

# 项目管理工具

## maven

## 学习目标

掌握 Maven 的传递依赖

使用 Maven 构建 SSM 项目

分模块构建 Maven 工程

掌握 Maven 私服的搭建

掌握私服中 jar 包的上传和下载

## 1 Maven 的传递依赖

### 1.1 什么是传递依赖

当 A 依赖 B、B 依赖 C，在 A 中导入 B 后会自动导入 C，C 是 A 的传递依赖，如果 C 依赖 D 则 D 也可能是 A 的传递依赖。

### 1.2 依赖版本冲突解决

#### 1.2.1 什么是依赖版本冲突

当一个项目依赖的构件比较多时，它们相互之前存在依赖，当你需要对依赖版本统一管理时如果让 maven 自动来处理可能并不能如你所愿。

## 1.2.2 依赖调解原则

maven 自动按照下边的原则调解：

### 1、第一声明者优先原则

在 pom 文件定义依赖，先声明的依赖为准。

### 2、路径近者优先原则

例如：A 依赖 spirng-beans-4.2.4，A 依赖 B 依赖 spirng-beans-3.0.5，则 spring-beans-4.2.4 优先被依赖在 A 中，因为 spring-beans-4.2.4 相对 spirng-beans-3.0.5 被 A 依赖的路径最近。

## 1.2.3 排除依赖

依赖版本冲突的问题可以通过排除依赖的方式来解决，将传递依赖中不合理的依赖排除，使用合理的依赖来构建我们的工程。

排除依赖可以在依赖的配置 dependency 中使用 exclusion 标签来进行排除操作。

## 1.2.4 锁定版本

面对众多的依赖，有一种方法不用考虑依赖路径、声明优化等因素可以采用直接锁定版本的方法确定依赖构件的版本，版本锁定后则不考虑依赖的声明顺序或依赖的路径，以锁定的版本的为准添加到工程中，此方法在企业开发中常用。锁定依赖的版本可以使用 dependencyManagement 标签来设置。

需要注意的是在工程中锁定依赖的版本并不代表在工程中添加了依赖，如果工程需要添加锁定版本的依赖则需要单独添加 <dependencies> </dependencies> 标签。

## 2 maven构建ssh工程

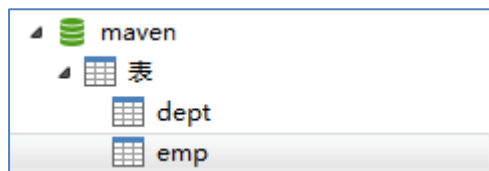
### 2.1 需求

在 web 工程的基础上实现 SSM 工程构建，实现对员工和部门的管理。

### 2.2 数据库环境

创建数据库：maven

导入，maven.sql 创建表



### 2.3 定义 pom.xml

maven 工程首先要识别依赖，web 工程实现 SSM 整合，需要依赖 Spring、springMVC、Mybatis 等，在 pom.xml 添加工程如下依赖：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.igeek.maven</groupId>
  <artifactId>ssm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <!-- 添加工程的依赖 -->
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
    <!-- servlet-api JSP页面编译时需要的包 -->
    <dependency>
```

```
<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>3.0-alpha-1</version>
<scope>provided</scope>
</dependency>
<!-- Spring 以及 SpringMVC需要引入的包，自动引入需要参照的包 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.3.6.RELEASE</version>
</dependency>
<!-- 持久层的包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.2</version>
</dependency>
<!-- Spring 和 Mybatis的整合包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.1</version>
</dependency>
<!-- Mysql驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.19</version>
</dependency>
<!-- druid数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.28</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.10</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
```

```
<version>4.3.6.RELEASE</version>
</dependency>
<!-- 打日志的 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.24</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.7.24</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
    <scope>provided</scope>
</dependency>
</dependencies>
<build>
    <finalName>ssm</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>/**/*.xml</include>
                <include>/**/*.properties</include>
            </includes>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>/**/*.xml</include>
                <include>/**/*.properties</include>
                <include>/**/*.ini</include>
            </includes>
        </resource>
    </resources>
</build>
</project>
```

```
        </includes>
    </resource>
</resources>
<plugins>
    <!-- 设置编译版本为1.8 -->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
            <encoding>UTF-8</encoding>
        </configuration>
    </plugin>

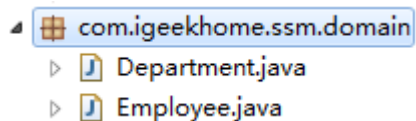
    <!-- Jetty插件，提供一种web容器 -->
    <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.4.2.v20170220</version>
        <configuration>
            <httpConnector>
                <!-- 配置运行的端口号 -->
                <port>80</port>
            </httpConnector>
            <!-- 配置扫描的时间间隔 -->
            <scanIntervalSeconds>1</scanIntervalSeconds>
            <webApp>
                <!-- 配置上下文 -->
                <contextPath>/ssm</contextPath>
            </webApp>
        </configuration>
    </plugin>
</plugins>
</build>
</project>
```

## 2.4dao

使用 Mybatis 作为持久层框架，可以使用 Mybatis 的逆向工程来生成我们需要的代码。

## 2.4.1 domain 模型类

在 src/main/java 创建模型类



部门实体

```
package com.igeekhome.ssm.domain;
import java.io.Serializable;
public class Department implements Serializable{
    private Integer deptno;
    private String dname;
    private String loc;
    public Integer getDeptno() {
        return deptno;
    }
    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }
    public String getDname() {
        return dname;
    }
    public void setDname(String dname) {
        this.dname = dname == null ? null : dname.trim();
    }
    public String getLoc() {
        return loc;
    }
    public void setLoc(String loc) {
        this.loc = loc == null ? null : loc.trim();
    }
}
```

员工实体

```
package com.igeekhome.ssm.domain;
import java.io.Serializable;
import java.math.BigDecimal;
import java.util.Date;
public class Employee implements Serializable{
    private Integer empno;
    private String ename;
    private String job;
    private Integer mgr;
```

```
private Date hiredate;
private BigDecimal sal;
private BigDecimal comm;
private Integer deptno;
public Integer getEmpno() {
    return empno;
}
public void setEmpno(Integer empno) {
    this.empno = empno;
}
public String getEname() {
    return ename;
}
public void setEname(String ename) {
    this.ename = ename == null ? null : ename.trim();
}
public String getJob() {
    return job;
}
public void setJob(String job) {
    this.job = job == null ? null : job.trim();
}
public Integer getMgr() {
    return mgr;
}
public void setMgr(Integer mgr) {
    this.mgr = mgr;
}
public Date getHiredate() {
    return hiredate;
}
public void setHiredate(Date hiredate) {
    this.hiredate = hiredate;
}
public BigDecimal getSal() {
    return sal;
}
public void setSal(BigDecimal sal) {
    this.sal = sal;
}
public BigDecimal getComm() {
    return comm;
}
public void setComm(BigDecimal comm) {
```



```
        this.comm = comm;
    }
    public Integer getDeptno() {
        return deptno;
    }
    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }
}
```

## 2.4.2Mapper 配置文件

Department 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.igeekhome.ssm.dao.DepartmentMapper" >
    <resultMap id="BaseResultMap" type="com.igeekhome.ssm.domain.Department" >
        <id column="deptno" property="deptno" jdbcType="INTEGER" />
        <result column="dname" property="dname" jdbcType="VARCHAR" />
        <result column="loc" property="loc" jdbcType="VARCHAR" />
    </resultMap>
    <sql id="Base_Column_List" >
        deptno, dname, loc
    </sql>
    <select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer" >
        select
        <include refid="Base_Column_List" />
        from dept
        where deptno = #{deptno,jdbcType=INTEGER}
    </select>
    <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >
        delete from dept
        where deptno = #{deptno,jdbcType=INTEGER}
    </delete>
    <insert id="insert" parameterType="com.igeekhome.ssm.domain.Department" >
        insert into dept (deptno, dname, loc
        )
        values (#{deptno,jdbcType=INTEGER}, #{dname,jdbcType=VARCHAR},
        #{loc,jdbcType=VARCHAR}
        )
    </insert>
```

```
<insert id="insertSelective" parameterType="com.igeekhome.ssm.domain.Department" >
    insert into dept
    <trim prefix="(" suffix=)" suffixOverrides="," >
        <if test="deptno != null" >
            deptno,
        </if>
        <if test="dname != null" >
            dname,
        </if>
        <if test="loc != null" >
            loc,
        </if>
    </trim>
    <trim prefix="values (" suffix=)" suffixOverrides="," >
        <if test="deptno != null" >
            #{deptno,jdbcType=INTEGER},
        </if>
        <if test="dname != null" >
            #{dname,jdbcType=VARCHAR},
        </if>
        <if test="loc != null" >
            #{loc,jdbcType=VARCHAR},
        </if>
    </trim>
</insert>

<update id="updateByPrimaryKeySelective"
parameterType="com.igeekhome.ssm.domain.Department" >
    update dept
    <set >
        <if test="dname != null" >
            dname = #{dname,jdbcType=VARCHAR},
        </if>
        <if test="loc != null" >
            loc = #{loc,jdbcType=VARCHAR},
        </if>
    </set>
    where deptno = #{deptno,jdbcType=INTEGER}
</update>

<update id="updateByPrimaryKey"
parameterType="com.igeekhome.ssm.domain.Department" >
    update dept
    set dname = #{dname,jdbcType=VARCHAR},
        loc = #{loc,jdbcType=VARCHAR}
    where deptno = #{deptno,jdbcType=INTEGER}
```

```
</update>
</mapper>
```

