

## SpringData

对于数据访问层，无论是 SQL(关系型数据库) 还是 NOSQL(非关系型数据库)，Spring Boot 底层都是采用 Spring Data 的方式进行统一处理。

Spring Boot 底层都是采用 Spring Data 的方式进行统一处理各种数据库，Spring Data 也是 Spring 中与 Spring Boot、Spring Cloud 等齐名的知名项目。

Spring Data 官网：<https://spring.io/projects/spring-data>

数据库相关的启动器：可以参考官方文档：

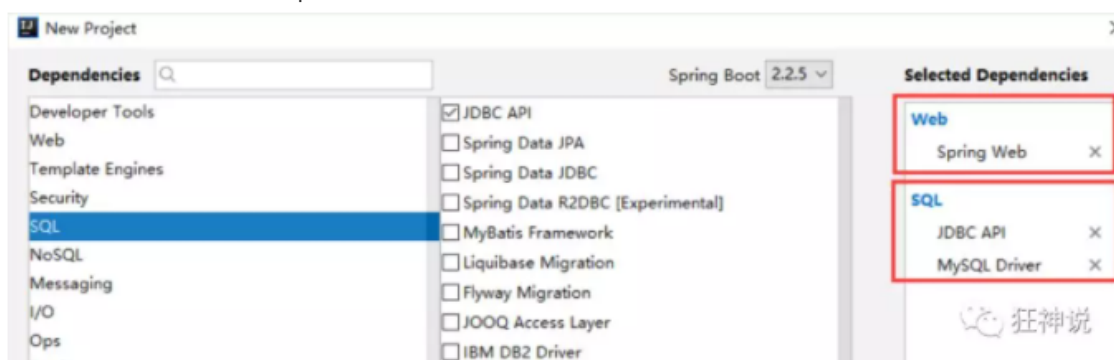
<https://docs.spring.io/spring-boot/docs/2.2.5.RELEASE/reference/htmlsingle/#using-boot-starter>

## 整合JDBC

先装好mysql，新建database，名字叫springboot。

### 创建测试项目测试数据源

1. 新建模块，引入相应的dependencies



2. 建好之后，发现自动导入了如下启动器，也就配置好了要用到的依赖：

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```

3. 编写yaml配置文件，配置数据库连接参数：

```
spring:
  datasource:
    username: root
    password: liu123
    url: jdbc:mysql://localhost:3306/springboot?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.cj.jdbc.Driver
```

其实driver-class-name不用指定，因为 Spring boot 可以从 `url` 推导出大多数数据库。

4. 配置完这一些东西后，我们就可以直接去使用了，因为SpringBoot已经默认帮我们进行了自动配置；去测试类测试一下

```
@SpringBootTest
class Module03jdbcApplicationTest {
    //DI注入数据源
    @Autowired
    DataSource dataSource;

    @Test
    void contextLoads() throws SQLException {
        // 查看默认数据源
        System.out.println(dataSource.getClass());

        // 获得链接
        Connection connection = dataSource.getConnection();

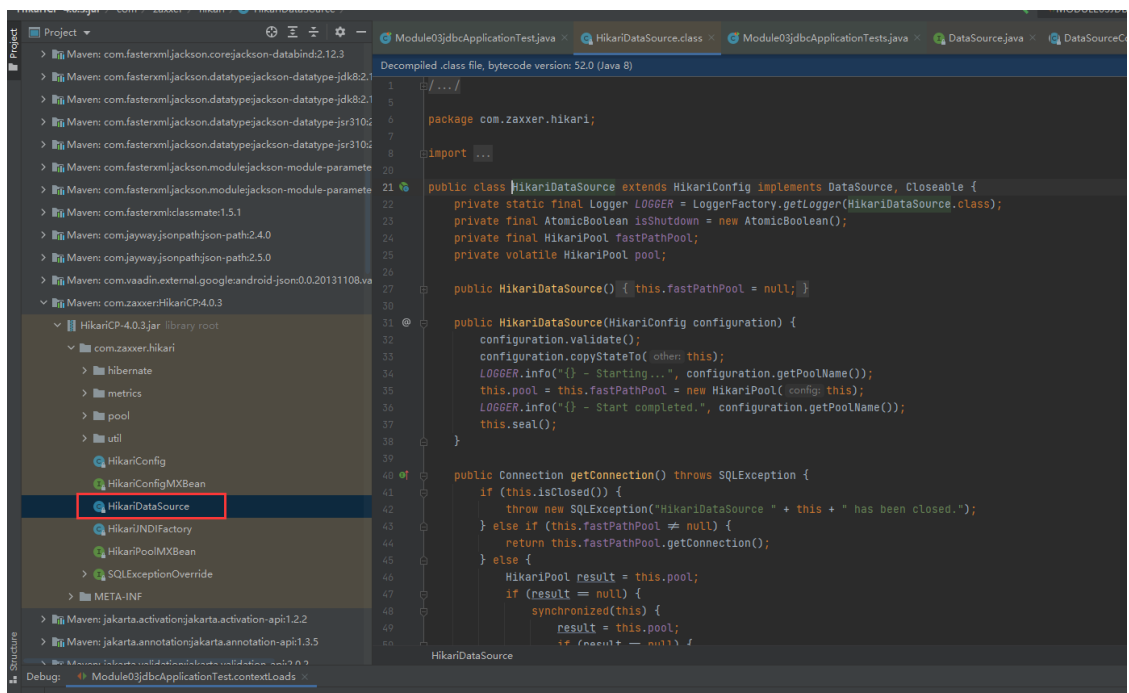
        System.out.println(connection);

        connection.close();
    }
}
```

