

J2EE与中间件技术

——Spring

Spring

- ◆ Spring简介
- ◆ Spring模块构成
- ◆ Spring版Hello World

Spring简介

- ◆ 轻量级
- ◆ 反向控制——IoC
- ◆ 面向切面——AOP
- ◆ 容器
- ◆ 框架

“轻量级”

- ◆ 大小——1MB JAR包；
- ◆ 系统开支——处理开支很小；
- ◆ 非侵入式——基于Spring开发的系统对象一般不依赖于Spring的类；无需代码中涉及Spring专有类，即可将其纳入Spring容器进行管理
 - 不对容器依赖
 - 具有配置能力
 - 不同的产品，部署过程相同，易通用
 - 轻量级容器，接受任何JavaBean

反向控制——IoC

- ◆ 实现松耦合
- ◆ 对象被动接收依赖类而不是自己主动寻找
- ◆ JNDI 的反转
 - 对象不是从容器中查找它的依赖类，而是容器在实例化对象时主动将依赖类注入给它

面向切面---AOP

- ◆ 将业务逻辑从系统服务中分离出来
- ◆ 内聚开发
- ◆ 将服务模块化，并把它们声明式地应用在需要它们的地方。结果是这些组件更加专注于自身的业务，不需要涉及其它系统问题

容器

◆ 包含并管理系统对象的生命周期和配置

■ 可以通过配置来设定Bean

- ◆ 单一实例（singleton）/每次请求产生一个实例（prototype）
- ◆ 设定它们之间的关联关系

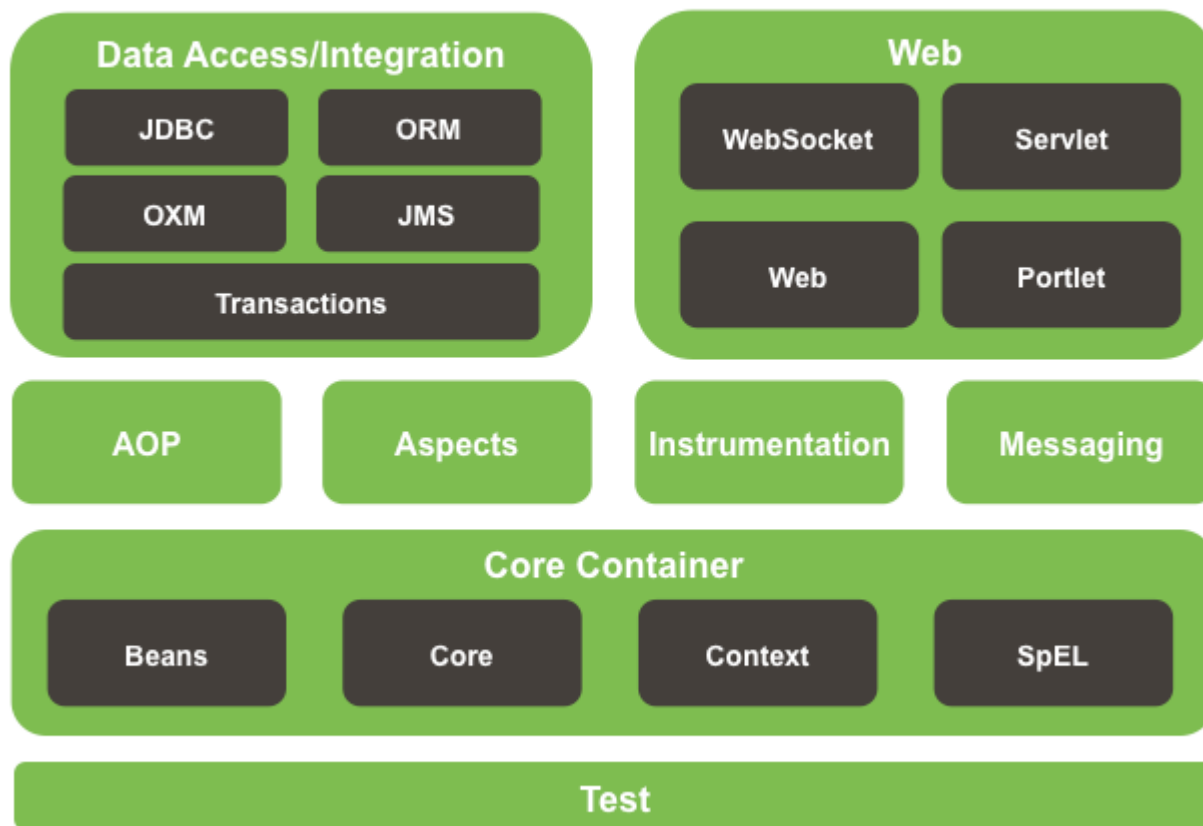
框架

- ◆使用简单的组件配置组合成为一个复杂的系统
- ◆系统的对象是通过XML文件配置组合起来
- ◆提供了基础功能—事务管理，持久层集成等，开发人员则专注于开发应用逻辑

Spring模块



Spring Framework Runtime



Spring模块

- ◆ 可以自由选择适合自己系统的模块，而不使用其他模块
- ◆ 所有模块都建立在核心容器之上，容器规定如何创建、配置、管理Bean，以及一些具体细节
 - 当你配置系统的时候，就隐式地使用了这些类
 - 作为开发人员，则关注那些提供服务的模块，如AOP模块等

核心容器

- ◆ Core和Beans模块提供基本部分的框架, 包括IoC和Dependency Injection功能
- ◆ org.springframework.beans包
 - BeanWrapper接口及它的实现
BeanWrapperImpl
 - BeanFactory接口及实现类

Bean Wrapper

- ◆ 封装了一个bean的行为
- ◆ 提供了设置和获取单个Bean的属性值（单个或者批量的），获取属性信息，查询只读/可写属性等功能

HelloWorld Bean类

```
public class HelloWorld{  
    private String msg = null;  
  
    public helloWorld() {  
    }  
  
    public void setMsg(String msg) {  
        this.msg = msg;  
    }  
  
    public String getMsg() {  
        return this.msg;  
    }  
}
```

测试程序

```
public class TestHelloWorld{
    public static void main(String[] args) throws
InstantiationException, IllegalAccessException, ClassNotFoundException{
        Object obj =
Class.forName("HelloWorld").newInstance();
        BeanWrapper bw = new BeanWrapperImpl(obj);

        bw.setPropertyValue("msg", "HelloWorld");

        System.out.println(bw.getPropertyValue("msg"));
    }
}
```

