基于xml整合第三方框架

xml整合第三方框架有两种整合方案：

- 不需要自定义命名空间，不需要使用Spring的配置文件配置第三方框架本身内容，例如，MyBatis；
- 需要引入第三方框架命名空间，需要使用Spring的配置文件配置第三方框架本身内容，例如，Dubbo。

命名空间指的是Spring配置的xml文件中的命名空间，例如，

```
xmlns="http://www.springframework.org/schema/beans"
```

用于解析xml配置文件。

1. 整合MyBatis

MyBatis提供了mybatis-spring.jar专门用于两大框架的整合，步骤如下：

- 导入MyBatis整合Spring的相关坐标；
- 编写Mapper和Mapper.xml;
- 配置SqlSessionFactoryBean和MapperScannerConfigurer；

```
<!--配置SqlSessionFactoryBean，提供SqlSessionFactory-->
<bean class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource"/>
</bean>

<!--MapperScannerConfigurer，扫描指定的包，产生Mapper对象-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
  <!--要设置多个basePackage，value中的多个路径用,隔开-->
  <property name="basePackage" value="com.example.mapper"/>
</bean>

<!--配置数据源-->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
  <property name="url" value="jdbc:mysql://192.168.224.100:3306/test"/>
  <property name="username" value="study"/>
  <property name="password" value="123456"/>
</bean>
```

- 编写测试代码。

```
public class UserServiceImpl implements UserService ... {
    ...
    private UserMapper userMapper;
    ...
    public void setUserMapper(UserMapper userMapper) {
        this.userMapper = userMapper;
    }
    ...
    public void show() {
        List<User> all = userMapper.findAll();
        all.forEach(System.out::println);
    }
    ...
}


@Test
public void test02() {
    ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
    UserServiceImpl userService = applicationContext.getBean("userService",
UserServiceImpl.class);
    userService.show();
}
```

输出如下：

```
User{id=1, name='Tom1', age=19, status=1, gender=男}
User{id=2, name='Tom2', age=25, status=0, gender=男}
User{id=3, name='Tom3', age=19, status=1, gender=男}
User{id=5, name='Tom4', age=80, status=1, gender=男}
User{id=6, name='Tom5', age=119, status=1, gender=男}
```

Spring整合MyBatis的jar包里，提供了一个SqlSessionFactoryBean和一个扫描Mapper的配置对象，SqlSessionFactoryBean一旦被实例化，就开始扫描Mapper，并通过动态代理产生Mapper的实现类，存储到Spring容器中。相关的有如下四个类：

- SqlSessionFactoryBean，需要进行配置，用于提供SqlSessionFactory
- MapperScannerConfigurer，需要进行配置，用于扫描指定mapper注册BeanDefinition
- MapperFactoryBean，Mapper的FactoryBean，获得指定Mapper时调用getObject方法
- ClassPathMapperScanner，definition.setAutowireMode(2) 修改了自动注入状态，所以MapperFactoryBean中的setSqlSessionFactory会自动注入进去

MyBatis未与Spring整合时，执行的过程如下：

- mybatis-config.xml中配置数据库连接信息；

- 定义PoJo，例如，User
- 定义Mapper接口，UserMapper
- 定义Mapper配置文件，UserMapper.xml
- 执行如下代码：

```
try (InputStream in = Resources.getResourceAsStream("mybatis-config.xml")) {
    SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
    SqlSessionFactory sqlSessionFactory = builder.build(in);

  try (SqlSession sqlSession = sqlSessionFactory.openSession()) {
      UserMapper mapper = sqlSession.getMapper(UserMapper.class);
      List<User> all = mapper.findAll();
      all.forEach(System.out::println);
    }
}
```

整合后，可将builder、sqlSessionFactory、sqlSession等的创建省略。

2. 整合需要引入命名空间的第三方框架

以引入spring-context标签为例，

- 引入命名空间：

```
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context.xsd
```

- 使用标签，将数据库配置信息存储在properties文件，使用context读取

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

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
  <property name="url" value="${jdbc.url}"/>
  <property name="username" value="${jdbc.username}"/>
  <property name="password" value="${jdbc.password}"/>
</bean>
```

自定义命名空间有两个要点：

- jar包下含有spring.handlers，指明NamespaceHandler
- Spring会调用指定NamespaceHandler的init方法和parse方法

parse方法一般处理两个事情：

- 注册BeanDefinition
- 注册BeanPostProcessor，对所有注册SpringBean进行处理

外部命名空间标签的执行过程，如下：

- 将**自定义标签约束**与**物理约束文件**与**网络约束名称约束**，以键值对的形式存储到一个**spring.schemas**文件里，该文件存储在类加载路径的META-INF里，Spring会自动加载
- 将**自定义命名空间名称**与**自定义命名空间处理器**的映射关系，以键值对的形式存储到一个**spring.handlers**文件里，该文件存储在类加载路径的META-INF里，Spring会自动加载到
- 准备好**NamespaceHandler**，如果命名空间只有一个标签，那么直接在parse方法中解析即可，一般解析结果就是注册该标签对应的BeanDefinition。如果命名空间里有多个标签，那么可以在init方法中为每个标签都注册一个**BeanDefinitionParser**，在执行NamespaceHandler的parse时，分流给不同的BeanDefinitionParser进行解析（重写doParse方法）