# 百度JAVA编码规范

## 目录

# 0 前言

编码风格没有太多的好坏之分，最重要的是风格保持一致，编码规范有助于规范我们编码的风格，使代码具有更好的可读性。

Java 在百度内部已经有多个项目组在使用，但是却一直没有统一的编码规范支持，编码风格百家齐放，不利于我们代码的维护和传承， 根据大家平时的开发情况，制定了此 java 编码规范。

每项规范前面的(强制) 代表该规范需要强制执行，(建议)代表推荐执行但不强制。

Java 编码规范制定人:

王靓 姜安琦 王珺 徐鑫 高友峰 张建勋刘泽胤 王继平

# 1 代码书写

**1-1：程序块要采用K&R代码风格编写，缩进的空格数建议为 4 个。【建议】**

说明：不同的缩进风格对代码的可读性影响很大，以 tab 为缩进单位在不同的 tab step 下可读性也相差很多，所以将缩进定为一个 soft tab 即 4 个空格，这样在所有环境下缩进都会保持一致。

**1-2：一行程序以小于 120 字符为宜【建议】，超长的语句应该在一个操作符之前折行，并在下一行加入适当的空格进行缩进；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之：【规则】**

a) 在一个逗号后面断开

如：

```
someMethod(longExpression1, longExpression2, longExpression3,
                longExpression4, longExpression5);     //','后换行
```

b) 在一个操作符后面断开

如：

```
longName1 = longName2 * (longName3 + longName4 - longName5) +
                4 * longname6;        //运算符号后换行
```

c) 新的一行应该与上一行同一级别表达式的开头处对齐

d) 如果以上规则导致你的代码混乱或者使你的代码都堆挤在右边，那就代之以缩进若干空格。

说明：这样可读性更好，逻辑更一目了然。

**1-3：if、while、for、do语句的执行体总是用"{"和"}"括起来，即使单条语句也是。并且在较长（超过一屏）的判断或者循环语句的结尾应该有注释语句做出标识。【规则】**

```
if (a > b) {
    temp = a;
    //……
 }//if (a >b)结束
```

**1-4：每行仅包含一条语句。语法意义相互独立的代码将用空行分隔。【规则】。**

说明：这样做可读性更好。

如：

```
a++;    b++;        //!!! 避免使用

if (a > b) {
    value = a;   //行间不用空行间隔
}

int ret = value;    //与上一代码行使用空行间隔
```

# 2 命名

2-1：Package 的名字应该都是由一个小写单词组成。【规则】

如：

```
package com.baidu.rigel.service;
```

2-2：类(class)命名规则：类名是个一名词，采用大小写混合的方式，每个单词的首字母大写。使用完整单词，避免缩写词(除非该缩写词被更广泛使用，像URL，HTML)。【规则】

如：

```
public class BaseDaoSupport {
      //……
}
```

2-3：参数和变量的名字必须用一个小写字母开头，后面的单词用大写字母开头【规则】,此外建议变量的名字最后带有表示该变量类型的后缀。

如：

```
List customerDataList;
```

2-4：Final 变量和ENUM变量的名字应该都大写，并且指出完整含义,每个单词之间使用“_”分割。【规则】

```
final String CUSTOMER_STATUS_REFUSE;
```

2-5：各种标识符的命名要使用有实际意义的英文单词或英文单词的缩写，切忌使用阿拉伯数字进行命名。【建议】

# 3 注释

**3-1：要求类、接口、方法都必须添加注释【规则】，实现逻辑、判断或者循环语句、全局变量、关键的局部变量等可以视重要程度添加注释。【建议】**

类、接口的注释，说明该类或者接口的主要作用。

如：

```
/**
* 字符串处理类
* @author Baidu R&D Center
* @version 1.0
*/
public class StringUtils {
    ……
}
```

**3-2：方法注释采用标准的 javadoc 注释规范，注释中必须提供方法说明，参数说明和返回值说明。【规则】**

如：

```
/**
* The doGet method of the servlet.
* This method is called when a form has its tag value method
* equals to get.
* @param request
* the request send by the client to the server
* @param response
* the response send by the server to the client
* @throws ServletException
* if an error occurred
* @throws IOException
* if an error occurred
*/
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
doPost(request, response);
}
```