Bean Wrapper

- ◆ 支持嵌套属性，可以不受嵌套深度限制对子属性的值进行设置
 - 属性名可包含层次，对于属性名“address.zipcode”，BeanWrapper会调用 `getAddress().setZipcode()` 方法

BeanFactory

- ◆ 负责创建并维护*Bean*实例
- ◆ 将系统配置和依赖关系从代码中独立出来
 - 使Spring成为容器

BeanFactory 配置

◆ 可配置的项目

- Bean属性值及依赖关系（对其他Bean的引用）
- Bean创建模式（是否Singleton模式）
- Bean初始化和销毁方法

```
<bean id="testBean"  
class="edu.nju.TestBeanImpl"  
scope="singleton" />
```

◆ scope的值:常用singleton, prototype

- singleton表示该bean全局只有一个实例, **默认值**.
 - ◆ 适用于Service层和DAO层
- prototype表示该bean在每次被注入的时候, 都要重新创建一个实例, 适用于**有状态**的Bean.
 - ◆ 适用于Action层

示例：GreetingService接口

◆ GreetingService接口

- 定义了服务类需要提供的服务

◆ GreetingServiceImpl实现类

- **属性**greeting
 - ◆ 通过构造器和set方法被设定
- sayGreeting() 方法
 - ◆ 打印greeting属性
- 由Spring容器来设置greeting属性的值
 - ◆ hello.xml——**配置文件**
 - ◆ Spring容器通过调用Bean的**set**Greeting方法来设置属性

GreetingService接口

```
package hello;  
public interface GreetingService{  
    public void sayGreeting();  
}
```

GreetingServiceImpl

```
package hello;
public class GreetingServiceImpl implements GreetingService{
    private String greeting;

    public GreetingServiceImpl(){ }

    public GreetingServiceImpl(String greeting){
        this.greeting = greeting;
    }

    public void sayGreeting(){
        System.out.println(greeting);
    }

    public void setGreeting(){
        this.greeting = greeting;
    }
}
```

配置文件 hello.xml

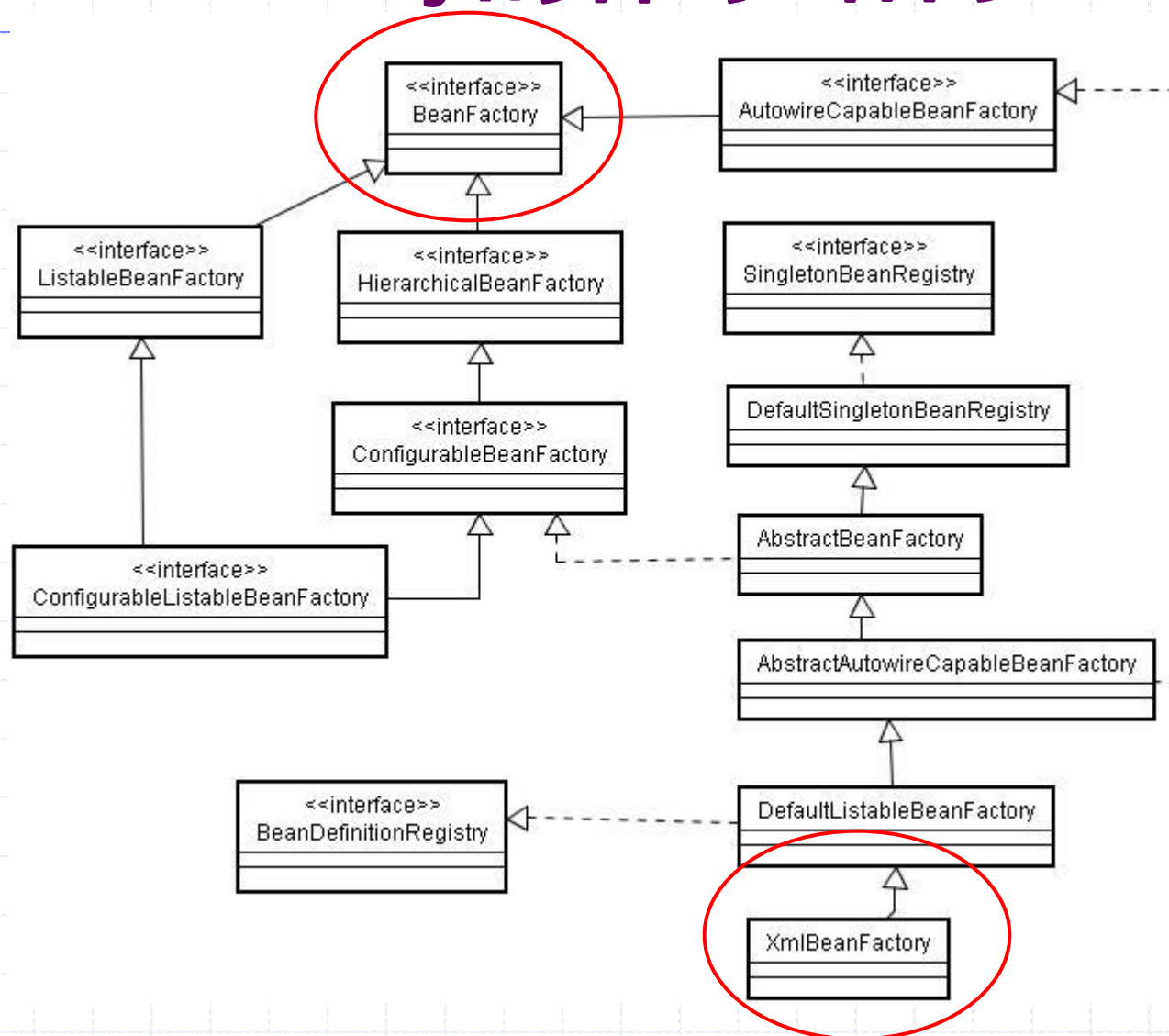
```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
  <bean id="greetingService"
    class="hello.GreetingServiceImpl">
    <property name="greeting">
      <value>Hello World!</value>
    </property>
  </bean>
</beans>
```

测试程序

```
public class HelloApp{  
    public static void main(String[] args) throws Exception{  
        BeanFactory factory =  
            new XmlBeanFactory(new FileInputStream("hello.xml"));  
  
        GreetingService greetingService  
        =(GreetingService)factory.getBean("greetingService"); //得到引用  
  
        greetingService.sayGreeting();  
    }  
}
```

