
上海协同上海协同

<负荷管理系统>
编程指南

版本 <1.0>

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

修订历史记录

日期	版本	说明	作者
2007-11-22	<1.0>	创建	蔡源

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

目录

1.	简介	4
1.1	目的	4
1.2	适用人员	4
1.3	定义、首字母缩写词和缩略语	4
1.4	参考资料	4
1.5	概述	4
2.	企业应用框架	5
3.	系统框架	5
3.1	系统架构	5
3.2	系统层次图	6
4.	代码编写	6
4.1	DAO 层	8
	注意点:	8
	Spring 配置图	8
4.2	Service 层	8
	注意点:	8
	Spring 配置图	8
4.3	Action 层	9
	注意点:	9
	Spring 配置图	9
4.4	JSP	9
	Extremetable 标签扩展	9
4.5	常用工具类	错误！未定义书签。
5.	错误处理和异常事件	11
6.	日志处理	12
7.	其他	13
8.	附录：指南概要	14

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

编程指南

1. 简介

1.1 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

1.2 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。

1.3 定义、首字母缩写词和缩略语

1.4 参考资料

1.5 概述

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

2. 企业应用框架

- ✓ 持久性 (persistence)：实现数据存储、处理，数据与对象映射，数据缓存 (caching)
- ✓ 事务 (transaction)：确保一组关联操作正常、完整的执行
- ✓ 安全性 (security)：保证系统的通信安全、数据安全
- ✓ 日志 (logging)：记录系统运行情况和异常，记录特定用户操作
- ✓ 业务逻辑 (business logic/rules)：实现业务规则和业务逻辑
- ✓ 监控 (system monitoring/management)：监控系统运行状况，设置系统参数
- ✓ 负载均衡 (load balance)：在大量并发访问时，保持系统可用

3. 系统框架

3.1 系统架构

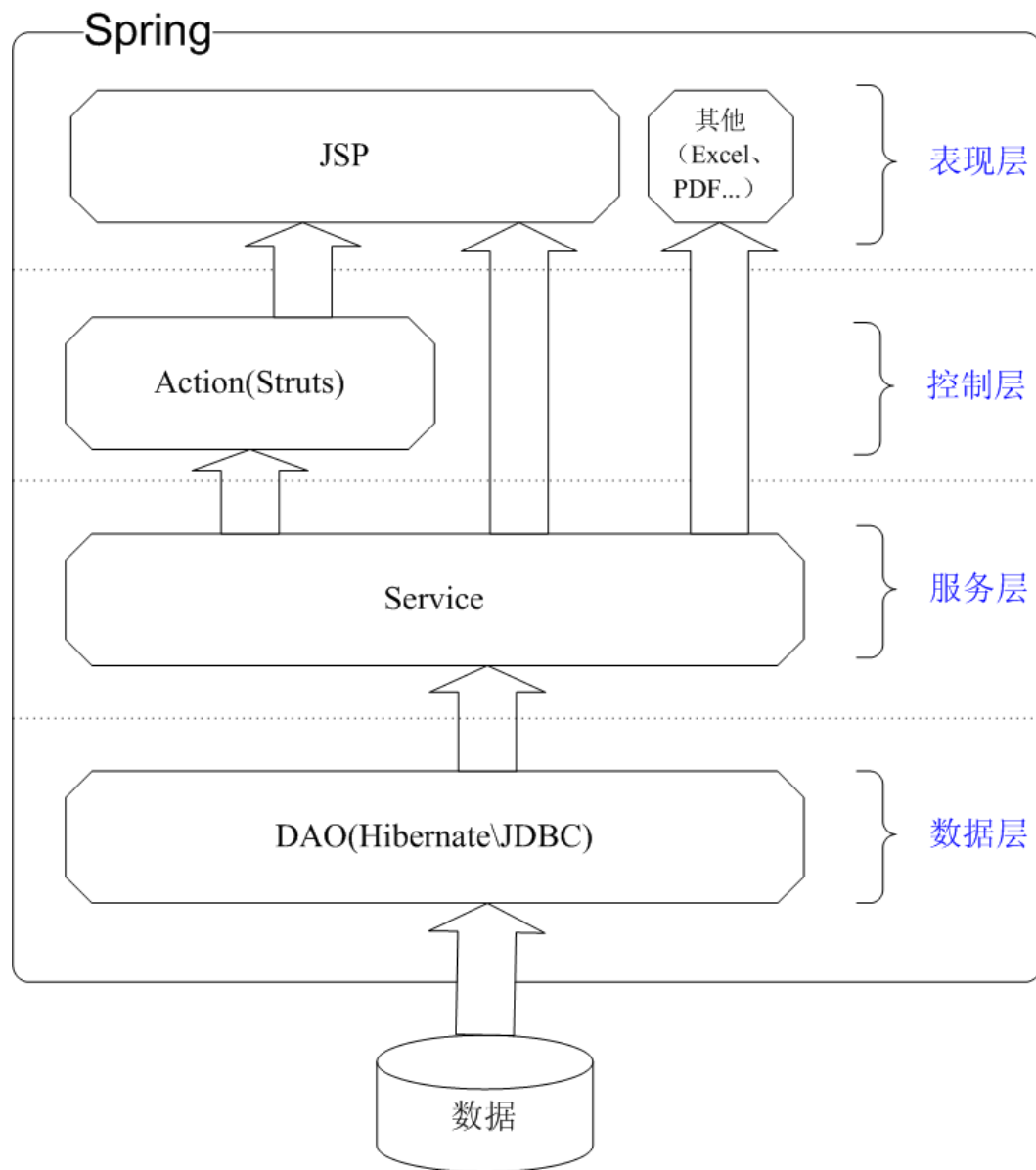
依照严格的 J2EE 架构标准，依据 J2EE 应用模式定义使用四层架构进行实现，它避免了传统两层结构的局限性，能够提供足够的可伸缩性、可配置型、可接入性和可管理性。

