注入映射器

与其在数据访问对象（DAO）中手工编写使用 `SqlSessionDaoSupport` 或 `SqlSessionTemplate` 的代码，还不如让 Mybatis-Spring 为你创建一个线程安全的映射器，这样你就可以直接注入到其它的 bean 中了：

```
<bean id="fooService" class="org.mybatis.spring.sample.service.FooServiceImpl">
  <constructor-arg ref="userMapper" />
</bean>
```

注入完毕后，映射器就可以在你的应用逻辑代码中使用了：

```
public class FooServiceImpl implements FooService {

  private final UserMapper userMapper;

  public FooServiceImpl(UserMapper userMapper) {
    this.userMapper = userMapper;
  }

  public User doSomeBusinessStuff(String userId) {
    return this.userMapper.getUser(userId);
  }
}
```

注意代码中并没有任何的对 `SqlSession` 或 MyBatis 的引用。你也不需要担心创建、打开、关闭 session，MyBatis-Spring 将为你打理好一切。

注册映射器

注册映射器的方法根据你的配置方法，即经典的 XML 配置或新的 3.0 以上版本的 Java 配置（也就是常说的 `@Configuration`），而有所不同。

XML 配置

在你的 XML 中加入 `MapperFactoryBean` 以便将映射器注册到 Spring 中。就像下面一样：

```
<bean id="userMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
  <property name="mapperInterface" value="org.mybatis.spring.sample.mapper.UserMapper" />
  <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

如果映射器接口 UserMapper 在相同的类路径下有对应的 MyBatis XML 映射器配置文件，将会被 `MapperFactoryBean` 自动解析。不需要在 MyBatis 配置文件中显式配置映射器，除非映射器配置文件与接口类不在同一个类路径下。 参考 `SqlSessionFactoryBean` 的 `configLocation` 属性以获取更多信息。

注意 `MapperFactoryBean` 需要配置一个 `SqlSessionFactory` 或 `SqlSessionTemplate`。它们可以分别通过 `sqlSessionFactory` 和 `sqlSessionTemplate` 属性来进行设置。 如果两者都被设置， `SqlSessionFactory` 将被忽略。由于 `SqlSessionTemplate` 已经设置了一个 session 工厂， `MapperFactoryBean` 将使用那个工厂。

Java 配置

```
@Configuration
public class MyBatisConfig {
  @Bean
  public MapperFactoryBean<UserMapper> userMapper() throws Exception {
    MapperFactoryBean<UserMapper> factoryBean = new MapperFactoryBean<>(UserMapper.class);
    factoryBean.setSqlSessionFactory(sqlSessionFactory());
    return factoryBean;
  }
}
```

发现映射器

不需要一个个地注册你的所有映射器。你可以让 MyBatis-Spring 对类路径进行扫描来发现它们。

有几种办法来发现映射器：

- 使用 `<mybatis:scan/>` 元素
- 使用 `@MapperScan` 注解
- 在经典 Spring XML 配置文件中注册一个 `MapperScannerConfigurer`

`<mybatis:scan/>` 和 `@MapperScan` 都在 MyBatis-Spring 1.2.0 中被引入。`@MapperScan` 需要你使用 Spring 3.1+。

从 2.0.2 版本开始，mapper 扫描机制支持控制 mapper bean 的懒加载（`lazy-initialization`），这个选项是可选的。 添加这个选项是为了支持 Spring Boot 2.2 中的懒加载特性。 默认的选项值为 `false` （即不开启懒加载）。 如果开发者想使用懒加载的特性，需要显式地将其设置为 `true`。

IMPORTANT 如果开发者想使用懒加载的特性，需要首先知道其局限性。 如果有下列情况，懒加载将在你的应用中不起作用：

- 当使用 `<association>`（`@One`）与 `<collection>`（`@Many`）指向**其它的 mapper** 时
- 当使用 `<include>` 将**其它的 mapper** 的一部分包含进来时
- 当使用 `<cache-ref>`（`@CacheNamespaceRef`）指向**其它的 mapper** 的缓存时
- 当使用 `<select resultMap="...">`（`@ResultMap`）指向**其它的 mapper** 的结果集时

NOTE 然而，通过使用 `@DependsOn`（Spring 的特性）在初始化依赖 bean 的同时，可以使用懒加载，如下所示：

