链滴

学习笔记 --Spring 框架

作者：KioLuo

原文链接：https://ld246.com/article/1492568343596

来源网站：链滴

//概述

轻量级，一站式，开发框架

IoC，Inversion of Control，控制反转

DI，Dependency Injection，依赖注入

AOP，Aspect-Oriented Programming，面向切面编程：业务逻辑与非业务逻辑分离，如日志、安全..

IoC容器：

对象创建、装配

对象生命周期管理

上下文环境

//IoC容器

IoC = ApplicationContext（org.springframework.context，spring-context)

初始化

ApplicationContext context = new ClassPathXmlApplicationContext("application-context.xml");

或

ApplicationContext context = new FileSystemXmlApplicationContext("/home/user/conf/application-context.xml");

或在web.xml中

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:application-context.xml</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

Bean定义

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
```

```
    http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="screwDriver" class="com.netease.course.ScrewDriver"></bean>

</beans>
```

Bean使用

```
//初始化容器
ApplicationContext context = new ClassPathXmlApplicationContext("application-context.xml");
//获取对象
ScrewDriver screwDriver = context.getBean("screwDriver", ScrewDriver.class);
//使用对象
screwDriver.use();
```

Bean作用域

singleton，单例

```
<bean id="screwDriver" class="com.netease.course.ScrewDriver" scope="singleton"></bean
```

默认为单例
prototype，每次引用创建一个实例

```
<bean id="screwDriver" class="com.netease.course.ScrewDriver" scope="prototype"></bea
>
```

request scope, requestBean
session scope, sessionBean
application scope, appBean
global scope

Bean生命周期回调

创建，申请资源
可以通过实现接口

```
public interface InitializingBean {
    void afterPropertiesSet() throws Exception;
}
```

或者直接在application-context.xml中配置

```
<bean id="screwDriver" class="com.netease.course.ScrewDriver" init-method="init"></bean
```

对应代码

```
public class ScrewDriver {
   public void init() {
       System.out.println("Init screwDriver");
   }
}
```

销毁

可以通过实现接口

```
public interface DisposableBean {
    void destroy() throws Exception;
}
```

或者直接在application-context.xml中配置

```
<bean id="screwDriver" class="com.netease.course.ScrewDriver" destroy-method="cleanup"
</bean>
```

对应代码

```
public class ScrewDriver {
   public void cleanup() {
       System.out.println("Cleanup screwDriver");
   }
}
```

关闭Bean

```
((ConfigurableApplicationContext) context).close();
```

依赖注入

构造函数，强依赖

Setter函数，可选依赖

配置bean的类的构造函数的参数

```
 <bean id="straightHeader" class="com.netease.course.StraightHeader">
     <constructor-arg value="red"></constructor-arg>
     <constructor-arg value="15"></constructor-arg>
  </bean>
```

或

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg index="0" value="red"></constructor-arg>
   <constructor-arg index="1" value="15"></constructor-arg>
</bean>
```

或

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg type="java.lang.String" value="red"></constructor-arg>
   <constructor-arg type="int" value="15"></constructor-arg>
</bean>
```

或

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg name="color" value="red"></constructor-arg>
   <constructor-arg name="size" value="15"></constructor-arg>
</bean>
```

需要传递集合类型的构造函数参数（如map）时

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg>
      <map>
         <entry key="color" value="red"></entry>
         <entry key="size" value="15"></entry>
      </map>
   </constructor-arg>
   <constructor-arg name="size" value="15"></constructor-arg>
</bean>
```

传入list时

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg>
      <list>
         <value>red</value>
         <value>15</value>
      </list>
   </constructor-arg>
   <constructor-arg name="size" value="15"></constructor-arg>
</bean>
```

传入Properties时

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
   <constructor-arg>
      <props>
         <prop key="color">red</prop>
         <prop key="size">15</prop>
```

```
      </props>
    </constructor-arg>
    <constructor-arg name="size" value="15"></constructor-arg>
</bean>
```

从外部倒入配置时