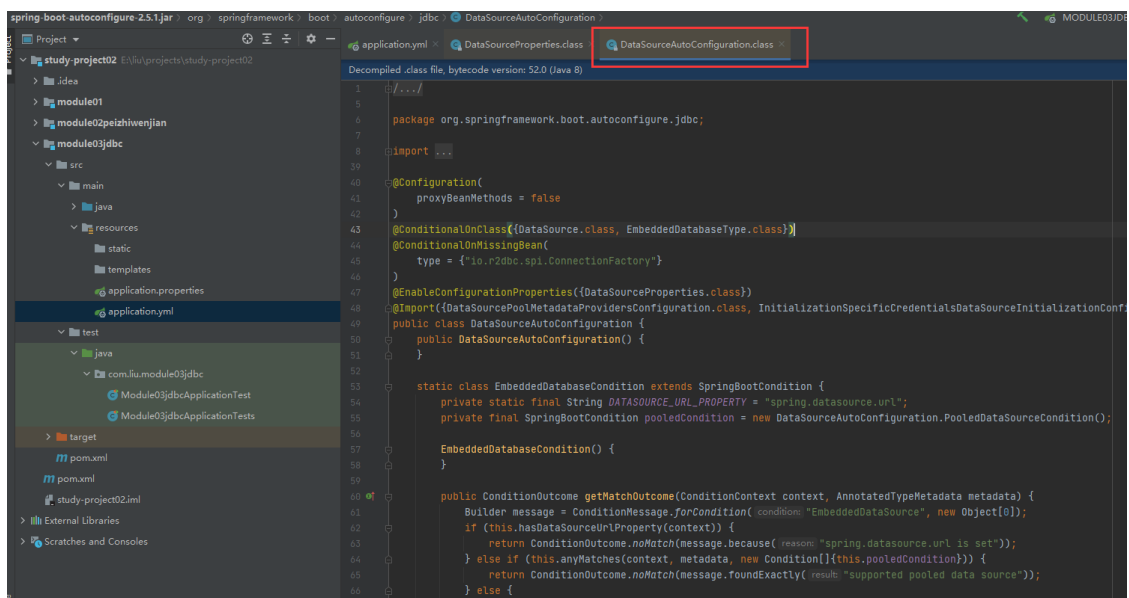
5. 运行结果：

```
class com.zaxxer.hikari.HikariDataSource
HikariProxyConnection@631410926 wrapping
com.mysql.cj.jdbc.ConnectionImpl@37d0d373
```

可以看到默认的数据源是HikariDataSource，去Libraries找一下，发现了：  
连接池可以自己替换



6. 再从yml文件的DataSource进去，发现了配置文件对应的DataSourceProperties类，根据名字查找对应的配置类xxxAutoConfiguration



