Spring Boot集成MyBatis

1.整合Durid数据源

1、引入Jar包

```

<dependencies>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-jdbc</artifactId>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <!--mybatis的场景启动器
  因为MyBatis它是Spring data jpa
  <dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>2.1.4</version>
  </dependency>
  -->
  <dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.2.3</version>
  </dependency>
```

```xml

 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <scope>runtime</scope>
 </dependency>
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-test</artifactId>
 <scope>test</scope>
 </dependency>
</dependencies>
```

2.application.yml配置

```yaml
#数据源
spring:
 datasource:
 username: root
 password: 123456
 url: jdbc:mysql://localhost:3306/springboot_mybatis?characterEncoding=utf8&useSSL=false&serverTimezone=UTC&
 driver-class-name: com.mysql.cj.jdbc.Driver
 type: com.alibaba.druid.pool.DruidDataSource

  # 数据源其他配置
  initialSize: 5
  minIdle: 5
  maxActive: 20
  maxWait: 60000
  timeBetweenEvictionRunsMillis: 60000
  minEvictableIdleTimeMillis: 300000
  validationQuery: SELECT 1 FROM DUAL
  testWhileIdle: true
  testOnBorrow: false
  testOnReturn: false
  poolPreparedStatements: true
  # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
  filters: stat,wall
  maxPoolPreparedStatementPerConnectionSize: 20
  useGlobalDataSourceStat: true
  connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500
  schema: classpath:sql/mybatis.sql
  initialization-mode: ALWAYS
```

3.读取配置类DruidConfig

```
/**
 * 数据源配置类
 */
@Configuration
public class DruidConfig {
 // 将所有前缀为spring.datasource下的配置项都加载到DataSource中
 @ConfigurationProperties(prefix = "spring.datasource")
 @Bean
 public DataSource druidDataSource() {
  return new DruidDataSource();
  }

  @Bean
  public ServletRegistrationBean statViewServlet() {
  ServletRegistrationBean servletRegistrationBean = new ServletRegistrationBean(new StatViewServlet(), "/druid/*");
  // 添加IP白名单
  servletRegistrationBean.addInitParameter("allow", "127.0.0.1");
  // 添加IP黑名单，当白名单和黑名单重复时，黑名单优先级更高
  servletRegistrationBean.addInitParameter("deny", "127.0.0.1");
  // 添加控制台管理用户
  servletRegistrationBean.addInitParameter("loginUsername", "admin");
  servletRegistrationBean.addInitParameter("loginPassword", "123456");
```

```java
// 是否能够重置数据
servletRegistrationBean.addInitParameter("resetEnable", "false");
return servletRegistrationBean;
}

/**
* 配置服务过滤器
*
* @return 返回过滤器配置对象
*/
@Bean
public FilterRegistrationBean statFilter() {
FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean(new WebStatFilter());
// 添加过滤规则
filterRegistrationBean.addUrlPatterns("/*");
// 忽略过滤格式
filterRegistrationBean.addInitParameter("exclusions", "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*,");
return filterRegistrationBean;
}
}
```

其实没有必要一个个手动去配置， druid 启动starter

```xml
<dependency>
 <groupId>com.alibaba</groupId>
 <artifactId>druid-spring-boot-starter</artifactId>
 <version>1.2.3</version>
</dependency>
```

druid 自动配置类

```java

@Configuration
@ConditionalOnClass(DruidDataSource.class)
@AutoConfigureBefore(DataSourceAutoConfiguration.class)
```

```java
@EnableConfigurationProperties({DruidStatProperties.class, DataSourceProperties.class})
@Import({DruidSpringAopConfiguration.class,
 DruidStatViewServletConfiguration.class,
 DruidWebStatFilterConfiguration.class,
 DruidFilterConfiguration.class})
public class DruidDataSourceAutoConfigure {

 private static final Logger LOGGER = LoggerFactory.getLogger(DruidDataSourceAutoConfigure.class);

 @Bean(initMethod = "init")
 @ConditionalOnMissingBean
 public DataSource dataSource() {
 LOGGER.info("Init DruidDataSource");
 return new DruidDataSourceWrapper();
 }
}
```