### Employee 配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.igeekhome.ssm.dao.EmployeeMapper" >
    <resultMap id="BaseResultMap" type="com.igeekhome.ssm.domain.Employee" >
        <id column="empno" property="empno" jdbcType="INTEGER" />
        <result column="ename" property="ename" jdbcType="VARCHAR" />
        <result column="job" property="job" jdbcType="VARCHAR" />
        <result column="mgr" property="mgr" jdbcType="INTEGER" />
        <result column="hiredate" property="hiredate" jdbcType="TIMESTAMP" />
        <result column="sal" property="sal" jdbcType="DECIMAL" />
        <result column="comm" property="comm" jdbcType="DECIMAL" />
        <result column="deptno" property="deptno" jdbcType="INTEGER" />
    </resultMap>
    <sql id="Base_Column_List" >
        empno, ename, job, mgr, hiredate, sal, comm, deptno
    </sql>
    <select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer" >
        select
        <include refid="Base_Column_List" />
        from emp
        where empno = #{empno,jdbcType=INTEGER}
    </select>
    <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer" >
        delete from emp
        where empno = #{empno,jdbcType=INTEGER}
    </delete>
    <insert id="insert" parameterType="com.igeekhome.ssm.domain.Employee" >
        insert into emp (empno, ename, job,
            mgr, hiredate, sal,
            comm, deptno)
        values (#{empno,jdbcType=INTEGER}, #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=INTEGER}, #{hiredate,jdbcType=TIMESTAMP},
            #{sal,jdbcType=DECIMAL},
            #{comm,jdbcType=DECIMAL}, #{deptno,jdbcType=INTEGER})
    </insert>
    <insert id="insertSelective" parameterType="com.igeekhome.ssm.domain.Employee" >
        insert into emp
        <trim prefix="(" suffix=")" suffixOverrides="," >
```

```
<if test="empno != null" >
    empno,
</if>
<if test="ename != null" >
    ename,
</if>
<if test="job != null" >
    job,
</if>
<if test="mgr != null" >
    mgr,
</if>
<if test="hiredate != null" >
    hiredate,
</if>
<if test="sal != null" >
    sal,
</if>
<if test="comm != null" >
    comm,
</if>
<if test="deptno != null" >
    deptno,
</if>
</trim>
<trim prefix="values (" suffix= ")" suffixOverrides="," >
    <if test="empno != null" >
        #{empno,jdbcType=INTEGER},
    </if>
    <if test="ename != null" >
        #{ename,jdbcType=VARCHAR},
    </if>
    <if test="job != null" >
        #{job,jdbcType=VARCHAR},
    </if>
    <if test="mgr != null" >
        #{mgr,jdbcType=INTEGER},
    </if>
    <if test="hiredate != null" >
        #{hiredate,jdbcType=TIMESTAMP},
    </if>
    <if test="sal != null" >
        #{sal,jdbcType=DECIMAL},
    </if>
```

```
<if test="comm != null" >
    #{comm,jdbcType=DECIMAL},
</if>
<if test="deptno != null" >
    #{deptno,jdbcType=INTEGER},
</if>
</trim>
</insert>
<update id="updateByPrimaryKeySelective"
parameterType="com.igeekhome.ssm.domain.Employee" >
    update emp
    <set >
        <if test="ename != null" >
            ename = #{ename,jdbcType=VARCHAR},
        </if>
        <if test="job != null" >
            job = #{job,jdbcType=VARCHAR},
        </if>
        <if test="mgr != null" >
            mgr = #{mgr,jdbcType=INTEGER},
        </if>
        <if test="hiredate != null" >
            hiredate = #{hiredate,jdbcType=TIMESTAMP},
        </if>
        <if test="sal != null" >
            sal = #{sal,jdbcType=DECIMAL},
        </if>
        <if test="comm != null" >
            comm = #{comm,jdbcType=DECIMAL},
        </if>
        <if test="deptno != null" >
            deptno = #{deptno,jdbcType=INTEGER},
        </if>
    </set>
    where empno = #{empno,jdbcType=INTEGER}
</update>
<update id="updateByPrimaryKey" parameterType="com.igeekhome.ssm.domain.Employee" >
    update emp
    set ename = #{ename,jdbcType=VARCHAR},
        job = #{job,jdbcType=VARCHAR},
        mgr = #{mgr,jdbcType=INTEGER},
        hiredate = #{hiredate,jdbcType=TIMESTAMP},
        sal = #{sal,jdbcType=DECIMAL},
        comm = #{comm,jdbcType=DECIMAL},
```

```
deptno = #{deptno,jdbcType=INTEGER}
where empno = #{empno,jdbcType=INTEGER}
</update>
</mapper>
```

## 2.4.3 Mapper 接口

Department 持久层接口

```
package com.igeekhome.ssm.dao;
import com.igeekhome.ssm.domain.Department;
public interface DepartmentMapper {
    int deleteByPrimaryKey(Integer deptno);
    int insert(Department record);
    int insertSelective(Department record);
    Department selectByPrimaryKey(Integer deptno);
    int updateByPrimaryKeySelective(Department record);
    int updateByPrimaryKey(Department record);
}
```

Employee 持久层接口

```
package com.igeekhome.ssm.dao;
import com.igeekhome.ssm.domain.Employee;
public interface EmployeeMapper {
    int deleteByPrimaryKey(Integer empno);
    int insert(Employee record);
    int insertSelective(Employee record);
    Employee selectByPrimaryKey(Integer empno);
    int updateByPrimaryKeySelective(Employee record);
    int updateByPrimaryKey(Employee record);
}
```

## 2.4.4 定义配置文件

在 src/main/resources 配置 db.properties

```
driverClassName=com.mysql.jdbc.Driver
jdbc_url=jdbc:mysql://localhost:3306/maven?useUnicode=true&characterEncoding=utf8
jdbc_username=root
jdbc_password=root
```

在 src/main/resources 配置 log4j.properties

```
### direct log messages to stdout ###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

### set log levels - for more verbose logging change 'info' to 'debug' ###
#在开发阶段日志级别使用debug
log4j.rootLogger=debug, stdout
### 在日志中输出sql的输入参数 ###
log4j.logger.org.hibernate.type=TRACE
```

在 src/main/resources 创建 applicationContext.xml 定义 Spring 的配置

```
<?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:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">
    <!-- 自动扫描, 排除扫描控制器, 控制器交给SpringMVC进行扫描 -->
    <context:component-scan base-package="com.igeekhome.ssm">
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <!-- 引入属性文件 -->
    <context:property-placeholder location="classpath:db.properties" />
    <!-- 配置数据源 -->
    <bean name="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
init-method="init" destroy-method="close">
        <property name="url" value="${jdbc_url}" />
        <property name="username" value="${jdbc_username}" />
        <property name="password" value="${jdbc_password}" />
        <!-- 初始化连接大小 -->
        <property name="initialSize" value="0" />
        <!-- 连接池最大使用连接数量 -->
        <property name="maxActive" value="5" />
        <!-- 连接池最大空闲 -->
        <property name="maxIdle" value="5" />
        <!-- 连接池最小空闲 -->
        <property name="minIdle" value="0" />
```

```
<!-- 获取连接最大等待时间 -->
<property name="maxWait" value="60000" />
<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->
<property name="timeBetweenEvictionRunsMillis" value="60000" />
<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->
<property name="minEvictableIdleTimeMillis" value="25200000" />
<!-- 打开removeAbandoned功能 -->
<property name="removeAbandoned" value="true" />
<!-- 1800秒，也就是30分钟 -->
<property name="removeAbandonedTimeout" value="1800" />
<!-- 关闭abandoned连接时输出错误日志 -->
<property name="LogAbandoned" value="true" />
<!-- 监控数据库 -->
<property name="filters" value="mergeStat" />
</bean>

<!-- 配置事务管理器 -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
<!-- 拦截器方式配置事物 -->
<tx:advice id="transactionAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="append*" propagation="REQUIRED" />
        <tx:method name="insert*" propagation="REQUIRED" />
        <tx:method name="save*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="modify*" propagation="REQUIRED" />
        <tx:method name="edit*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
        <tx:method name="remove*" propagation="REQUIRED" />
        <tx:method name="repair" propagation="REQUIRED" />
        <tx:method name="delAndRepair" propagation="REQUIRED" />
        <tx:method name="get*" propagation="SUPPORTS" />
        <tx:method name="find*" propagation="SUPPORTS" />
        <tx:method name="load*" propagation="SUPPORTS" />
        <tx:method name="search*" propagation="SUPPORTS" />
        <tx:method name="datagrid*" propagation="SUPPORTS" />
        <tx:method name="*" propagation="SUPPORTS" />
    </tx:attributes>
</tx:advice>
<aop:config>
```



```
<aop:pointcut id="transactionPointcut" expression="execution(*
com.igeekhome.ssm.service..*Impl.*(..))" />
<aop:advisor pointcut-ref="transactionPointcut"
advice-ref="transactionAdvice" />
</aop:config>

<!-- 配置druid监控spring jdbc -->
<bean id="druid-stat-interceptor"
class="com.alibaba.druid.support.spring.stat.DruidStatInterceptor">
</bean>
<bean id="druid-stat-pointcut"
class="org.springframework.aop.support.JdkRegexpMethodPointcut" scope="prototype">
<property name="patterns">
<list>
<value>com.igeekhome.ssm.service.*</value>
</list>
</property>
</bean>
<aop:config>
<aop:advisor advice-ref="druid-stat-interceptor"
pointcut-ref="druid-stat-pointcut" />
</aop:config>
</beans>
```

在 src/main/resources 创建 mybatis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
</configuration>
```

在 src/main/resources 创建 spring-mybatis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">
<!-- myBatis文件 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="configLocation" value="classpath:mybatis-config.xml" />
<!-- 自动扫描entity目录, 省掉Configuration.xml里的手工配置 -->
<property name="mapperLocations"
```