## JDBC 模板

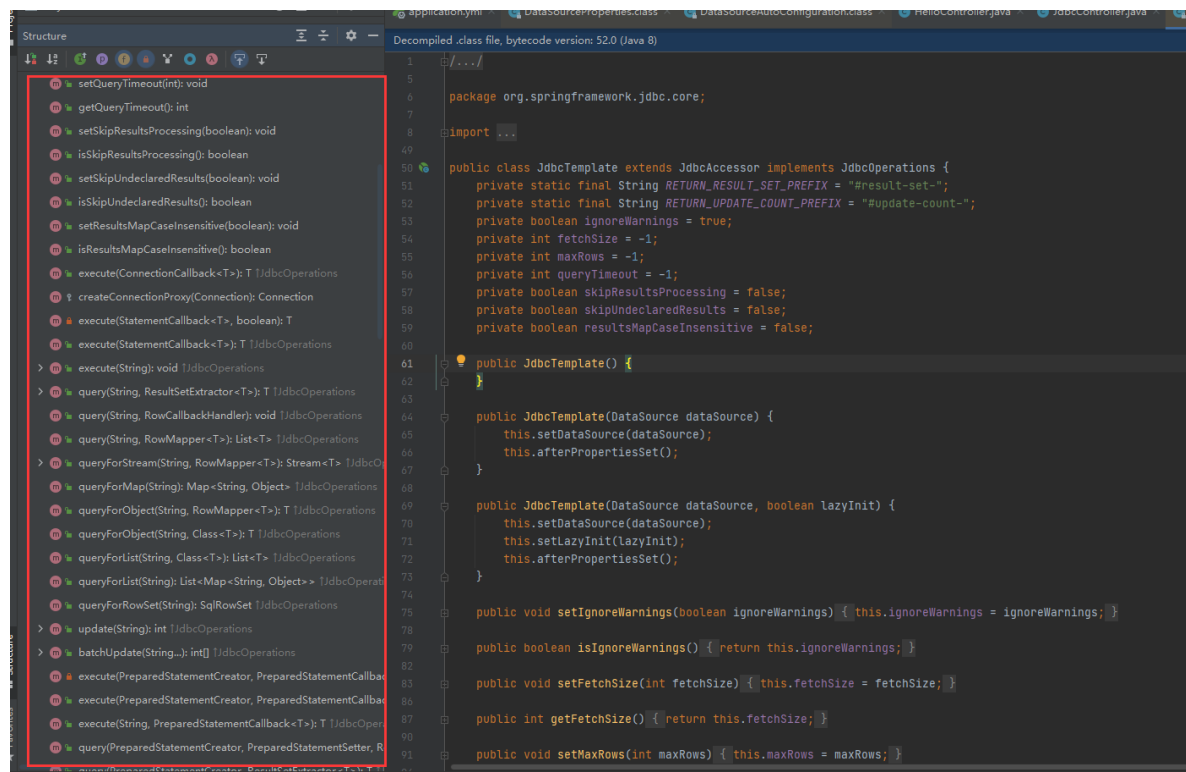
springboot有很多以xxxtemplate为名的模板bean，拿来即用，例如jdbcTemplate可以直接用来CRUD，

```
@RestController
public class JdbcController {

    @Autowired
    JdbcTemplate jdbcTemplate;

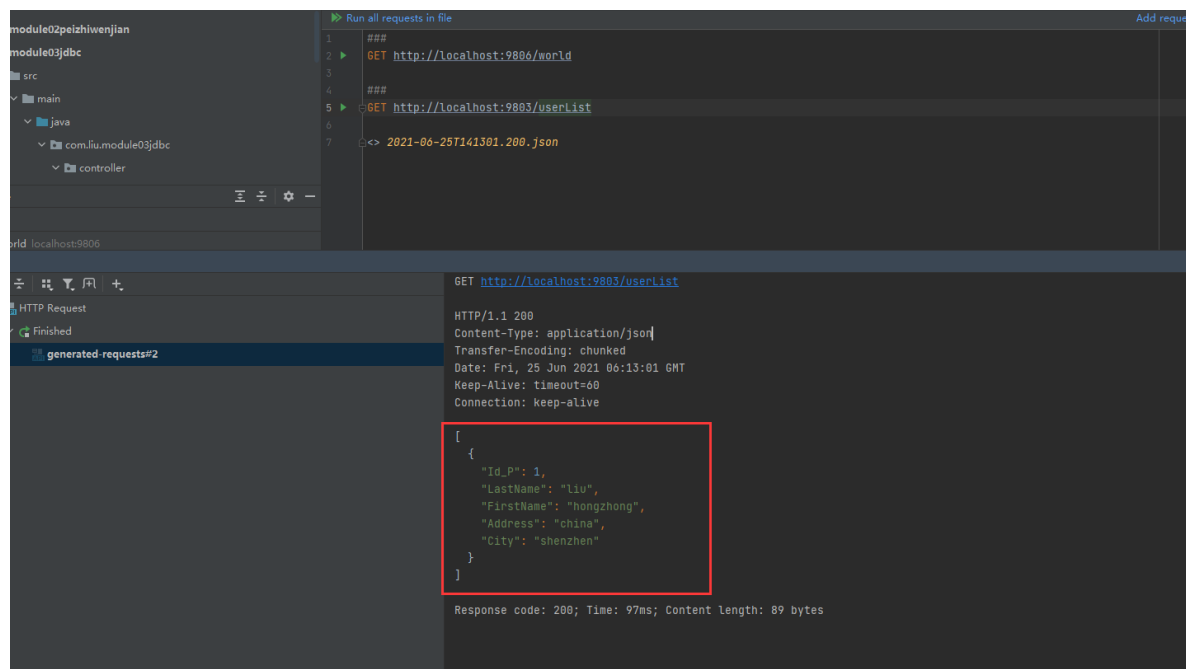
    @GetMapping("/userList")
    public List<Map<String, Object>> userList() {
        String sql = "select * from persons";
        return jdbcTemplate.queryForList(sql);
    }
}
```