2.整合MyBatis

2.1生成MyBatis代码:

pom.xml

```xml
<!-- Mybatis-Generator插件，自动生成代码 -->
<plugin>
<groupId>org.mybatis.generator</groupId>
<artifactId>mybatis-generator-maven-plugin</artifactId>
<version>1.3.5</version>
<configuration>
<configurationFile>${project.basedir}/src/main/resources/generatorConfig.xml</configurationFile>
<verbose>true</verbose>
<overwrite>true</overwrite>
 </configuration>
 <dependencies>
 <!--必須要引入数据库驱动-->
 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <!--必须制定版本-->
 <version>8.0.22</version>
 </dependency>
 </dependencies>
 </plugin>
 </plugins>
</build>
```

generatorConfig.xml

```


<!DOCTYPE generatorConfiguration PUBLIC
 "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
 "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>


 <!--如果需要使用 command的方式生成需要配置数据库驱动的jar包路径
 <classPathEntry location="指定数据驱动的磁盘路径"/>-->

 <!--context 生成上下文 配置生成规则
 id 随意写
 targetRuntime 生成策略
 MyBatis3DynamicSql 默认的，会生成 动态生成sql的方式（没有xml)
 MyBatis3 生成通用的查询，可以指定动态where条件
 MyBatis3Simple 只生成简单的CRUD
 -->
 <context id="simple" targetRuntime="MyBatis3Simple">


 <commentGenerator>
```

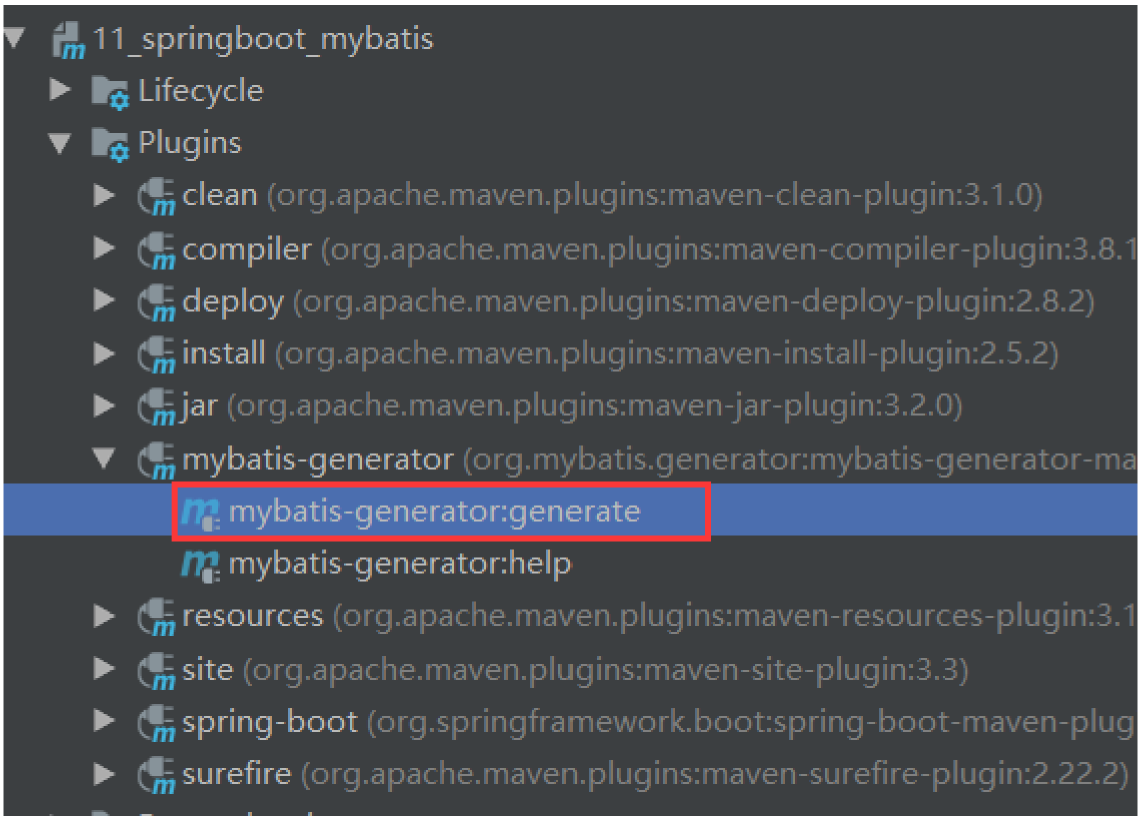
```xml
<!--设置是否生成注释 true 不生成 注意： 如果不生成注释，下次生成代
就不会进行合并-->
<property name="suppressAllComments" value="true"/>
</commentGenerator>
<!--数据源 -->
<jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://localhost:3306/mybatis"
userId="root"
password="123456"/>

<!--pojo
javaModelGenerator java实体生成规则(POJO)
targetPackage 生成到哪个包下
targetProject 生成到当前文件的哪个相对路径下
-->
<javaModelGenerator targetPackage="cn.tulingxueyuan.pojo" targetProject="src/main/java"/>
<!--mapper xml映射文件
sqlMapGenerator mapper xml映射文件生成规则
targetPackage 生成到哪个包下
targetProject 生成到当前文件的哪个相对路径下
-->
<sqlMapGenerator targetPackage="cn.tulingxueyuan.mapper" targetProject="src/main/resources"></sqlMapGenerator>
<!--mapper接口
javaClientGenerator mapper mapper接口生成规则
type 指定生成的方式
1.使用注解的方式生成
2.使用接口绑定的方式生成（要配置sqlMapGenerator）
targetPackage 生成到哪个包下
targetProject 生成到当前文件的哪个相对路径下-->
<javaClientGenerator type="XMLMAPPER" targetPackage="cn.tulingxueyuan.mapper" targetProject="src/main/java"/>


<!--配置哪些表需要进行代码生成
```

```
 tableName 表名
 domainObjectName pojo类名
 mapperName 对应mapper接口的类名 和 mapper xml文件名
 -->
 <table tableName="emp" domainObjectName="Emp" mapperName="EmpMapper" />
 <table tableName="dept" domainObjectName="Dept" mapperName="DeptMapper" />
 </context>
</generatorConfiguration>
```