```
  <bean id="straightHeader" class="com.netease.course.StraightHeader">
      <constructor-arg name="color" value="${color}"></constructor-arg>
      <constructor-arg name="size" value="${size}"></constructor-arg>
  </bean>
  <bean id="headerProperties" class="org.springframework.beans.factory.config.PropertyPlac
holderConfigurer">
    <property name="location" value="classpath:header.properties" />
</bean>
```

在一个bean中添加所依赖的bean

```
<bean id="screwDriver" class="com.netease.course.ScrewDriver">
    <constructor-arg>
        <ref bean="straightHeader" />
    </constructor-arg>
</bean>
```

通过Setter方法注入依赖

```
<bean id="straightHeader" class="com.netease.course.StraightHeader">
    <property name="color" value="${color}"></property>
    <property name="size" value="${size}"></property>
</bean>
```

自动装配

constructor是按byType方式注入

```
   <bean id="screwDriver" class="com.netease.course.ScrewDriver" autowire="constructor">
    </bean>
```

或

```
   <bean id="screwDriver" class="com.netease.course.ScrewDriver" autowire="byName">
    </bean>
```

或

```
 <bean id="screwDriver" class="com.netease.course.ScrewDriver" autowire="byType">
  </bean>
```

Annotation

@Component：定义Bean，或@Component("name")

@Value：properties注入

@Autowired & @Resource：自动装配依赖

@PostConstruct & @PreDestroy：生命周期回调

在xml中加入

```
<context:component-scan base-package="com.netease.course" />
```

//AOP技术

AOP术语

Aspect：日志、安全等功能

Join point：函数执行或者属性访问

Advice：在某个函数执行点上要执行的切面功能

Pointcut：匹配横切目标函数的表达式

Advice类型

Before：函数执行之前

After returning：函数正常返回之后

After throwing：函数抛出异常之后

After finally：函数返回之后

Around：函数执行前后

Spring AOP

非完整AOP实现

整合AOP和与IoC

XML schema-based AOP

@AspectJ annotation-based AOP

@AspecsJ annotation-based AOP

aspectjweaver.jar

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">

    <aop:aspectj-autoproxy />

</beans>
```

定义Aspect

```
  <bean id="loggingAspect" class="com.netease.course.LoggingAspect">
        <!-- configure properties of aspect here as normal -->
  </bean>
```

相应代码中类名前加入@Aspect

定义Pointcut

```
@Pointcut("execution(* com.netease.course.Calculator.*(..))")

private void arithmetic() {}
```

Pointcut表达式

designator(modifiers? return-type declaring-type? name(param) throws?)

designator: execution, within

modifiers: public, private

return-type: 返回类型，*

declaring-type: 包名，类名

name: 函数名，*

param: 参数列表，()无参，(..)任意参数

throws: 异常类型

可以组合

定义Advice

```
@Before("com.netease.course.LoggingAspect.arithmetic()")

public void doLog() {

  //

}
```

或

```
@Before("execution(* com.netease.course.Calculator.*(..))")
public void doLog() {
    //
}
```

或

```
@AfterReturning("com.netease.course.LoggingAspect.arithmetic()")
public void doLog() {
    //
}
```

或

```
@AfterThrowing("com.netease.course.LoggingAspect.arithmetic()")
public void doLog() {
    //
}
```

或

```
@After("com.netease.course.LoggingAspect.arithmetic()")
public void doLog() {
    //
}
```

Advice参数

函数上下文信息

```
@Before("com.netease.course.LoggingAspect.arithmetic()")
public void doLog(JoinPoint jp) {
    System.out.println(jp.getSignature() + ", " + jp.getArgs());
```

```
}
```

或

```
@Around("com.netease.course.LoggingAspect.arithmetic()")
public void doLog(ProceedingJoinPoint pjp) {
    System.out.println("start method: " + pjp.toString());
    Object retVal = pjp.proceed();
    System.out.println("stop method: " + pjp.toString());
}
```

