方法。“动作+属性”的方法。并且，动作以小写字母开始，属性以大写字母开始。常用的动作有：is、get、set、add、 update、del 等。

   例：getName、setName、isSysManager、saveXXX、mdfXXX、delXXX 等。

2. 使用 Camel 大小写。
3. 遇到缩写如 XML 时，仅首字母大写。

   例：loadXmlDocument()而不是 loadXMLDocument()

4. 为了基于接口编程，不采用首字母为 I 或加上 IF 后缀的命名方式，

   例：IBookDao,BookDaoIF。

5. 方法长度 150 行以内

   例：

前缀名	意义	举例
create	创建	createOrder()
delete	删除	deleteOrder()
add	创建，暗示新创建的对象属于某个集合	addPaidOrder()
remove	删除	removeOrder()
init 或 initialize	初始化，暗示会做些诸如获取资源等特殊动作	initializeObjectPool
destroy	销毁，暗示会做些诸如释放资源的特殊动作	destroyObjectPool
open	打开	openConnection()
close	关闭	closeConnection()<
read	读取	readUserName()
write	写入	writeUserName()
get	获得	getName()
set	设置	setName()
prepare	准备	prepareOrderList()
copy	复制	copyCustomerList()
modity	修改	modifyActualTotalAmount()
calculate	数值计算	calculateCommission()
前缀名	意义	举例

do	执行某个过程或流程	doOrderCancelJob()
dispatch	判断程序流程转向	dispatchUserRequest()
start	开始	startOrderProcessing()
stop	结束	stopOrderProcessing()
send	发送某个消息或事件	sendOrderPaidMessage()
receive	接受消息或时间	receiveOrderPaidMessgae()
respond	响应用户动作	responseOrderListItemClicked()
find	查找对象	findNewSupplier()
update	更新对象	updateCommission()

15、参数（parameter）

以下规则概述参数的命名指南：

1. 使用描述性参数名称。参数名称应当具有足够的描述性，以便参数的名称及其类型可用于在大多数情况下确定它的含义。使用 Camel 大小写。
2. 使用描述参数的含义的名称，而不要使用描述参数的类型的名称。开发工具将提供有关参数的类型的有意义的信息。因此，通过描述意义，可以更好地使用参数的名称。少用基于类型的参数名称，仅在适合使用它们的地方使用它们。
3. 不要使用保留的参数。保留的参数是专用参数，如果需要，可以在未来的版本中公开它们。相反，如果在类库的未来版本中需要更多的数据，请为方法添加新的重载。
4. 方法（构造器）参数在 5 个以内 ，太多的方法（构造器）参数影响代码可读性。考虑用值对象代替这些参数或重新设计。
5. 不要给参数名称加匈牙利语类型表示法的前缀。

   例：
   Type getType(string typeName)
   String format(String format, String[] args)

16、集合（collection）

集合是一组组合在一起的类似的类型化对象，如哈希表、查询、堆栈、字典和列表，集合的命名建议用复数，小写 s 结尾。

17、页面（page）

以下规则概述页面的命名指南：

1. 页面以页面功能命名。使用 Camel 大小写。

   例：

   - userInfoList：用户列表页面
   - userInfoDetail：用户详细页面

- userInfoEdit：用户编辑页面

2. 页面部件名建议命名为：btnOK、lblName。其中 btn、lbl 缩写代表按钮（Button）、标签（Label）

18、关键字（keyword）

避免使用和以下关键字冲突的标识符。

AddHandler	AddressOf	Alias	And	Ansi	instanceof
As	Assembly	Auto	Base	Boolean	package
ByRef	Byte	ByVal	Call	Case	var
Catch	CBool	CByte	Cchar	CDate	With
CDec	CDbl	Char	Cint	Class	extends
CLng	CObj	Const	Cshort	CSng	While
CStr	CType	Date	Decimal	Declare	Eval
Default	Delegate	Dim	Do	Double	When
Each	Else	ElseIf	End	Enum	Xor
Erase	Error	Event	Exit	ExternalSource	volatile
False	Finalize	Finally	Float	For	WriteOnly
Friend	Function	Get	GetType	Goto	To
Handles	If	Implements	Imports	In	Until
Inherits	Integer	Interface	Is	Let	WithEvents
Lib	Like	Long	Loop	Me	Throw
Mod	Module	MustInherit	MustOverride	MyBase	Unicode
MyClass	Namespace	New	Next	Not	Then
Nothing	NotInheritable	NotOverridable	Object	On	TypeOf
Option	Optional	Or	Overloads	Overridable	SyncLock
Overrides	ParamArray	Preserve	Private	Property	Try
Protected	Public	RaiseEvent	ReadOnly	ReDim	Sub
Region	REM	RemoveHandler	Resume	Return	True
Select	Set	Shadows	Shared	Short	Structure
Single	Static	Step	Stop	String	