本系统共分为：**数据层**、**服务层**、**控制层**、**表现层**，共四个逻辑层次。术语表示为：DAO(Hibernate\JDBC)、Service、Action(Struts)、JSP。

- DAO:简单的数据库操作由 Hibernate 实现，复杂的业务逻辑如批处理等由 JDBC 调用 SQL 或存储过程实现，每个存储过程都是一个独立的完整的业务，事务管理由存储过程实现。
- Service:该层作为给表现层提供基础的业务服务,处理相关业务逻辑,由 Spring 提供事务支持。Service 层调用底层抽取上的数据，并相互关联形成一个完成的业务操作；因此，开放给上层的方法都应该是一个完成的业务。
- Action:Struts 的 Action 采用 DispatchAction 作为基类,每一个方法都是一个独立的完整的业务操作。一般情况下数据更新的操作由于受 Spring 事务管理，每个方法都只对应 Service 中的一个方法。
- JSP:作为表现层，应该尽可能的简单，专注于数据展示。

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

3.2 系统层次图



4. 技术框架

4.1 Spring

4.2 Hibernate

- Hibernate 配置文件统一存放到<模块\model>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）。

4.3 Struts

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

4.4 POJO、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。所有中间环节需要封装成简单对象的，统一存放到<模块\pojo>包下

4.5 DWR

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

5. 代码编写

5.1 DAO层

注意点:

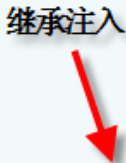
1. 所有 DAO 继承于 **BaseDAO**。BaseDAO 统一引入了 Spring 的 HibernateTemplate、JdbcTemplate 两个模板类，封装了 Spring 底层数据库操作实现，将复杂的数据库操作进一步封装。常用数据库操作变为简单的参数传递，并统一了分页处理。减轻 DAO 层的开发强度和重复劳动并提高易用性和扩展性。

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.shxt.core.dao.BaseDao"
    abstract="true">
    <property name="hibernateTemplate">
        <ref bean="hibernateTemplate" />
    </property>
    <property name="jdbcTemplate">
        <ref bean="jdbcTemplate" />
    </property>
</bean>

<bean id="menuDao" class="com.shxt.system.dao.impl.MenuDaoImpl" parent="baseDao"/>
```



5.2 Service层

注意点:

1. 所有 Service 继承于 **BaseManager**。
2. 一个 DAO 只能对应一个 Service。不同模块间交流只能通过 Service 开放的方法。
3. 事务处理方面，预定义了以 **save、update、delete、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 BusinessException 的受查异常，Service 层统一只能抛出此异常，如有其他受查异常请转换成此异常。
4. 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）

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。

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

```

<bean id="menuManager" parent="baseTransactionProxy">
  <property name="target">
    <bean
      class="com.shxt.system.service.impl.MenuManagerImpl">
        <property name="menuDao">
          <ref bean="menuDao" />
        </property>
      </bean>
    </property>
  </bean>
</bean>

```

5.3 Action层

注意点:

1. 所有 Action 继承于 **BaseAction**。
2. 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 层中处理。
3. 采用简单 JAVA 类 (pojo) 作为 Form 对象，而不继承 ActionForm 的方式。
4. 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。

Spring配置图

如图为增量开发的基础。

```

<bean name="/menu" class="com.shxt.system.web.MenuAction">
  <property name="menuManager" ref="menuManager"/>
</bean>

```

5.4 JSP

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>

```

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

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"`

5.5 Javascript

5.6 CSS

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

6. 错误处理和异常事件

6.1 DAO

6.2 Service

- ✓ 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。

6.3 Action

- ✓ NotLoginException: 未登陆异常。

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

7. 日志处理

- 各层基类中已定义 log 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。
- 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，下面给出了代码示例。

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

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

8. 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。

<负荷管理系统>	Version: <1.0>
编程指南	Date: 2007-11-22

9. 附录：指南概要