返回值

```
@AfterReturning(pointcut="com.netease.course.LoggingAspect.arithmetic()"),
                returning="retVal")
public void doLog(Object retVal) {
    //
}
```

异常

```
@AfterThrowing(pointcut="com.netease.course.LoggingAspect.arithmetic()"),
                throwing="ex")
public void doLog(IllegalArgumentException ex) {
    //
}
```

目标函数参数

```
@Before("com.netease.course.LoggingAspect.arithmetic() && args(a, ..)")
public void doLog(JoinPoint jp, int a) {
    //
}
```

XML schema-based AOP

定义Aspect和PointCut

```
 <aop:config>
    <aop:aspect id="loggingAspect" ref="loggingBean">
       <aop:pointcut id="arithmetic" expression="execution(* com.netease.course.Calculator.*(
.))" />
    </aop:aspect>
 </aop:config>
```

定义Advice

```
 <aop:config>
    <aop:aspect id="loggingAspect" ref="loggingBean">
       <aop:before pointcut-ref="arithmetic" method="doLog" />
    </aop:aspect>
 </aop:config>
```

或

```
 <aop:config>
    <aop:aspect id="loggingAspect" ref="loggingBean">
       <aop:before pointcut="execution(* com.netease.course.Calculator.*(..))" method="doLo
" />
    </aop:aspect>
 </aop:config>
```

或

```
 <aop:config>
    <aop:aspect id="loggingAspect" ref="loggingBean">
       <aop:after-returning pointcut-ref="arithmetic" returning="retVal" method="doLog" />
    </aop:aspect>
 </aop:config>
```

或

```
 <aop:config>
    <aop:aspect id="loggingAspect" ref="loggingBean">
       <aop:after-throwing pointcut-ref="arithmetic" throwing="ex" method="doLog" />
    </aop:aspect>
 </aop:config>
```

或

```
   <aop:aspect id="loggingAspect" ref="loggingBean">
      <aop:around pointcut-ref="arithmetic" method="doLog" />
   </aop:aspect>
```

//数据访问

DAO，Data Access Object

数据访问相关接口

ORM，Object Relation Mapping

对象关系映射

DataSource (javax.sql)

DriverManagerDataSource (org.springframework.jdbc.datasource)

BasicDataSource (org.apache.commons.dbcp)

```
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-meth
d="close">
      <property name="driverClassName" value="${jdbc.driverClassName}" />
      <property name="url" value="${jdbc.url}" />
      <property name="username" value="${jdbc.username}" />
      <property name="password" value="${jdbc.password}" />
  </bean>

  <context:property-placeholder location="db.properties" />
```

JdbcTemplate (org.springframework.jdbc.core)

设置JdbcTemplate的数据库配置信息

```
private JdbcTemplate jdbcTemplate;

@Autowired
public void setDataSource(DataSource dataSource) {
   this.jdbcTemplate = new JdbcTemplate(dataSource);
}
```

查询操作

```
   int rowCount = this.jdbcTemplate.queryForObject("select count(*) from user", Integer.class);
   int countOfNamedJoe = this.jdbcTemplate.queryForObject("select count(*) from user where
first_name = ?", Integer.class, "Joe");
   String lastName = this.jdbcTemplate.queryForObject("select last_name from user where id
 ?", new Object[]{1212L}, String.class);
```

更改操作

```
this.jdbcTemplate.update("insert into user (first_name, last_name) values (?, ?)", "Meimei", "Ha
");
或
this.jdbcTemplate.execute("create table user (id integer, first_name varchar(100), last_name va
char(100))");
```

对象匹配

```
User user = this.jdbcTemplate.queryForObject("select last_name from user where id = ?",
        new Object[]{1212L},
        new RowMapper<User>() {
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User user = new User();
        user.setFirstName(rs.getString("first_name"));
        user.setLastName(rs.getString("last_name"));
        return user;
    }
});
```

