J2EE B/S 应用软件开发技术指南

（讨论稿）

二〇一七年十月

目　　录

1 前言

目前 Java 作为最完整的开发生态，广泛应用于企业管理开发、互联网开发、大数据开发、安卓开发等领域，相关的框架、解决方案层出不穷。相比于其他编程语言，比如 C、C++、C#、Python、Ruby 等，Java 具有以下几个方面的优势：

1、简单易学

首先 Java 是一个面向对象的编程语言，容易理解，语法简单，略去了重载、指针等难以理解的概念，实现了自动垃圾回收，大大简化了程序设计。其次 Java 学习资料较多，从 CSDN 或其他在线网站中就可以找到很多 Java 的学习资料，另外也有很多培训机构提供免费的 Java 视频课程。

2、跨平台

跨平台是 Java 最大的优势。Java 运行在 JVM（Java 虚拟机）上，在任何平台只要安装了 JVM，就可以运行 Java 应用。JVM 部署在操作系统之上，屏蔽了底层的差异。

3、安全

Java 中没有指针，应用程序无法直接访问内存，不容易出现内存泄露等异常情况。

4、多线程

Java 内置对多线程的支持，可以通过多线程等技术来实现对系统资源的充分利用。

5、丰富类库支持

经过 10 多年的积累和沉淀，出现了很多优秀的开源社区，如 Apache 和 Spring。这些优秀的社区提供了很多框架，借助这些框架可以不用去关注 Java 底层的开发，而只需关注业务逻辑的实现。

6、应用广泛

Java 普遍应用于企业开发、互联网开发等领域，在做 Java 开发时如果遇到问题，可以很容易从网上找到解决方案。

目前铁路以 Java 作为开发语言的建设项目较多，研发人员也都具备基础的 Java 编程能力，因此需要制定相应的 Java 研发规范，约束实际项目开发。

2 开发规范制定说明

为全面提升铁路软件研发生产水平，整合共享技术资源，掌控核心技术，规范编码标准，达到技术统一、资源共享，特制定以下开发技术指南，供各软件研发单位参照执行。

该指南适用范围：企业内部 J2EE B/S 架构应用系统的研发。

3 技术规范

3.1 Java 开发规范

3.1.1 文件编码规范

【强制】所有 Java 代码、JS 代码、CSS 代码、HTML 代码必须以 UTF-8 作为标准编码格式，禁止采用 GBK 和 GB2312 等其他编码格式。

3.1.2 整体命名规范

1、 【强制】所有编程相关命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

反例：_name, $name, name_, name$

2、 【推荐】所有编程相关的命名不宜使用拼音与英文混合的方式，更不建议直接使用中文的方式（注：已在国际通用的名称可视为英文），鉴于铁路行业的特殊性，把业务系统分为强业务系统与弱业务系统，可根据实际情况选择命名方式。

（1）强业务系统，比如调度、货检等业务系统可以根据业务需要采用已被约定俗成的拼音缩写方式，比如行调（xd）。

（2）弱业务系统，比如办公、财务等系统必须采用国际通用的英文单词命名。

3、 【强制】禁止完全不规范的缩写，避免望文不知义，影响可读性。

反例：AbstractServlet 缩写为 AbsClass；condition 缩写为 condi。

4、 【强制】中括号是数组类型的一部分，数组定义如下：String[] args。

反例：禁止使用 String args[]的方式来定义。

3.1.3 VO/DTO/DO/PO/model 类型规范

1、 VO：View Object, Value Object 用于前端展示或查询条件等进行参数传递。

2、 DTO：Data Transfer Object 用于服务接口数据传输。

3、 DO：Domain Object 用于描述业务对象、处理业务逻辑，可用于数据持久化。

4、 POJO：Persistent Object 用于数据库持久化，一般用在 Hibernate 中

5、 Model：数据库字段与 Java 对象间的映射，各个变量对应数据库的字段。

示例：在页面向 Server 端进行传值时，server 端接收的对象可以是 VO 对象，比如查询条件封装。在 service 层及各方法进行字段组合传输时，如果传输字段是由多个 model 组装，宜采用 DTO。