```
value="classpath:com/igeekhome/ssm/mapping/*.xml" />
</bean>
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.igeekhome.ssm.dao" />
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
</bean>
</beans>
```

## 2.4.5 单元测试

在 src/test/java 创建单元测试类  
测试 Employee 的 Dao 方法

```
package com.igeekhome.ssm.tests;
import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.igeekhome.ssm.dao.EmployeeMapper;
import com.igeekhome.ssm.domain.Employee;

public class EmployeeTest {

    @Test
    public void testFindEmployeeById(){
        //加载配置文件
        ClassPathXmlApplicationContext applicationContext = new
ClassPathXmlApplicationContext(new String[]{"applicationContext.xml",
"spring-mybatis.xml"});
        //获取dao
        EmployeeMapper employeeMapper =
applicationContext.getBean(EmployeeMapper.class);
        //查询数据
        Employee employee = employeeMapper.selectByPrimaryKey(7369);
        //查看数据
        System.out.println(employee.getEname());
        //关闭上下文
        applicationContext.close();
    }
}
```

测试 Department 的 Dao 方法

```
package com.igeekhome.ssm.tests;
import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.igeekhome.ssm.dao.DepartmentMapper;
import com.igeekhome.ssm.domain.Department;
```

```
public class DepartmentTest {  
    @Test  
    public void testFindEmployeeById(){  
        //加载配置文件  
        ClassPathXmlApplicationContext applicationContext = new  
ClassPathXmlApplicationContext(new String[]{"applicationContext.xml",  
"spring-mybatis.xml"});  
        //获取dao  
        DepartmentMapper departmentMapper =  
applicationContext.getBean(DepartmentMapper.class);  
        //查询数据  
        Department department = departmentMapper.selectByPrimaryKey(10);  
        //查看数据  
        System.out.println(department.getDname());  
        //关闭上下文  
        applicationContext.close();  
    }  
}
```

## 2.5Service

### 2.5.1 定义 Service 接口

EmployeeService 接口

```
package com.igeekhome.ssm.service;  
  
import com.igeekhome.ssm.domain.Employee;  
  
public interface EmployeeService {  
    int deleteByPrimaryKey(Integer empno);  
    int insert(Employee record);  
    int insertSelective(Employee record);  
    Employee selectByPrimaryKey(Integer empno);  
    int updateByPrimaryKeySelective(Employee record);  
  
    int updateByPrimaryKey(Employee record);  
}
```

DepartmentService 接口

```
package com.igeekhome.ssm.service;  
  
import com.igeekhome.ssm.domain.Department;
```

```
public interface DepartmentService {  
    int deleteByPrimaryKey(Integer deptno);  
    int insert(Department record);  
    int insertSelective(Department record);  
    Department selectByPrimaryKey(Integer deptno);  
    int updateByPrimaryKeySelective(Department record);  
    int updateByPrimaryKey(Department record);  
}
```

## 2.5.2 定义 Service 接口的实现类

EmployeeService 接口实现类

```
package com.igeekhome.ssm.service.impl;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import com.igeekhome.ssm.dao.EmployeeMapper;  
import com.igeekhome.ssm.domain.Employee;  
import com.igeekhome.ssm.service.EmployeeService;  
  
@Service  
public class EmployeeServiceImpl implements EmployeeService {  
    @Autowired  
    private EmployeeMapper employeeMapper;  
    @Override  
    public int deleteByPrimaryKey(Integer empno) {  
        return employeeMapper.deleteByPrimaryKey(empno);  
    }  
    @Override  
    public int insert(Employee record) {  
        return employeeMapper.insert(record);  
    }  
    @Override  
    public int insertSelective(Employee record) {  
        return employeeMapper.insertSelective(record);  
    }  
    @Override  
    public Employee selectByPrimaryKey(Integer empno) {  
        return employeeMapper.selectByPrimaryKey(empno);  
    }  
    @Override  
    public int updateByPrimaryKeySelective(Employee record) {  
        return employeeMapper.updateByPrimaryKeySelective(record);  
    }  
}
```

```
    }  
    @Override  
    public int updateByPrimaryKey(Employee record) {  
        return employeeMapper.updateByPrimaryKey(record);  
    }  
}
```

DepartmentServiceImpl 接口实现类

```
package com.igeekhome.ssm.service.impl;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import com.igeekhome.ssm.dao.DepartmentMapper;  
import com.igeekhome.ssm.domain.Department;  
import com.igeekhome.ssm.service.DepartmentService;  
  
@Service  
public class DepartmentServiceImpl implements DepartmentService {  
    @Autowired  
    private DepartmentMapper departmentMapper;  
    @Override  
    public int deleteByPrimaryKey(Integer deptno) {  
        return departmentMapper.deleteByPrimaryKey(deptno);  
    }  
    @Override  
    public int insert(Department record) {  
        return departmentMapper.insert(record);  
    }  
    @Override  
    public int insertSelective(Department record) {  
        return departmentMapper.insertSelective(record);  
    }  
    @Override  
    public Department selectByPrimaryKey(Integer deptno) {  
        return departmentMapper.selectByPrimaryKey(deptno);  
    }  
    @Override  
    public int updateByPrimaryKeySelective(Department record) {  
        return departmentMapper.updateByPrimaryKeySelective(record);  
    }  
    @Override  
    public int updateByPrimaryKey(Department record) {  
        return departmentMapper.updateByPrimaryKey(record);  
    }  
}
```

## 2.6Controller

### 2.6.1 定义控制器

```
package com.igeekhome.ssm.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.igeekhome.ssm.domain.Employee;
import com.igeekhome.ssm.service.EmployeeService;

@Controller
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    @RequestMapping(value="/emp",method=RequestMethod.GET)
    public String emp(){
        return "employee";
    }

    @RequestMapping(value="/emp",method=RequestMethod.GET)
    public String findEmployeeByEmpno(int empno,ModelMap map){
        Employee employee = employeeService.selectByPrimaryKey(empno);
        map.put("employee", employee);
        return "employee_info";
    }
}
```

### 2.6.1 定义 SpringMVC 配置文件

在 src/main/resources 配置 spring-mvc.xml

```
<?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:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd
    http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd">

    <!-- 自动扫描controller包下的所有类，使其认为spring mvc的控制器 -->
    <context:component-scan base-package="com.igeekhome.ssm.controller" />

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

    <mvc:default-servlet-handler/>
    <mvc:annotation-driven></mvc:annotation-driven>
</beans>
```

定义 web.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>

        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    >
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:spring-mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

```
<context-param>
    <param-name>contextConfigLocation</param-name>

    <param-value>classpath:applicationContext.xml,classpath:spring-mybatis.xml</param-value>
</context-param>

<listener>

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

<filter>
    <description>字符集过滤器</description>
    <filter-name>encodingFilter</filter-name>

    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

    <init-param>
        <description>字符集编码</description>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
```

## 2.7Jsp

创建 employee.jsp 如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>员工查询</title>
```



```
</head>
<body>
<form action="emp" method="get">
    <input type="text" name="empno" placeholder="请输入员工编号"><input type="submit"
value="查询">
</form>
</body>
</html>
```