或

```
List<User> users = this.jdbcTemplate.query("select last_name from user where id = ?",
        new Object[]{1212L},
        new RowMapper<User>() {
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User user = new User();
        user.setFirstName(rs.getString("first_name"));
        user.setLastName(rs.getString("last_name"));
        return user;
    }
});
```

定义JdbcTemplate

```
public class JdbcExampleDao implements ExampleDao {
    private JdbcTemplate jdbcTemplate;
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
    //...DAO接口实现
}
```

可以在xml中配置相应的bean

也可以用annotation的方法，如下：

```
@Repository
public class JdbcExampleDao implements ExampleDao {
    private JdbcTemplate jdbcTemplate;
    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
    //...DAO接口实现
}
```

NamedParameterJdbcTemplate (org.springframework.jdbc.core)

```
private NamedParameterJdbcTemplate namedParameterJdbcTemplate;
```

```
@Autowired
public void setDataSource(DataSource dataSource) {
    this.namedParameterJdbcTemplate= new NamedParameterJdbcTemplate (dataSource);
}

public int countOfUserByFirstName(String firstName) {
    String sql = "select count(*) from usertest where first_name = :first_name";
    Map<String, String> namedParameters = Collections.singletonMap("first_name", firstName
;
    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters, Integer.c
ass);
}
```

其他api接口

```
queryForObject(String sql, Map<String, ?> paramMap, RowMapper<T> rowMapper)
queryForObject(String sql, SqlParameterSource paramSource, Class<T> requiredType)

SqlParameterSource: MapSqlParameterSource, BeanPropertySqlParameterSource (org.springf
amework.jdbc.core.namedparam)
如:
public int countOfUserByFirstName(User user) {
    String sql = "select count(*) from usertest where first_name = :first_name";
    SqlParameterSource namedParameters = new BeanPropertySqlParameterSource(user);
    return this.namedParameterJdbcTemplate.queryForObject(sql, namedParameters, Integer.c
ass);
}
```

异常处理

DataAccessException, "unchecked", 是一个基类 （org.springframework.dao）

//事务管理

spring事务管理

统一的事务编程模型，编程式事务及声明式事务（AOP）

```
public interface PlatformTransactionManager {

    TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionExcep
ion;

    void commit(TransactionStatus status) throws TransactionException;

    void rollback(TransactionStatus status) throws TransactionException;

}
```

事务管理器

PlatformTransactionManager (org.springframework.transaction):

DataSourceTransactionManager (org.springframework.jdbc.datasource), JDBC

HibernateTransactionManager (org.springframework.orm.hibernate), Hibernate

TransactionDefinition

getName：事务名称

getIsolationLevel：隔离级别

getPropagationBehavior：传播行为

getTimeout：超时时间

isReadOnly：是否只读

TransactionStatus

isNewTransaction：是否是新事务

hasSavePoint：是否有savepoint（诊断，NESTED）

isCompleted：是否完成

isRollbackOnly：事务结果是否是rollback-only
setRollbackOnly：设置事务为rollback-only

隔离级别

ISOLATION_READ_UNCOMMITTED：读未提交

ISOLATION_READ_COMMITTED：读提交

ISOLATION_REPEATABLE_READ：重复读

ISOLATION_SERIALIZABLE：串行化

ISOLATION_DEFAULT：默认

传播行为

PROPAGATION_MANDATORY：必须在一个事务中运行，不存在则抛异常

PROPAGATION_NEVER：不应该在事务中运行，存在则抛异常

PROPAGATION_NOT_SUPPORTED：不应该在事务中运行，存在则挂起

PROPAGATION_SUPPORTS：不需要事务，有则在事务中执行

PROPAGATION_REQUIRED：必须在事务中执行，如果不存在，则启动新事务（内部事务会影响外事务）

PROPAGATION_NESTED：必须在事务中执行，如果不存在，则启动新事务（事务之间互不影响）

PROPAGATION_REQUIRES_NEW：必须在新事务中执行，挂起当前事务（独立physical事务）

声明式事务

添加schema

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springf
amework.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-2.0.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-2.0.xsd">
```

定义事务管理器

