数据访问

1 简介

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。

Sping Data 官网：https://spring.io/projects/spring-data

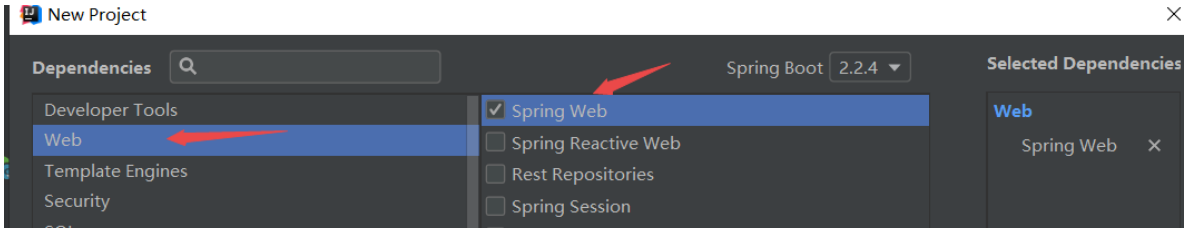
数据库相关的启动器： 可以参考官方文档：https://docs.spring.io/spring-boot/docs/2.1.7.RELEASE/reference/htmlsingle/#using-boot-starter

`spring-boot-starter-data-cassandra`	Starter for using Cassandra distributed database and Spring Data Cassandra	Pom
`spring-boot-starter-data-cassandra-reactive`	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive	Pom
`spring-boot-starter-data-couchbase`	Starter for using Couchbase document-oriented database and Spring Data Couchbase	Pom
`spring-boot-starter-data-couchbase-reactive`	Starter for using Couchbase document-oriented database and Spring Data Couchbase Reactive	Pom
`spring-boot-starter-data-elasticsearch`	Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch	Pom
`spring-boot-starter-data-jpa`	Starter for using Spring Data JPA with Hibernate	Pom
`spring-boot-starter-data-ldap`	Starter for using Spring Data LDAP	Pom
`spring-boot-starter-data-mongodb`	Starter for using MongoDB document-oriented database and Spring Data MongoDB	Pom
`spring-boot-starter-data-mongodb-reactive`	Starter for using MongoDB document-oriented database and Spring Data MongoDB Reactive	Pom
`spring-boot-starter-data-neo4j`	Starter for using Neo4j graph database and Spring Data Neo4j	Pom
`spring-boot-starter-data-redis`	Starter for using Redis key-value data store with Spring Data Redis and the Lettuce client	Pom
`spring-boot-starter-data-redis-reactive`	Starter for using Redis key-value data store with Spring Data Redis reactive and the Lettuce client	Pom
`spring-boot-starter-data-rest`	Starter for exposing Spring Data repositories over REST using Spring Data REST	Pom
`spring-boot-starter-data-solr`	Starter for using the Apache Solr search platform with Spring Data Solr	Pom

