整合的原则：

1、先将Spring整合到Web项目中去并保证ioc可以正常运行

2、如果有用到AOP则进一步确定AOP可以正常运行

3、加入Struts2确保可以独立正常运行

4、关联Spring与Struts2

5、加入Hibernate确保可以独立正常运行

6、关联Spring与Hibernate

整合的入口：先从Spring开始入手

Spring整合到WEB工程中

步骤：

1、创建Web工程，并导入页面及JSCS等静态资源

2、导入Spring的相关JAR(IOC——>6个、AOP——>4个、TX——>3个、Web——>1个、junit测试——>2个)共16个。

3、写Spring的配置文件，导入相应约束并配置IOC的资源管理

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd">
```

4、（可以不用）在使用IOC资源的类里通过WebApplicationContextUtils获取Spring容器

```
ApplicationContext ac = WebApplicationContextUtils.getWebApplicationContext(this.getServletContext());
```

5、建立项目包结构

action 对应**JSP**的动作

dao 持久层数据库操作

service 业务层，事务的处理

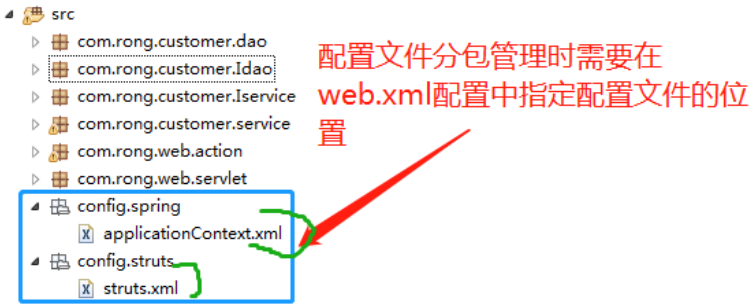
domain 实体类（数据库表对应）

utist 相关工具类

test 项目测试（**junit**）

6、配置web.xml中Spring的监听器（让应用一加载时创建spring容器。保证一个应用只有一个spring容器 该监听器默认只能加载WEB-INF目录中一个名称为applicationContext.xml的配置文件）

7、配置web.xml统一管理Spring的配置文件(配置全局初始化参数：指定spring配置文件的位置)



8、先写业务层，然后持久层。配置Spring的ioc资源管理及注入

```
<bean id="customerDao" class="com.itheima.dao.impl.CustomerDaoImpl">
    <!-- 注入sessionFactory的 extends HibernateDaoSupport -->
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>
```

持久层extends HibernateDaoSupport 所以注入SessionFactory

```
<bean id="customerService" class="com.itheima.service.impl.CustomerServiceImpl">
    <!-- 注入dao -->
    <property name="customerDao" ref="customerDao"></property>
</bean>

<!-- 把action也交给spring来管理 -->
<bean id="customerAction" class="com.itheima.web.action.CustomerAction" scope="prototype">
    <property name="customerService" ref="customerService"></property>
</bean>
```

aciton配置为单例模式

9、配置数据库连接信息

sessionFactory

说明一、配置sessionFactory分以下两种情况

情况一：hibernate的主配置文件（保留hibernate主文件），让sessionFactory读取hibernate主配置文件

```
<!-- 配置sessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
<!-- 创建sessionFactory必须的三部分信息：都在hibernate的主配置文件中，需要告知spring的是：hibernate主配置文件的位置-->
<property name="configLocation" value="classpath:config/hibernate/hibernate.cfg.xml"></property>
</bean>
```

```
<!-- 配置sessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <!-- 创建sessionFactory必须的三部分信息：都在hibernate的主配置文件中呢。
        需要告知spring的是：hibernate主配置文件的位置-->
    <property name="configLocation" value="classpath:config/hibernate/hibernate.cfg.xml"></property>
</bean>
```

情况二：数据源配置数据库连接的基本信息，不保留hibernate主配置文件

1、通过C3P0注入数据库连接基本信息

2、在*sessionFactory*注入*C3P0*

3、在*sessionFactory*里配置*hibernate*的基本配置及配置*hibernate*实体文件

详见：applicationContext-sessionFactory数据源.xml