**3-3：修改代码时候必须仔细阅读和修改附近的代码注释；如果在函数前有函数体注释，必须更新函数注释。【规则】**

注释规范遵守 javadoc 注释规范，更多请参见: http://java.sun.com/j2se/javadoc/index.jsp

# 4 常量和变量

**4-1：具有全局性逻辑功能的常量值需要定义为常量，而不要将数值直接出现在代码中。此外程序中多处使用到的常量一定要在公共文件中进行定义，并且必须使用该常量进行赋值。【规则】**

如:

```
public static final int MAX_VALUE = 100;
……
System.out.println(MAX_VALUE);
```

**4-2：一行一个声明，避免在一个语句中给多个变量赋值。【规则】**

如:

```
int indexBegin = indexEnd = 0;    //!!!避免使用
indexBegin = 5; indexEnd = 10;    //!!!避免使用
```

**4-3：只在代码块的开始处声明变量，内外层的代码声明不要重名。【规则】**

如:

```
 void myMethod(){
 int int1 = 0;         // 函数开头声明函数内使用的局部变量

     if (condition){
         int int2 = 0;    // 代码块开头声明代码块内使用的局部变量，且不得与外层声明的变量同名
         ……
      }
}
```

**4-4：数组定义方法如下： 数组元素类型紧跟着”[]”,其后是数组标识。【规则】**

如:

```
byte[] buffer = null; //提倡的使用
byte buffer[]  = null; //!避免使用
```

# 5 函数

**5-1：函数划分原则【规则】**

a) 按照逻辑功能合理划分函数。
b) 避免重复代码。在多个地方需要重复使用的代码，使用独立的函数来封装。

**5-2：尽量控制函数总长度小于 500 行。当一个函数代码超过 500 行，一般将其中相对独立的功能使用单独的函数来实现。【建议】**

**5-3：设计代码时，尽量减少函数入口、出口参数的数量，以及各个参数所代表含义的复杂度。【建议】**

**5-4：判断函数入口参数的合法性，特别是 public 和 protected 的函数，一定要检查入口参数是否合法。【规则】**

如:

```
public static String getStringWithLen(String str, int len) {
    if((str != null) && (str.length() > len)){
        str = str.substring(0, len);
    }
        return str;
}
```

**5-5：使用一个由其他函数返回的对象之前，先判断是否为 null，以免产生 NullPointerException 异常**

如:

```
//不规范写法:
User user = getUser();
System.out.print(user.getName());


//应写为:
User user = getUser();
if(user == null) {
   //提示错误
   ……
 }
System.out.print(user.getName());
```

**5-6：不要在函数中对函数的参数进行赋值。【规则】**

说明：private String getCustomerIdsFromFormattedArray(Object object) ，不要在函数体中使用“object =”之类的赋值语句，在函数返回后，参数的值实际上不会改变。

**5-7：只公布必须公布的方法，尽量使方法私有化。【建议】**

说明：只公布自己必须公布的，就可放心大胆地改变其他任何东西。如果修改一个没有必要公开的方法，就可能破坏其他人现有的代码，使他们不得不重新编写和设计。

# 6 编码原则

**6-1：尽量使用接口而不是一个具体的类。【建议】**

例如有以下需求：给定一个 SQL 语句，返回一个对象的列表，实现中用 java.util.ArrayList 实现，于是定义方法为：

```
public java.util.ArrayList getObjectItems(String sql){
    ……
}
```

上面的方法存在一个问题，当 getObjectItems 内改用 Vector 或 LinkedList 实现，外部类必须做相应更改。一个更好的方法是定义返回值为 java.util.AbstractList 更合适：

```
public java.util.AbstractList getObjectItems(String sql){
    ……
}
```

这样即使更改实现，外部类也不必做相应更改。

**6-2：对于可能发生异常的代码，都应在尽可能低的层次进行捕获。若低层不知如何处理，则将异常继续抛给高层处理，但最终必须得到处理。【规则】**

**6-3：防止下标越界，在使用数组下标之前先判断下标的合法性，比如数组类型，集合类型。【规则】**

如：

```
// 以下为不规范的写法：
String[] ary = getAry();
System.out.print(ary[2]);


// 应写为：
String[] ary = getAry();
    if ((ary != null) && (ary.length > 2)){
        System.out.print(ary[2]);
}
```

**6-4：谨防数据类型越界。【规则】**

如：

```
int milsec = 25 * 86400 * 1000;
System.out.println(milsec);
// 打印结果是-2134967296 （超过 int 最大值范围 2147483647）
```