BeanFactory的体系结构



BeanWrapper 与 BeanFactory

- ◆ BeanWrapper 实现了针对单个 Bean 的属性设定操作
- ◆ BeanFactory 则是针对多个 Bean 的管理容器，根据给定的配置文件，BeanFactory 从中读取类名、属性名/值，然后通过 Reflection 机制进行 Bean 加载和属性设定

上下文模块

- ◆ 建立在Core和Bean模块的基础上, 使Spring成为框架
- ◆ 从Beans模块继承其功能, 添加了国际化, 事件传播, 资源装载和透明的创建上下文
- ◆ 支持Java EE的功能, 比如EJB、JMX和基本的远程访问
- ◆ ApplicationContext接口

ApplicationContext

- ◆ BeanFactory提供了针对Java Bean的管理功能，而ApplicationContext提供了一个更为框架化的实现
- ◆ ApplicationContext覆盖了BeanFactory的所有功能，并提供了更多的特性
 - **ClassPathXmlApplicationContext类**
 - ◆ 读取web-info/classes目录下的配置文件
 - **FileSystemXmlApplicationContext类**
 - ◆ 文件系统下的配置文件

示例

```
ApplicationContext applicationContext=new  
ClassPathXmlApplicationContext("applicationConte  
xt.xml");
```

```
GreetingService greetingService  
=(GreetingService)  
applicationContext.getBean("greetingService");
```

onlineStock示例——注册用户

- ◆ edu.nju.onlinestock.dao

- Spring Bean
- 未使用Hibernate

- ◆ edu.nju.onlinestock.Service

- Spring Bean

- ◆ applicationContext.xml

- ◆ 不再需要Factory包

applicationContext.xml

◆ 配置

- 注入Bean
 - ◆ Bean之间的依赖关系（对其他Bean的引用）
- 两种方式
 - ◆ XML配置文件注入
 - 如: *UserManageService*
 - ◆ 注解注入
 - **@Repository**: 将数据访问层(DAO)的类标识为Spring Bean, 如: UserDao

配置文件注入Bean

.....

```
<bean id="UserManageService"  
class="edu.nju.onlinestock.service.Use  
rManageServiceBean"/>
```

.....

Servlet获取Spring容器中的Bean

```
@WebServlet("/register.user")
public class RegisterUserServlet extends HttpServlet {
    private static ApplicationContext apliationContext;
    private static UserManagerService userService;

    public void init() throws ServletException {
        super.init();
        apliationContext=new
        ClassPathXmlApplicationContext("applicationContext.xml");
        userService=(UserManagerService)apliationContext.getBean("UserManageService"
);
    }
    .....

    private void execute(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
        .....
        userService.registerUser(user);
        .....
    }
}
```


定义注解注入Bean

.....

<!-- 扫描有注解的文件 base-package 包
路径 -->

<context:component-scan base-
package="edu.nju.onlinestock" />

.....

UserDaoImpl——使用JDBC

```
.....
/*Repository 将数据访问层(DAO)的类标识为Spring Bean,
将所标注的类中抛出的数据访问异常封装为Spring的数据访问异常类型*/
@Repository
public class UserDaoImpl implements UserDao{
    .....
    public void save(User user){
        .....
        try {
            .....
            stmt=con.prepareStatement("insert into
users(id, user id, password, name, birthday, phone, email, bankid, account)
values(?, ?, ?, ?, ?, ?, ?, ?, ?)");
            .....
            stmt.executeUpdate();
        }catch (Exception e) {
            .....
        }
    }
    .....
}
```

注入userDao—— UserManageServiceImpl

.....

/* Component 是一个泛化的概念，仅仅表示一个组件（Bean），可以作用在任何层次；Service 通常作用在业务层，功能与 Component 相同*/

@Service

```
public class UserManageServiceImpl implements UserManageService {
```

```
    /** * Autowired 自动装配 相当于get() set() */
```

```
    @Autowired
```

```
    private UserDao userDao;
```

```
    public String registerUser(User user) {
```

.....

```
        userDao.save(user);
```

```
        return message;
```

```
    }
```

.....

```
}
```

@Autowired 与 @Resource 的区别

◆ @Autowired 属于 spring

- 默认按类型装配
- 要求依赖对象必须存在
 - ◆ 如果允许null，设置：
`@Autowired(required=false)`

◆ @Resource 属于 J2EE

- 默认按名称装配
 - ◆ `@Resource(name="baseDao")`

下载Spring

- ◆ <http://springsource.io>
- ◆ <http://repo.spring.io/release/org/springframework/spring>
- ◆ <http://sourceforge.net/projects/springframework>
- ◆ 下载Spring Framework
- ◆ 最新：5.1