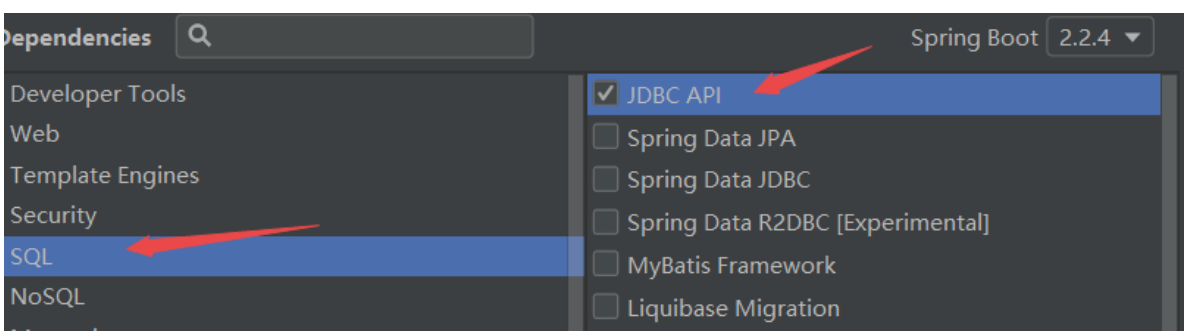
2 搭建环境

- 一样的创建流程，但是由于要使用数据库进行连接，所有要添加支持：

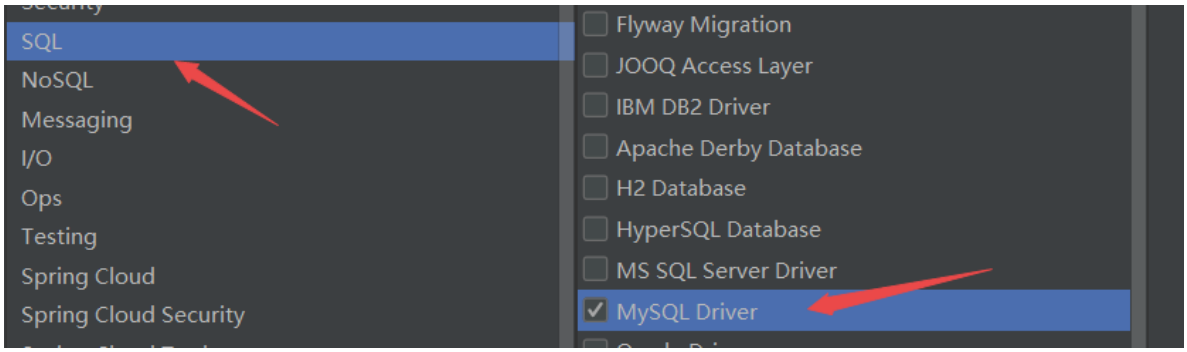
web：【固定的】



jdbc：



mysql驱动：



3 使用原生的JDBC

3.1 配置文件里写数据源的配置

- application.yaml:

```yaml
spring:
  datasource:
    username: root
    password: mynewroot
    # 假如时区报错了，就添加时区的配置：serverTimezone=UTC
    url: jdbc:mysql://localhost:3306/mybatis?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
```

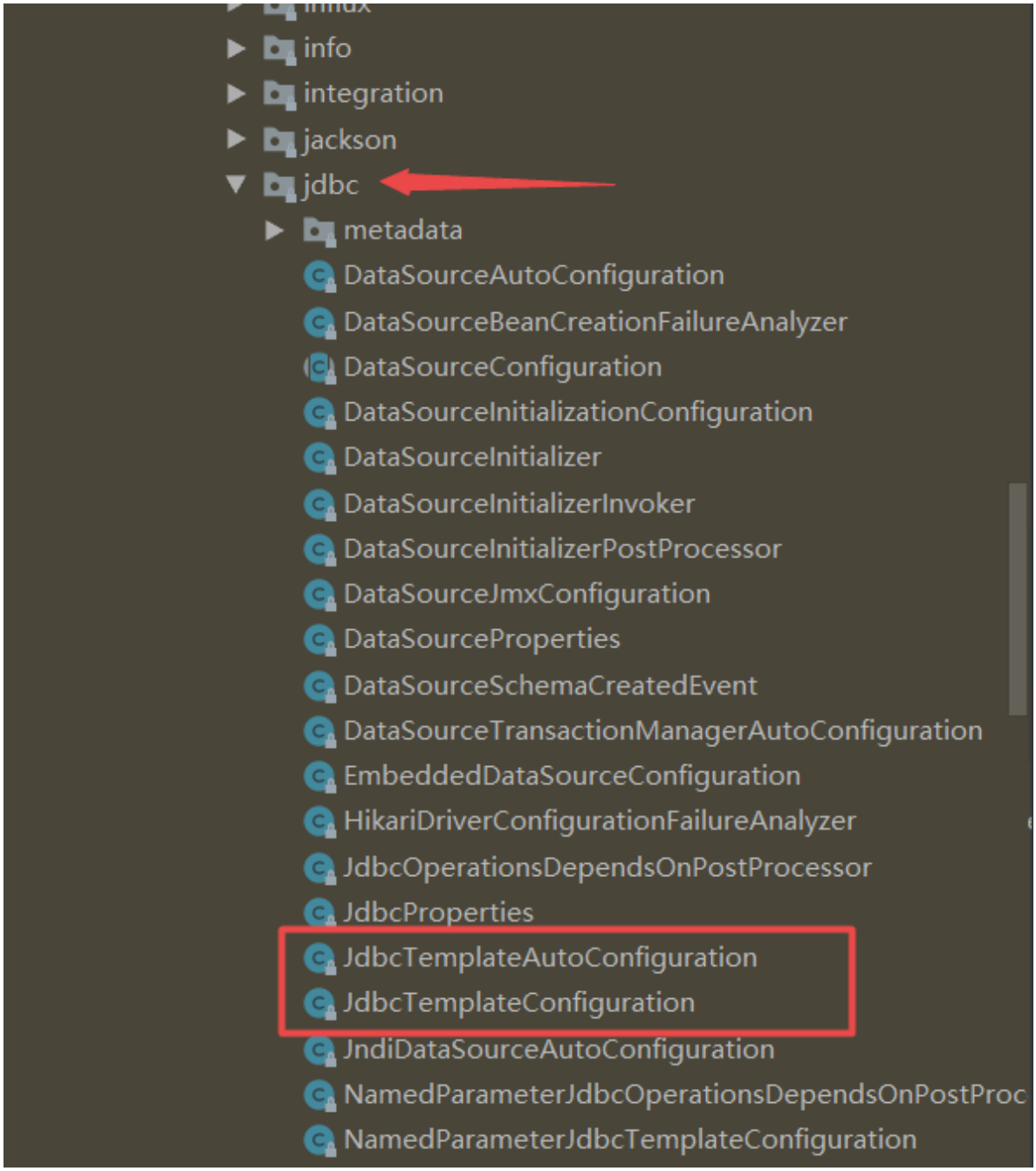
可以进去看一下源码（点击username之类的属性）：

```
 * @since 1.1.0
 */
@ConfigurationProperties(prefix = "spring.datasource")
public class DataSourceProperties implements BeanClassLoaderAware, InitializingBean {

    private ClassLoader classLoader;

    /**
     * Name of the datasource. Default to "testdb" when using an embedded database.
     */
    private String name;

    /**
```