```
<bean id="txManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
</bean>
<context:property-placeholder location="db.properties" />
```

定义事务Advice

```
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="get*" read-only="true" />
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

定义Pointcut

```
<aop:config>
    <aop:pointcut id="daoOperation"
        expression="execution(* com.netease.course.AccountDao.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="daoOperation" />
</aop:config>
```

配置<tx:method />

name: 匹配的函数名称，支持*匹配

propagation：事务传播行为

isolation：事务隔离级别

timeout：超时

read-only：是否只读事务

rollback-for：触发回滚的异常，用逗号分隔

no-rollback-for：不触发回滚的异常

采用annotation方法

@Transactional

xml中添加

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

相应代码

```
@Transactional(propagation=Propagation.REQUIRED, rollbackFor=Exception.class)
public boolean deleteClusterByClusterId(Sring clusterId) {
    // do work
}
```

@transactional

value：使用的TransactionManager

propagation：事务传播行为

isolation：事务隔离级别

timeout：超时

readOnly：是否只读

rollbackFor：触发回滚的异常类对象数组

rollbackForClassName：触发回滚的异常类名称数组

noRollbackFor：不触发回滚的异常类对象数组

noRollbackForClassName：不触发回滚的异常类名称数组

编程式事务

定义TransactionTemplate

```
public class SimpleService implements Service {
    private final TransactionTemplate transactionTemplate;
    public SimpleService(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
        this.transactionTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_READ_UNC
MMITTED);
        this.transactionTemplate.setTimeout(30);
    }
}
```

使用TransactionTemplate

```
public Object someMethod() {
    return transactionTemplate.execute(new TransactionCallback() {
        public Object doInTransaction(TransactionStatus status) {
            updateOperation1();
            return resultOfUpdateOperation2();
        }
    });
}
```

或（不返回结果）

```
public Object someMethodWithoutResult() {
    return transactionTemplate.execute(new TransactionCallbackWithoutResult() {
        protected void doInTransactionWithoutResult(TransactionStatus status) {
            updateOperation1();
            updateOperation2();
        }
    });
}
```

或（设置为遇到异常时只能回滚）

```
public Object someMethodWithoutResult() {
    return transactionTemplate
            .execute(new TransactionCallbackWithoutResult() {
                protected void doInTransactionWithoutResult(
                        TransactionStatus status) {
                    try {
                        updateOperation1();
                        updateOperation2();
                    } catch (SomeBusinessException e) {
                        status.setRollbackOnly();
                    }
                }
            });
}
```

PlatformTransactionManager的实现

```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
def.setName("TxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

TransactionStatus status = txManager.getTransaction(def);
try {
    //do something
} catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
```

```
}
txManager.commit(status);
```

整合MyBatis

SqlSessionFactory

添加mybatis-spring依赖

```
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>1.2.3</version>
    </dependency>
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.3.0</version>
    </dependency>
```

定义SqlSessionFactoryBean

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:sqlMapConfig.xml" />
    <property name="mapperLocations" value="classpath*:sample/config/mappers/**/*.xml"
>
</bean>
```

定义Mapper

```
public interface UserMapper {

    @Select("SELECT * FROM users WHERE id = #{userId}")
    User getUser(@Param("userId") String userId);
}
```

配置结果映射

```
@Results({
    @Result(property="id", column="id"),
    @Result(property="firstName", column="first_name"),
    @Result(property="lastName", column="last_name")
})
```

或者采用xml的方法，见Mybatis部分

定义Mapper Bean

```
<bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface" value="com.netease.course.UserMapper" />
```

```
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

或采用自动发现的机制

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

    <mybatis:scan base-package="com.netease.course" />
</beans>
```

当需要指定SqlSessionFactory时