运行插件---生成代码



```
11_springboot_mybatis
  Lifecycle
  Plugins
    clean (org.apache.maven.plugins:maven-clean-plugin:3.1.0)
    compiler (org.apache.maven.plugins:maven-compiler-plugin:3.8.1
    deploy (org.apache.maven.plugins:maven-deploy-plugin:2.8.2)
    install (org.apache.maven.plugins:maven-install-plugin:2.5.2)
    jar (org.apache.maven.plugins:maven-jar-plugin:3.2.0)
    mybatis-generator (org.mybatis.generator:mybatis-generator-ma
      mybatis-generator:generate
      mybatis-generator:help
    resources (org.apache.maven.plugins:maven-resources-plugin:3.1
    site (org.apache.maven.plugins:maven-site-plugin:3.3)
    spring-boot (org.springframework.boot:spring-boot-maven-plug
    surefire (org.apache.maven.plugins:maven-surefire-plugin:2.22.2)
```

2.2 整合Mybatis

1.引入jar包

```

1、引入Jar包
<dependency>
 <groupId>org.mybatis.spring.boot</groupId>
 <artifactId>mybatis-spring-boot-starter</artifactId>
 <version>1.3.2</version>
```

```
</dependency>


```

application.yml

```
mybatis:
# 映射文件所在路径
mapper-locations: classpath:mappers/*.xml
# pojo类所在包路径
type-aliases-package: com.cx.user.model
```