自动配置类：

```
 */
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class })
@EnableConfigurationProperties(DataSourceProperties.class)
@Import({ DataSourcePoolMetadataProvidersConfiguration.class, DataSourceInitializ
public class DataSourceAutoConfiguration {

    @Configuration(proxyBeanMethods = false)
    @Conditional(EmbeddedDatabaseCondition.class)
    @ConditionalOnMissingBean({ DataSource.class, XADataSource.class })
    @Import(EmbeddedDataSourceConfiguration.class)
    protected static class EmbeddedDatabaseConfiguration {

    }
```

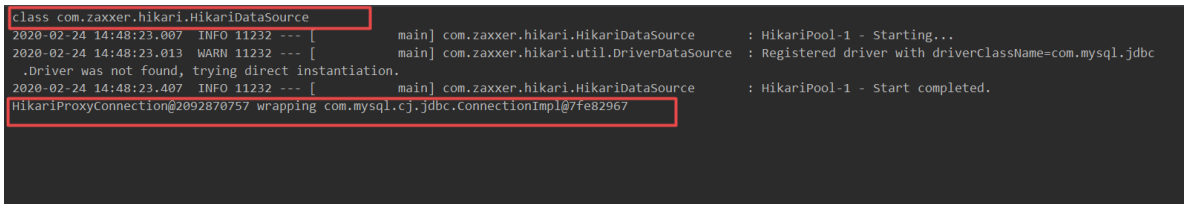
之后会遇到xxxTemplate：这是SpringBoot已经配置好的模板bean，可以拿来即用！



- 测试一下

```java
package com.kuang;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

@SpringBootTest
class Springboot05DataApplicationTests {

    @Autowired
    DataSource dataSource;

    @Test
    void contextLoads() throws SQLException {
```

```
        //查看默认的数据源：class com.zaxxer.hikari.HikariDataSource
        System.out.println(dataSource.getClass());

        //获得数据库的连接
        Connection connection = dataSource.getConnection();
        System.out.println(connection);

        //关闭
        connection.close();
    }

}
```