**6-5：确保释放数据库连接资源。【规则】**

说明：使用完 Connection 后注意 close(),如果使用连接池则使用后应及时归还给连接池，防止数据库连接资源被耗尽。其它在 JDBC 中还需要检查是否关闭的有：ResultSet、Statement。

**6-6：确保释放网络连接、文件资源和 runtime 调用的 process。【规则】**

说明：使用完 Socket 后注意 close(),如果使用连接池则使用后应及时归还给连接池，及时关闭打开的文件，防止超出允许打开的最大文件句柄(可以用 ulimit -n 查看和改变)。

**6-7：循环编程：尽量选取 for 循环而不是 while 循环；如果使用 while 循环，最好能用 iterator；while 循环的终止条件需慎重，以避免死循环的可能。【建议】**

如:

```
/** 不正确的用法：当interval取零或者负数时将可能出现死循环* */
while (i <= max) {
    i += interval;
}
```

**6-8：字符串比较时，将常量置于 equals 之前，避免 null 异常。【建议】**

**6-9：一般而言，在含有多种运算符的表达式中使用圆括号来避免运算符优先级问题，是个好方法。即使运算符的优先级对你而言可能很清楚，但对其他人未必如此。你不能假设别的程序员和你一样清楚运算符的优先级。【建议】**

如:

```
if (a == b && c == d)   //!!!避免使用
if ((a == b) && (c == d))  //正确的使用方法
```

**6-10：对于简单的 if-else 语句，可以使用简洁的三元运算符“？ :”。【建议】**

如:

```
if (condition) {
    return x;
}
return y;
```

可以写做：

```
return (condition ? x : y);
```

# 附录 JAVA安全编码规范

## 基本出发点

目前 web 系统面临的主要安全问题在于如下两点：

1． web 页面的 sql 注入攻击。

2．利用文件上传功能上传后门文件的攻击。

基于这两点，在 rd 开发时需要遵守的原则如下：

1．为了防止 sql 输入的攻击，需要避免直接根据页面的输入参数拼接 sql 的情况出现，推荐使用 PreparedStatement 进行 sql 的查询；并且需要对输入的参数进行特殊字符的过滤。

2．为了防止利用文件上传功能上传后门文件进行攻击，在设计阶段涉及到对上传文件的处理时，需要考虑对文件的扩展名进行判断，对文件的内容进行判断，并将上传的文件按照一定规则改名并放到其它目录下保存。同时满足这三个条件以便彻底避免用户上传后门程序。

## 编码方面

编码方面的原则主要是为了防止用户的 sql 注入引发的攻击，需要遵守的原则和代码样例如下：

1 为了避免用户通过输入参数对系统进行 SQL 注入的攻击，对页面的输入必须对参数进行合法性验证。对于数字和字符串，需要分别进行参数的类型转换，并过滤特殊字符。可以根据使用框架的不同选择下面描述的任何一个样例参考开发：

1） 如果使用 struts1.x 框架，通常我们通过 struts form 封装前台返回的数据。此时可以借助 struts 本身提供的 validation 机制，对前台数据进行验证，包括对数据类型，数据格式，数据长度等进行限制。此时可以将 FormBean 的类型定义为 org.apache.struts.validator.DynaValidatorForm ， 并同时在 struts-config.xml 与 validation.xml 配置文件中添加对应内容即可。以下为一个 validation.xml 片段样本：

```
<form name="addNormalManagerForm">
  <field property="managerName" depends="required,maxlength ">
    <arg0     key="manager.info.managername"/>
    <arg1     key="${var:maxlength}" name="maxlength" resource="false"/>
    <var>
      <var-name>maxlength</var-name>
      <var-value>24</var-value>
    </var>
  </field>
```

```
<!—这里省略对其他属性的验证 -->
</form>
```

