# SSM整合

## 01. 环境准备

### 1.1 创建数据库和表结构

```
create table account(
    id int primary key auto_increment,
    name varchar(100),
    money double(7,2)
 )
```

### 1.2 创建Maven工程，在pom文件中导入坐标

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

<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.dgut</groupId>
  <artifactId>SSM-01</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>SSM-01 Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <spring.version>5.1.6.RELEASE</spring.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjweaver</artifactId>
      <version>1.9.4</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>



  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
  </dependency>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.3</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.17</version>
  </dependency>

  <dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.2</version>
```

```
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.1.6.RELEASE</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>SSM-01</finalName>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven
defaults (may be moved to parent pom) -->
      <plugins>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

### 1.3 编写实体类 Account

```
public class Account {
    private Integer id;
    private String name;
    private Float money;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Float getMoney() {
        return money;
    }

    public void setMoney(Float money) {
        this.money = money;
    }

    @Override
    public String toString() {
        return "Account{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", money=" + money +
                '}';
    }
}
```

### 1.4 编写持久层接口

```
public interface IAccountDao {

    /**
     * 保存账户
     * @param account
```

```
     */
    void saveAccount(Account account);

    /**
     * 查询所有账户
     */
    List<Account> findAllAccount();
}
```

**1.5编写业务层接口及实现**

```
public interface IAccountService {
    /**
     * 保存账户 * @param account
     */
    void saveAccount(Account account);

    /**
     * 查询所有账户 * @return
     */
    List<Account> findAllAccount();
}

public class AccountServiceImpl implements IAccountService {
    public void saveAccount(Account account) {
        System.out.println("业务层保存account");
    }

    public List<Account> findAllAccount() {
        System.out.println("业务层查找所有account");
        return null;
    }
}
```

# 02. 整合spring框架

**2.1 保证Spring框架在web工程中独立运行**

创建**applicationContext.xml**，用来配置spring相关

```
<?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"
       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">

    <!-- 配置spring创建容器时要扫描的包 -->
    <context:component-scan base-package="com.dgut">
        <!--制定扫包规则，不扫描@Controller注解的JAVA类，其他的还是要扫描 -->
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

</beans>
```

### 2.2 使用注解配置业务层

```
@Service("accountService")
public class AccountServiceImpl implements IAccountService {
    public void saveAccount(Account account) {
        System.out.println("业务层保存account");
    }

    public List<Account> findAllAccount() {
        System.out.println("业务层查找所有account");
        return null;
    }
}
//持久层实现类代码：  此时不要做任何操作，就输出一句话。目的是测试spring框架搭建的结果。
```

### 2.3 测试spring能否独立运行

```
  @Test
    public void test(){
        ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
        IAccountService service = (IAccountService)
ac.getBean("accountService");
        service.findAllAccount();
        service.saveAccount(null);
    }
```

# 03. 整合SpringMVC

### 3.1 配置核心控制器

```
  <!-- 配置spring mvc的核心控制器 -->
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
      <!-- 配置初始化参数，用于读取springmvc的配置文件 -->
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:springmvc.xml</param-value>
    </init-param>
    <!-- 配置servlet的对象的创建时间点：应用加载时创建。取值只能是非0正整数，表示启动顺序
-->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <!-- 配置springMVC编码过滤器 -->
  <filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
    <!--
    forceEncoding=true 强制所有的请求响应都使用encoding编码。
     forceEncoding=false 如果请求头中包含charset，则使用chartset编码，否则使用
encoding编码。
     -->
    <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
```

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

### 3.2 编写SpringMVC的配置文件

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

    <!-- 配置创建spring容器要扫描的包 -->
    <context:component-scan base-package="com.dgut">
        <!-- 制定扫包规则 ,只扫描使用@Controller注解的JAVA类 -->
        <context:include-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <!-- 配置视图解析器 -->
    <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <!--放行资源-->
    <mvc:resources mapping="/js/**" location="/js/"/>

    <!-- 开启注解 -->
    <mvc:annotation-driven />

</beans>
```

### 3.3 编写Controller和jsp页面

```
@Controller
@RequestMapping("/account")
public class UserController {
    @RequestMapping("/findAll")
    public String findAll(){
        return "list";
    }
```

```
}
```

index.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

<a href="account/findAll">查找所有用户</a>
</body>
</html>
```

# 04. 整合Spring和SpringMVC

同学们发现，现在我们加载了springmvc的配置文件，但是spring的配置文件我们一直都是用测试代码加载的，那么如何将spring的配置文件纳入到我们servlet的容器管理呢?