```
<mybatis:scan base-package="com.netease.course" factory-ref="sqlSessionFactory" />
```

或者

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.netease.course" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
```

使用Mapper

```
public class SomeService {
    @Autowired
    private UserMapper userMapper;
    public User getUser(String userId) {
        return userMapper.getUser(userId);
    }
}
```

定义SqlSessionTemplate使用

```
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

SqlSessionTemplate使用

```
public class UserDao {
    @Autowired
    private SqlSession sqlSession;
    public User getUser(String userId) {
```

```
        return (User) sqlSession.selectOne("com.netease.course.UserMapper.getUser", userId);
    }
}
```

//Web框架

DispatcherServlet

[servlet-name]-servlet.xml

HandlerMapping

Controllers

View解析相关

WebApplicationContext

ContextLoaderListener

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext*.xml</param-value>
</context-param>
```

Servlet WebApplicationContext (containing controllers, view resolvers, and other web-related beans)

Root WebAppliacationContext (containing middler-tier services, datasources, etc.)

实现Controller

```
@Controller
@RequestMapping(value = "/hello")
public class HelloController {
    @RequestMapping(value = "/spring")
    public void spring(HttpServletResponse response) throws IOException {
        response.getWriter().write("Hello, Spring Web!!");
    }
}
```

定义Controller

自动发现

```
<context:component-scan base-package="com.netease.course" />
```

@RequestMapping

name: 名称

value & path: 路径，如"/hello"

method: 请求方法，如"GET"

params: 请求参数

headers: 请求头

consumes: 请求的媒体类型，"Content-Type"

produces: 响应的媒体类型，"ACCEPT"

注入路径中的变量

```
@RequestMapping(value="/users/{userId}")
public String webMethod(@PathVariable String userId) {
    //do work
}
```

或

```
@RequestMapping(value="/users/{userId:[a-z]+}")
public String webMethod(@PathVariable String userId) {
    //do work
}
```

函数参数

HttpServletRequest / HttpServletResponse, HttpSession (Servlet API)

Reader / Writer

@PathVariable

@RequestParam

@RequestHeader

HttpEntity

@RequestBody

Map / Model / ModelMap

函数返回值

void

String: view名称，@ResponseBody

HttpEntity

View

Map

Model

ModelAndView

函数实现

```
@RequestMapping(value="/spring/{user}")
```

```
public void helloSpring(
        @PathVariable("user") String user,
        @RequestParam("msg") String msg,
        @RequestHeader("host") String host,
        HttpServletRequest request,
        Writer writer) throws IOException {
    writer.write("URI: " + request.getRequestURI());
    writer.write("Hello, " + user + ": " + msg + ", host=" + host);
}
```

或

```
@RequestMapping(value="/spring/login")
public void login(@ModelAttribute User user, Writer writer) {
    //do work
}
```

或

```
@RequestMapping(value="/users/login")
public String login(@RequestParam("name") String name, @RequestParam("password") Strin
 password, ModelMap map) {
    map.addAttribute("name", name);
    map.addAttribute("password", "******");
    return "user";
}
```

ModelMap

ModelMap的实例是由bboss mvc框架自动创建并作为控制器方法参数传入，用户无需自己创建。

```
public String xxxxmethod(String someparam,ModelMap model)
{
    //省略方法处理逻辑若干
     //将数据放置到ModelMap对象model中,第二个参数可以是任何java类型
     model.addAttribute("key",someparam);
```

```
    ......
    //返回跳转地址
     return "path:handleok";
}
```

ModelAndView

ModelAndView的实例是由用户手动创建的，这也是和ModelMap的一个区别。

```
public ModelAndView xxxxmethod(String someparam)
{
    //省略方法处理逻辑若干
     //构建ModelAndView实例，并设置跳转地址
     ModelAndView view = new ModelAndView("path:handleok");
     //将数据放置到ModelAndView对象view中,第二个参数可以是任何java类型
     view.addObject("key",someparam);
    ......
    //返回ModelAndView对象view
     return view;
}
```