创建 employee\_info.jsp 如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>员工信息</title>
</head>
<body>
员工编号: ${employee.empno}<br>
员工姓名: ${employee.ename}<br>
员工工作: ${employee.job}<br>
员工薪资: ${employee.sal}<br>
员工奖金: ${employee.comm==null?0:employee.comm}<br>
员工入职日期: <fmt:formatDate value="${employee.hiredate}" pattern="yyyy-M-d" /><br>
</body>
</html>
```

## 3分模块构建工程

基于上边的三个工程分析,我们将持久层,业务层、控制器和视图表现层可以分为三个不同的模块来处理,创建一个 parent 工程将通用的 pom 配置抽取出来,然后聚合多个模块运行。

### 3.1 需求

#### 3.1.1 需求描述

将 SSM 工程拆分为多个模块开发:

Dao 模块

Service 模块

Web 模块

### 3.1.2 理解继承和聚合

通常继承和聚合同时使用。

- 何为继承？

继承是为了消除重复，如果将 `dao`、`service`、`web` 分开创建独立的工程则每个工程的 `pom.xml` 文件中的内容存在重复，如设置编译版本、锁定 `spring` 的版本等等，可以将这些重复的配置提取出来在父工程的 `pom.xml` 中定义。

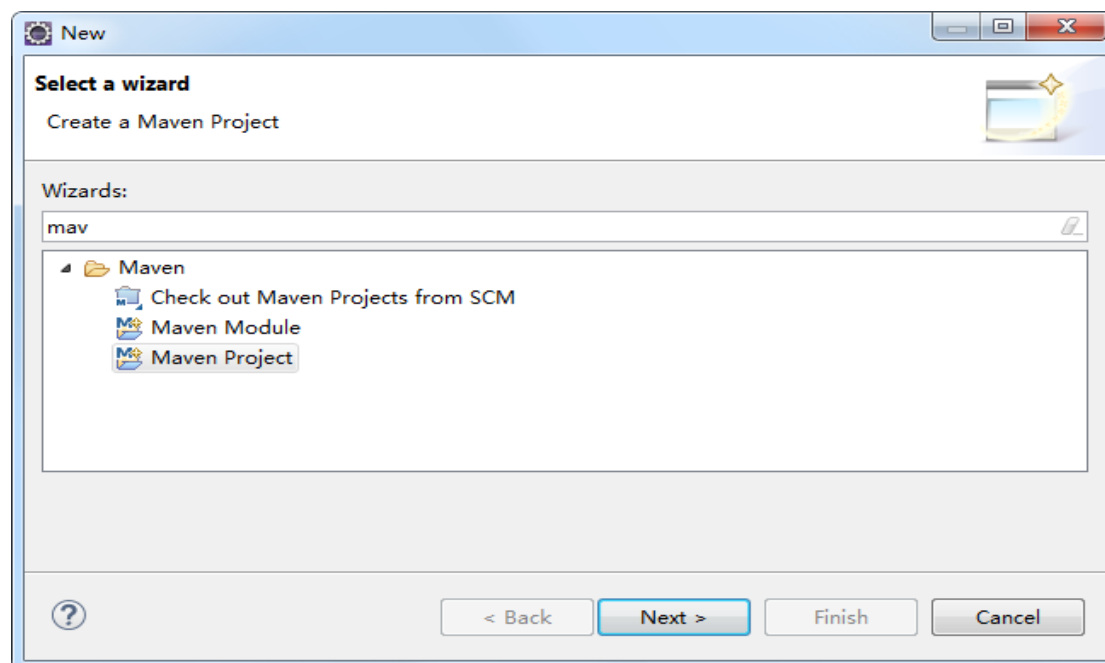
- 何为聚合？

开发通常是分组分模块开发，每个模块开发完成要运行整个工程需要将每个模块聚合在一起运行，比如：`dao`、`service`、`web` 三个工程最终会打一个独立的 `war` 运行。

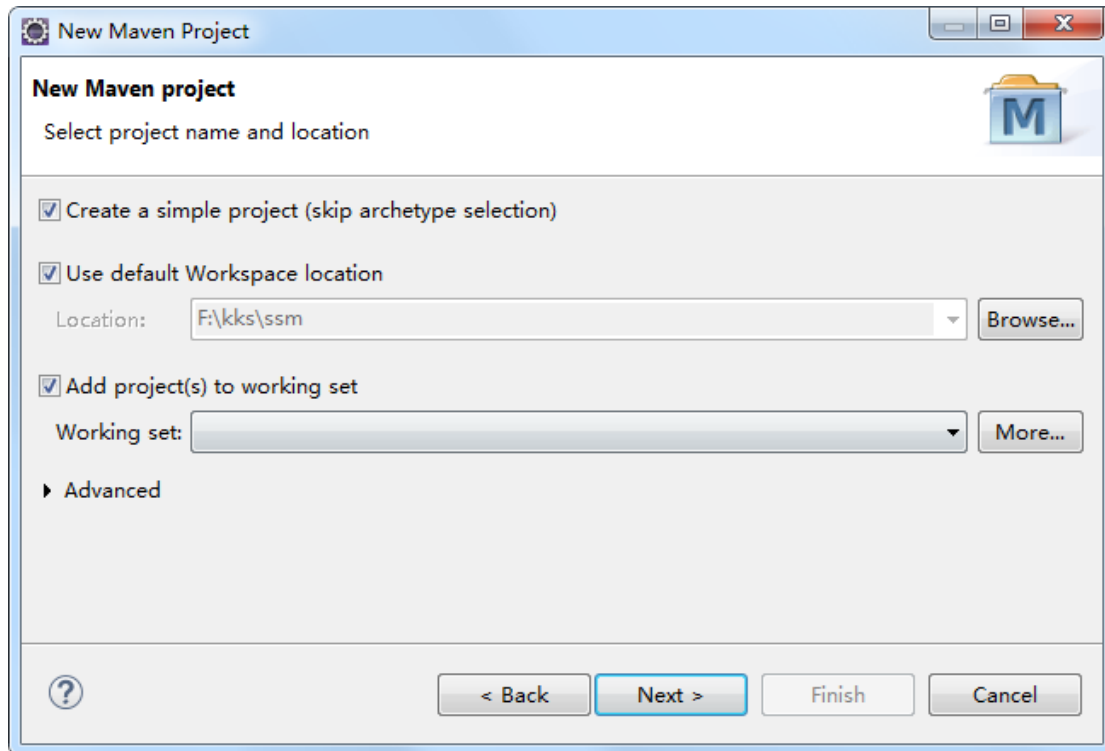
## 3.2 案例实现

### 3.2.1 ssm-parent 父模块

#### 3.2.1.1 创建父工程



这里选择“跳过骨架选择”



New Maven Project

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: F:\kks\ssm Browse...

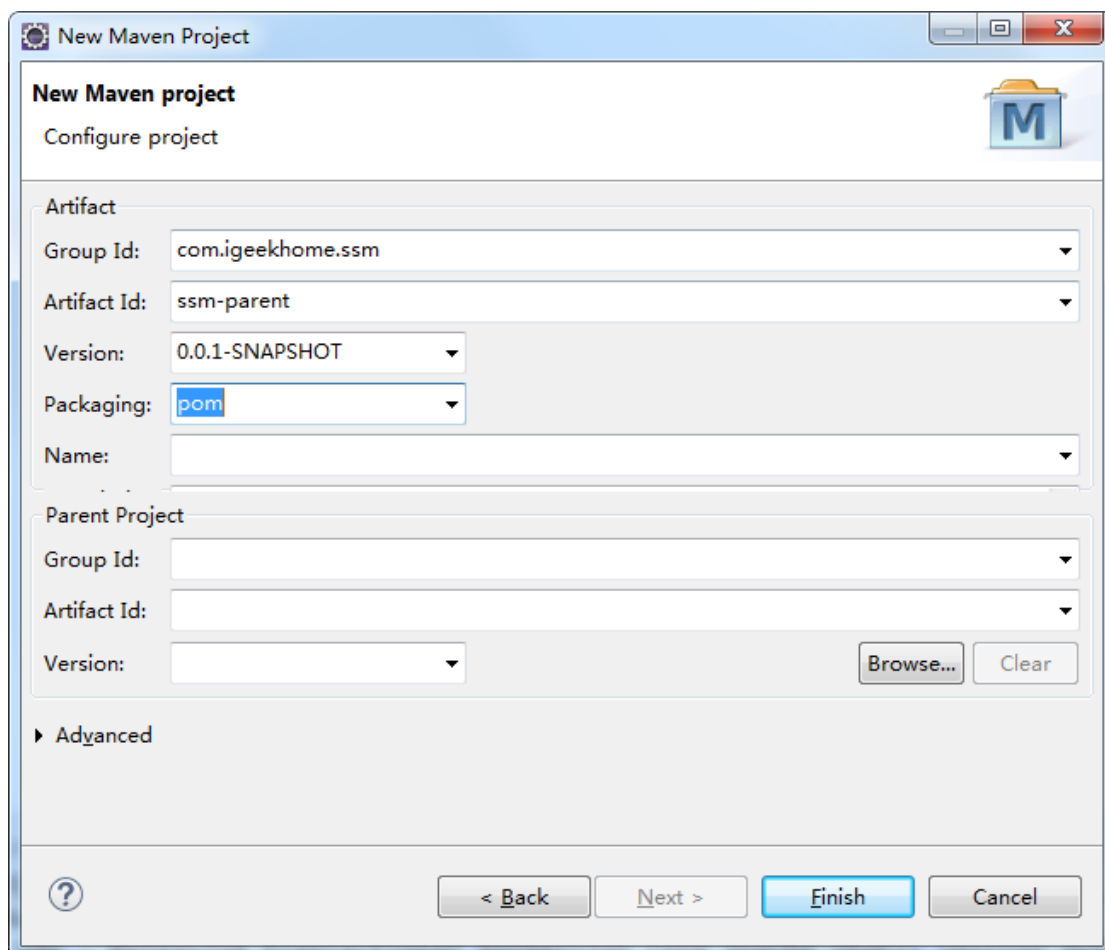
☒ Add project(s) to working set

Working set: More...

▶ Advanced

? < Back Next > Finish Cancel

定义坐标:



New Maven Project

Configure project

Artifact

Group Id: com.igeekhome.ssm

Artifact Id: ssm-parent

Version: 0.0.1-SNAPSHOT

Packaging: pom

Name:

Parent Project

Group Id:

Artifact Id:

Version: Browse... Clear

▶ Advanced

? < Back Next > Finish Cancel

### 3.2.1.2 定义 pom.xml

在父工程的 pom.xml 中抽取一些重复的配置的, 比如: 锁定 jar 包的版本、设置编译版本等。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.igeekhome.ssm</groupId>
  <artifactId>ssm-parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>

  <properties>
    <junit.version>4.12</junit.version>
    <servlet-api.version>3.0-alpha-1</servlet-api.version>
    <spring.version>4.3.6.RELEASE</spring.version>
    <mybatis.version>3.4.2</mybatis.version>
    <mybatis-spring.version>1.3.1</mybatis-spring.version>
    <mysql.version>5.1.19</mysql.version>
    <druid.version>1.0.28</druid.version>
    <aspectjweaver.version>1.8.10</aspectjweaver.version>
    <slf4j.version>1.7.24</slf4j.version>
    <log4j.version>1.2.17</log4j.version>
    <jstl.version>1.2</jstl.version>
  </properties>

  <!-- 添加工程的依赖 -->
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
      </dependency>
    <!-- servlet-api JSP页面编译时需要的包 -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>${servlet-api.version}</version>
      <scope>provided</scope>
    </dependency>
  </dependencyManagement>
</project>
```

```
<!-- Spring 以及 SpringMVC需要引入的包，自动引入需要参照的包 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
</dependency>
<!-- 持久层的包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>${mybatis.version}</version>
</dependency>
<!-- Spring 和 Mybatis的整合包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>${mybatis-spring.version}</version>
</dependency>
<!-- Mysql驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
</dependency>
<!-- druid数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>${druid.version}</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${aspectjweaver.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
</dependency>
<!-- 打日志的 -->
<dependency>
    <groupId>org.slf4j</groupId>
```

