迅扬电子

<持续集成开发平台>
编程指南

版本 **<1.2>**

修订历史记录

日期	版本	说明	作者
2007-11-22	<1.0>	创建	蔡源
2008-09-27	<1.1>	统一样式，完善内容	蔡源
2008-10-21	<1.2>	完成基本查询页面及表单页面样式	蔡源

目录

修订历史记录 2
目录 3
简介 5
目的 5
适用人员 5
定义、首字母缩写词和缩略语 6
参考资料 6
概述 6
如何从需求到实现到交付 6
流程图 6
注意点 8
总体配置 9
Spring 9
Hibernate 9
Struts 9
ActionForm 9
POJO 9
DWR 9
代码编写 11
DAO 层 11
规范 11
Spring 配置图 11
注意点 11
Service 层 12
规范 12
Spring 配置图 12
注意点 13
Action 层 13
规范 13
Spring 配置图 14
Struts-config 配置图 14
注意点 14
JSP 15
规范 15
Extremetable 标签扩展 16
Javascript 16
规范 16
common.js 17
common-form.js 17
common-form-validator.js 17
CSS 17
规范 17
common.css 17

layout.css 18
print.css 18
externalLinks.css 18
reset.css 18
form.css 18
Widgets 18
LittTable 18
错误处理和异常事件 19
公用异常 19
DAO 19
Service 19
Action 19
核心组件包使用 19
日志处理 20
其他 21
附录：指南概要 22