设置开发环境

◆ 解压

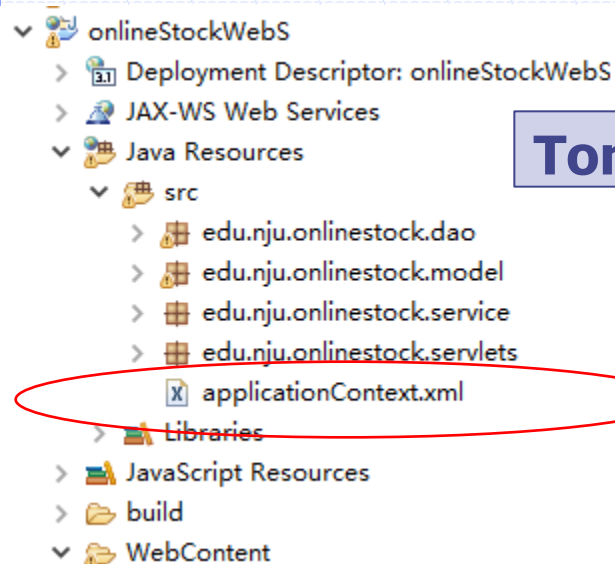
■ dist: 发布包

- ◆ *.jar
- ◆ 根据需要, 选择相关的JAR文件复制到WEB-INF/lib

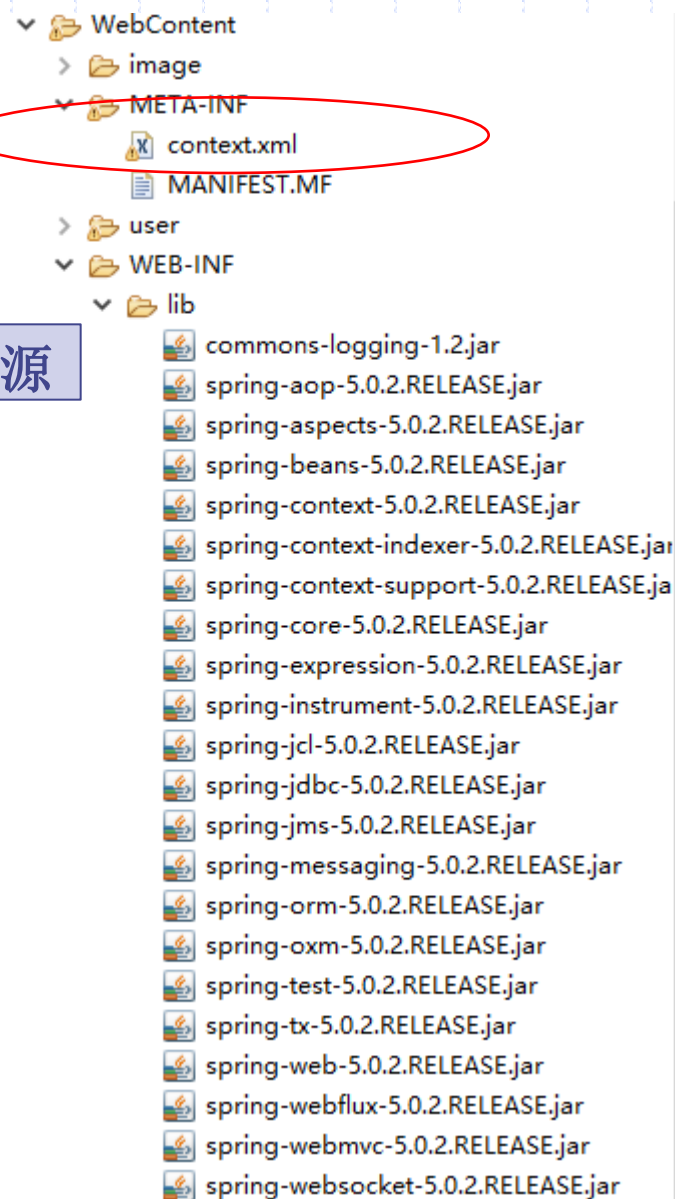
其他需下载的JAR文件

- ◆ Spring使用了来自Jakarta Commons项目的大量组件
- ◆ 需在所有基于Spring的应用里包含这些JAR文件，如
 - commons-logging.jar

onlineStock示例



Tomcat配置数据源



作业8

◆ 修改作业7中数据访问层和Service的设计

- edu.nju.onlinestock.servlets
 - ◆ 或使用SpringMVC的Controller
- edu.nju.onlinestock.model
 - ◆ 使用JDBC，或集成Hibernate
- edu.nju.onlinestock.dao
 - ◆ Spring
- edu.nju.onlinestock.Service
 - ◆ Spring

数据访问/集成模块——JDBC模块

◆ 包括JDBC、ORM、OXM、JMS和Transaction模块

■ JDBC模块

- ◆ 提供了不需要编写冗长的JDBC代码和解析数据库厂商特有的错误代码的JDBC-抽象层

JDBC模块

◆核心：JdbcTemplate类

- 模板设计模式
- 消除冗长的代码，如连接的创建及关闭等
- 对可变部分采用回调接口方式实现，如
 - ◆ ConnectionCallback，通过回调接口返回给一个Connection，从而可以使用该连接做任何事情
 - ◆ StatementCallback，返回一个Statement

JdbcTemplate类

◆方法：

- execute方法：执行任何SQL语句；
- update/batchUpdate方法：执行新增、修改、删除等语句；
- query/queryForXXX方法：执行查询相关语句；
- call方法：执行存储过程、函数相关语句。

applicationContext.xml

◆ 配置

- 数据源
 - ◆ 如Jdbc data Source
- JdbcTemplate
- 注入Bean

配置数据源（jdbc）

.....

```
<!-- 配置dataSource -->
```

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerData
Source">
```

```
    <property name="driverClassName"
value="com.mysql.jdbc.Driver" />
```

```
    <property name="url"
value="jdbc:mysql://localhost:3306/onlinestockdb?autoReconne
ct=true" />
```

```
    <property name="username" value="root" />
```

```
    <property name="password" value="" />
```

```
</bean>
```

配置JdbcTemplate

```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.J
dbcTemplate">
    <property name =
"dataSource" ref="dataSource"/>
</bean>
.....
```

示例： UserDaoImpl——使用 JdbcTemplate

```
.....
@Repository
public class UserDaoImpl implements UserDao
{
    //注入JdbcTemplate
    @Autowired
    private JdbcTemplate jdbcTemplate;
    .....

    public void save(User user) {
        String sql = "insert into
users(id,userid,password,name,birthday,phone,email,bankid,account)
values(?, ?, ?, ?, ?, ?, ?, ?, ?) ";
        jdbcTemplate.update(sql, new PreparedStatementSetter() {
            @Override
            public void setValues(PreparedStatement ps) throws SQLException {
                ps.setString(1, user.getId());
                ps.setString(2, user.getUserid());
                .....
            }
        });
    }
    .....
}
```


集成hibernate5

◆ Hibernate5 重复代码 (***)Dao) :

```
config = new Configuration().configure();
config.addAnnotatedClass(User.class);
serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(config.getProperties()).build();
sessionFactory=config.buildSessionFactory(serviceRegistry);
session=sessionFactory.openSession();
Transaction tx=session.beginTransaction();
session.save(user); //保存Entity到数据库中
tx.commit();
session.close();
sessionFactory.close();
```

数据访问/集成模块——ORM、事务模块

- ◆ **ORM模块**提供了对象-关系映射API集成层，包含JPA, JDO, **Hibernate**
 - 使用ORM包，可以使用所有的O/R映射框架结合所有Spring提供的特性，比如简单声明式事务管理功能
- ◆ **事务模块**提供了编程和声明式的事务管理的支持，利用了**AOP模块**

applicationContext.xml

◆ 配置

■ 数据源

- ◆ Jdbc
- ◆ Hibernate c3p0连接池
- ◆

■ SessionFactory

■ 事务管理，两种方式

- ◆ 声明式
- ◆ 注解方式

■ 注入Bean，两种方式

- ◆ 注解注入
- ◆ XML配置文件注入

◆ 不需要hibernate.cfg.xml文件

配置数据源（以jdbc为例）

.....