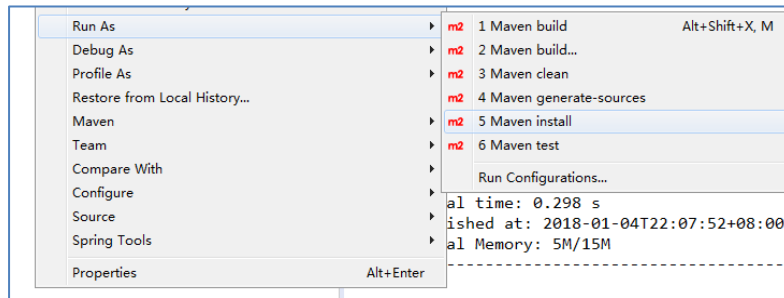
```
        <artifactId>slf4j-log4j12</artifactId>
        <version>${slf4j.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>${slf4j.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>${log4j.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
</dependencyManagement>
<build>
    <finalName>ssm</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>/**/*.xml</include>
                <include>/**/*.properties</include>
            </includes>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>/**/*.xml</include>
                <include>/**/*.properties</include>
                <include>/**/*.ini</include>
            </includes>
        </resource>
    </resources>
</pluginManagement>
```

```
<plugins>
  <!-- 设置编译版本为1.8 -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
      <encoding>UTF-8</encoding>
    </configuration>
  </plugin>

  <!-- Jetty插件, 提供一种web容器 -->
  <plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.4.2.v20170220</version>
    <configuration>
      <httpConnector>
        <!-- 配置运行的端口号 -->
        <port>80</port>
      </httpConnector>
      <!-- 配置扫描的时间间隔 -->
      <scanIntervalSeconds>1</scanIntervalSeconds>
      <webApp>
        <!-- 配置上下文 -->
        <contextPath>/ssm</contextPath>
      </webApp>
    </configuration>
  </plugin>
</plugins>
</pluginManagement>
</build>
</project>
```

### 3.2.1.3 将父工程发布至仓库

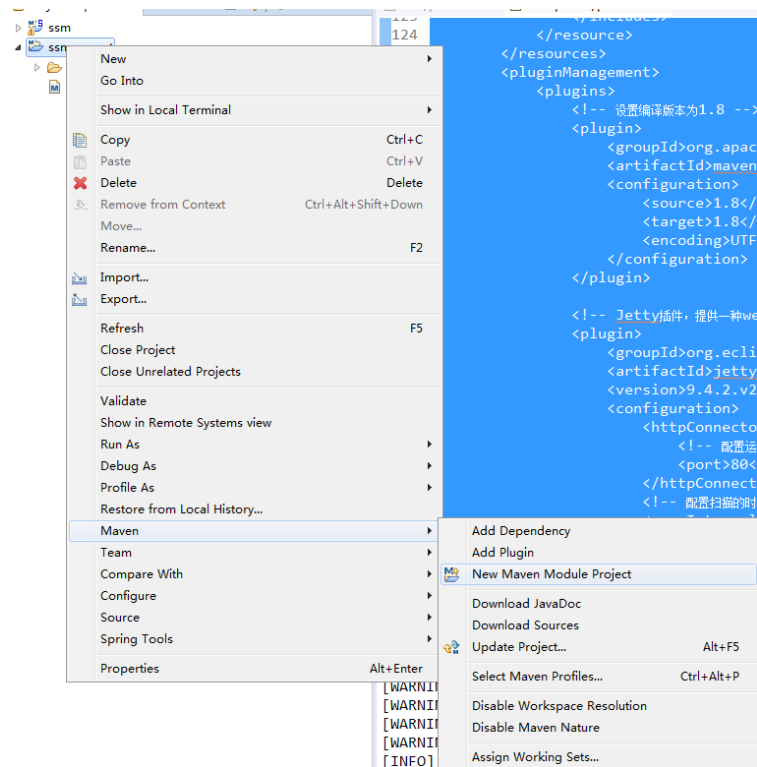
父工程创建完成执行 `maven-install` 将父工程发布到仓库方便子工程继承:



## 3.2.2ssm-dao 子模块

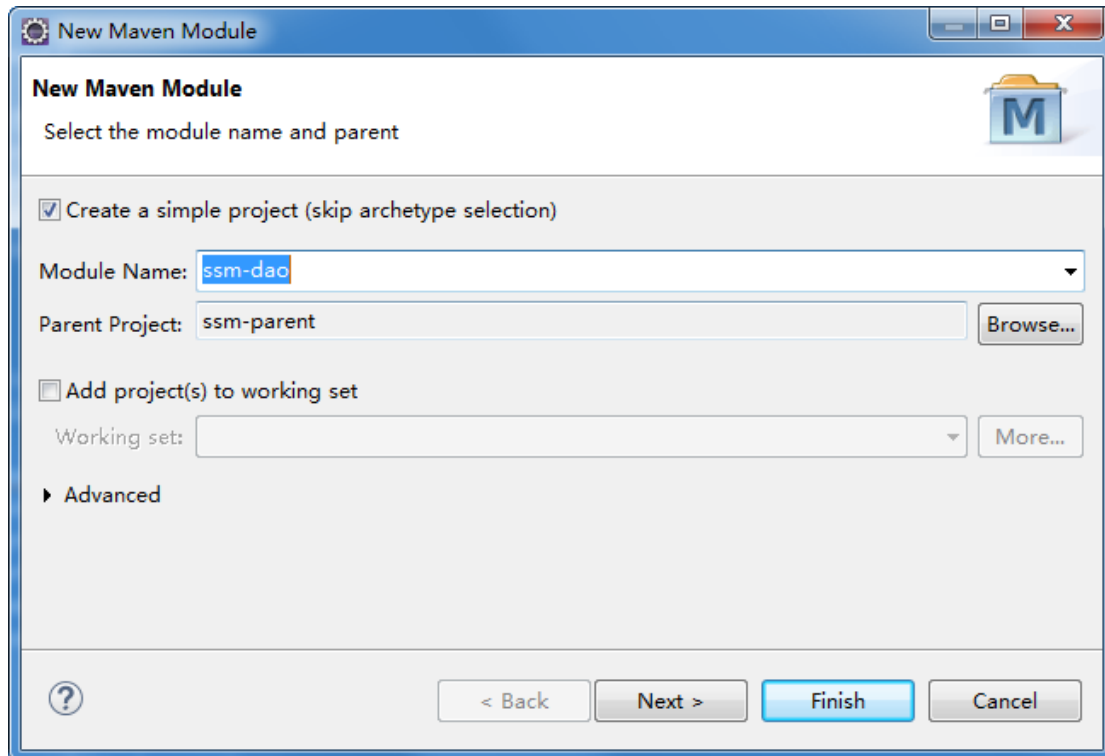
### 3.2.2.1 创建 dao 子模块

选择 ssm-parent 工程添加模块



这里指定模块名称，选择“跳过骨架选择”，并设置模块名称





### 3.2.2.2 定义 pom.xml

dao 模块的 pom.xml 文件中需要继承父模块，添加持久层需要的依赖坐标：

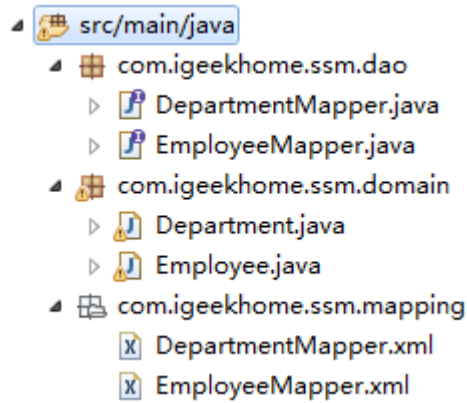
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.igeekhome.ssm</groupId>
    <artifactId>ssm-parent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>ssm-dao</artifactId>
  <packaging>jar</packaging>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
  <!-- 持久层的包 -->
```

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
</dependency>
<!-- Spring 和 Mybatis的整合包 -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
</dependency>
<!-- Mysql驱动包 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<!-- druid数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
</dependency>
</dependencies>
</project>
```

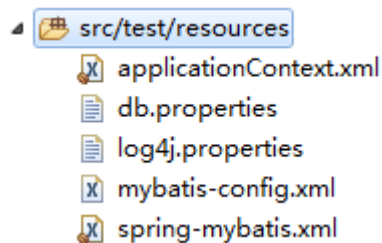
### 3.2.2.3dao

将 ssm 工程中的 dao 接口，mapper 配置文件及 domain 类拷贝到 src/main/java 中：



### 3.2.2.4 配置文件

拷贝 ssm 工程中如下配置文件到 dao 工程:



### 3.2.2.5 单元测试

```
package com.igeekhome.ssm.tests;

import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.igeekhome.ssm.dao.EmployeeMapper;
import com.igeekhome.ssm.domain.Employee;

public class EmployeeTest {

    @Test
    public void testFindEmployeeById(){
        //加载配置文件
        ClassPathXmlApplicationContext applicationContext = new
        ClassPathXmlApplicationContext(new String[]{"applicationContext.xml",
        "spring-mybatis.xml"});
        //获取dao
        EmployeeMapper employeeMapper =
        applicationContext.getBean(EmployeeMapper.class);
        //查询数据
```

```
Employee employee = employeeMapper.selectByPrimaryKey(7369);  
//查看数据  
System.out.println(employee.getEname());  
//关闭上下文  
applicationContext.close();  
}  
}
```