```
    <!-- 配置sessionFactory -->
    <bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <!-- 1、注入  连接数据库的基本信息 -->
        <property name="dataSource" ref="c3p0DataSource"></property>
        <!-- 2、hibernate的基本配置 -->
        <property name="hibernateProperties">
            <props>
<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
<prop key="hibernate.show_sql">true</prop>
<prop key="hibernate.format_sql">true</prop>
<prop key="hibernate.hbm2ddl.auto">update</prop>
<prop key="hibernate.current_session_context_class">org.springframework.orm.hibernate5.SpringSessionContext
            </prop>
        </props>
        </property>
        <!-- 3、映射文件的位置
            mappingResources：它用于指定映射文件位置。有几个映射文件需要写几个。
            mappingLocations：它用于指定映射文件位置。它可以使用通配符。
            mappingDirectoryLocations：它用于指定映射文件的目录位置。  -->
        <property name="mappingLocations">
            <array>  <value>classpath:com/itheima/domain/*.hbm.xml</value>   </array>
        </property>
    </bean>

<!-- 配置连接池,连接数据库的基本信息 -->
<bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/crm_szee03"></property>
    <property name="user" value="root"></property>
    <property name="password" value="1234"></property>
</bean>
```

说明二、单独使sessionFactory配置，DAO持久层的事务要手动去控制提交，通过sessionFactory获取Session对象实现数据库的操作

注入：配置Dao注入sessionFactory，Dao类定义sessionFactory，通过sessionFactory.getCurrentSession()获取Session对象

```
public class CustomerDao implements ICustomerDao {

    private SessionFactory sessionFactory;

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Override
    public void addUICustomer(Customer c) {
        Session s = null;
        Transaction tx = null;
        try{
            s = sessionFactory.getCurrentSession();
            tx = s.beginTransaction();
            s.save(c);
            tx.commit();
        }catch(Exception e){
            tx.rollback();
            e.printStackTrace();
        }
    }
```

注入

手动提交事务

HibernateTemplate：使用配合事务可以放弃sessionFactory手动提交事务操作JDBC

想使用注解IOC配置那么必须使HibernateTemplate在Dao里使用，如果没有使用HibernateTemplate可以使用Dao继承HibernateDaoSupport，HibernateDaoSupport里已封装HibernateTemplate

为HibernateTemplate注入sessionFactory，

为DAO注入HibernateTemplate，使用Hibernate模板可使用注解方式来获取资源

```
<!-- 配置HibernateTemplate 注入SessionFactory-->
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate5.HibernateTemplate">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

public class CustomerDao implements ICustomerDao {


    private HibernateTemplate hibernateTemplate;

    public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {
        this.hibernateTemplate = hibernateTemplate;
    }


  @Override
  public void addUICustomer(Customer c) {

      hibernateTemplate.save(c);

  }
```

Dao持久层继承HibernateDaoSupport

1、自定义Dao类继承HibernateDaoSupport

2、类中使用 **getHibernateTemplate().save(****);**

```
public class CustomerDaoImpl extends HibernateDaoSupport implements ICustomerDao {

    @Override
    public void saveCustomer(Customer customer){
        getHibernateTemplate().save(customer);
    }


    @Override
    public List findAllCustomer() {
        return getHibernateTemplate().find("from Customer");
    }
```

配置：

1、继承HibernateDaoSupport的Dao类注入*sessionFactory*

2、*HibernateTemplate*的配置可以注释掉不用了

```
<bean id="customerDao" class="com.itheima.dao.impl.CustomerDaoImpl">
    <!-- 注入sessionFactory的 extends HibernateDaoSupport -->
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<!-- 配置HibernateTemplate 注入SessionFactory
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate5.HibernateTemplate">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>  -->
```

由于没有使用所以可以删除或注释掉

10、配置事务

transactionManager， transactionManager的创建需要注入sessionFactory