简介

目的

- ✓ 指导开发人员熟悉系统框架
- ✓ 运用示例快速进入开发角色

适用人员

本文档的适用人员包括 J2EE 项目开发组全体成员：技术管理人员、系统设计人员、系统开发人员、系统测试人员、系统维护人员、推广培训人员及其他相关人员。

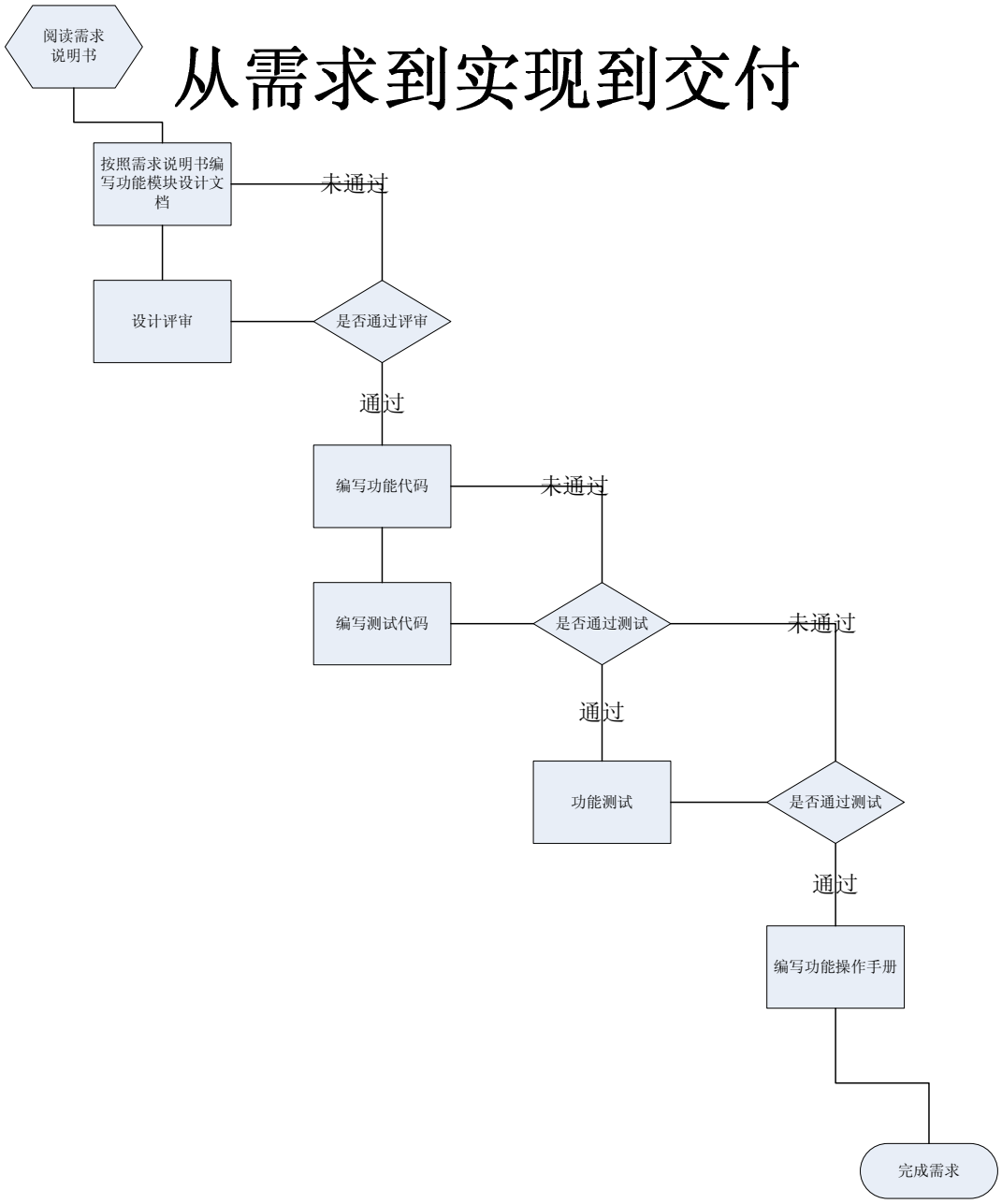
定义、首字母缩写词和缩略语

参考资料

概述

如何从需求到实现到交付

流程图


从需求到实现到交付
阅读需求说明书
按照需求说明书编写功能模块设计文档
未通过
设计评审
是否通过评审
通过
编写功能代码
未通过
编写测试代码
是否通过测试
未通过
通过
功能测试
是否通过测试
通过
编写功能操作手册
完成需求

注意点

- ✓ 所有业务功能接口在设计时就需要定义完成，一切实现以此为依据。开发过程中有变动的需同时修订设计文档。
- ✓ 为保证测试质量，测试代码覆盖度需 90%以上
- ✓ 功能测试要尽可能测试到所有可能的情况

总体配置

Spring

- 配置文件统一存放于 conf 包下，按其子系统和功能分：
  - applicationContext-common.xml
    存放公用 Bean 的定义，如数据源配置、事务管理配置
  - applicationContext-dao.xml
    存放 DAO 层 Bean 的定义
  - applicationContext-service.xml
    存放 Service 层 Bean 的定义
  - applicationContext-quartz.xml
    存放用 quartz 实现定时任务 Bean 的定义
  - jdbc.properties
    数据库连接属性定义

Hibernate

- Hiberenate 配置文件统一存放到<\子系统\模块\po>包下。
- 配置文件由 MyEclipse 自动生成，取消对象之间的关联，每个 PO 对象都是一个独立的个体（对象间关系通过代码手工实现）。

Struts

- 配置文件存放 WEB-INF 下
  - action-servlet.xml
    存放了 Action 的 Bean 定义
  - struts-config.xml
    Struts 配置信息

ActionForm

- 为了最大程度上增加类可重用性，借鉴已有项目的成功经验，取消 ActionForm 的强制继承关系，而改用 DynaValidatorForm 内嵌 POJO 的方式。

POJO

- 统一存放到<\子系统\模块\pojo>包下。

DWR

- 配置文件存放 WEB-INF 下
  - dwr.xml

代码编写

DAO层

规范

命名规范

- ✓ 实现类命名按：**表名+DaoImpl**，例如：OperatorDaoImpl
- ✓ 接口命名按：**I+表名+Dao**，例如：IOperatorDao
- ✓ 方法命名按：**动词或（动词+名词形式）**，例如：
  - ■ 新增功能命名：save()
  - ■ 修改功能命名：update()
  - ■ 删除功能命名：delete()
  - ■ 审核功能命名：check()
  - ■ 查询功能命名：list()
  - ■ 分页查询功能命名：listByPage()
  - ■ 读取单条命名：load()
- ✓

Spring配置图

所有实现类需继承 baseDao，继承的同时也获得了父类的属性注入。方括号内为增量开发的基础。

```
<bean id="baseDao" class="com.litt.core.dao.BaseDaoImpl"
    abstract="true">
    <property name="hibernateTemplate">
        <ref bean="hibernateTemplate" />
    </property>
    <property name="jdbcTemplate">
        <ref bean="jdbcTemplate" />
    </property>
</bean>
<!-- 操作员DAO -->
<bean id="operatorDao"
    class="com.cidp.system.dao.impl.OperatorDaoImpl" parent="baseDao" />
```