```
<!-- 配置dataSource -->
```

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerData
Source">
```

```
    <property name="driverClassName"
value="com.mysql.jdbc.Driver" />
```

```
    <property name="url"
value="jdbc:mysql://localhost:3306/onlinestockdb?autoReconne
ct=true" />
```

```
    <property name="username" value="root" />
```

```
    <property name="password" value="" />
```

```
</bean>
```

或者 c3p0 连接池数据源

.....

<!-- 定义数据源的信息, 使用c3p0连接池 -->

```
<bean id="dataSource"
class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass"
value="com.mysql.jdbc.Driver" />
        <property name="jdbcUrl"
value="jdbc:mysql://localhost:3306/onlinestock?useSSL=false" />
        <property name="user" value="root" />
        <property name="password" value="mysql" />
    </bean>
```

.....

或者 JNDI 数据源

```
<bean id= "dataSource"  
class= "org.springframework.jndi.JndiOb  
jectFactoryBean" >  
    <property name= "jndiName" >  
  
        <value>java:com/env/jdbc/trainingData  
Source</value>  
    </property>  
</bean>
```

配置SessionFactory

```
<!-- 配置sessionFactory -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="edu.nju.onlinestock.model" />
    <property name="hibernateProperties">
        <props>
            <prop
key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.connection.autocommit">true</prop>
        </props>
    </property>
</bean>
```

配置TransactionManager

```
<!-- 配置transactionManager -->
```

```
<bean id="transactionManager"  
      class="org.springframework.orm.hib  
ernate5.HibernateTransactionManager">  
    <property name="sessionFactory"  
ref="sessionFactory" />  
</bean>
```


容器事务管理——声明式

```
<!-- 声明式容器事务管理, transaction-manager 指定事务管理器为 transactionManager -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*User" propagation="REQUIRED" />
        <tx:method name="*" propagation="NOT_SUPPORTED" read-only="true" />
    </tx:attributes>
</tx:advice>

<!-- 定义切面, 在 edu.nju.onlinestock.service 包及子包中所有以 Service 结尾的方法中, 执行有关的
hibernate session 的事务操作 -->
<aop:config>
    <!-- 只对业务逻辑层实施事务 -->
    <aop:pointcut id="serviceOperation"
        expression="execution( *
            edu.nju.onlinestock.service.*Service.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceOperation" />
</aop:config>
```

AOP模块

◆ AOP 封装包

- 提供了符合AOP Alliance规范的面向方面的编程（aspect-oriented programming）的实现
- ◆ 从逻辑上减弱代码的功能耦合，而且利用source-level的元数据功能，可以将各种行为信息合并到你的代码中

或者注解方式

```
<tx:annotation-driven transaction-  
manager="transactionManager" />  
</beans>
```

- 在具体的类（或类的方法）上使用
@Transactional 注解

注入Bean

```
<bean id="UserManageService"  
class="edu.nju.onlinestock.service.impl.Use  
rManageServiceImpl">  
</bean>
```

<!-- 扫描有注解的文件 base-package 包路径 -->

```
<context:component-scan base-  
package="edu.nju.onlinestock" />
```

BaseDao设计

- 重复的数据库操作方法CRUD
- 以save为例

```
/*openSession()方法，获取的session不受  
spring管理*/
```

```
/*getCurrentSession()方法，获取的session受  
spring管理，不需要session.close() */
```

```
session=sessionFactory.getCurrentSession();  
session.save(object);  
//session.close();
```

注入sessionFactory—— BaseDaoImpl

.....

/*Repository 将数据访问层(DAO)的类标识为Spring Bean, 将所标注的类中抛出的数据访问异常封装为Spring的数据访问异常类型*/

@Repository

```
public class BaseDaoImpl implements BaseDao {  
    /** * Autowired 自动装配 相当于get() set() */  
    @Autowired  
    protected SessionFactory sessionFactory;
```

.....

```
    public Session getSession() {  
        return sessionFactory.getCurrentSession();  
    }  
}
```

BaseDaoImpl

```
public void save(Object bean) {  
    try {  
        getSession. save(bean) ;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

```
.....
```

```
}
```

注入baseDao——UserDaoImpl

.....

@Repository

public class UserDaoImpl implements UserDao{

@Autowired

private **BaseDao** baseDao;

public void save(User user){

try {

baseDao. save (user) ;

}catch (Exception e) {

e.printStackTrace();

}

}

.....

}

注入userDao—— UserManageServiceImpl

.....

/* Component 是一个泛化的概念，仅仅表示一个组件（Bean），可以作用在任何层次；Service 通常作用在业务层，功能与 Component 相同*/

@Service

public class UserManageServiceImpl implements UserManageService {

@Autowired

private UserDao userDao;

public String registerUser(User user) {

.....

userDao.save(user);

return message;

}

.....

}


Servlet获取Spring容器中的Bean

```
@WebServlet("/register.user")
public class RegisterUserServlet extends HttpServlet {
    private static ApplicationContext apliationContext;
    private static UserManagerService userService;

    public void init() throws ServletException {
        super.init();
        apliationContext=new
        ClassPathXmlApplicationContext("applicationContext.xml");
        userService=(UserManagerService)apliationContext.getBean("UserManageService"
);
    }
    .....

    private void execute(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
    .....