点进去看，很多方法在模板里面：



往springboot数据库添加一个persons表用来查询；

启动 测试：



## 整合Mybatis

官方文档：<http://mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/>

mybatis文档：<https://mybatis.org/mybatis-3/zh/getting-started.html#>

狂神博客：[https://mp.weixin.qq.com/s?\\_\\_biz=Mzg2NTAzMTEyNg==&mid=2247483788&idx=1&sn=aabf8cf31d7d45be184cc59cdb75258c&scene=19#wechat\\_redirect](https://mp.weixin.qq.com/s?__biz=Mzg2NTAzMTEyNg==&mid=2247483788&idx=1&sn=aabf8cf31d7d45be184cc59cdb75258c&scene=19#wechat_redirect)

Maven仓库地址：<https://mvnrepository.com/artifact/org.mybatis.spring.boot/mybatis-spring-boot-starter/2.1.1>（有时候没用，另外再找吧）

1. 导入mybatis需要的依赖（当然还有jdbc），还是利用springboot自动配置：

```
<dependencies>
  <dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.21</version>
  </dependency>
  <!--lombok-->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
</dependencies>
```

2. 配置数据库连接信息

```
spring:
  datasource:
    username: root
    password: liu123
    #?serverTimezone=UTC解决时区的报错
    url: jdbc:mysql://localhost:3306/springboot?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    type: com.alibaba.druid.pool.DruidDataSource

    #Spring Boot 默认是不注入这些属性值的，需要自己绑定
    #druid 数据源专有配置
    initialSize: 5
    minIdle: 5
    maxActive: 20
    maxWait: 60000
    timeBetweenEvictionRunsMillis: 60000
    minEvictableIdleTimeMillis: 300000
```

```

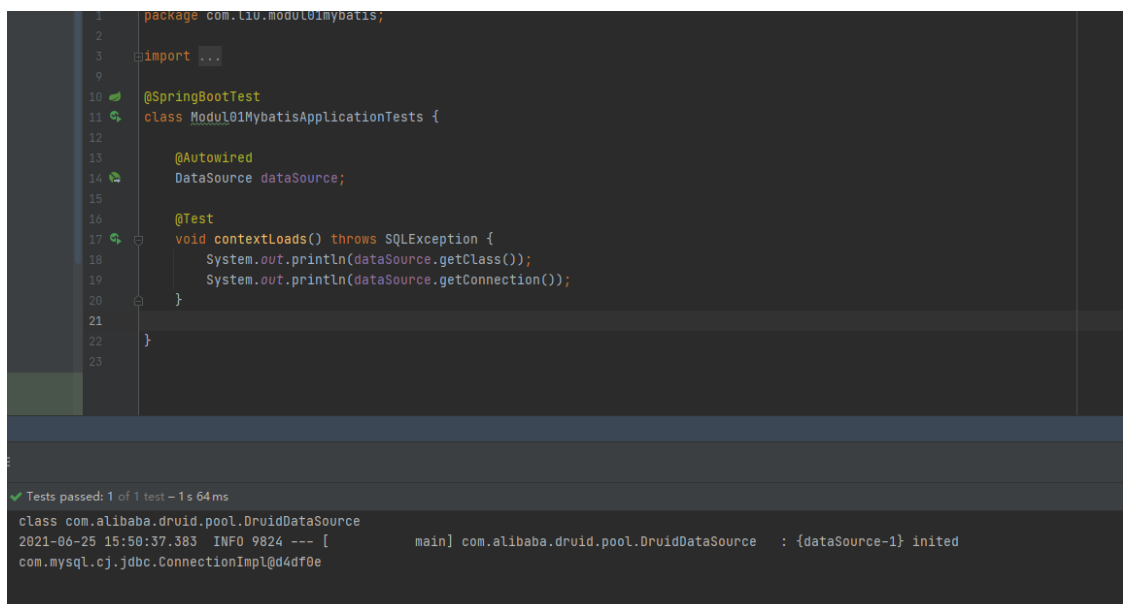
validationQuery: SELECT 1 FROM DUAL
testWhileIdle: true
testOnBorrow: false
testOnReturn: false
poolPreparedStatements: true

#配置监控统计拦截的filters，stat:监控统计、log4j: 日志记录、wall: 防御sql注入
#如果允许时报错 java.lang.ClassNotFoundException:
org.apache.log4j.Priority
#则导入 log4j 依赖即可，Maven 地址：
https://mvnrepository.com/artifact/log4j/log4j
filters: stat,wall,log4j
maxPoolPreparedStatementPerConnectionSize: 20
useGlobalDataSourceStat: true
connectionProperties:
druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500

#mybatis配置
#mapper-locations
#指定 mapper.xml 映射文件的地址，可以有多个。可以是类路径，或者文件系统。** 表示匹
配多级目录。
#路径别写错了
mybatis:
mapper-locations: classpath:mybatis/mapper/*.xml
type-aliases-package: com.liu.pojo