MyBatis自动配置原理

```
@Bean
@ConditionalOnMissingBean
public SqlSessionFactory sqlSessionFactory(DataSource
dataSource) throws Exception {
 SqlSessionFactoryBean factory = new SqlSessionFactoryBean();
 factory.setDataSource(dataSource);
 factory.setVfs(SpringBootVFS.class);

 // 设置Mybaits的全局配置文件
 if (StringUtils.hasText(this.properties.getConfigLocation())) {
factory.setConfigLocation(this.resourceLoader.getResource(this.pro
perties.getConfigLocation()));
  }
  // ? 有另一种定制方式的体现
  applyConfiguration(factory);
  // 相当于mybatis全局配置文件中
  /*<properties>
  <property name="" value=""/>
 </properties>*/
  if (this.properties.getConfigurationProperties() != null) {
  factory.setConfigurationProperties(this.properties.getConfigur
ationProperties());
  }
  // 就是配置插件-拦截器 只需要配置一个实现了Interceptor的接口为Bean
```

```
if (!ObjectUtils.isEmpty(this.interceptors)) {
factory.setPlugins(this.interceptors);
}
// 设置数据库厂商id
if (this.databaseIdProvider != null) {
factory.setDatabaseIdProvider(this.databaseIdProvider);
}
// 设置别名：去application.yml中獲取mybatis.typeAliasesPackage
if (StringUtils.hasLength(this.properties.getTypeAliasesPackage())) {
factory.setTypeAliasesPackage(this.properties.getTypeAliasesPackage());
}
// 可以通过父类过滤哪些类需要使用别名
比如：pojo.user extends basePojo
pojo.user2
去application.yml中设置mybatis.typeAliasesSuperType: com.tulingxueyuan.pojo.basePojo
if (this.properties.getTypeAliasesSuperType() != null) {
factory.setTypeAliasesSuperType(this.properties.getTypeAliasesSuperType());
}
// 设置类型处理器
<typeHandlers>
<package name=""/>
</typeHandlers>
if (StringUtils.hasLength(this.properties.getTypeHandlersPackage())) {
factory.setTypeHandlersPackage(this.properties.getTypeHandlersPackage());
}
// 设置类型处理器
<typeHandlers>
<typeHandler handler=""
</typeHandlers>
if (!ObjectUtils.isEmpty(this.typeHandlers)) {
factory.setTypeHandlers(this.typeHandlers);
}
// 设置mapper.xml映射文件：mapper-locations: classpath:com/tulingxueyuan/mapper/*Mapper.xml
if (!ObjectUtils.isEmpty(this.properties.resolveMapperLocations())) {
factory.setMapperLocations(this.properties.resolveMapperLocations());
}
Set<String> factoryPropertyNames = Stream
.of(new BeanWrapperImpl(SqlSessionFactoryBean.class).getPropertyDescriptors()).map(PropertyDescriptor::getName)
.collect(Collectors.toSet());
Class<? extends LanguageDriver> defaultLanguageDriver = this.properties.getDefaultScriptingLanguageDriver();
if (factoryPropertyNames.contains("scriptingLanguageDrivers") && !ObjectUtils.isEmpty(this.languageDrivers)) {
// Need to mybatis-spring 2.0.2+
factory.setScriptingLanguageDrivers(this.languageDrivers);
if (defaultLanguageDriver == null && this.languageDrivers.length == 1) {
defaultLanguageDriver = this.languageDrivers[0].getClass();
}
}
if (factoryPropertyNames.contains("defaultScriptingLanguageDriver")) {
// Need to mybatis-spring 2.0.2+
factory.setDefaultScriptingLanguageDriver(defaultLanguageDriver);
}

return factory.getObject();
}
```

- 如果依然放不下mybatis全局配置文件， springboot 还是支持的：
  - 配置application.yml

```
mybatis:
  config-location: classpath:mybatis-config.xml
```

- mybatis-config.xml

```
<?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">
<!--就是DOCTYPE后面对应的根节点-->
<configuration>

 <!--mybatis的设置选项 可以改变mybatis运行时行为-->
 <settings>
  <setting name="mapUnderscoreToCamelCase" value="true"/>
  </settings>

  <!--类型别名可为 Java 类型设置一个缩写名字。 它仅用于 XML 配置，意在降低冗余的全限定类名书写-->
  <typeAliases>
  <package name="com.tulingxueyuan.pojo"/>
  </typeAliases>

 </configuration>
```

- 如果要设置mybatis的settings怎么设置呢?
  - 1.可以通过mybatis全局配置文件设置
  - 2. 也可以通过在application.yml中配置configuration
    - configuration 它封装mybatis所有信息

```
configuration:
 mapUnderscoreToCamelCase: true
```

- configuration 什么情况=null呢?
  - 没有在application.yml中配置configuration 就会为null
- 如果没有在application.yml中配置config-location 就会new new Configuration();
- 要定制mybatis

- 1. 使用mybatis全局配置文件
- 2. 可以使用application.yml中配置configuration + ConfigurationCustomizer
- 要么使用mybatis的东西，要么使用springboot的， 只能用1种

```java
private void applyConfiguration(SqlSessionFactoryBean factory) {
  Configuration configuration = this.properties.getConfiguration();
  if (configuration == null && !StringUtils.hasText(this.properties.getConfigLocation())) {
  configuration = new Configuration();
  }
  if (configuration != null && !CollectionUtils.isEmpty(this.configurationCustomizers)) {
  for (ConfigurationCustomizer customizer : this.configurationCustomizers) {
  customizer.customize(configuration);
  }
  }
  factory.setConfiguration(configuration);
}
```