【强制】在编写 Model 的变量时，采用驼峰写法，且字段要和数据库字段对应。

数据库字段为 created_by，则各实体类命名字段为 createdBy

正例：`private String createdBy;//创建人`

反例：`private String created_by;//创建人`

【强制】上述各类都要增加 toString()方法，以便于后续的错误追踪、日志记录等。

3.1.4 包命名规范

1、【强制】包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。

2、【强制】包名统一使用单数形式。

正例：com.critc.sys.user

反例：com.critc.sys.users

3、【推荐】常见包名宜采用通用的命名方式，比如 controller、dao、service、model、vo 等

3.1.5 类命名规范

1、【强制】类名使用 upperCamelCase 风格，命名必须遵从驼峰形式。

正例：XmlService, SysUserDao

反例：XMLService, sysUserDao,Sys_user_dao

2、【强制】抽象类名使用 Abstract 开头。

3、【强制】异常类名使用 Exception 结尾。

正例：GeneralException

反例：GeneralExcep

4、【强制】单元测试类名以 Test 开头，紧跟待测试的类的名称。

正例：TestSysUserService

反例：SysUserServiceTest

5、【强制】枚举类型使用 Enum 结尾。

正例：UserTypeEnum

反例：EnumUserType

6、【强制】类名一律采用单数形式，不用复数，比如用户对应资源。

正例：SysUserResource

反例：SysUserResources

7、【强制】接口类的实现类使用 Impl 结尾，以区别接口。

正例：SysUserServiceImpl

反例：SysUserServiceImp

3.1.6 方法命名规范

1、【强制】方法名使用 lowerCamelCase 风格，必须遵从驼峰形式。

正例：getHttpMessage()

反例：get_http_message()

2、【强制】方法名一律采用动词+名词，如果方法是对数据库对象的增删改，可以直接是动词，禁止使用名词作为方法名。

正例：listUser(),get(),getById()

反例：userList(),user(),get_by_id()

常用方法命名一览：

（1） 获取单个对象的方法用 get 做前缀

（2） 按条件获取单个对象的方法用 getBy 做前缀

（3） 获取多个对象的方法用 list 做前缀

（4） 按条件查询获取多个对象的方法用 listBy 做前缀

（5） 获取统计值的方法用 count 做前缀

（6） 插入的方法用 add（推荐）或 save 做前缀

（7） 删除的方法用 delete（推荐）或 remove 做前缀

（8） 修改的方法用 update 做前缀

（9） 导入数据以 import 做前缀

（10） 审核、修改状态等以 save 做前缀

（11） 导出数据以 export 做前缀

3、【强制】接口类中的方法禁止加任何修饰符，包括 public。

正例：int getById(int id)

反例：public int getById(int id)

3.1.7 成员变量、参数、局部变量命名规范

1、【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清晰。

正例：MAX_STOCK_COUNT

反例：MAX_COUNT

2、 【强制】成员变量、参数、局部变量名统一使用驼峰风格。

正例：int userClickCount

反例：int user_click_count

3、 【强制】类型为 boolen/Boolean 的变量，禁止添加 is/has 等前缀，防止语义错误。

正例：closed

反例：isClosed

4、 枚举类型的成员名称应遵从常量命名规范。

正例：USER_TYPE_MANAGER，USER_TYPE_NORMAL

反例：Uerr_type_manager,User_type_normal

3.1.8 警告类代码编写规范

1、 【强制】禁用使用过期方法（类或方法标注为@Deprecated），必须改为新的使用方式。

正例：URL.decode(String str,String chartset)

反例：URL.decode(String str)

2、 【强制】各容器类（List、ArrayList、Collection、Iterator、Entry、Map、HashMap），如果非泛型，必须标明容器内数据类型。

正例：List<String> list=new ArrayList<>();

反例：List list=new ArrayList();

3.1.9 Sql 编写规范

1、 【强制】insert、update、delete、search 语句各参数都要采用占位符，禁止直接拼写 sql，参数赋值通过程序来写，防止注入攻击。#安全#

正例：select username,password from t_sys_user where username=?

反例：select username,password from t_sys_user where username='test'

2、 【强制】select 查询语句的查询字段要按照实际使用来拼写，禁止直接 select *

正例：select username,mobile,realname from t_sys_user

反例：select * from t_sys_user

3、 【推荐】不宜在 sql 语句中放入过多业务逻辑，影响 sql 的整体可读性。比如放入过多的 case when 语句，放入过多的 decode 语句等。

正例：select status from t_user (status 的后续处理由 Java 程序来完成)

反例：select decode(status,1,'正常',2,'禁用') from t_user

3.1.10 注释规范

1、【强制】类注释：类注释必须标明该类的用途、编写人、编写时间，编写人用中文，编写时间格式为 yyyy-MM-dd。

例如：

```
/**
 * 说明：日志操作类，在系统中如果需要输入日志，调用该类的 info 方法
 * @author 孔垂云
 * @date 2017-05-23
 */
```

