http://www.mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/index.html

**mybatis-spring-boot-autoconfigure**

## Introduction

### What is MyBatis-Spring-Boot-Starter?

The MyBatis-Spring-Boot-Starter help you build quickly MyBatis applications on top of the [Spring Boot](http://projects.spring.io/spring-boot/).

By using this module you will achieve:

* Build standalone applications.
* Reduce the boilerplate to almost zero.
* Less XML configuration.

### Requirements

The MyBatis-Spring-Boot-Starter requires Java 6 or higher and the following MyBatis-Spring and Spring Boot versions:

| **MyBatis-Spring-Boot-Starter** | [**MyBatis-Spring**](http://www.mybatis.org/spring/index.html#Requirements) | **Spring Boot** |
| --- | --- | --- |
| **1.3.x (1.3.1)** | 1.3 or higher | 1.5 or higher |
| **1.2.x (1.2.1)** | 1.3 or higher | 1.4 or higher |
| **1.1.x (1.1.1)** | 1.3 or higher | 1.3 or higher |
| **1.0.x (1.0.2)** | 1.2 or higher | 1.3 or higher |

### Installation

To use the MyBatis-Spring-Boot-Starter module, you just need to include the mybatis-spring-boot-autoconfigure.jar file and its dependencies(mybatis.jar, mybatis-spring.jar and etc ...) in the classpath.

If you are using Maven just add the following dependency to your pom.xml:

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>1.3.2</version>

</dependency>

If using gradle add this to your build.gradle:

dependencies {

compile("org.mybatis.spring.boot:mybatis-spring-boot-starter:1.3.2")

}

### Quick Setup

As you may already know, to use MyBatis with Spring you need at least an SqlSessionFactory and at least one mapper interface.

MyBatis-Spring-Boot-Starter will:

* Autodetect an existing DataSource.
* Will create and register an instance of a SqlSessionFactory passing that DataSource as an input using the SqlSessionFactoryBean.
* Will create and register an instance of a SqlSessionTemplate got out of the SqlSessionFactory.
* Autoscan your mappers, link them to the SqlSessionTemplate and register them to Spring context so they can be injected into your beans.

Suppose we have the following mapper:

@Mapper

public interface CityMapper {

@Select("SELECT \* FROM CITY WHERE state = #{state}")

City findByState(@Param("state") String state);

}

You just need to create a normal Spring boot application and let the mapper be injected like follows(available on Spring 4.3+):

@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"));

}

}

This is all you have to do. You application can now be run as a normal Spring Boot application.

### Advanced scanning

The MyBatis-Spring-Boot-Starter will search, by default, for mappers marked with the @Mapper annotation.

You may want to specify a custom annotation or a marker interface for scanning. If so, you must use the @MapperScan annotation. See more about it in the [MyBatis-Spring reference page.](http://www.mybatis.org/spring/mappers.html#scan)

The MyBatis-Spring-Boot-Starter will not start the scanning process if it finds at least one MapperFactoryBean in the Spring's context so if you want to stop the scanning at all you should register your mappers explicitly with @Bean methods.

### Using an SqlSession

An instance of a SqlSessionTemplate is created and added to the Spring context, so you can use the MyBatis API letting it be injected into your beans like follows(available on Spring 4.3+):

@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);

}

}

### Configuration

As any other Spring Boot application a MyBatis-Spring-Boot-Application configuration parameters are stored inside the application.properties(or application.yml).

MyBatis uses the prefix mybatis for its properties

Available properties are:

| **Property** | **Description** |
| --- | --- |
| config-location | Location of MyBatis xml config file. |
| check-config-location | Indicates whether perform presence check of the MyBatis xml config file. |
| mapper-locations | Locations of Mapper xml config file. |
| type-aliases-package | Packages to search for type aliases. (Package delimiters are ",; \t\n") |
| type-handlers-package | Packages to search for type handlers. (Package delimiters are ",; \t\n") |
| executor-type | Executor type: SIMPLE, REUSE, BATCH. |
| configuration-properties | Externalized properties for MyBatis configuration. Specified properties can be used as placeholder on MyBatis config file and Mapper file. For detail see the [MyBatis reference page](http://www.mybatis.org/mybatis-3/configuration.html#properties) |
| configuration | A MyBatis Configuration bean. About available properties see the [MyBatis reference page](http://www.mybatis.org/mybatis-3/configuration.html#settings). **NOTE** This property cannot be used at the same time with the config-location. |

For example:

# 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

...

### Using a ConfigurationCustomizer

The MyBatis-Spring-Boot-Starter provide opportunity to customize a MyBatis configuration generated by auto-configuration using Java Config. The MyBatis-Spring-Boot-Starter will search beans that implements the ConfigurationCustomizerinterface by automatically, and call a method that customize a MyBatis configuration. (Available since 1.2.1 or above)

For example:

// @Configuration class

@Bean

ConfigurationCustomizer mybatisConfigurationCustomizer() {

return new ConfigurationCustomizer() {

@Override

public void customize(Configuration configuration) {

// customize ...

}

};

}

### Detecting MyBatis components

The MyBatis-Spring-Boot-Starter will detects beans that implements following interface provided by MyBatis.

* [Interceptor](http://www.mybatis.org/mybatis-3/configuration.html#plugins)
* [DatabaseIdProvider](http://www.mybatis.org/mybatis-3/configuration.html#databaseIdProvider)

### Running Samples

The project provides two samples so you play and experiment with them:

| **Sample** | **Description** |
| --- | --- |
| [1st Sample](https://github.com/mybatis/spring-boot-starter/tree/master/mybatis-spring-boot-samples/mybatis-spring-boot-sample-annotation) | Show the simplest scenario with just a mapper and a bean where the mapper is injected into. This is the sample we saw in the Quick Setup section. |
| [2nd Sample](https://github.com/mybatis/spring-boot-starter/tree/master/mybatis-spring-boot-samples/mybatis-spring-boot-sample-xml) | Shows how to use a Mapper that has its statements in an xml file and Dao that uses an SqlSesionTemplate. |