## 3.2.3ssm-service 子模块

### 3.2.3.1 创建 service 子模块

方法同 ssm-dao 模块创建方法，模块名称为 ssm-service。

### 3.2.3.2 定义 pom.xml

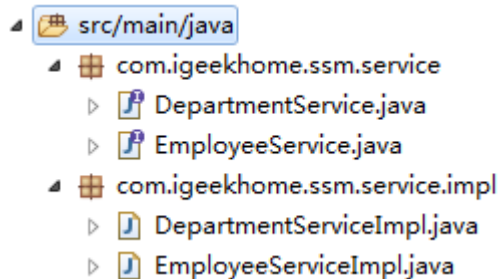
service 模块的 pom.xml 文件中需要继承父模块，service 依赖 dao 模块：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  <parent>  
    <groupId>com.igeekhome.ssm</groupId>  
    <artifactId>ssm-parent</artifactId>  
    <version>0.0.1-SNAPSHOT</version>  
  </parent>  
  <artifactId>ssm-service</artifactId>  
  <packaging>jar</packaging>  
  
  <dependencies>  
    <dependency>  
      <groupId>junit</groupId>  
      <artifactId>junit</artifactId>  
      <scope>test</scope>  
    </dependency>  
    <dependency>  
      <groupId>com.igeekhome.ssm</groupId>  
      <artifactId>ssm-dao</artifactId>  
      <version>0.0.1-SNAPSHOT</version>  
    </dependency>  
  </dependencies>
```

```
</project>
```

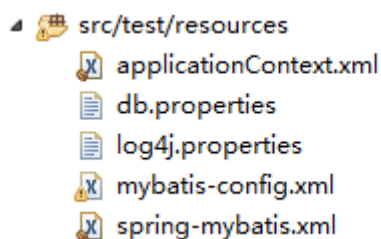
### 3.2.3.3 service 接口

将 ssm 工程中的 service 接口拷贝到 src/main/java 中:



### 3.2.3.4 配置文件

拷贝 ssm 工程中如下配置文件到 service 工程:



### 3.2.3.5 单元测试

```
package com.igeekhome.ssm.tests;

import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.igeekhome.ssm.domain.Employee;
import com.igeekhome.ssm.service.EmployeeService;

public class EmployeeTest {

    @Test
    public void testFindEmployeeById(){
        //加载配置文件
        ClassPathXmlApplicationContext applicationContext = new
        ClassPathXmlApplicationContext(new String[]{"applicationContext.xml",
        "spring-mybatis.xml"});
        //获取dao
        EmployeeService employeeService =
```

```
applicationContext.getBean(EmployeeService.class);

    //查询数据
    Employee employee = employeeService.selectByPrimaryKey(7369);
    //查看数据
    System.out.println(employee.getEname());
    //关闭上下文
    applicationContext.close();
}
}
```

## 3.2.4ssm-web 子模块

### 3.2.4.1 创建 web 子模块

方法同 ssm-dao 模块创建方法，模块名称为 ssm-web。

### 3.2.4.2 定义 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.igeekhome.ssm</groupId>
    <artifactId>ssm-parent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>ssm-web</artifactId>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <scope>test</scope>
    </dependency>
    <!-- servlet-api JSP页面编译时需要的包 -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <scope>provided</scope>
```

```
</dependency>
<!-- Spring 以及 SpringMVC需要引入的包，自动引入需要参照的包 -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
</dependency>
<!-- 打日志的 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.igeekhome.ssm</groupId>
    <artifactId>ssm-service</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
</dependencies>
<build>
    <finalName>ssm</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>/**/*.xml</include>
                <include>/**/*.properties</include>
            </includes>
        </resource>
        <resource>
```




```
<directory>src/main/resources</directory>
<includes>
  <include>**/*.xml</include>
  <include>**/*.properties</include>
  <include>**/*.ini</include>
</includes>
</resource>
</resources>
<plugins>
  <!-- 设置编译版本为1.8 -->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
      <encoding>UTF-8</encoding>
    </configuration>
  </plugin>

  <!-- Jetty插件, 提供一种web容器 -->
  <plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <configuration>
      <httpConnector>
        <!-- 配置运行的端口号 -->
        <port>80</port>
      </httpConnector>
      <!-- 配置扫描的时间间隔 -->
      <scanIntervalSeconds>1</scanIntervalSeconds>
      <webApp>
        <!-- 配置上下文 -->
        <contextPath>/ssm</contextPath>
      </webApp>
    </configuration>
  </plugin>
</plugins>
</build>
</project>
```










### 3.2.4.3 Controller

将 ssm 工程中的 controller 拷贝到 src/main/java 中：











- ▲  src/main/java
  - ▲  com.igeekhome.ssm.controller
    - ▶  EmployeeController.java

### 3.2.4.4 配置文件

将 ssm 工程中的配置文件拷贝到 src/main/resources 中

- ▲  src/main/resources
  -  applicationContext.xml
  -  db.properties
  -  log4j.properties
  -  mybatis-config.xml
  -  spring-mvc.xml
  -  spring-mybatis.xml

将 ssm 工程中的 WEB-INF 中的文件拷贝到项目的 webapp 中

- ▲  src
  - ▲  main
    - ▶  java
    - ▶  resources
    - ▲  webapp
      - ▲  WEB-INF
        - ▲  jsp
          -  employee\_info.jsp
          -  employee.jsp
        -  web.xml

### 3.2.5 运行调试

方法 1：在 maven-web 工程的 pom.xml 中配置 tomcat 插件运行

运行 maven-web 工程它从本地仓库下载依赖的 jar 包，所以当 maven-web 依赖的 jar 包内容修改了必须及时发布到本地仓库，比如：maven-web 依赖的 maven-service 修改了，需要及时将 maven-service 发布到本地仓库。

方法 2：在父工程的 pom.xml 中配置 tomcat 插件运行，自动聚合并执行

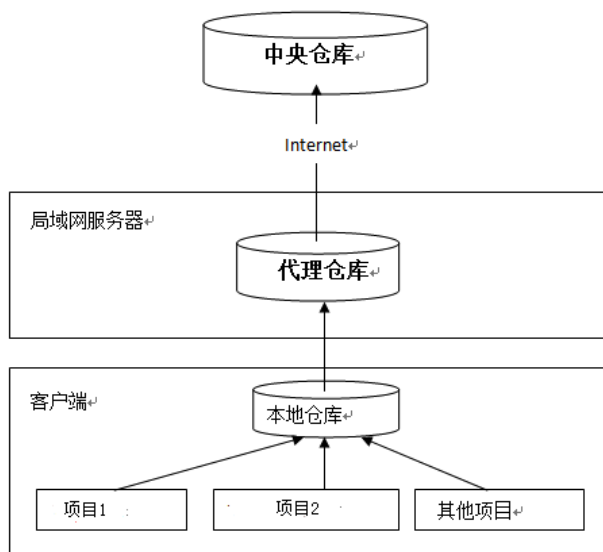
推荐方法 2，如果子工程都在本地，采用方法 2 则不需要子工程修改就立即发布到本地仓库，父工程会自动聚合并使用最新代码执行。

**注意：**如果子工程和父工程中都配置了 tomcat 插件，运行的端口和路径以子工程为准。

## 4maven私服

正式开发，不同的项目组开发不同的工程。dao 工程开发完毕，发布到私服。service 从私服下载 dao。

公司在自己的局域网内搭建自己的远程仓库服务器，称为私服，私服服务器即是公司内部的 maven 远程仓库，每个员工的电脑上安装 maven 软件并且连接私服服务器，员工将自己开发的项目打成 jar 并发布到私服服务器，其它项目组从私服服务器下载所依赖的构件（jar）。私服还充当一个代理服务器，当私服上没有 jar 包会从互联网中央仓库自动下载，如下图：



### 4.1 搭建私服环境

#### 4.1.1 下载 nexus

Nexus 是 Maven 仓库管理器，通过 nexus 可以搭建 maven 仓库，同时 nexus 还提供强大的仓库管理功能，构件搜索功能等。

下载 Nexus，下载地址：<http://www.sonatype.org/nexus/archived/>

## 2 Download a Distribution

Once you've selected a version in Step 1, you can download a distribution from the following list.

Download Nexus 2.12.0-01

NEXUS OSS (TGZ)

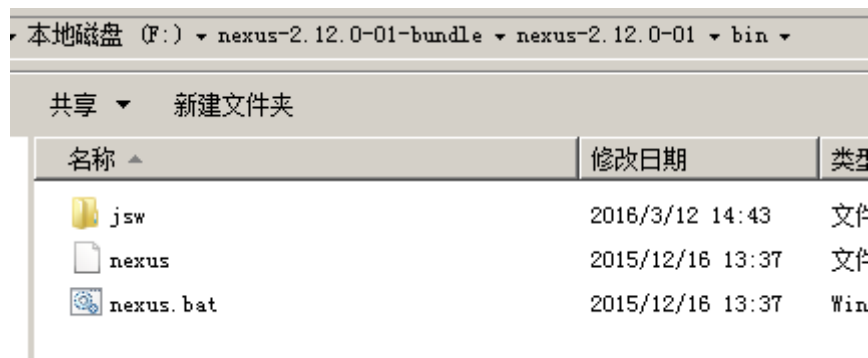
Checksums: MD5 SHA Signature: PGP

NEXUS OSS (ZIP)

Checksums: MD5 SHA Signature: PGP

## 4.1.2 安装 nexus

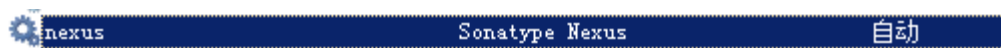
解压 nexus-2.12.0-01-bundle.zip，本教程将它解压在 F 盘，进入 bin 目录：



cmd 进入 bin 目录，执行 nexus.bat install

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat install
wrapper ! nexus installed.
```

安装成功在服务中查看有 nexus 服务：



## 4.1.3 卸载 nexus

cmd 进入 nexus 的 bin 目录，执行：nexus.bat uninstall