2、【强制】方法注释：必须标明方法用途（如果后续修改，必须写清楚原因）、输入参数含义、输出参数含义。

例如：

```
/**
 * 把对象转成 json 串
 *
 * @param obj 对象，可以是 VO、List、HashMap 等等
 * @return 返回生成的 json 值
 */
```

其中 get、set、toString 等方法不必写注释。

3、【强制】Model/vo/dto 类变量注释：必须标明各个字段的含义，例如：

```
private int id;//用户 id
private int roleId;//所属角色
private String roleName;//角色描述
private String username;//登录账号
private String password;//登录密码
private String randomcode;//随机数
private int status;//账号状态
private String realname;//姓名
private String mobile;//手机号
private String createdBy;//创建人
private Date createdAt;//创建时间
```

4、【强制】流程控制注释：在涉及状态转变，参数赋值等流程处理时，必须标明注释。

例如：

```
if(true){
    status=1;//操作正确，状态变为 1
}else{
    status=2;//操作失败，状态变为 2
}
```

5、【强制】准备实现但尚未实现的方法，必须添加 TODO 注释，并说明预计处理时间和处理人。

6、【推荐】关于待删除的接口或者公共类或则方法，必须添加@Deprecated 注解，并说明替换的类和方法、预计删除或变更时间与处理人。使用相关接口或者方法的开发人员，应及时处理变更。

3.1.11 日志处理规范

1、【强制】所有的日志输出都必须通过公共的方法来调用。业务系统所有日志输出直接调用 LogUtil.info()进行输出，不允许自定义 logger 进行输出。

2、【推荐】日志分为应用日志、系统日志、操作日志、异常日志等，各类都应分别输出各自内容，不允许混用。

3、【强制】禁止使用 System.out.println/System.err.println 输出日志。

3.1.12 异常处理规范

1、【强制】所有资源类对象必须在 finally 中提供关闭操作。例如：

```
  public static String readFile(String filePath) {
    StringBuilder sb = new StringBuilder("");
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(new InputStreamReader(new
FileInputStream(filePath), "UTF-8"));
        String tempString = null;
        // 一次读入一行，直到读入 null 为文件结束
        while ((tempString = reader.readLine()) != null) {
            sb.append(tempString).append("\r\n");
        }
        reader.close();
    } catch (IOException e) {
        LogUtil.error(e);
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e1) {
                LogUtil.error(e1);
            }
        }
    }
    return sb.toString();
}
```

2、【强制】所有异常的 catch 均须打印日志，不允许直接抛弃，反例：

```
try{
}
catch (IOException e) {}
```

3、【强制】不允许在 finally 中返回操作值

反例：

```
int a=0;
try{}
catch(Exception e)
{
LogUtil.error(e);
}
finally {
    return a;
}
```

4、【强制】不允许通过异常来处理业务逻辑

反例：新增数据，直接 insert，如果抛出异常，则提示数据存在。应该先执行查询，如果有数据，则提示数据存在。

5、【强制】service 层的操作，禁止对涉及数据库操作的方法进行 try、catch 操作，否则事务不起作用

反例：SysUserService

```
public int update(SysUser sysUser) {
        int flag = 0;
        try {
                flag = sysUserDao.update(sysUser);
        } catch (Exception e) {
                LogUtil.error(e);
        }
 return flag;
}
```

3.2 单元测试规范

1、【强制】好的单元测试必须遵守 AIR 原则。

说明：好的单元测试宏观上来说，具有自动化、独立性、可重复（AIR）执行的特点。

A：Automatic（自动化）

I：Independent（独立性）

R：Repeatable（可重复）

2、【强制】保持单元测试的独立性。为了保证单元测试稳定可靠且便于维护，单元测试用例之间决不能互相调用，也不能依赖执行的先后次序。

反例：method2 需要依赖 method1 的执行，将执行结果做为 method2 的输入。

3、【强制】单元测试是可以重复执行的，不能受到外界环境的影响。

说明：单元测试通常会被放到持续集成中，每次有代码 check in 时单元测试都会被执行。如果单测对外部环境（网络、服务、中间件等）有依赖，容易导致持续集成机制的不可用。