继承注入

注意点

- ✓ 所有 DAO 继承于 **BaseDaoImpl**。BaseDaoImpl 统一注入了 Spring 的 HibernateTemplate、JdbcTemplate

两个模板类，封装了 Spring 底层数据库操作实现，将复杂的数据库操作进一步封装。常用数据库操作变为简单的参数传递，并统一了分页处理。减轻 DAO 层的开发强度和重复劳动并提高易用性和扩展性。

- ✓ 由于 DAO 与表是一一对应的，因此方法命名时可省略后面的名词。特殊情况如主从表合并使用一个 DAO 时，从表的方法需采用动词+名词形式的命名方式

Service层

规范

命名规范

- ✓ 实现类命名按：**表名+ServiceImpl**，例如：OperatorServiceImpl
- ✓ 接口命名按：**I+表名+Service**，例如：IOperatorService
- ✓ 方法命名按：**动词+名词形式**（功能+类别)），例如：
  - ■ 新增功能命名：saveXXX()
  - ■ 修改功能命名：updateXXX()
  - ■ 删除功能命名：deleteXXX()
  - ■ 审核功能命名：checkXXX()
  - ■ 查询功能命名：listXXX()
  - ■ 分页查询功能命名：listByPage()
  - ■ 读取单条命名：loadXXX()
- ✓

注释规范

- ✓ 引用他人模块方法的，需同时在引用处和被引用处同时添加注释

Spring配置图

所有实现类需继承 baseTransactionProxy，继承的同时也获得了父类的事务管理支持。如图为增量开发的基础。

```
<!-- 操作员Service -->
<bean id="operatorService" parent="baseTransactionProxy">
    <property name="target">
        <bean
            class="com.cidp.system.service.impl.OperatorServiceImpl">
            <property name="operatorDao">
                <ref bean="operatorDao" />
            </property>
            <property name="operatorDetailDao">
                <ref bean="operatorDetailDao" />
            </property>
        </bean>
    </property>
</bean>
```

注意点

- ✓ 所有 Service 继承于 BaseServiceImpl。
- ✓ Service 与 DAO 应一一对应，不同模块间通信只能通过 Service 接口开放的方法。
- ✓ 事务处理方面，预定义了以 save、update、delete、list、do 开头命名的方法将受事务控制。默认情况下运行时异常发生时将回滚事务；另外我们也定义了 BusiException 的受查异常，Service 层统一只能抛出此异常，如有其他需要控制事务的异常请转换成此异常再抛出。
- ✓ 该层是底层数据库操作和上层表现层展现连接的纽带。特殊情况下可以在表现层中直接调用该层的业务方法，如（AJAX）

Action层

规范

命名规范

- ✓ Action 命名按：**模块名+Action**，例如操作员管理对应的是：OperatorAction.java
- ✓ ActionForm 命名按：**对象名+Form**，例如操作员管理对应的是：OperatorForm
- ✓ 方法命名按：**动词+名词形式**（功能+类别），例如：
  - ■ 保存新增功能：saveXXX()
  - ■ 保存修改功能：updateXXX()
  - ■ 删除功能：deleteXXX()
  - ■ 审核功能：checkXXX()
  - ■ 查询功能：listXXX()
- ✓ 页面跳转命名按：**to+功能名**，例如：
  - ■ 跳转到新增页面：toAdd
  - ■ 跳转到修改页面：toEdit

- 跳转到查询页面：toList
- 跳转到成功页面：toSuccess

✓

Spring配置图

如图为增量开发的基础。

```
<!-- 操作员信息表 -->
<bean name="/operator" class="com.cidp.system.web.OperatorAction">
    <property name="menuService" ref="menuService"></property>
    <property name="operatorService" ref="operatorService"></property>
    <property name="operatorDetailService" ref="operatorDetailService"></property>
</bean>
```

Struts-config配置图

```
<!-- 操作员管理 -->
<form-bean name="operatorForm" type="org.apache.struts.action.DynaActionForm">
  <form-property name="operatorVo" type="com.cidp.system.pojo.OperatorVo"></form-property>
</form-bean>
```