        userService.registerUser(user);
    .....
    }
}
```

- 
- ◆ Spring AOP是基于AOP Alliance标准API实现的，使用Spring的AOP或基于AOP的任何特性，需要
 - aopalliance.jar

◆使用Spring的AOP，集成AspectJ LTW织入器，需要

- aspectjweaver.jar

Spring5+Hibernate5

- onlineStockWebSH
 - Deployment Descriptor: onlineStockWebSH
 - JAX-WS Web Services
 - Java Resources
 - src
 - edu.nju.onlinestock.dao
 - edu.nju.onlinestock.dao.impl
 - edu.nju.onlinestock.model
 - edu.nju.onlinestock.service
 - edu.nju.onlinestock.service.impl
 - edu.nju.onlinestock.servlets
 - applicationContext.xml
 - Libraries
 - JavaScript Resources
 - build
 - WebContent
 - image
 - META-INF
 - user
 - ErrorMessage.jsp
 - register.html
 - RegUser.jsp
 - WEB-INF
 - lib

- WebContent
 - image
 - META-INF
 - user
 - WEB-INF
 - lib
 - activation-1.1.jar
 - antlr-2.7.7.jar
 - aopalliance-1.0.jar
 - aspectjweaver.jar
 - classmate-1.3.0.jar
 - commons-logging-1.2.jar
 - dom4j-1.6.1.jar
 - hibernate-commons-annotations-5.0.1.Final.jar
 - hibernate-core-5.2.12.Final.jar
 - hibernate-jpa-2.1-api-1.0.0.Final.jar
 - jarindex-2.0.3.Final.jar
 - javassist-3.20.0-GA.jar
 - javax.xml.bind.jar
 - jaxb-impl-2.1.jar
 - jboss-logging-3.3.0.Final.jar
 - jboss-transaction-api-1.2_spec-1.0.1.Final.jar
 - spring-aop-5.0.2.RELEASE.jar
 - spring-aspects-5.0.2.RELEASE.jar
 - spring-beans-5.0.2.RELEASE.jar
 - spring-context-5.0.2.RELEASE.jar
 - spring-context-indexer-5.0.2.RELEASE.jar
 - spring-context-support-5.0.2.RELEASE.jar
 - spring-core-5.0.2.RELEASE.jar
 - spring-expression-5.0.2.RELEASE.jar
 - spring-instrument-5.0.2.RELEASE.jar
 - spring-jcl-5.0.2.RELEASE.jar
 - spring-jdbc-5.0.2.RELEASE.jar
 - spring-jms-5.0.2.RELEASE.jar
 - spring-messaging-5.0.2.RELEASE.jar
 - spring-orm-5.0.2.RELEASE.jar
 - spring-oxm-5.0.2.RELEASE.jar
 - spring-test-5.0.2.RELEASE.jar
 - spring-tx-5.0.2.RELEASE.jar
 - spring-web-5.0.2.RELEASE.jar
 - spring-webflux-5.0.2.RELEASE.jar
 - spring-webmvc-5.0.2.RELEASE.jar
 - spring-websocket-5.0.2.RELEASE.jar

或者

- ◆ 使用Spring的XML配置文件注入Bean，不使用注解注入

- 注：Bean之间的依赖关系

applicationContext.xml

.....

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
```

```
    <property name="configLocation">
```

```
        <value>classpath:hibernate.cfg.xml</value>
```

```
    </property>
```

```
</bean>
```

```
<bean id="BaseDao"
class="edu.nju.onlinestock.dao.impl.BaseDaoImpl">
```

```
    <property name="sessionFactory">
```

```
        <ref bean="sessionFactory" />
```

```
    </property>
```

```
</bean>
```

```
<bean id="UserDao"
class="edu.nju.onlinestock.dao.impl.UserDaoImpl">
    <property name="BaseDao">
        <ref bean="BaseDao" />
    </property>
</bean>

<bean id="UserManageService"
class="edu.nju.onlinestock.service.UserManageServiceBean">
    <property name="UserDao">
        <ref bean="UserDao" />
    </property>
</bean>
.....
```


BaseDaoImpl

```
import org.hibernate.SessionFactory;

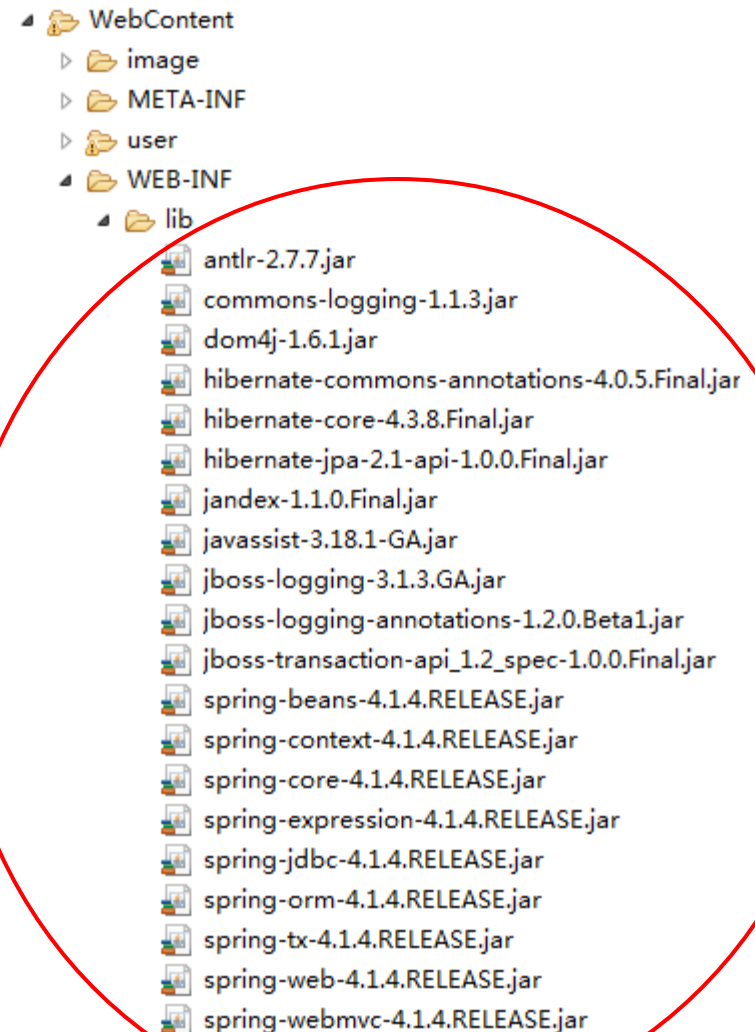
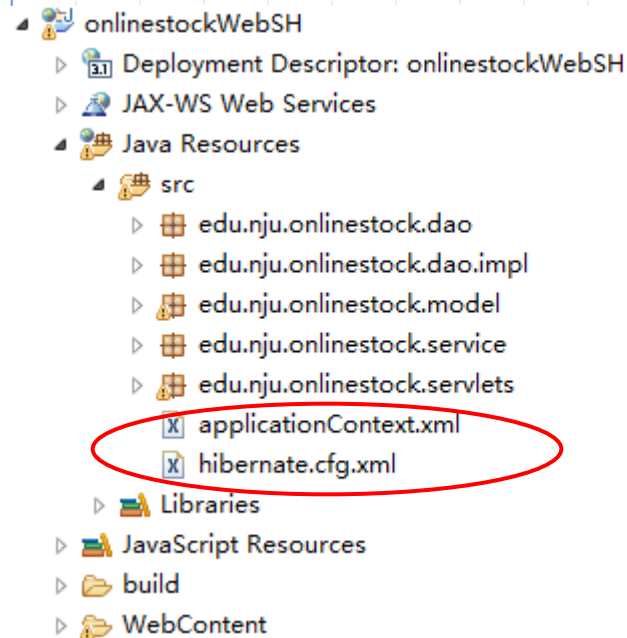
public class BaseDaoImpl implements BaseDao {
    private SessionFactory sessionFactory;

    public SessionFactory getSessionFactory() {
        return sessionFactory;
    }

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

    .....
}
```