4、【强制】对于单元测试，要保证测试粒度足够小，有助于精确定位问题。单测粒度至多是类级别，一般是方法级别。

说明：只有测试粒度小才能在出错时尽快定位到出错位置。单测不负责检查跨类或者跨系统的交互逻辑，那是集成测试的领域。

5、【强制】核心业务、核心应用、核心模块的增量代码确保单元测试通过。

说明：新增代码及时补充单元测试，如果新增代码影响了原有单元测试，请及时修正

6、【强制】单元测试代码必须写在如下工程目录：src/test/java，不允许写在业务代码目录下。

说明：源码构建时会跳过此目录，而单元测试框架默认是扫描此目录。

7、【推荐】对于数据库相关的查询，更新，删除等操作，不能假设数据库里的数据是存在的，或者直接操作数据库把数据插入进去，请使用程序插入或者导入数据的方式来准备数据。

反例：删除某一行数据的单元测试，在数据库中，先直接手动增加一行作为删除目标，但是这一行新增数据并不符合业务插入规则，导致测试结果异常。

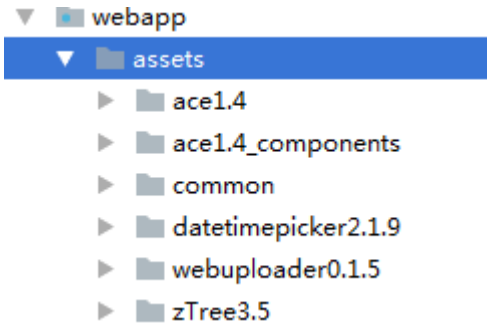
8、【推荐】和数据库相关的单元测试，可以设定自动回滚机制，不给数据库造成脏数据。或者对单元测试产生的数据有明确的前后缀标识。

3.3 UI 界面设计规范

3.3.1 静态资源存放路径规范

1、【推荐】系统所有静态资源，包括 JS、CSS、HTML、图片等，统一存在在 webapp/assets 下面，按照类型分别存储。应用的各种插件，都要标名版本号，便于后续进行动静分

离处理。



例如：ace1.4 存放 ace 框架的基本文件

ace1.4_componets 存放依赖的组件

datetimepicker2.19 存放日期框组件

3.3.2 静态资源引用规范

1、 【强制】静态资源的引用都要采用全路径方式，禁止采用相对路径。

正例：<script src="http://localhost:8080/test/assets/login.js"></script>

反例：<script src="../assets/login.js"></script>

3.3.3 系统避免 JS、CSS 缓存处理规范

1、 【强制】在引用自定义 JS、CSS 时，都要标明版本号，防止客户端 JS 缓存。

正例：<script src="http://localhost:8080/test/assets/login.js?version=1.0"></script>

上述标红部分由系统统一控制。

反例：<script src="http://localhost:8080/test/assets/login.js"></script>

3.3.4 JSP 路径规范

1、 【强制】系统所有 JSP 都统一存放在 webapp/WEB-INF/views 下面，禁止直接访问 JSP 路径。

3.3.5 浏览器兼容规范

1、 【推荐】BS 程序都要满足 IE9+、Chrome25+等内核的浏览器兼容测试。

2、 【推荐】系统要进行分辨率兼容性测试，最小分辨率不低于 1024*768，最大分辨率 1920×1200。

3.3.6 页面布局规范

1、 【推荐】在采用 js 表格 datagrid、html table 进行页面布局时，列宽要适中（根据显示内容设定显示宽度），列头显示文字要与采集页面显示文字保持一致，查询条件出现的文字标签要与显示（或采集）项文字标签保持一致，避免出现前后叫法不一、逻

辑矛盾等现象。

3.3.7 页面表单校验规范

1、【强制】表单元素在进行提交时，必须要进行前端校验，建议采用 jQuery-validator 等校验工具。

2、【强制】表单校验、操作提示时必须采用友好的交互方式，禁止使用浏览器原生提示，比如 alert。

3.3.8 页面元素及方法命名规范

1、【推荐】页面的元素分为文本框、下拉框、复选框、单选框、文本域、form、datagrid、dialog 等等，采用元素类型缩写+具体含义的命名原则。

- 文本框：txt
- 下拉框：cmb
- 复选框：chk
- 单选框：radio

3.3.9 url 传输中文规范

1、【强制】特殊情况下需要采用 url 进行中文传输，这时候必须对 url 进行 encode 操作

正例：http://localhost:8080/text/post.do?name=%E6%B5%8B%E8%AF%95

反例：http://localhost:8080/text/post.do?name=测试

3.4 数据库设计规范

3.4.1 总体规范

- 少用存储过程、函数、视图
- 不宜使用触发器
- 不宜使用数据链路
- 不宜使用同义词
- 禁止多库间的表进行关联查询，如果存在跨库查询，利用程序的多数据源进行处理
- 数据库表及字段都要标明对应的 comments 说明