```
F:\nexus-2.12.0-01-bundle\nexus-2.12.0-01\bin>nexus.bat uninstall
```

查看 window 服务列表 nexus 已被删除。

## 4.1.4 启动 nexus

方法 1：

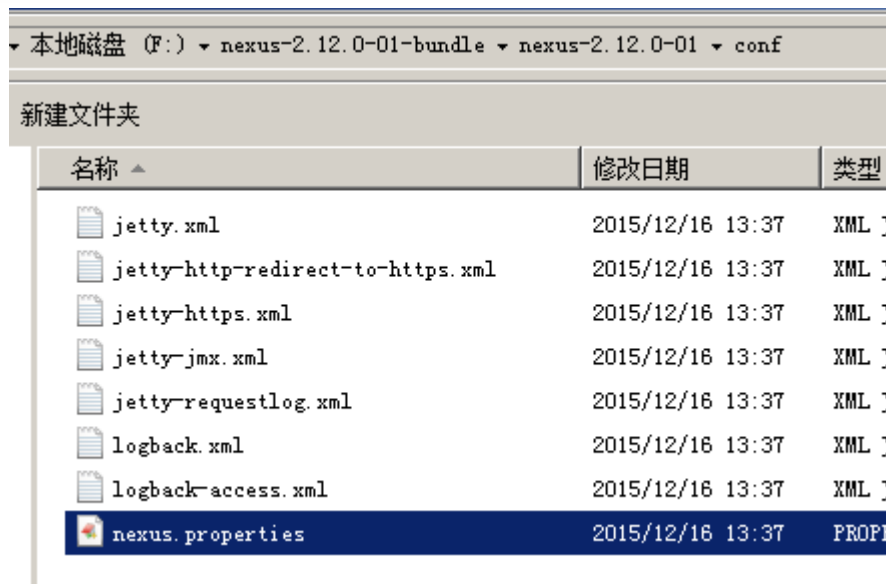
cmd 进入 bin 目录，执行 nexus.bat start

方法 2：

直接启动 nexus 服务



查看 nexus 的配置文件 conf/nexus.properties



# Jetty section

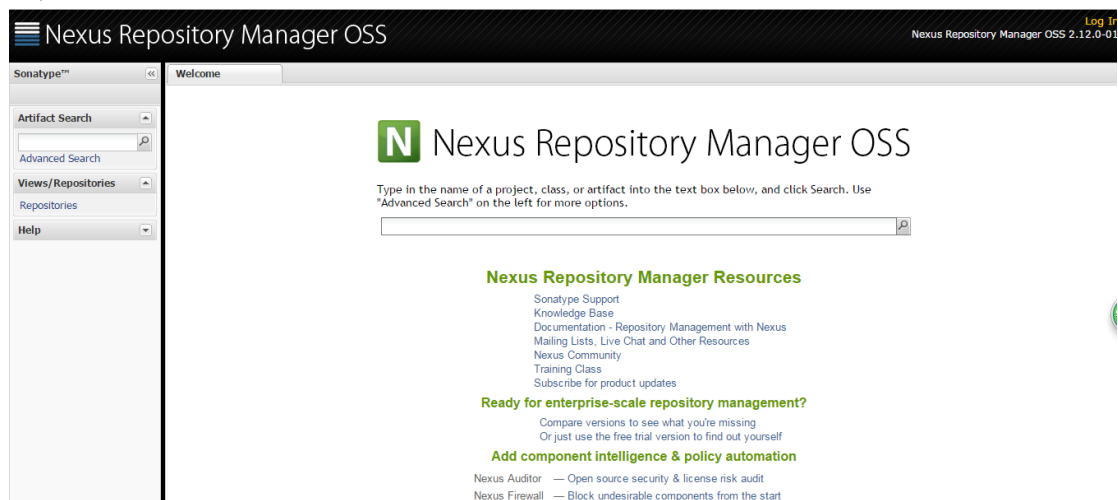
```
application-port=8081      # nexus 的访问端口配置
application-host=0.0.0.0  # nexus 主机监听配置(不用修改)
nexus-webapp=${bundleBasedir}/nexus  # nexus 工程目录
nexus-webapp-context-path=/nexus    # nexus 的 web 访问路径
```

# Nexus section

```
nexus-work=${bundleBasedir}/../sonatype-work/nexus  # nexus 仓库目录
runtime=${bundleBasedir}/nexus/WEB-INF  # nexus 运行程序目录
```

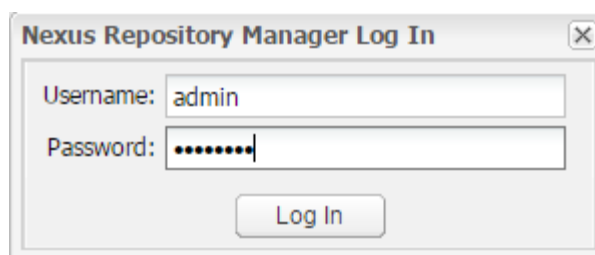
访问:

http://localhost:8081/nexus/



使用 Nexus 内置账户 admin/admin123 登陆:

点击右上角的 Log in, 输入账号和密码 登陆



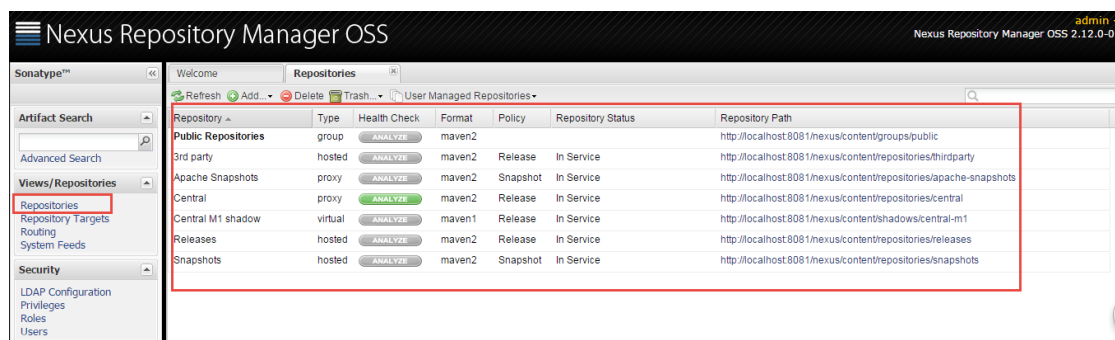
登陆成功:



## 4.1.5 仓库类型

nexus

查看 nexus 的仓库:



nexus 的仓库有 4 种类型:

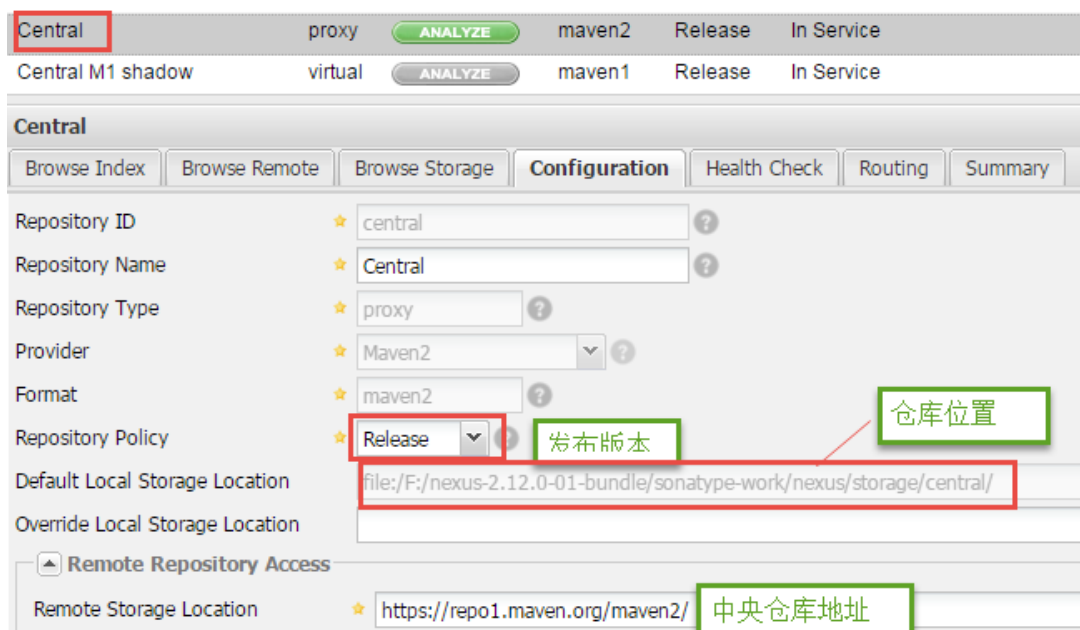
Repository	Type
Public Repositories	group
3rd party	hosted
Apache Snapshots	proxy
Central	proxy
Central M1 shadow	virtual
Releases	hosted
Snapshots	hosted

1. hosted, 宿主仓库, 部署自己的 jar 到这个类型的仓库, 包括 releases 和 snapshot 两部分, Releases 公司内部发布版本仓库、 Snapshots 公司内部测试版本仓库
2. proxy, 代理仓库, 用于代理远程的公共仓库, 如 maven 中央仓库, 用户连接私服, 私服自动去中央仓库下载 jar 包或者插件。

3. **group**, 仓库组, 用来合并多个 hosted/proxy 仓库, 通常我们配置自己的 maven 连接仓库组。
4. **virtual(虚拟)**: 兼容 Maven1 版本的 jar 或者插件  
nexus 仓库默认在 sonatype-work 目录中:



✓ **central**: 代理仓库, 代理中央仓库



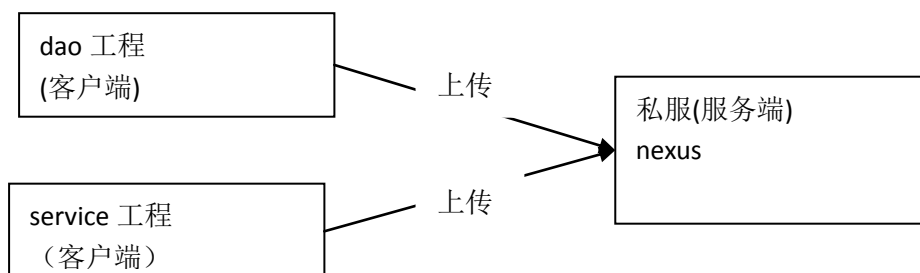
- ✓ **apache-snapshots**: 代理仓库  
存储 snapshots 构件, 代理地址 <https://repository.apache.org/snapshots/>
- ✓ **central-m1**: virtual 类型仓库, 兼容 Maven1 版本的 jar 或者插件
- ✓ **releases**: 本地仓库, 存储 releases 构件。
- ✓ **snapshots**: 本地仓库, 存储 snapshots 构件。
- ✓ **thirdparty**: 第三方仓库
- ✓ **public**: 仓库组

## 4.2 将项目发布到私服

### 4.2.1 需求

企业中多个团队协作开发通常会将一些公用的组件、开发模块等发布到私服供其它团队或模块开发人员使用。

本例子假设多团队分别开发 dao、service、web，某个团队开发完在 dao 会将 dao 发布到私服供 service 团队使用，本例子会将 dao 工程打成 jar 包发布到私服。



### 4.2.2 配置

第一步：需要在客户端即部署 dao 工程的电脑上配置 maven 环境，并修改 settings.xml 文件，配置连接私服的用户和密码。

此用户名和密码用于私服校验，因为私服需要知道上传的账号和密码是否和私服中的账号和密码一致。