上传文件

定义bean

```
<bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="100000" />
</bean>
```

相应代码

```
@RequestMapping(value="/form", method=RequestMethod.POST)
public String handleFormUpload(@RequestParam("file") MultipartFile file) {
    // save the file
}
```

相应依赖

```
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>
```

HttpEntity

```
@RequestMapping(value="/something")
public ResponseEntity<String> handle(HttpEntity<byte[]> requestEntity) {
    String requestHeader = requestEntity.getHeaders().getFirst("MyRequestHeader");
    byte[] requestBody = requestEntity.getBody();
```

```
    // do something with requestHeader and requestBody
    HttpHeaders responseHeaders = new HttpHeaders();
    responseHeaders.set("MyResponseHeader", "MyValue");
    return new ResponseEntity<String>("hello spring", responseHeaders, HttpStatus.CREATED);
}
```

@RequestBody & @ResponseBody

```
@RequestMapping(value="/spring")
@ResponseBody
public String spring(@RequestBody String body) throws IOException {
    return "hello" + body;
}
```

MessageConverter，返回Java对象的转化

RequestBody ---> Object: 参数

ResponseBody <--- Object: 返回值

xml文件配置

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.spr
ngframework.org/schema/context"
    xmlns:mvc= “http://www.springframework.org/schema/mvc”
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
    <context:component-scan base-package="com.netease.course" />
    <mvc:annotation-driven />
</beans>
```

添加相应依赖，如JSON相关依赖

```
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.6.4</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.6.4</version>
    </dependency>
```

View解析

String　　名称

View

ModelAndView

Map

Model

ViewResolver (org.springframework.web.servlet)：

InternalResourceViewResolver (org.springframework.web.servlet.view)

FreeMarkerViewResolver (org.springframework.web.servlet.view.freemarker)

ContentNegotiatingViewResolver (org.springframework.web.servlet.view)

InternalResourceViewResovler

Servlet, JSP

bean定义

```
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

resultView -------> /WEB-INF/jsp/resultView.jsp

FreeMarkerViewResolver

FreeMarker

bean定义

```
<bean id="freemarkerConfig"
    class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
    <property name="templateLoaderPath" value="/WEB-INF/freemarker/" />
</bean>
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
    <property name="cache" value="true" />
    <property name="prefix" value="" />
    <property name="suffix" value=".ftl" />
    <property name="contentType" value="text/html; charset=utf-8" />
</bean>
```

ContentNegotiatingViewResolver

ViewResovler的组合

扩展名：user.json, user.xml, user.pdf

Accept头（media types）：application / json, application / xml

bean定义

```
<bean
   class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
   <property name="viewResolvers">
      <list>
         <bean id="viewResolver"
            class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
            <property name="cache" value="true" />
            <property name="prefix" value="" />
            <property name="suffix" value=".ftl" />
         </bean>
      </list>
   </property>
   <property name="defaultViews">
      <list>
         <bean
            class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />
      </list>
   </property>
</bean>
```