Struts1.1 validation.xml 片段样本

2） 如果使用 struts2.x 框架，验证相比显得更加方便，在 validators.xml 可以定制工程中需要引入的多种数据的验证方式（验证机制的实现有 struts2 提供），包括：必填项、数据类型、email 地址、正则表达式、日期数据格式等。当需要为不同的 yourAction（your 为 Action 的名字）定制化验证时，仅需要在与 yourAction 文件同一目录下放置 yourAction--validation.xml 文件即可。以下分别给出了 validators.xm 文件片段与 yourAction--validation.xml 文件片段：

```
<validators>
    <validator name="required"
class="com.opensymphony.xwork2.validator.validators.RequiredFieldValidator"/>
    <validator name="int"
class="com.opensymphony.xwork2.validator.validators.IntRangeFieldValidator"/>
    <validator name="email"
class="com.opensymphony.xwork2.validator.validators.EmailValidator"/>
    <validator name="regex"
class="com.opensymphony.xwork2.validator.validators.RegexFieldValidator"/>
    <validator name="datestring"
class="com.baidu.rigel.util.web.validator.DateStringFieldValidator"/>
    <!—这里省略对其他验证的定义 -->
</validators>
```

Struts2.x validation.xml 片段样例

```
<validators>
    <!-- 公司名称,必填项,长度不能大于 200 -->
    <field name="cust.custName">
        <field-validator type="requiredstring" short-circuit="true">
            <message>${getText("errors.required",{'公司名称'})}</message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="trim">true</param>
            <param name="maxLength">200</param>
            <message>${getText("errors.stringlength",{'  公  司  名  称
```

```
','200'})}</message>
            </field-validator>
        </field>
<!—这里省略对其他属性的验证 -->
</ validators>
```

Struts2.x yourAction-validation.xml 片段样例

3） 针对 Java Web 应用，也可以直接通过在后台应用入口处通过编码验证数据的正确性，例如在 SERVLET 的 doGet、doPost 或者 Struts-Action 的 execute 方法内，对用户的输入进行验证，例如在下面的 execute 方法中，对请求参数的有效性进行了检查，包括“非空验证”，与“类型转化验证”：

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {
        String callidValue = request.getParameter(CALLID); // 电话 ID
        String eventidValue = request.getParameter(EVENTID); // 事件 ID
        //省略其他参数的获取
        Long eventid = StringUtils.isNullOrSpace(eventidValue) == true ? null: new
Long(eventidValue);
        Long callid = StringUtils.isNullOrSpace(callidValue) == true ? null: new
Long(callidValue);
        //省略其他参数的处理过程
}
```

通过编码完成的输入数据验证样例

2 为了避免 SQL 注入攻击，必须限制所有 Database 操作，使用 SQL 进行数据库查询时，需要使用[com.mysql.jdbc.PreparedStatement]进行参数绑定，原则上不允许直接拼接 SQL 进行数据库查询。如果使用 Hibernate 接口进行数据库操作时，我们通常用 HQL 代替 SQL，同时为了避免参数注入，同样需要通过参数绑定的方式，代替 HQL 的拼接。请参考下面的样例：

```
public void deleteSelectedManagers(Long skillGroupId,Long[] managerIds){
        if(managerIds!=null&&managerIds.length>0){
        List<SkillAccountRange>
  skillAccountRanges=getHibernateTemplate().find("from SkillAccountRange sar where
  sar.skillAccount.skillGroup.skillGroupId=? and sar.managerid in ?",new
  Object[]{skillGroupId, managerIds});
    }
```

```
}
```

HQL 参数绑定样本[SQL 类似]

另外可以通过对用户输入的合理限制，避免恶意输入导致的 SQL 注入问题，例如采用以下黑名单处理方法：

```
<!-- 在配置文件中增加以下“查询条件”的黑名单内容： →
        <!--  SQL INJECTION AVOID//which must be split by "," -->
        <sql_query_avoid>',;,and,(,),update,count,*,%</sql_query_avoid>
//相应地，通过以下代码片段就可以检查输入的“查询条件”是否包含黑名单中内容。

public static boolean isQueryStringSafe(String queryString){
          boolean result = true;
          if(!StringUtils.isNullOrSpace(queryString)){//判断非空
// SysProp.SQL_QUERY_AVOID 就是配置文件中定义的内容
              String[] dangerSA =
           SysProp.SQL_QUERY_AVOID.split(CCConstants.COMMON_SPLIT_STRING);
              for(String dString : dangerSA){
                  if(queryString.indexOf(dString) > -1){//query contains danger String,forbidded
                      result = false;
                      break;
                  }
              }
          }
          return result;
        }
public ActionForward query(ActionMapping mapping, ActionForm form,
              HttpServletRequest request, HttpServletResponse response) {
        if(!StringUtils.isNullOrSpace(queryEventForm.getQueryvalue())
```