3.4.2 表命名规范

1、【强制】表名长度不能超过 30 个字符，禁止采用中文，推荐采用“T_MODULENAME_TABLENAME”形式，MODULENAME 、TABLENAME 使用能体现表名意义的英文单词或英文单词缩写，

2、【强制】表名中含有单词全部采用单数形式，单词字母均使用大写，除“_”分隔符外，单词间不用任何连接符号。

正例：T_SYS_USER

反例：T_SYS_USERS,T_SYSUSER

3.4.3 字段命名规范

1、【强制】数据库字段名全部采用大写英文单词，单词之间用“_”隔开，尽量不要超过三个单词。

2、【强制】如果数据表有审计字段，包括以下四个，统一命名：created_at(创建时间)、created_by（创建人）、last_modified_at(最后修改时间)、last_modified_by（最后修改人）

3、【强制】字段值若为可列举类型，必须在 comments 中描述每个值对应的中文含义

3.4.4 字段类型长度规范

1、【强制】字段类型应采用标准类型，以 Oracle 数据库类型为例，字段类型采用标准的 VARCHAR2、NUMBER、DATE 等类型，不允许采用 DOUBLE、DECIMAL 等非标准类型。长度尽量满足使用要求，不允许过长。比如操作人字段，长度建议设为 10 即可。

2、【推荐】建议日期类型说明：一般统计日期都应采用字符型，操作时间、入库时间等需要精确计算的采用 DATE 类型。Timestamp 类型在有高精度时间计算时采用，其余情况尽量少用。

3、【推荐】文件、图片等二进制数据，一般不宜以 BLOB 形式存数据库，而应存在文件服务器上。

3.4.5 索引规范

1、【强制】索引一般是针对于数据表中的一个字段或多个字段所建立，采用“IDX_表名+序号”形式，例如：IDX_T_SYS_USER1。如果有多个索引，顺序排即可。

3.4.6 序列规范

1、【强制】其命名采用“SEQ_表名”形式，例如：SEQ_T_DEP。

3.4.7 存储过程、函数规范

1、为了便于程序的可读性，系统开发不宜采用存储过程及函数。

下述内容不要放进去

说明：

（1） 存储过程不便于阅读，不便于调试

（2） 存储过程无法采用版本管理工具记录历史

（3） 存储过程无法和 spring 更好的整合，因为 spring 的事务管理无法控制存储过程

（4） 存储过程无法采用多线程处理，所有处理压力完全依赖数据库，不能利用应用服务器的资源

（5） 现在的开发人员多数对存储过程不熟悉

（6） 未来的开发趋势是数据库只做数据存储读取用，业务处理由应用服务器来做。

3.4.8 视图规范

1、 不建议系统开发采用视图。

3.4.9 触发器规范

1、 不建议使用触发器，开发应用程序替代相关功能。

说明：触发器导致程序处理不可控，会产生意想不到的结果。

3.4.10 Job 规范

1、 不建议使用 Job，开发定时器程序替代相关功能。

说明：在不使用存储过程的前提先下，Job 也有意义。

3.4.11 同义词规范

1、 为提高程序可读性，同义词尽量少用，如果使用，一定要和普通表区分开来，比如以’SYN’开头。

3.5 文档规范

3.5.1 数据库文档规范

1、 【强制】需要记录版本号、修改时间、修改人。

2、 【强制】需要记录每次修改内容。

3、 【强制】需要记录所有数据表的名称、字段名、说明、类型、长度、是否为空，详细描述。

4、 【强制】需要标注对应的索引即索引字段。

5、 【强制】如果有其他说明，应详细标注。

6、 【强制】数据库结构的变化，要带上升级脚本。

7、 【推荐】绘制 ER 关系图，提高文档可读性。

具体例子参见附件：《数据库设计文档示例》

3.5.2 接口文档规范

1、【强制】需要记录版本号、修改时间、修改人

2、【强制】需要记录每次修改内容

3、【强制】需要标注接口使用方式

4、【强制】需要标注接口地址、参数列表、返回结果

5、【强制】需要描述数据示例

具体例子参见附件：《接口设计文档示例》