```
@DependsOn("vendorMapper")
public interface GoodsMapper {
  // ...
}
```

2.0.6 起，开发者可以通过 mapper 扫描的特性，使用（`default-scope` 的选项和作用域注解来指定扫描的 bean 的作用域（`@Scope`、`@RefreshScope` 等）。 添加这个可选项是为了支持 Spring Cloud 的 `refresh` 作用域的特性。可选项默认为空（相当于指定 `singleton` 作用域）。 当被扫描的 bean 定义在 `singleton` 作用域（默认作用域），且若最终的作用域不是 `singleton` 时，为其创建一个基于作用域的代理 bean， `default-scope` 作用于 mapper bean（`MapperFactoryBean`）。

<mybatis:scan>

`<mybatis:scan/>` 元素会发现映射器，它发现映射器的方法与 Spring 内建的 `<context:component-scan/>` 发现 bean 的方法非常类似。

下面是一个 XML 配置样例：

```
<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="org.mybatis.spring.sample.mapper" />

  <!-- ... -->

</beans>
```

`base-package` 属性允许你设置映射器接口文件的基础包。通过使用逗号或分号分隔，你可以设置多个包。并且会在你所指定的包中递归搜索映射器。

注意，不需要为 `<mybatis:scan/>` 指定 `SqlSessionFactory` 或 `SqlSessionTemplate`，这是因为它将使用能够被自动注入的 `MapperFactoryBean`。但如果你正在使用多个数据源（`DataSource`），自动注入可能不适合你。 在这种情况下，你可以使用 `factory-ref` 或 `template-ref` 属性指定你想使用的 bean 名称。

`<mybatis:scan/>` 支持基于标记接口或注解的过滤操作。在 `annotation` 属性中，可以指定映射器应该具有的特定注解。而在 `marker-interface` 属性中，可以指定映射器应该继承的父接口。当这两个属性都被设置的时候，被发现的映射器会满足这两个条件。 默认情况下，这两个属性为空，因此在基础包中的所有接口都会被作为映射器被发现。

被发现的映射器会按照 Spring 对自动发现组件的默认命名策略进行命名（参考 the Spring reference document(Core Technologies -Naming autodetected components-)🔗）。 也就是说，如果没有使用注解显式指定名称，将会使用映射器的首字母小写非全限定类名作为名称。但如果发现映射器具有 `@Component` 或 JSR-330 标准中 `@Named` 注解，会使用注解中的名称作为名称。 提醒一下，你可以设置 `annotation` 属性为你自定义的注解，然后在你的注解上设置 `org.springframework.stereotype.Component` 或 `jakarta.inject.Named`（需要使用 Jakarta EE 以上）注解，这样你的注解既可以作为标记，也可以作为一个名字提供器来使用了。

提示 `<context:component-scan/>` 无法发现并注册映射器。映射器的本质是接口，为了将它们注册到 Spring 中，发现器必须知道如何为找到的每个接口创建一个 `MapperFactoryBean`。

@MapperScan

当你正在使用 Spring 的基于 Java 的配置时（也就是 `@Configuration`），相比于使用 `<mybatis:scan/>`，你会更喜欢用 `@MapperScan`。

`@MapperScan` 注解的使用方法如下：

```
@Configuration
@MapperScan("org.mybatis.spring.sample.mapper")
public class AppConfig {
  // ...
}
```

这个注解具有与之前见过的 `<mybatis:scan/>` 元素一样的工作方式。它也可以通过 `markerInterface` 和 `annotationClass` 属性设置标记接口或注解类。 通过配置 `sqlSessionFactory` 和 `sqlSessionTemplate` 属性，你还能指定一个 `SqlSessionFactory` 或 `SqlSessionTemplate`。

NOTE 从 2.0.4 起，如果 `basePackageClasses` 或 `basePackages` 没有定义， 扫描将基于声明这个注解的类所在的包。

MapperScannerConfigurer

`MapperScannerConfigurer` 是一个 `BeanDefinitionRegistryPostProcessor`，这样就可以作为一个 bean，包含在经典的 XML 应用上下文中。为了配置 `MapperScannerConfigurer`，使用下面的 Spring 配置：

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

如果你需要指定 `sqlSessionFactory` 或 `sqlSessionTemplate`，那你应该要指定的是 **bean 名**而不是 bean 的引用，因此要使用 `value` 属性而不是通常的 `ref` 属性：

```
<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />
```

提示 在 MyBatis-Spring 1.0.2 之前， `sqlSessionFactoryBean` 和 `sqlSessionTemplateBean` 属性是唯一可用的属性。 但由于 `MapperScannerConfigurer` 在启动过程中比 `PropertyPlaceholderConfigurer` 运行得更早，经常会产生错误。基于这个原因，上述的属性已被废弃，现在建议使用 `sqlSessionFactoryBeanName` 和 `sqlSessionTemplateBeanName` 属性。