答案：用监听器实现启动服务创建容器

### 4.1 配置监听器实现启动服务创建容器

```
  <!-- 配置spring提供的监听器，用于启动服务时加载容器 。
    该间监听器只能加载WEB-INF目录中名称为applicationContext.xml的配置文件 -->
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <!-- 手动指定spring配置文件位置 -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
  </context-param>
```

### 4.2 业务层注入到控制层

```
@Controller
@RequestMapping("/account")
public class UserController {

    @Autowired
    private IAccountService service;

    @RequestMapping("/findAll")
```

```
    public String findAll(){
        List<Account> list = service.findAllAccount();
        System.out.println(list);
        return "list";
    }
}
```

# 05. MyBatis框架

> 注意：我们使用代理**dao**的方式来操作持久层，所以此处**Dao**的实现类就是多余的了。

### 5.1 编写AccountDao映射配置文件

```
<?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>
    <environments default="mysql">
        <environment id="mysql">
            <transactionManager type="JDBC"></transactionManager>
            <dataSource type="pooled">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///ssm"/>
                <property name="username" value="root"/>
                <property name="password" value="xieman123"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>

<!--        <mapper class="com.dgut.dao.IAccountDao"/>-->
        <package name="com.dgut.dao"/>
    </mappers>
</configuration>
```

### 5.2 编写dao接口类

```
public interface IAccountDao {

    /**
     * 保存账户
     * @param account
     */
    @Insert("insert into account(name,money) values (#{name},#{money})")
    void saveAccount(Account account);

    /**
     * 查询所有账户
     */
```

```
    @Select("select * from account")
    List<Account> findAllAccount();
}
```

5.3 编写测试类

```
public class MyBatisTest {

    @Test
    public void mybatisTestFind() throws IOException {
      //加载配置文件
        InputStream inputStream =
Resources.getResourceAsStream("sqlMapConfig.xml");
      //获取session工厂类
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
      //获取session
        SqlSession session = sessionFactory.openSession();
      //根据session获取实现
        IAccountDao dao = session.getMapper(IAccountDao.class);
        List accounts = dao.findAllAccount();
        System.out.println(accounts);
    }
    @Test
    public void mybatisTestInsert() throws IOException {
        Account account = new Account();
        account.setName("xiaohei");
        account.setMoney(100f);
        InputStream inputStream =
Resources.getResourceAsStream("sqlMapConfig.xml");
        SqlSessionFactory sessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        SqlSession session = sessionFactory.openSession();
        IAccountDao dao = session.getMapper(IAccountDao.class);
        dao.saveAccount(account);
      //提交事务
        session.commit();
    }
}
```

# 06. Spring整合MyBatis

整合思路：

把mybatis配置文件（sqlMapConfig.xml）中内容配置到spring配置文件中

同时，把mybatis配置文件的内容清掉。

### 6.1 导入mybatis-spring坐标 及 缓存池

```
 <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>2.0.2</version>
    </dependency>
<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->
<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.5.4</version>
</dependency>
```

### 6.2 Spring接管MyBatis的Session工厂

```
   <!-- 配置数据源 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="com.mysql.cj.jdbc.Driver"/>
        <property name="jdbcUrl" value="jdbc:mysql:///ssm"/>
        <property name="user" value="root"/>
        <property name="password" value="xieman123"/>
    </bean>

    <bean id="sessionFactoryBean"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

### 6.3 配置自动扫描所有Mapper接口和文件

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.dgut.dao"/>
    </bean>
```

### 6.4 配置spring的事务

```
<!-- 配置事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

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

    <!-- 配置aop -->
    <aop:config >
        <!-- 建立通知和切入点表达式的关系 -->
        <aop:advisor advice-ref="txAdvice" pointcut="execution(*
com.dgut.service.impl.*.*(..))"/>
    </aop:config>
```