（form 定义）

```
<!-- 操作员管理 -->
<action path="/operator" scope="request" parameter="method" name="operatorForm">
    <forward name="toList" path="/pages/system/operator/operator_list.jsp"></forward>
    <forward name="toAdd" path="/pages/system/operator/operator_add.jsp"></forward>
    <forward name="toEdit" path="/pages/system/operator/operator_edit.jsp"></forward>
    <forward name="toWatch" path="/pages/system/operator/operator_watch.jsp"></forward>
    <forward name="toSuccess" path="/pages/system/success.jsp"></forward>
</action>
```

（Action 定义）

注意点

- ✓ 所有 Action 继承于 **BaseAction**。
- ✓ Action 中一个方法可以调用多个 Service 中的方法，但是包含数据更新逻辑的则应当将这个逻辑放到 Service 层中封装。
- ✓ 采用简单 JAVA 类（pojo）作为 Form 对象，而不继承 ActionForm 的方式。
- ✓ 所有 Action 继承于 BaseAction，BaseAction 附加了一些基本方法，如获取登陆用户信息等。

JSP

规范

命名规范

- ✓ 页面命名按：**名词+动词形式**（类别+功能），不同单词间用下划线分隔，例如：
  - ■ 新增页面命名：XXX_add.jsp
  - ■ 修改页面命名：XXX_edit.jsp
  - ■ 审核页面命名：XXX_check.jsp
  - ■ 查询页面命名：XXX.jsp
  - ■ 详细内容查看页面命名：XXX_watch.jsp
  - ■ 功能查询页面命名：XXX_list_forsomething.jsp
- ✓ 按钮快捷键命名

按钮名称	快捷键	英文含义
查询(F)	f	Find
新建(N)	n	New
修改(U)	u	Update
删除(D)	d	Delete
确定(O)	o	Operate
保存(S)	s	Save
返回(B)	b	Back
取消(C)	c	Cancel
重置		reset
查看(I)	i	Info
打印(P)	p	Print
打印预览(V)	v	preView
刷新(R)	r	Refresh
帮助(H)	h	Help

其他一律不使用快捷键
弹出页面使用"取消"按钮，跳转页面使用"返回"按钮

格式规范

- ✓ 统一 DOCTYPE 标准，代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

✓ title 标签作为了解页面功能的快速手段，编写格式：**${SYSTEM_TITLE } + 模块名**，示例如下：

```
<title>${SYSTEM_TITLE } - 系统配置</title>
```

✓ 查询条件或是表单的名称标签后，统一使用半角冒号，示例如下：

```
<td>名称:</td>
<td><input name=”userName”/></td>
```

Extremetable标签扩展

switchColumn(标志分支标签)
<ec:switchColumn property="FLAG" title="标志" defaultValues="0,1,2"> 　　标志 1 　　|| 　　标志 2 　　|| 　　标志 3 </ec:switchColumn>

NumberCell(金额格式化)
用法：<ec:column property="MONEY" title="金额" cell="currency"/>

DateCellTag(日期格式化显示标签)
用法：<ec:column property="DATE" title="时间" cell="dateCell" format="integer"/> 参数：format 可等于"integer","datetime","date","time"

Javascript

规范

命名规范

✓ 表单验证方法统一命名为：validateForm(form)
✓ onchange 事件方法统一按：**change+属性名**，如：changeOpType(this)

common.js

common-form.js

common-form-validator.js

CSS

规范

命名规范

✓ 容器命名按：**类型+Container**，例如：mainContainer、leftContainer

common.css

全局公用样式。

- 默认页面背景
- 内容容器的基本样式#mainContainer
- 默认字体样式：类型、大小、行高、颜色
- 默认加粗字体的磅值
- 默认 em 的样式
- 默认超链样式：颜色
- 默认 H1~H6 标题的样式
- 引用块样式(blockquote)
- 常用浮动
  - .left
  - .right
  - clearIt
- 常用字体大小
  - .small: =10px
  - .large: =14px
  - .larger: =16px
- 隐藏样式(.hide)

layout.css

布局样式。

print.css

打印类样式。

externalLinks.css

外部链接样式。给各种外部链接增加右侧图标显示，如：A,mailto,aim:,.DOC,.PDF,.RSS,.rdf

reset.css

重置样式。（通常无需修改）

form.css

表单页面样式。

Widgets

LittTable

简单表格控件，

错误处理和异常事件

这里只列举一些系统分层中典型的异常处理方式，实际项目开发过程中，需要针对不同业务需要，进行特定的异常划分和处理。

公用异常

- UnimplementedException – 未实现功能异常
  由于采用模块化开发，或是采用更高级的组件化开发，其基本的设计思想都是先定义接口后实现具体功能。因此，为了在开发过程中显示的标注出哪些功能代码尚未实现，在这些方法上需抛出此异常。此异常为 RuntimeException，故不需要显示的捕获。