```

### 3. 测试连接是否成功



```

1 package com.liu.module.mybatis;
2
3 import ...
4
5
6
7
8
9
10 @SpringBootTest
11 class Modul01MybatisApplicationTests {
12
13     @Autowired
14     DataSource dataSource;
15
16     @Test
17     void contextLoads() throws SQLException {
18         System.out.println(dataSource.getClass());
19         System.out.println(dataSource.getConnection());
20     }
21 }
22
23

```

```

✓ Tests passed: 1 of 1 test - 1s 64ms
class com.alibaba.druid.pool.DruidDataSource
2021-06-25 15:50:37.383 INFO 9824 --- [main] com.alibaba.druid.pool.DruidDataSource : {dataSource-1} inited
com.mysql.cj.jdbc.ConnectionImpl@d4df0e

```

### 4. 准备数据表,编写对应实体类

```

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Department {
    private int id;
    private String name;
}

```

### 5. 创建mapper目录及对应的mapper接口

```
// @Mapper : 表示本类是一个 MyBatis 的 Mapper
@Mapper
@Repository
public interface DepartmentMapper {

    // 获取所有部门信息
    List<Department> getDepartments();

    // 通过id获得部门
    Department getDepartment(Integer id);
}
```

## 6. 编写测试接口：

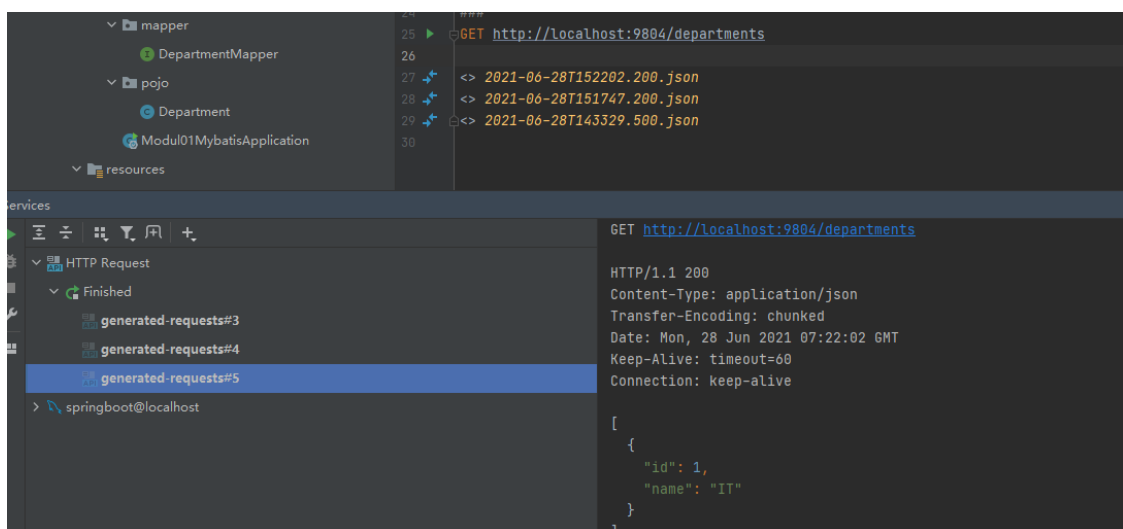
```
@RestController
public class DepartmentController {

    @Autowired
    DepartmentMapper departmentMapper;

    // 查询全部部门
    @GetMapping("/departments")
    public List<Department> getDepartments(){
        return departmentMapper.getDepartments();
    }

    // 查询全部部门
    @GetMapping("/department/{id}")
    public Department getDepartment(@PathVariable("id") Integer id){
        return departmentMapper.getDepartment(id);
    }
}
```