```
<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">

<!-- 注入SessionFactory -->

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>
```

```
<!-- 配置事务管理器 -->
<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <!-- 注入SessionFactory -->
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>
```

注入：sessionFactory

*txAdvice*事务制作

```
<!-- 制作事务的通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <!-- 配置事务的属性 -->
        <tx:attributes>
            <tx:method name="*" propagation="REQUIRED" read-only="false"/>
            <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
        </tx:attributes>
    </tx:advice>
```

```
<!-- 制作事务的通知 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <!-- 配置事务的属性 -->
    <tx:attributes>
        <tx:method name="*" propagation="REQUIRED" read-only="false"/>
        <tx:method name="find*" propagation="SUPPORTS" read-only="true"/>
    </tx:attributes>
</tx:advice>
```

name ： 对业务层 哪个方法的事务属性配置

AOP配置通知和切入点

```
<aop:config>
        <!-- 指定通知和切入点表达式的对应关系 -->
        <aop:pointcut expression="execution(* com.itheima.service.impl.*.*(..))" id="pt1"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"/>
</aop:config>
```

```
<!-- aop配置 -->
<aop:config>
    <!-- 指定通知和切入点表达式的对应关系 -->
    <aop:pointcut expression="execution(* com.itheima.service.impl.*.*(..))" id="pt1"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"/>
</aop:config>
```

Struts2整合到WEB工程中

步骤：

1、导入Struts2的必备JAR包（13个JAR包）

log4j-core-2.2.jar

ognl-3.0.6.jar

struts2-core-2.3.24.jar

xwork-core-2.3.24.jar

asm-3.3.jar

asm-commons-3.3.jar

asm-tree-3.3.jar

commons-fileupload-1.3.1.jar

commons-io-2.2.jar

commons-lang3-3.2.jar

freemarker-2.3.22.jar

javassist-3.11.0.GA.jar

log4j-api-2.2.jar

2、创建Struts2的配置文件 struts.xml

3、对配置文件导入约束

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
```

4、配置Struts2的通用配置

开发者模式：<constant name="struts.devMode" value="true"></constant>

动作后缀设置：<constant name="struts.action.extension" value="action"></constant>

Struts2使用的标签默认主题修改：<constant name="struts.ui.theme" value="simple"></constant>

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <!-- 开启开发者模式 -->
    <constant name="struts.devMode" value="true"></constant>

    <!-- 改变struts2默认拦截的url后缀为.action -->
    <constant name="struts.action.extension" value="action"></constant>

    <!-- 配置struts2的主题，使用最简单的模式 -->
    <constant name="struts.ui.theme" value="simple"></constant>
```

5、配置Struts2的核心控制器，指定配置文件的位置（在web.xml里，web项目必须要的一步）

```
<filter>
        <filter-name>struts2</filter-name>
<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
        <init-param>
     <param-name>config</param-name>
     <param-value>struts-default.xml,struts-plugin.xml,config/struts/struts.xml</param-value>
        </init-param>
</filter>

<filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

配置文件分包管理时需要在web.xml配置中指定配置文件的位置

```
<!-- 配置struts2的核心控制器：过滤器
     它默认会去类的根路径下加载一个名称为struts.xml的配置文件
     我们可以通过手动设置，来指定配置文件的位置
     指定的方式是：配置过滤器的初始化参数
             参数的key：config
             参数的value：
                 config/struts/struts.xml
                 当我们手动指定了struts2的配置文件，必须在写完自己的配置文件位置之后
                 还得写上：struts-default.xml和struts-plugin.xml
                 不写会报错                                                    -->
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>struts-default.xml,struts-plugin.xml,config/struts/struts.xml</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

指定Struts2配置文件的位置，配置文件在SRC根目录里可以不用这个配置

5、改JSP/HTML的超连接部分（将请求参数的形式改为请求动作的形式，符合Struts2格式即可）