DAO

Dao 层主要跟数据库打交道，由于使用了 Spring 的模板类进行数据操作，相关的异常处理都已统一封装。因此在这一层中，我们通常不需要做异常处理。

Service

Service 层也就是业务逻辑层，也是整个应用系统中最复杂、代码最多、处理方式最多变、最容易出错的地方；同时基于 Spring 的声明式事务管理也是覆盖在这个层次上的，如何保障事务的有效、完整处理，也使这一层次的开发增加了很多难度。

- BusiException - 业务异常类
  由于业务需要在业务流程中终止流转时使用的异常，该异常发生后如果当前方法受事务管理，会回滚事务。

Action

- NotLoginException - 未登陆异常
  某些功能中，可能需要对用户未登录的情况做特殊处理，如错误页面跳转等，该异常类就是做这个作用。

核心组件包使用

具体编程示例请参阅【知识库】→ 项目开发指南 → 核心组件库(littcore.jar)

日志处理

- 各层基类中已定义了一个“logger”对象，实现类中可直接使用该对象进行日志记录。
- 所有的异常（Exception）都必须写在日志中，当系统抛出异常时，除了使用 log.error()记录定制信息外，还必须打印异常树信息，例如 log.error(定制信息, e)。
- 系统出现问题时，必须抛出异常，在处理异常时记录日志，且日志级别必须是 Error。
- 当同时有多条日志记录时，请使用 isXXXEnabled 方法进行预判断。
  - 示例：

```
if(logger.isDebugEnabled()) {
    logger.debug(“123”);
    logger.debug(“abc”);
}
```

- 对于 debug、info 级别的日志而言，如果日志中存在类似：logger.info("Entry User: " + User.name + " is " + String.valueOf(entry))语句，则它必须出现在 log.isInfoEnabled()判断中。
  - 示例：

```
if (logger.isInfoEnabled()){
    logger.info("Entry User: " + UserA.name + " is " + String.valueOf(entryA));
    logger.info("Entry User: " + UserB.name + " is " + String.valueOf(entryB));
}
```

- 对于 error 级别的日志而言，不应该出现类似：logger.error("Entry User: " + Object + " is " + String.valueOf(entry))语句。在输出日志之前完成字符串操作，禁止一些结构复杂的对象被直接放入日志中作字符串处理。这种调用方式会引起过多的构建信息参数的花费。定制错误信息的组合必须在输出日志之前完成字符串的操作，可使用 StringBuffer 提高效率。
- 在应用系统中，必须使用 log4j NDC(Nested Diagnostic Contexts)来处理多线程的日志记录。当程序响应请求时，通过 Log4J NDC(Nested Diagnostic Contexts)机制，将日志（logger）推入栈中，当程序完成处理或抛出异常时，NDC 弹栈，以保证同一线程日志的一致性。
- 为使 NDC 机制生效，配置文件中的布局格式中一定要加上%x。
- 日志的输出格式(不包括异常树信息)规定如下：日期(yyyy-M-d) 时间(HH:mm:ss) NDC 标志[级别](对应程序文件名:行号)定制信息。即，%d{yyyy-M-d HH:mm:ss}%x[%5p](%F:%L) %m%n

其他

- 在保证软件系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。
- 警惕"分析瘫痪"。请记住，无论如何都要提前了解整个项目的状况，再去考察其中的细节。由于把握了全局，可快速认识自己未知的一些因素，防止在考察细节的时候陷入"死逻辑"中。
- 警惕"过早优化"。首先让它运行起来，再考虑变得更快--但只有在自己必须这样做、而且经证实在某部分代码中的确存在一个性能瓶颈的时候，才应进行优化。除非用专门的工具分析瓶颈，否则很有可能是在浪费自己的时间。性能提升的隐含代价是自己的代码变得难于理解，而且难于维护。

附录：指南概要