第六章 语句

1、每行一个语句

每行最多包含一个语句。

例：

```
a++;      //推荐
b--;      //推荐
a++; b--; //不推荐
```

2、复合语句

复合语句是指包含“父语句{子语句; 子语句;}”的语句，使用复合语句应遵循以下几点

1. 子语句要缩进。
2. 左花括号“{”与复合语句父语句同行，并用一个空格间隔。
3. 即使只有一条子语句要不要省略花括号“{}”。

例：

```
while (d == s) {
n++;
}
```

3、return 语句

return 语句中不使用括号，除非它能使返回值更加清晰。

例：

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

4、if、if-else、if-else if-else 语句

if、if-else、if else-if 语句使用格式

例：

- if 语句

```
if (condition) {
    //Do something here
}
```

- if-else 语句

```
if (condition) {
    //Do something here
} else {
    //Do something here
}
```

- if-else if-else 语句

```
if (condition) {
    //Do something here
} else if (condition) {
    //Do something here
} else {
    //Do something here
}
```

注意：

1. if 语句的嵌套层数 3 层以内
2. 不要再对 boolean 值做 true false 判断

例：

```
if (order.isPaid() == true) {
    // Do something here
}
```

不如写成：

```
if (order.isPaid()) {
    //Do something here
}
```

5、while、do-while 语句

1. while 语句使用格式

例：

```
while (condition) {
    //Do something here
}
```

2. do-while 语句使用格式

例：

```
do {
    //Do something here
} while (condition);
```

6、for、foreach 语句

1. for 语句使用格式，

例：

```
for (initialization; condition; update) {
    //Do something here
}
```

2. foreach 语句使用格式

例：

```
for (object obj : array) {
    //Do something here
}
```

注意:

1) 在循环过程中不要修改循环计数器。

2) 对每个空循环体给出确认性注释。

7、switch-case 语句

switch-case 语句使用格式

例：

```
switch (condition) {
    case 1:
        //Do something here
        break;
    case 2:
        //Do something here
        break;
    default:
        //Do something here
        break;
}
```

注意：

1) 语句 switch 中的每个 case 各占一行。

2) 语句 switch 中的 case 按字母顺序排列。

3) 为所有 switch 语句提供 default 分支。

4) 所有的非空 case 语句必须用 break; 语句结束。

8、异常处理语句

1. try-catch 语句使用格式

例：

```
try {
    //Do something here
} catch (ExceptionClass e) {
    //Do something here
} finally {
    //Do something here
}
```

如果属于正常异常的空异常处理块必须注释说明原因，否则不允许空的 catch 块

2. throw 语句

重新抛出的异常必须保留原来的异常，

例：

throw new NewException("message", e);

而不能写成 throw new NewException("message")。

9、方法体

尽量不要用参数来带回方法运算结果

例：

```
public void calculate(Order order) {
    int result = 0;
    //do lots of computing and store it in the result
    order.setResult(result);
}
public void action() {
    order = orderDao.findOrder();
    calculate(order);
    // do lots of things about order
}
```

例子中 calculate 方法通过传入的 order 对象来存储结果， 不如如下写：

```
public int calculate(Order order) {
    int result = 0;
    //do lots of computing and store it in the result
    return result;
}
public void action() {
    order = orderDao.findOrder();
    order.setResult(calculate(order));
    // do lots of things about order
}
```

第七章 框架

1、目录结构

1. 实体层：entity
2. 数据访问层：dao、dao.impl
3. 业务逻辑层：biz、biz.impl
4. Servlet：web
5. Struts2：action
6. 自定义标签：tag
7. 后台页面：root/admin
8. 前台页面：root 根目录下
9. javaScript：js
10. 样式表：css
11. 图片，flash：images

2、struts.xml

1. package 的 name 和 namespace 属性使用 Pascal 大小写。
2. action 的 name 属性使用 Camel 大小写。
3. 后台操作，每个增删改查最多 2 个页面，分为为列表页面和编辑页面（新增和修改）
4. package 和 action 的命名必须符合该模块的功能
5. result 的 name 尽量与 action 中的 method 一致，可以使用通配符。

例：

```
<package name="UserInfo" namespace="/UserInfo" extends="struts-default">
    <action name="userInfo_*"class="org.tcedu.action.UserInfoAction"
method="{1}">                    <result name="list">/admin/userInfo/userInfo_list.jsp</result>
        <result name="edit">/admin/userInfo/userInfo_edit.jsp</result>
        <result name="delete">/admin/userInfo/userInfo_delete.jsp</result>
        <result name="load">/admin/userInfo/userInfo_load.jsp</result>
    </action>
</package>
```

或

```
<package name="UserInfo" namespace="/UserInfo" extends="struts-default">
    <action name="userInfo_*"class="org.tcedu.action.UserInfoAction"
method="{1}">                    <result
name="{1}" >/admin/userInfo/userInfo_{1}.jsp</result>
    </action>
</package>
```