Spring+Hibernate4



Spring整合Hibernate3

◆ Hibernate3以下

◆ DAO抽象

- 提供了一致的抽象，让不同的数据访问对象中使用相同的基类，统一了Dao类中的模板使用，开发者可以直接使用基类中提供的方法，而无需关注与具体方法的实现方式
- 从设计层面上讲，隐藏了实现的细节，将开发者的任务集中在业务实现上，如：
 - ◆ JdbcDaoSupport, HibernateDaoSupport

示例

- ◆ 对照hibernate. ppt
- ◆ 表tblUser
- ◆ User类
- ◆ User. hbm. xml映射文件
- ◆ applicationContext.xml : Spring
bean 配置文件
- ◆ UserManager类: DAO

UserManager

```
public class UserManager extends
HibernateDaoSupport {
    public UserManager () { super (); }
    public User getUser (String userName) throw
Exception {
        User u=new User ();
        u. setUsername (userName) ;
        List
list=this. getHibernateTemplate (). findByExample (
u) ;
```

```
        if (list.isEmpty())
            throw new Exception( "No Such Record" );
        else
            return (User) list.get(0);
    }
    public void add(User user) {
        super. getHibernateTemplate().save(u);
    }
    public void updateUser(User user) {
        this. getHibernateTemplate().update(u);
    }
    public void deleteUser(User user) {
        this. getHibernateTemplate().delete(u);
    }
}
```

applicationContext.xml

◆ 以Bean的方式定义JDBC DataSource、
Hibernate SessionFactory

.....


<bean>

```
<bean id= "datasource"  
class= "org.apache.commons.dbcp.BasicDataS  
ource" >
```

```
    <property name= "driverClassName" >
```

```
        <value>com.mysql.jdbc.Driver</value>
```

```
    </property>
```



```
<property name= "url" >  
    <value> jdbc:mysql://localhost:3306/abcdef  
    </value>  
</property>  
<property name= "username" >  
    <value> root </value>  
</property>  
<property name= "password" >  
    <value> </value>  
</property>  
</bean>
```



```
<bean id= "sessionFactory"  
class= "org.springframework.com.hibernate3  
.LocalSessionFactoryBean" >  
    <property name= "dataSource" >  
        <ref bean= "dataSource" />  
    </property>  
    <property name= "mappingResources" >  
        <list>  
            <value>User.hbm.xml</value>  
        </list>  
    </ property >
```

```
<property name= "hibernateProperties" >
  <props>
    <prop key= "hibernate.dialect" >
      org.hibernate.dialect.MySQLDialect
    </prop>
    <prop key= "hibernate.show_sql" >
      true
    </prop>
  </props>
</property>
</bean>
```

```
<bean id= "UserManager"  
class= "edu.nju.hbn.UserManager" >  
  <property name= "sessionFactory" >  
    <ref bean= "sessionFactory" />  
  </property>  
</bean>  
</bean>
```

◆ 在Spring中注册UserManager，然后向它注入sessionFactory

测试程序MainTest

```
public static void main(String[] args.) {  
    try{  
        //获得Spring应用上下文  
        ApplicationContext context=new  
        FileSystemXmlApplicationContext( “applicationContext.xml” );  
        User u=new User();  
        u.setUsername( “user005” );  
        .....  
        ((UserManager)context.getBean( “UserManager” )).add(u);  
    }catch(Exception e)  
    {  
        e.printStackTrace();  
    }  
}
```

Web模块

- ◆ 建立在应用上下文模块基础上，提供适合Web系统的上下文
- ◆ 支持多项面向web的任务，如
 - 透明处理多文件上传请求
 - 自动将请求参数绑定到业务对象中
- ◆ 对Jakarta **Struts**的集成支持
 - Struts. ppt

