Maven实战

# 传递依赖冲突解决

传递依赖：A（项目）依赖B，B依赖C（１.１版本），B是A的直接依赖，C是Ａ的传递依赖。

　　　　　　　　　依赖Ｄ，Ｄ依赖Ｃ（１.２版本）

## Maven自己的调解原则

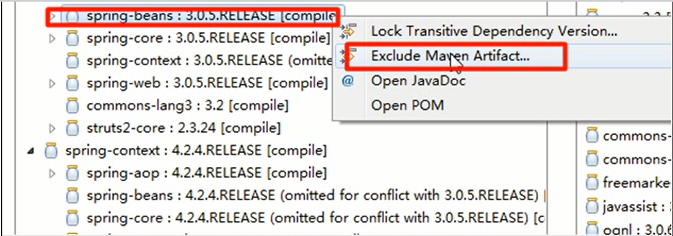
### 第一声明者优先原则

谁先声明就用谁的传递依赖

### 路径近者优先

直接依赖级别高于传递依赖

## 排除依赖



<!-- 将struts2-spring-plugin.jar中spring-beans的传递依赖排除掉 -->

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

</exclusion>

</exclusions>

## 版本锁定

<!-- 版本锁定 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

</dependencies>

</dependencyManagement>

将版本信息定义为变量

<!-- 属性 -->

<properties>

<webVersion>3.1</webVersion>

<spring.version>4.2.4.RELEASE</spring.version>

</properties>

<!-- 版本锁定 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

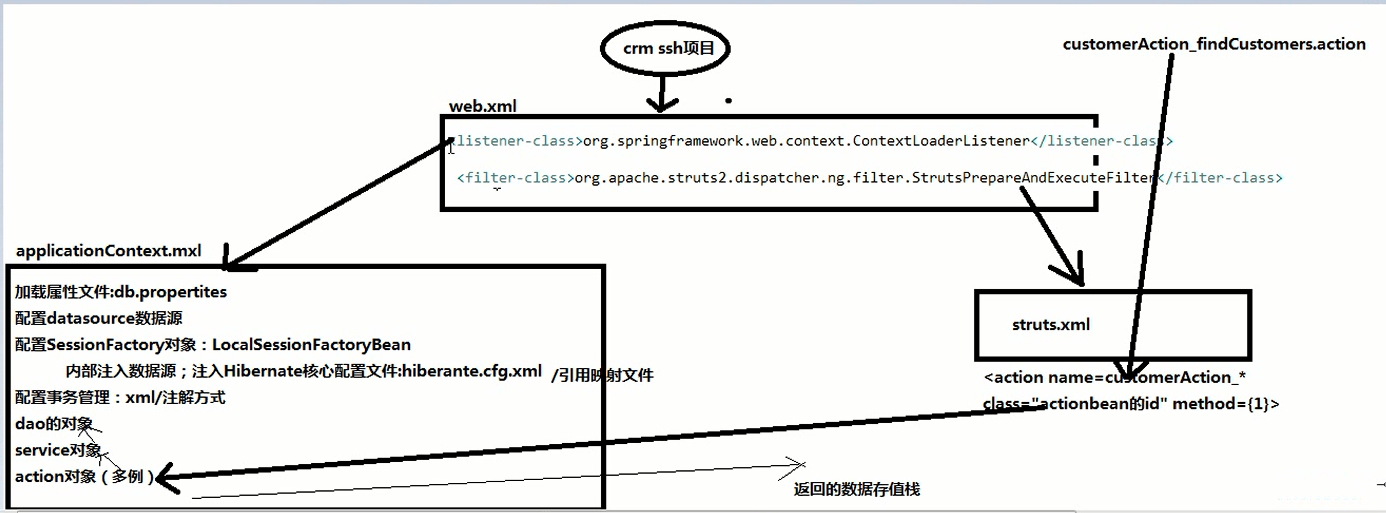
<version>${spring.version}</version>

</dependency>

</dependencies>

</dependencyManagement>

# Maven整合SSH框架



## 搭建struts2环境

### 创建struts2配置文件

### 在web.xml中配置struts2的核心过滤器

<!-- 配置struts2框架核心过滤器 -->

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

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

</filter-mapping>

## 搭建spring环境

### 创建spring配置文件applicationContext.xml

### 在web.xml中配置监听器ContextLoaderListener

<!-- 配置监听器 -->

<listener>

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

</listener>

<!-- 通过上下文参数指定spring配置文件的路径 -->

<context-param>

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

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

## 搭建Hibernate环境

### 创建hibernate核心的配置文件

<?xml version=*'1.0'* encoding=*'utf-8'*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<!-- SQL dialect -->