```

<server>
  <id>releases</id>
  <username>admin</username>
  <password>admin123</password>
</server>
<server>
  <id>snapshots</id>
  <username>admin</username>
  <password>admin123</password>
</server>
  
```

releases 连接发布版本项目仓库

snapshots 连接测试版本项目仓库

Releases	hosted	ANALYZE	maven2	Release	In Service
Snapshots	hosted	ANALYZE	maven2	Snapshot	In Service

第二步：配置项目 pom.xml

配置私服仓库的地址，本公司的自己的 jar 包会上传到私服的宿主仓库，根据工程的版本号决定上传到哪个宿主仓库，如果版本为 release 则上传到私服的 release 仓库，如果版本为 snapshot 则上传到私服的 snapshot 仓库

```
<distributionManagement>
  <repository>
    <id>releases</id>
    <url>http://localhost:8081/nexus/content/repositories/releases/</url>
  </repository>
  <snapshotRepository>
    <id>snapshots</id>
    <url>http://localhost:8081/nexus/content/repositories/snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

注意：pom.xml 这里<id> 和 settings.xml 配置 <id> 对应！

## 4.2.3 测试

将项目 dao 工程打成 jar 包发布到私服：

- 1、首先启动 nexus
- 2、对 dao 工程执行 deploy 命令

根据本项目 pom.xml 中 version 定义决定发布到哪个仓库，如果 version 定义为 snapshot，执行 deploy 后查看 nexus 的 snapshot 仓库，如果 version 定义为 release 则项目将发布到 nexus 的 release 仓库，本项目将发布到 snapshot 仓库：

F:\nexus-2.12.0-01-bundle\sonatype-work\nexus\storage\snapshots\cn\itcast\maven\maven-dao\0.0.1-SNAPSHOT\

名称	修改日期	类型	大小
maven-dao-0.0.1-20160313.061752-1.jar	2016/3/13 14:17	Executable Jar...	9
maven-dao-0.0.1-20160313.061752-1...	2016/3/13 14:17	MD5 文件	1

也可以通过 http 方式查看：

localhost:8081/nexus/content/repositories/snapshots/cn/itcast/maven/maven-dao/0.0.1-SNAPSHOT/

### Index of /repositories/snapshots/cn/itcast/maven/maven-d

Name	Last Modified	Size	Description
<a href="#">Parent Directory</a>			
<a href="#">maven-dao-0.0.1-20160313.061752-1.jar</a>	Sun Mar 13 14:17:52 CST 2016	8791	



## 4.3 从私服下载 jar 包

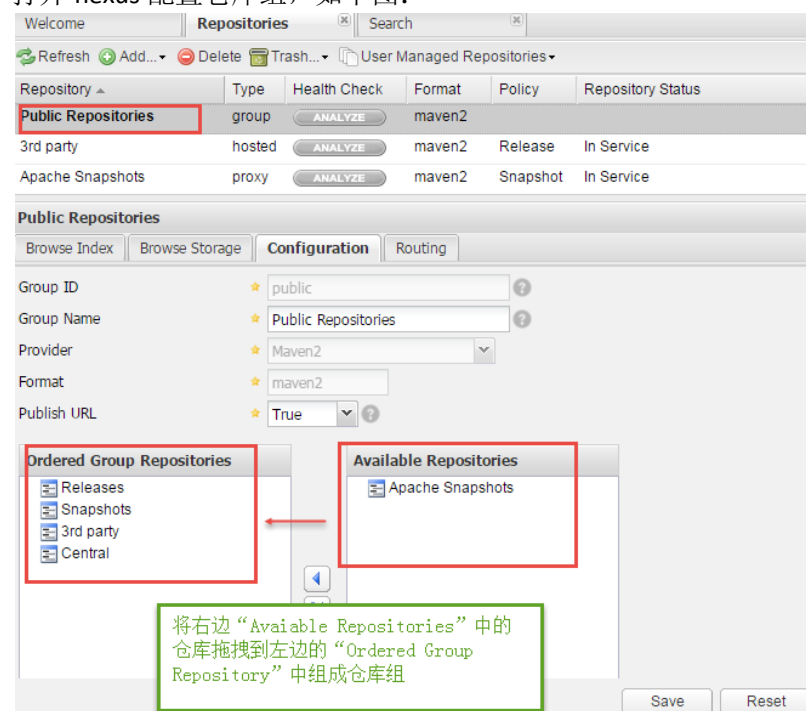
### 4.3.1 需求

没有配置 nexus 之前, 如果本地仓库没有, 去中央仓库下载, 通常在企业中会在局域网内部署一台私服服务器, 有了私服本地项目首先去本地仓库找 jar, 如果没有找到则连接私服从私服下载 jar 包, 如果私服没有 jar 包私服同时作为代理服务器从中央仓库下载 jar 包, 这样做的好处是一方面由私服对公司项目的依赖 jar 包统一管理, 一方面提高下载速度, 项目连接私服下载 jar 包的速度要比项目连接中央仓库的速度快的多。

### 4.3.2 管理仓库组

nexus 中包括很多仓库, hosted 中存放的是企业自己发布的 jar 包及第三方公司的 jar 包, proxy 中存放的是中央仓库的 jar, 为了方便从私服下载 jar 包可以将多个仓库组成一个仓库组, 每个工程需要连接私服的仓库组下载 jar 包。

打开 nexus 配置仓库组, 如下图:



上图中仓库组包括了本地仓库、代理仓库等。

### 4.3.3 在 setting.xml 中配置仓库

在客户端的 setting.xml 中配置私服的仓库, 由于 setting.xml 中没有 repositories 的配置标签需要使用 profile 定义仓库。

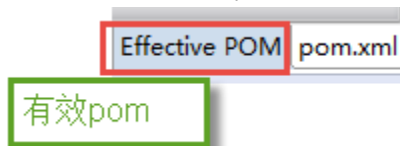
```
<profile>
  <!--profile 的 id-->
```

```
<id>dev</id>
<repositories>
  <repository>
    <!--仓库 id, repositories 可以配置多个仓库, 保证 id 不重复-->
    <id>nexus</id>
    <!--仓库地址, 即 nexus 仓库组的地址-->
    <url>http://localhost:8081/nexus/content/groups/public/</url>
    <!--是否下载 releases 构件-->
    <releases>
      <enabled>true</enabled>
    </releases>
    <!--是否下载 snapshots 构件-->
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
<pluginRepositories>
  <!-- 插件仓库, maven 的运行依赖插件, 也需要从私服下载插件 -->
  <pluginRepository>
    <!-- 插件仓库的 id 不允许重复, 如果重复后边配置会覆盖前边 -->
    <id>public</id>
    <name>Public Repositories</name>
    <url>http://localhost:8081/nexus/content/groups/public/</url>
  </pluginRepository>
</pluginRepositories>
</profile>
```

使用 profile 定义仓库需要激活才可生效。

```
<activeProfiles>
  <activeProfile>dev</activeProfile>
</activeProfiles>
```

配置成功后通过 eclipse 查看有效 pom, 有效 pom 是 maven 软件最终使用的 pom 内容, 程序员不直接编辑有效 pom, 打开有效 pom



有效 pom 内容如下:

下边的 pom 内容中有两个仓库地址, maven 会先从前边的仓库的找, 如果找不到 jar 包再从下边的找, 从而就实现了从私服下载 jar 包。

```
<repositories>
  <repository>
    <releases>
      <enabled>true</enabled>
```

```
</releases>
<snapshots>
  <enabled>true</enabled>
</snapshots>
<id>public</id>
<name>Public Repositories</name>
<url>http://localhost:8081/nexus/content/groups/public/</url>
</repository>
<repository>
  <snapshots>
    <enabled>false</enabled>
  </snapshots>
  <id>central</id>
  <name>Central Repository</name>
  <url>https://repo.maven.apache.org/maven2</url>
</repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>public</id>
    <name>Public Repositories</name>
    <url>http://localhost:8081/nexus/content/groups/public/</url>
  </pluginRepository>
  <pluginRepository>
    <releases>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </pluginRepository>
</pluginRepositories>
```