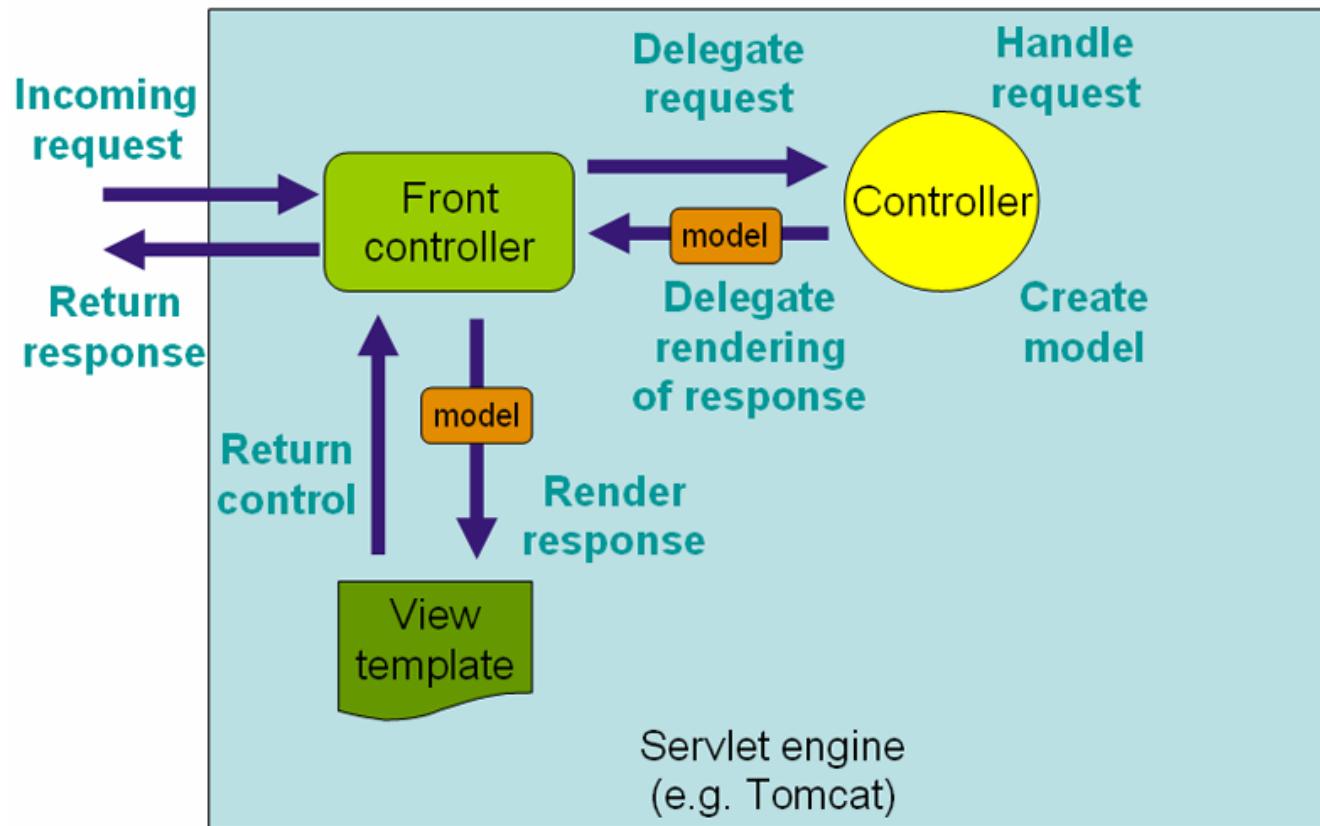
Spring MVC

◆ MVC封装包

- 提供了Web应用的Model-View-Controller实现

◆ Spring的MVC框架并不是仅仅提供一种传统的实现，它提供了一种清晰地分离模型。并且还可以借助Spring框架的其他特性——信息国际化，和验证服务等

Request Processing Workflow



The requesting processing workflow in Spring Web MVC (high level)

1. Front controller

- ◆ 在web.xml文件中配置

- ◆ DispatcherServlet, 功能

- 1、拦截匹配的请求, 分发到目标Controller处理
 - ◆ 扫描使用@Controller注解的类, 和类中使用@RequestMapping注解的方法, 匹配该请求的URL
 - ◆ 基于方法的拦截
- 2、根据Controller的返回值, 定位到相应的view/页面
 - ◆ 指明配置了视图解析器InternalResourceViewResolver的xml文件

web.xml

.....

<servlet>

 <servlet-name>spring</servlet-name>

 <servlet-class>org.springframework.web.servlet.**DispatcherServlet**</servlet-class>

 <init-param>

 <param-name>contextConfigLocation</param-name>

 <param-value>**/WEB-INF/spring-servlet.xml**</param-value>

 </init-param> </servlet>

<servlet-mapping>

 <servlet-name>spring</servlet-name>

 <url-pattern>**/**</url-pattern>

</servlet-mapping>

.....

spring-servlet.xml

.....

<!--自动扫描base-pack下或子包下的Java文件，如果扫描到有@Component
@Controller@Service等注解的类，则把这些类注册为bean -->

<context:component-scan base-package="edu.nju.onlinestock" />

<!-- don't handle the static resource -->

<mvc:default-servlet-handler />

<!-- if you use annotation you must configure following setting -->

<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.*InternalResourceViewResolver*">

 <property name="prefix" value="/user/" />

 <property name="suffix" value=".jsp" />

</bean>

.....

2. Controller

◆ @Controller

- 用于标注控制层组件
- 以注册用户为例
 - ◆ 提交注册请求——register.html
 - ◆ Controller——UserController.java
 - ◆ 错误页面——ErrorMessage.jsp
 - ◆ 注册成功页面——RegUser.jsp

register.html

.....

```
<form name="f1" id="f1" action="..register"  
method=post>
```

```
  <table align="center" border="0">
```

```
    <tr>
```

```
      <td>User id</td>
```

```
      <td><input type="text" name="user id"  
size=25></td>
```

```
    </tr>
```

```
      .....
```

```
    .....
```

```
.....
```

```
.....
```

UserController示例

.....

@Controller

public class UserController{

 @Autowired

 private UserManagerService userService; //注入Service

.....

@RequestMapping(value="/register", method= RequestMethod.*POST*)

 protected String doRegist(**@RequestParam**("userid") String id,
 //其他参数.....

ModelMap model, HttpServletRequest request) {

 model.addAttribute("mess", message); return "ErrorMessage";

 userService.registerUser(user);

 session = request.getSession(true); session.setAttribute("user",
user);

 return "RegUser";

 }

}

ModelMap

- ◆ 用于传递Controller方法的处理数据到结果页面
- ◆ 作用域类似于request对象
- ◆ 方法：
`addAttribute(String key, Object value);`

ErrorMessage.jsp

- 使用request对象获取ModelMap的属性

.....

```
<H1><%=request.getAttribute("mess")  
%></H1>
```

.....

SpringMVC

◆ Controller

- @Controller
- 不用servlet

◆ Model

- User
- Dao
 - ◆ 使用 *JdbcTemplate*
 - ◆ 在 `spring-servlet.xml` 中配置

■ Service

◆ View

- `/user/*.jsp`

- ▼ onlineStockWebSpringMVC
 - > Deployment Descriptor: onlineStockWebSpringMVC
 - > JAX-WS Web Services
 - ▼ Java Resources
 - ▼ src
 - ▼ edu.nju.onlinestock.controller
 - > UserController.java
 - ▼ edu.nju.onlinestock.dao
 - > UserDao.java
 - > UserDaoImpl.java
 - > edu.nju.onlinestock.model
 - > edu.nju.onlinestock.service
 - > Libraries
 - > JavaScript Resources
 - > build
 - ▼ WebContent
 - > image
 - ▼ META-INF
 - MANIFEST.MF
 - ▼ user
 - ErrorMessage.jsp
 - register.html
 - RegUser.jsp
 - ▼ WEB-INF
 - > lib
 - spring-servlet.xml
 - > web.xml

获取request等对象的方法

◆ Requet, Response, Session, InputStream, OutputStream等

◆ 1、直接作为Controller的方法的参数

- 如本示例

◆ 2、自动注入

@Autowired

```
private HttpServletRequest request;
```

SpringMVC的线程安全问题

◆ Controller默认是Singleton

- 应避免在controller中定义实例变量

◆ 解决方案

- 1、使用ThreadLocal变量；
- 2、声明 `scope="prototype"`；

`@Controller`

`@Scope("prototype")`

`public class XXController {...}`