## 7. 测试成功



[MyBatis-Plus \(opens new window\)](#) ( 简称 MP ) 是一个 [MyBatis \(opens new window\)](#)的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生

- 安装

maven:

```
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.4.1</version>
</dependency>
```

- 配置

springboot项目添加 MapperScan 注解

依赖下载失败的话，尝试更换maven仓库：

```
<repositories>
  <repository>
    <id>nexus-aliyun</id>
    <name>Nexus aliyun</name>
    <layout>default</layout>
    <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <releases>
      <enabled>true</enabled>
    </releases>
  </repository>
</repositories>
```

## 自动生成代码

- 添加依赖，MyBatis-Plus 从 3.0.3 之后移除了代码生成器与模板引擎的默认依赖，需要手动添加相关依赖：

```
<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-generator</artifactId>
  <version>3.4.1</version>
</dependency>
```

- 添加 模板引擎 依赖，MyBatis-Plus 支持 Velocity ( 默认 )、Freemarker、Beetl，用户可以选择自己熟悉的模板引擎，如果都不满足您的要求，可以采用自定义模板引擎。

默认：

```
<dependency>
  <groupId>org.apache.velocity</groupId>
  <artifactId>velocity-engine-core</artifactId>
  <version>2.3</version>
</dependency>
```



参考：<https://www.jianshu.com/p/e567c6b2b3ba>

用easyCode插件也可以自动生成代码：<https://www.jianshu.com/p/e4192d7c6844>

## CRUD

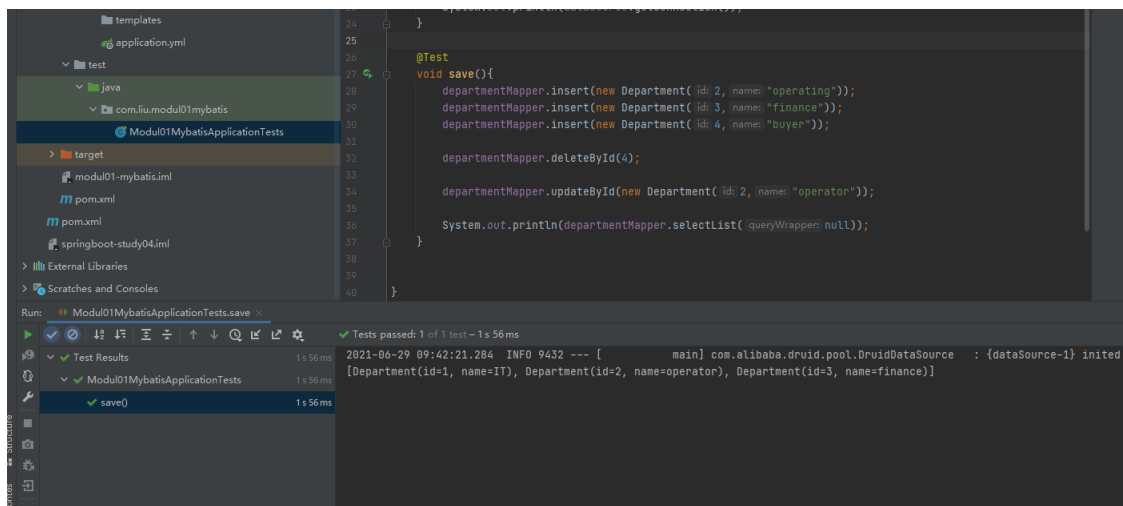
- 准备表和对应实体类：

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Department {
    private int id;
    private String name;
}
```

- 准备mapper继承BaseMapper()即可：

```
public interface DepartmentMapper extends BaseMapper<Department> {}
```

- 测试mapper可用：



如果用在service层使用，继承IService接口，会进一步封装CRUD；