```
                && !isQueryStringSafe(queryEventForm.getQueryvalue())){
        String failMessage = "Invalid Query Conditon: "+queryEventForm.getQueryvalue();
        saveErrors(request, WebUtil.errorForward(failMessage, 1));
        return mapping.findForward(ERROR_FORWARD);
      }
```

黑名单参数过滤样本

**设计方面**

1 在 web 的动态页面中，避免出现对系统命令的调用的情况。

2 为了避免用户上传后门程序到服务器上，用户上传文件的文件必须遵循三个步骤：1）文件上传后，根据文件头判断文件类型是否符合预期；2）文件上传后，需要对文件的扩展名进行判断，判断扩展名是否符合预期；3）文件上传后，需要对文件名按照一定规则重新命名并移至其他目录下。其中，步骤 1 可以根据实现情况酌情考虑是否实现，步骤 2、3 则是必须实现。

3 对于安全性要求较高的页面，尽量避免 get 请求，而选择使用 POST 请求，这样能够避免因数据长度，CSRF 漏洞等引发的安全问题。

4 在使用第三方程序、框架、工具之前，需要首先调研该程序是否存在安全性隐患，并明确需要使用的版本。目前已经确认存在安全隐患，并对 ECOM 产品线造成影响的第三方工具包括：旧版本 FCKEDITOR 2.3（包括）以前存在漏洞，请采用 2.4（包括）以后的版本。

4.1 在某模块中使用的 FCKEDITOR 文件上传组件版本较早，其中存在安全漏洞，加上配置不当，没有对允许上传的文件名进行限制，因此导致了文件上传的漏洞。此时可以采用升级 FCKEDITOR 的版本，或者采用设置文件扩展名的白名单可以避免问题(*修改对应的文件上传 SERVLET 配置文件，增加文件类型的白名单，限制上传文件的类型。)*。

5 重要信息的保存是否选用合适的加密算法。例如，以下给出了 Bad Case 与 Good Case：

| [BAD CASE] 页面中的超链接中参数并未进行加密，此时重要参数，（例如[eventide]）就 |
|---|

| |
|---|
| 容易暴露给使用者。<br>http://[host]:[port]/event_detail.jsp?**eventid=57073**&commentenabled=0 |
| [GOOD CASE] 当后台数据传递至前台时，使用 DES 加密，就可以将重要信息（例如[qmTaskId]）隐藏，当数据从前台提交至后台时再使用 DES 解密即可。<br>http://[host]:[port]/qm/qmTaskDetail.jsp?**qmTaskId=61BE5F817DC24C42** |

参数传递 Bad Case & Good Case

6 通讯时考虑是否选用安全的通讯方式，包括：

6.1 例如，使用 HTTPS

6.2 发送端，接收端，双方约定的加密方法，对传输数据进行加密。请求 The 3rd party 服务时采用双方约定的方法对数据加密

```
//请求第三方 服务样本
String seed = "RandomSeed";
String key = seed+"CrmT&IWeb";
MD5 md5 = new MD5();
md5.Update(key);
md5pwd = MD5.asHex( md5.Final());
http://[host]:[port]/baidu.htm?managerid=" + managerId.toString() + "&seed=" + seed + "&pwd=" + md5pwd;
```

传输数据加密样例

**安全开发的其它原则**

1 对所有从客户端传入的数据不信任：包括通信协议中从客户端传过来的一切变量，无论是客户手动填写的数据或是客户端浏览器或操作系统自动填写的数据；以及从数据库、文件、网络等，一些不直接来源于用户，但是又不是程序中定义好的常量数据。比如用户的输入经过层层转化输出到数据库或文件，后面又再次利用的时候，这时变量依然是不可信任的，但往往容易让程序员放松警惕。

2 最小化原则：用户输入最小化：尽可能少地使用用户的输入；用户输入范围的最小化：过滤参数的时候尽量使用白名单策略，对于可以明确定义范围的参数要检查参数的有效性。

3 安全的编码不依赖任何安全配置：即在编写程序的时候不能有侥幸心理，不能寄希望于配置文件的安全选项；必须将自身的程序置身最不安全的配置下进行考虑。

4 程序错误信息对外界透明：对程序的错误和异常进行专门的日志记录，避免直接输出给用户。特别对于与文件或数据库相关的操作，错误信息会对攻击者带来非常大的帮助。