简介

翻译

用户可以在以下翻译中了解 MyBatis-Spring-Boot-Starter：

英语
简体中文

什么是 MyBatis-Spring-Boot-Starter?

MyBatis-Spring-Boot-Starter 帮助您快速在 Spring Boot 上构建 MyBatis 应用程序。

通过使用此模块，您将实现：

- 构建独立应用程序
- 将样板代码减少到几乎为零
- 较少的 XML 配置

需求

MyBatis-Spring-Boot-Starter 需要以下版本：

MyBatis-Spring-Boot-启动器	MyBatis-Spring	Spring Boot	Java：Java
3.0	3.0	3.0 - 3.4	17 或更高
2.3	2.1	2.7	8 或更高
~~**2.2 (文件结束)**~~	~~2.0 (需要 2.0.6+以启用所有功能)~~	~~2.5 - 2.7~~	~~8 或更高~~
~~**2.1 (文件结束)**~~	~~2.0 (需要 2.0.6+以启用所有功能)~~	~~2.1 - 2.4~~	~~8 或更高~~
~~**2.0 (EOL)**~~	~~2.0~~	~~2.0 或 2.1~~	~~8 或更高~~
~~**1.3 (文件结束)**~~	~~1.3~~	~~1.5~~	~~6 或更高~~
~~**1.2 (文件结束)**~~	~~1.3~~	~~1.4~~	~~6 或更高~~
~~**1.1 (换行符)**~~	~~1.3~~	~~1.3~~	~~6 或更高~~
~~**1.0 (文件结束)**~~	~~1.2~~	~~1.3~~	~~6 或更高~~

安装

要使用 MyBatis-Spring-Boot-Starter 模块，您只需将 `mybatis-spring-boot-autoconfigure.jar` 文件及其依赖项（`mybatis.jar`、`mybatis-spring.jar` 等）包含在类路径中。

Maven

如果您正在使用 Maven，只需将以下依赖项添加到您的 `pom.xml` 中：

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>3.0.4</version>
</dependency>
```

Gradle

如果使用 gradle，请将以下内容添加到您的 `build.gradle` 中：

```
dependencies {
  implementation("org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.4")
}
```

快速设置

如您可能已经知道，要使用 MyBatis 与 Spring 集成，您至少需要一个 `SqlSessionFactory` 以及至少一个 mapper 接口。

MyBatis-Spring-Boot-Starter 将：

- 自动检测现有的 `数据源`
- 将创建并注册一个 `SqlSessionFactory` 实例，使用 `DataSource` 作为输入，通过 `SqlSessionFactoryBean` 进行操作
- 将创建并注册一个从 `SqlSessionFactory` 获取的 `SqlSessionTemplate` 实例
- 自动扫描您的映射器，将它们链接到 `SqlSessionTemplate` 并将其注册到 Spring 上下文中，以便它们可以注入到您的 Bean 中

假设我们有以下映射器：

```
@Mapper
public interface CityMapper {

  @Select("SELECT * FROM CITY WHERE state = #{state}")
  City findByState(@Param("state") String state);

}
```

您只需创建一个普通的 Spring Boot 应用程序，并让 mapper 按照以下方式注入（Spring 4.3+可用）：

```
@SpringBootApplication
public class SampleMybatisApplication implements CommandLineRunner {

  private final CityMapper cityMapper;

  public SampleMybatisApplication(CityMapper cityMapper) {
    this.cityMapper = cityMapper;
  }

  public static void main(String[] args) {
    SpringApplication.run(SampleMybatisApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    System.out.println(this.cityMapper.findByState("CA"));
  }

}
```

这是您需要做的全部。您现在可以将应用程序作为正常的 Spring Boot 应用程序运行。

高级扫描

MyBatis-Spring-Boot-Starter 默认会搜索带有 `@Mapper` 注解的映射器。

您可能需要指定一个自定义注解或标记接口进行扫描。如果是这样，您必须使用 `@MapperScan` 注解。更多相关信息请参阅MyBatis-Spring 参考页面 。

MyBatis-Spring-Boot-Starter 如果在 Spring 的上下文中找到至少一个 `MapperFactoryBean`，则不会启动扫描过程。因此如果您想完全停止扫描，应该使用 `@Bean` 方法型式注册您的映射器。

使用 SqlSession

一个 `SqlSessionTemplate` 实例被创建并添加到 Spring 上下文中，因此您可以使用 MyBatis API，将其注入到您的 Bean 中，如下所示（在 Spring 4.3+版本中可用）：

```
@Component
public class CityDao {

  private final SqlSession sqlSession;

  public CityDao(SqlSession sqlSession) {
    this.sqlSession = sqlSession;
  }

  public City selectCityById(long id) {
    return this.sqlSession.selectOne("selectCityById", id);
  }