```
"${pageContext.request.contextPath}/customer/addUICustomer.html"  target=main>- 新增客户</A>
"${pageContext.request.contextPath}/customer/CustomerServlet?method=listCustomer" target=main>- 客户
```

/customer：为名称空间
/addUICustomer：请求动作
.html：后缀（用户可见）

Struts2请求规则

通过请求获取参数的底层原始方式请求规则

原始请求规则获取参数方式：`String requestMethod = request.getParameter("method");`

6、创建动作类extends ActionSupport，没有实体类时不用实现数据封装接口

```
public class CustomerAction extends ActionSupport{

    //通过Spring的ioc注入获取对象
    private ICustomerService customerService;

    public void setCustomerService(ICustomerService customerService) {
        this.customerService = customerService;
    }

    //动作类动作方法处理用户请求
    public String addUICustomer(){
        System.out.println("struts2正常运行了");
        customerService.allListCustomer();
        return "addUICustomer";
    }
}
```

动作类

7、配置动作类与JSP/HTML的关系

```
<struts>
    <!-- 开启开发者模式 -->
    <constant name="struts.devMode" value="true"></constant>

    <!-- 改变struts2默认拦截的url后缀为.html -->
    <constant name="struts.action.extension" value="html"></constant>

    <!-- 动作配置 -->
    <package name="customer" extends="struts-default" namespace="/customer">
        <action name="addUICustomer" class="customerAction" method="addUICustomer">
            <result name="addUICustomer">/jsp/customer/add.jsp</result>
        </action>
    </package>
</struts>
```

名称空间对应页面里的动作前面部分

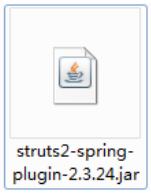
html/jsp里的动作

Spring管理Action类的类路径指引

处理请求动作的方法

8、整合Struts2与Spring分三步：

8.1、导入JAR包（1个 struts2-spring-plugin-2.3.24.jar）



ss整合必备包

由于Struts2生产Action的工厂不知道有Spring的存在，struts-default.xml这个配置里定义的生产Action的工厂并没有一个和Spring有关的

```
<bean class="com.opensymphony.xwork2.ObjectFactory" name="struts"/>
<bean type="com.opensymphony.xwork2.factory.ResultFactory" name="struts" class="org.apache.strut
<bean type="com.opensymphony.xwork2.factory.ActionFactory" name="struts" class="com.opensymphony
<bean type="com.opensymphony.xwork2.factory.ConverterFactory" name="struts" class="com.opensymph
<bean type="com.opensymphony.xwork2.factory.InterceptorFactory" name="struts" class="com.opensym
<bean type="com.opensymphony.xwork2.factory.ValidatorFactory" name="struts" class="com.opensymph
<bean type="com.opensymphony.xwork2.factory.UnknownHandlerFactory" name="struts" class="com.open
```

生产Action的工厂

所以要导入一个知道Spring的jar包

8.2、配置Struts2的Action让Spring统一管理Action的创建

```
<!-- 配置Action类让Spring来管理Action的创建 -->
<bean id="customerAction"
    class="com.rong.web.action.CustomerAction">
    <!-- 在Action类里注入Service -->
    <property name="customerService" ref="customerService"></property>
</bean>
```

```
<bean id="customerAction"
        class="com.rong.web.action.CustomerAction">
        <!-- 在Action类里注入Service -->
        <property name="customerService" ref="customerService"></property>
</bean>
```

8.3、配置Struts2的Action动作

```
<!-- 动作配置 -->
<package name="customer" extends="struts-default" namespace="/customer">
    <action name="addUICustomer" class="customerAction" method="addUICustomer">
        <result name="addUICustomer">/jsp/customer/add.jsp</result>
    </action>
</package>
```

通过Spring管理的来获取