DefaultRequestToViewNameTranslator

example/admin/index.html ---------> admin/index

example/display.html -------> display

ContentNegotiatingViewResolver 根据路径后缀，选择不同视图

方法一：使用扩展名

http://localhost:8080/learn/user.xml 获取xml类型数据

http://localhost:8080/learn/user.json 获取json类型数据

http://localhost:8080/learn/user 使用默认view呈现，如jsp

方法二：使用http 请求头的Accept

GET /user HTTP/1.1

Accept:application/xml

GET /user HTTP/1.1

Accept:application/json

方法三：使用参数

http://localhost:8080/learn/user?format=xml

http://localhost:8080/learn/user?format=json

同一资源，多种表述

```
<bean id="contentNegotiationManager" class="org.springframework.web.accept.ContentNe
```

```
otiationManagerFactoryBean">
    <!-- 是否启用扩展名支持，默认为true -->
    <property name="favorPathExtenion" value="true" />
    <!-- 是否启用参数支持，默认为true -->
    <property name="favorParameter" value="false" />
    <!-- 是否忽略掉accept header,默认为false -->
    <property name="ignoreAcceptHeader" value="true" />

    <!-- 扩展名到mimeType的映射 -->
    <property name="mediaTypes">
        <map>
            <!-- 例如：/user.json 中的 .json 会映射到 application/json -->
            <entry key="json" value="application/json" />
            <entry key="xml" value="application/xml" />
        </map>
    </property>

    <!-- 如果所有mediaType都没匹配上，就使用defaultContentType -->
    <property name="defaultContentType" value="text/html"/>
</bean>

<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
    <!-- 解析器的执行顺序 -->
    <property name="order" value="1" />
    <property name="contentNegotiationManager" ref="contentNegotiationManager" />

    <property name="defaultViews">
        <list>
            <bean class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"
</bean>
            <bean class="org.springframework.web.servlet.view.xml.MarshallingView">
                <constructor-arg>
                    <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
                        <property name="classesToBeBound">
                            <list>
                                <value>com.learn.model.User</value>
                            </list>
                        </property>
                    </bean>
                </constructor-arg>
            </bean>
        </list>
    </property>

</bean>
<!-- 上面没匹配到则会使用这个视图解析器 ,解析为jsp  -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="order" value="2" />
    <property name="prefix" value="/" />
    <property name="suffix" value=".jsp"/>
</bean>
```

Controller中可以用RequestMapping匹配多个路径后缀

```
@RequestMapping(value = {"/users", "/users.html", "/users.json"})
public String getUsersInfo(ModelMap map) {
    List users = userServiceImp.getUsers();
    map.addAttribute("users", users);
    return "users";
}
```

FreeMarker

模板引擎

数据模型

对象：hashes

基本类型：scalars

注释

<#-- 这是注释 -->

插值：表达式

${animal.name}

直接指定值：

字符串，如“Zoo”，'Zoo'

数字，如123.45

布尔值，如true，false

序列，如["zoo", "bar", 123]

值域，如0..9，0..<10

哈希表，如{"name":"green mouse", "price":150}

检索变量：

顶层变量：user

哈希表数据：user.name, user["name"]

列表数据：products[5]

连接操作：

```
users + "guest"
passwords + "joe":"secret123"
```

算术操作

逻辑操作

比较操作

FTL标签：指令

开始标签：<#directivename parameters>

结束标签：

if指令

```
<#if user=="Big Joe">, our beloved leader
```

list指令

```
<#list animals as animal>
  ${animal.name}${animal.price} Euros
```

include指令

```
<#include "/copyright_footer.html">
```

使用

创建配置

```
Configuration cfg = new Configuration(Configuration.VERSION_2_3_0);
cfg.setDirectoryForTemplateLoading(new File("/where/you/store/templates"));
cfg.setDefaultEncoding("UTF-8");
cfg.setTemplateExceptionHandler(TemplateExceptionHandler.RETHROW_HANDLER);
```

定义模板

定义数据模型

如

```
Map root = new HashMap<>();
root.put("user", "Big Joe");
```

```
Map latestProduct = new HashMap<>();
root.put("latestProduct", latestProduct);
latestProduct.put("url", "products/greenmouse.html");
latestProduct.put("name", "greenmouse");
```

添加依赖

```
  <dependency>
     <groupId>org.freemarkergroupId>
     <artifactId>freemarkerartifactId>
     <version>2.3.23version>
  dependency>
```

输出结果

```
    Template ftl = cfg.getTemplate("user.ftl");
    Writer outWriter = new OutputStreamWriter(System.out);
    ftl.process(root, outWriter);
```

工程模板

一般java目录

com.netease.course.dao

com.netease.course.meta

com.netease.course.service

com.netease.course.service.impl

com.netease.course.utils

com.netease.course.web.controller

com.netease.course.web.filter