<property name=*"dialect"*>org.hibernate.dialect.MySQLDialect</property>

<!-- Echo all executed SQL to stdout -->

<property name=*"show\_sql"*>true</property>

<!-- 是否对日志中输出的sql语句进行格式化 -->

<property name=*"hibernate.format\_sql"*>true</property>

<!-- Drop and re-create the database schema on startup -->

<property name=*"hbm2ddl.auto"*>update</property>

</session-factory>

</hibernate-configuration>

## Struts2和spring进行整合

整合的关键点：action对象的创建交给spring

### 创建action类

### 将action对象配置到spring配置文件中

### 在struts.xml中在action节点中class属性配置为spring工厂中action对象bean的id

## Spring和hibernate框架的整合

整合关键点：1. 数据源dataSource交给spring

2.SessionFactory对象创建交给spring创建

3.事务管理

### 配置数据源

<!-- 配置数据源 -->

<bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>

<property name=*"dirverClass"* vlaue=*"${jdbc.driverClass}"*></property>

<property name=*"jdbcUrl"* value=*"${jdbc.jdbUrl}"*></property>

<property name=*"user"* value=*"${jdbc.user}"* ></property>

<property name=*"password"* value=*"${jdbc.password}"*></property>

</bean>

### 配置sessionFactory对象

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<property name=*"configLocations"* vlaue=*"classpath:applicationContext.xml"*></property>

</bean>

### 事务管理

#### 配置事务管理器

<!-- 配置事务管理器 -->

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

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

</bean>

<!-- xml方式管理事务 -->

<!-- 配置通知 ：具体增强逻辑-->

<tx:advice id=*"txAdvice"*>

<tx:attributes>

<!-- 匹配业务类中方法的名称 -->

<tx:method name=*"save\*"*/>

<tx:method name=*"update\*"*/>

<tx:method name=*"delete\*"*/>

<tx:method name=*"find\*"* read-only=*"true"*/>

<tx:method name=*"\*"*/>

</tx:attributes>

</tx:advice>

<!-- 配置aop -->

<aop:config>

<!-- 配置切点：具体哪些方法要增强？（真正被增强的方法） -->

<aop:pointcut expression=*"execution(\* com.sk.service.\*.\*(..))"* id=*"cut"*/>

<!-- 配置切面 ：将增强逻辑作用到切点（通知+切入点）-->

<aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"cut"*/>

</aop:config>

<!-- 注解的方式管理事务 -->

<!-- 1. 开启一个注解的扫描 2.在service类上或者方法上使用注解@transactional-->

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

<!-- 注解的方式管理事务 -->

# 对maven项目进行拆分聚合

## 创建父工程

## 将父工程发布到本地仓库

## 创建子模块

### 将配置文件进行了拆分

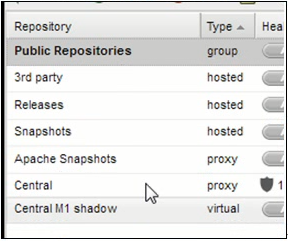
# 私服应用

## 私服安装

### 准备安装包

### 解压

## 私服仓库类型



Hosted：宿主仓库

存放本公司开发的jar包，正式和测试版本，第三方的：存在版权问题的

Proxy：代理仓库

代理中央仓库，apache下的测试版本的jar包

Group：组仓库

将来连接组仓库。包含hosted和proxy

## 上传jar包

## 下载jar包到本地

# 总结

## Maven整合SSH

## 拆分工程

1. 将每一层的代码&配置文件全部提取一个表现独立工程

## 私服

# Maven优点

## 依赖管理。项目不需要存放jar包。项目代码大小减小

## 环境统一。导入其他项目不会报错

## 代码耦合度降低

## 项目升级

## 节省人力成本