```
<package name="customer" extends="struts-default" namespace="/customer">

        <action name="addUICustomer" class="customerAction" method="addUICustomer">

            <result name="addUICustomer">/jsp/customer/add.jsp</result>

        </action>

    </package>
```

9、结合使用：

在Action里使用Spring的ioc的set注入方式获取业务层对象，由于使用了注入就不用再去获取Spring的容器来获取对象

```
public class CustomerAction extends ActionSupport{

    //通过Spring的ioc注入获取对象
    private ICustomerService customerService;

    public void setCustomerService(ICustomerService customerService) {
        this.customerService = customerService;
    }

    //动作类动作方法处理用户请求
    public String addUICustomer(){
        System.out.println("struts2正常运行了");
        customerService.allListCustomer();
        return "addUICustomer";
    }
```

有Spring的统一管理后，有SET注入Action类，就不用再去手动获取Spring的容器

Hibernate与Spring整合

1、导入Hibernate必备13个JAR包+1个JDBC数据库驱动JAR包+3个C3P0的数据库连接池JAR包

javassist-3.18.1-GA.jar

jboss-logging-3.3.0.Final.jar

log4j-1.2.16.jar

mysql-connector-java-5.1.7-bin.jar ———> JDBC数据库驱动JAR包

slf4j-api-1.6.1.jar

slf4j-log4j12-1.7.2.jar

antlr-2.7.7.jar

commons-beanutils-1.8.3.jar

dom4j-1.6.1.jar

geronimo-jta_1.1_spec-1.1.1.jar

hibernate-commons-annotations-5.0.1.Final.jar

hibernate-core-5.0.7.Final.jar

hibernate-jpa-2.1-api-1.0.0.Final.jar

jandex-2.0.0.Final.jar

以下是C3P0 jar包

c3p0-0.9.2.1.jar

hibernate-c3p0-5.0.7.Final.jar

mchange-commons-java-0.2.3.4.jar

2、配置实体类的映射文件　　类名.hbm.xml

3、配置Hibernate主配置文件（连接数据库信息及数据库相关文件），分两种情况：

情况一：有主配置文件，让Spring读取配置

sessionFactory的配置：

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
        <!-- 创建sessionFactory必须的三部分信息：都在hibernate的主配置文件中
            需要告知spring的是：hibernate主配置文件的位置-->
        <property name="configLocation" value="classpath:config/hibernate/hibernate.cfg.xml">
</property>
    </bean>
```

情况二：不要主配置文件，让Spring直接管理配置

sessionFactory的配置：注入C3P0数据源

```
<!-- 配置sessionFactory -->
 <bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
  <!-- 1、注入 连接数据库的基本信息 -->
  <property name="dataSource" ref="c3p0DataSource"></property>
  <!-- 2、hibernate的基本配置 -->
  <property name="hibernateProperties">
   <props>
    <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
    <prop key="hibernate.show_sql">true</prop>
    <prop key="hibernate.format_sql">true</prop>
    <prop key="hibernate.hbm2ddl.auto">update</prop>
    <prop
key="hibernate.current_session_context_class">org.springframework.orm.hibernate5.SpringSessionContext
   </prop>
  </props>
  </property>
  <!-- 3、映射文件的位置
   mappingResources：它用于指定映射文件位置。有几个映射文件需要写几个。
   mappingLocations：它用于指定映射文件位置。它可以使用通配符。
   mappingDirectoryLocations：它用于指定映射文件的目录位置。 -->
  <property name="mappingLocations">
   <array> <value>classpath:com/itheima/domain/*.hbm.xml</value> </array>
  </property>
 </bean>
```

```
<!-- 配置连接池,连接数据库的基本信息 -->
<bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
 <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
 <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/crm_szee03"></property>
 <property name="user" value="root"></property>
 <property name="password" value="1234"></property>
</bean>
```