}
```

配置

任何其他 Spring Boot 应用程序一样，MyBatis-Spring-Boot-Application 的配置参数存储在 `application.properties`（或 `application.yml`）中。

MyBatis 使用前缀 `mybatis` 来表示其属性。

可用属性包括：

财产	描述
`配置位置`	MyBatis xml 配置文件的位置。
`检查配置位置`	指示是否执行 MyBatis xml 配置文件的存活性检查。
`mapper-locations`	位置信息：Mapper xml 配置文件的位置。
`类型别名包`	搜索类型别名的包。（包分隔符为",; \t\n"）
`类型别名-超类型`	过滤类型别名的超类。如果此处未指定，MyBatis 将将所有从 `type-aliases-package` 搜索到的类视为类型别名。
`类型处理器包`	搜索类型处理器的包。（包分隔符为",; \t\n"）
`执行器类型`	执行器类型：`简单`，`重用`，`批量`
`default-scripting-language-driver`	默认脚本语言驱动类。此功能需要与 mybatis-spring 2.0.2+一起使用。

`configuration-properties`	外部化属性用于 MyBatis 配置。指定的属性可以在 MyBatis 配置文件和 Mapper 文件中使用作为占位符。详细信息请参阅MyBatis 参考页面。
`lazy-initialization`	是否启用映射器 bean 的懒加载。将 `true` 设置为启用懒加载。此功能需要与 mybatis-spring 2.0.2+一起使用。
`mapper-default-scope`	默认由自动配置扫描的 mapper bean 的作用域。此功能需要与 mybatis-spring 2.0.6+一起使用。
`inject-sql-session-on-mapper-scan`	设置是否注入 `SqlSessionTemplate` 或 `SqlSessionFactory` 实例（如果您想回到 2.2.1 或更早的行为，请指定 `false`）。如果您与 spring-native 一起使用，应设置为 `true`（默认）。
`configuration.*`	属性键为 MyBatis Core 提供的 `configuration` 实例。有关可用嵌套属性，请参阅 MyBatis 参考页面。提示：此属性不能与 `config-location` 同时使用。
`scripting-language-driver.thymeleaf.*`	属性键为 MyBatis Thymeleaf 提供的 `ThymeleafLanguageDriverConfig` 实例。有关可用嵌套属性，请参阅 MyBatis Thymeleaf 参考页面。
`scripting-language-driver.freemarker.*`	属性键为 MyBatis FreeMarker 提供的 `FreeMarkerLanguageDriverConfig` bean。有关可用嵌套属性，请参阅MyBatis FreeMarker 参考页面。此功能需要与 mybatis-freemarker 1.2.0+一起使用。
`scripting-language-driver.velocity.*`	属性键用于 MyBatis Velocity 提供的`VelocityLanguageDriverConfig` Bean。有关可用嵌套属性的详细信息，请参阅MyBatis Velocity 参考页面。此功能需要与 mybatis-velocity 2.1.0+一起使用。

例如：

```
# application.properties
mybatis.type-aliases-package=com.example.domain.model
mybatis.type-handlers-package=com.example.typehandler
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.configuration.default-fetch-size=100
mybatis.configuration.default-statement-timeout=30
...
```

```
# application.yml
mybatis:
    type-aliases-package: com.example.domain.model
    type-handlers-package: com.example.typehandler
    configuration:
        map-underscore-to-camel-case: true
        default-fetch-size: 100
        default-statement-timeout: 30
...
```

使用配置定制器

MyBatis-Spring-Boot-Starter 提供了使用 Java 配置自定义由自动配置生成的 MyBatis 配置的机会。MyBatis-Spring-Boot-Starter 将自动搜索实现 `ConfigurationCustomizer` 接口的 bean，并调用一个自定义 MyBatis 配置的方法。（自 1.2.1 版本起可用）

例如：

```
@Configuration
public class MyBatisConfig {
  @Bean
  ConfigurationCustomizer mybatisConfigurationCustomizer() {
    return new ConfigurationCustomizer() {
      @Override
      public void customize(Configuration configuration) {
        // customize ...
      }
    };
  }
}
```

使用 SqlSessionFactoryBeanCustomizer

MyBatis-Spring-Boot-Starter 提供了使用 Java 配置自定义由自动配置生成的 `SqlSessionFactoryBean` 的机会。MyBatis-Spring-Boot-Starter 将自动搜索实现 `SqlSessionFactoryBeanCustomizer` 接口的 bean，并调用一个自定义 `SqlSessionFactoryBean` 的方法。（自 2.2.2 版本起可用）

例如：

```
@Configuration
public class MyBatisConfig {
  @Bean
  SqlSessionFactoryBeanCustomizer sqlSessionFactoryBeanCustomizer() {
    return new SqlSessionFactoryBeanCustomizer() {
      @Override
      public void customize(SqlSessionFactoryBean factoryBean) {
        // customize ...
      }
    };
  }
}
```

使用 SpringBootVFS

MyBatis-Spring-Boot-Starter 提供了 `SpringBootVFS` 作为 `VFS` 的实现类。`VFS` 用于从应用程序（或应用程序服务器）中搜索类（例如类型别名的目标类、类型处理器类）。如果您使用可执行 jar 运行 Spring Boot 应用程序，则需要使用 `SpringBootVFS`。MyBatis-Spring-Boot-Starter 提供的自动配置功能会自动使用它，但手动配置（例如使用多个 `DataSource` 时）不会自动使用。

如何手动配置使用 `SpringBootVFS`：

```
@Configuration
public class MyBatisConfig {
  @Bean
  public SqlSessionFactory masterSqlSessionFactory() throws Exception {
    SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
    factoryBean.setDataSource(masterDataSource());
    factoryBean.setVfs(SpringBootVFS.class); // Sets the SpringBootVFS class into SqlSessionFactoryBean
    // ...
    return factoryBean.getObject();
  }
}
```

检测 MyBatis 组件

MyBatis-Spring-Boot-Starter 将检测实现以下由 MyBatis 提供的接口的 Bean。

- 拦截器
- 类型处理器
- 语言驱动程序（需要与 mybatis-spring 2.0.2+一起使用）
- 数据库 Id 提供者

```
@Configuration
public class MyBatisConfig {
  @Bean
  MyInterceptor myInterceptor() {
    return MyInterceptor();
  }
  @Bean
  MyTypeHandler myTypeHandler() {
    return MyTypeHandler();
  }
  @Bean
  MyLanguageDriver myLanguageDriver() {
    return MyLanguageDriver();
  }
  @Bean
  VendorDatabaseIdProvider databaseIdProvider() {
    VendorDatabaseIdProvider databaseIdProvider = new VendorDatabaseIdProvider();
    Properties properties = new Properties();
    properties.put("SQL Server", "sqlserver");
    properties.put("DB2", "db2");
    properties.put("H2", "h2");
    databaseIdProvider.setProperties(properties);
    return databaseIdProvider;
  }
}
```

提示：如果检测到 `LangaugeDriver` 的计数为一位，则自动设置为默认脚本语言。

语言驱动器的定制

如果您想自定义由自动配置创建的 `LanguageDriver`，请注册用户定义的 Bean。

Thymeleaf 语言驱动程序

```
@Configuration
public class MyBatisConfig {
  @Bean
  ThymeleafLanguageDriverConfig thymeleafLanguageDriverConfig() {
    return ThymeleafLanguageDriverConfig.newInstance(c -> {
      // ... customization code
    });
  }
}
```

FreeMarker 语言驱动器配置

```
@Configuration
public class MyBatisConfig {
  @Bean
  FreeMarkerLanguageDriverConfig freeMarkerLanguageDriverConfig() {
    return FreeMarkerLanguageDriverConfig.newInstance(c -> {
      // ... customization code
    });
  }
}
```

速度语言驱动器

```
@Configuration
public class MyBatisConfig {
  @Bean
```

```
  VelocityLanguageDriverConfig velocityLanguageDriverConfig() {
    return VelocityLanguageDriverConfig.newInstance(c -> {
      // ... customization code
    });
  }
}
```

运行示例

该项目提供了两个样本，以便您运行游戏和实验：

类别	样本	描述
核心	第一个样本 🥊	展示仅包含 mapper 和一个 bean 的最简单场景，其中 mapper 被注入到 bean 中。这是我们之前在快速设置部分看到的示例。
	第二个样本 🥊	展示如何使用一个其语句在 xml 文件中的 Mapper，以及使用 `SqlSessionTemplate` 的 Dao。
语言驱动程序	第 3 个样本 🥊	展示如何使用 Thymeleaf 的 mybatis-thymeleaf 语言驱动。
	第 4 个样本 🥊	展示如何使用 FreeMarker 语言驱动程序与 mybatis-freemarker 结合使用。
	第 5 个样本 🥊	展示如何使用 mybatis-velocity 与 Velocity 语言驱动程序。
Java 虚拟机语言	第 6 个样本 🥊	展示如何使用 Kotlin。
	第 7 个样本 🥊	展示如何使用 Groovy。
网络	第 8 个样本 🥊	展示如何使用网络环境。
	第九样本 🥊	展示如何使用 Web 环境与 war（=部署到应用服务器）相关联。