数据源的相关操作都用DataSource这个bean。

3.2 JdbcTemplate

原生的JDBC，在SpringBoot使用JdbcTemplate进行操作数据库。

```
package com.kuang.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.List;
import java.util.Map;

@Controller
public class JDBCController {
    @Autowired
    JdbcTemplate jdbcTemplate;

    @GetMapping("/userList")
    @ResponseBody
    public List<Map<String, Object>> userList() {
        String sql = "select * from user";
        List<Map<String, Object>> list = jdbcTemplate.queryForList(sql);

        return list;
    }

    @GetMapping("/addUser")
    public String addUser() {
        String sql = "insert into user(id,name,pwd)
values(7,'GGek','123')";
        jdbcTemplate.update(sql);
        return "redirect:/userList";
```

```
    }

    @GetMapping("/updateUser/{id}")
    public String updateUser(@PathVariable("id") int id) {
        String sql = "update user set name = ?,pwd = ? where id=" + id;
        //封装上面的参数
        Object[] objects = new Object[2];
        objects[0] = "小明2";
        objects[1] = "000";
        jdbcTemplate.update(sql, objects);
        return "redirect:/userList";
    }

    @GetMapping("/deleteUser/{id}")
    public String deleteUser(@PathVariable("id") int id) {
        String sql = "delete from mybatis.user where id = ?";

        jdbcTemplate.update(sql,id);
        return "redirect:/userList";
    }

}
```

使用原生的JDBC直接使用JdbcTemplate进行操作即可。

4 Druid数据源

4.1 简介

Druid 是阿里巴巴开源平台上一个数据库连接池实现，结合了 C3P0、DBCP、PROXOOL 等 DB 池的优点，同时加入了日志监控。

Druid 可以很好的监控 DB 池连接和 SQL 的执行情况，天生就是针对监控而生的 DB 连接池。

Spring Boot 2.0 以上默认使用 Hikari 数据源，可以说 Hikari 与 Driud 都是当前 Java Web 上最优秀的数据源，重点了解 Spring Boot 如何集成 Druid 数据源，如何实现数据库监控。

- 特点
    - 处理的数据量规模较大。
    - 可以进行数据的实时查询展示。
    - 它的查询模式是交互式的，这也说明其查询并发能力有限。
- 常见问题见官网：https://github.com/alibaba/druid/wiki/%E5%B8%B8%E8%A7%81%E9%97%AE%E9%A2%98

4.2 配置参数

com.alibaba.druid.pool.DruidDataSource

配置	缺省值	说明
name		配置这个属性的意义在于，如果存在多个数据源，监控的时候可以通过名字来区分开来。 如果没有配置，将会生成一个名字，格式是："DataSource-" + System.identityHashCode(this)
jdbcUrl		连接数据库的url，不同数据库不一样。例如： mysql : jdbc:mysql://10.20.153.104:3306/druid2 oracle : jdbc:oracle:thin:@10.20.149.85:1521:ocnauto
username		连接数据库的用户名
password		连接数据库的密码。如果你不希望密码直接写在配置文件中，可以使用ConfigFilter。详细看这里：https://github.com/alibaba/druid/wiki/%E4%BD%BF%E7%94%A8ConfigFilter
driverClassName	根据url自动识别	这一项可配可不配，如果不配置druid会根据url自动识别dbType，然后选择相应的driverClassName(建议配置下)
initialSize	0	初始化时建立物理连接的个数。初始化发生在显示调用init方法，或者第一次getConnection时
maxActive	8	最大连接池数量
maxIdle	8	已经不再使用，配置了也没效果
minIdle		最小连接池数量
maxWait		获取连接时最大等待时间，单位毫秒。配置了maxWait之后，缺省启用公平锁，并发效率会有所下降，如果需要可以通过配置useUnfairLock属性为true使用非公平锁。
poolPreparedStatements	false	是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。在mysql下建议关闭。
maxOpenPreparedStatements	-1	要启用PSCache，必须配置大于0，当大于0时，poolPreparedStatements自动触发修改为true。在Druid中，不会存在Oracle下PSCache占用内存过多的问题，可以把这个数值配置大一些，比如说100
validationQuery		用来检测连接是否有效的sql，要求是一个查询语句。如果validationQuery为null，testOnBorrow、testOnReturn、testWhileIdle都不会其作用。
testOnBorrow	true	申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能。
testOnReturn	false	归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能
testWhileIdle	false	建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测，如果空闲时间大于timeBetweenEvictionRunsMillis，执行validationQuery检测连接是否有效。

配置	缺省值	说明
timeBetweenEvictionRunsMillis		有两个含义： 1) Destroy线程会检测连接的间隔时间 2) testWhileIdle的判断依据，详细看testWhileIdle属性的说明
numTestsPerEvictionRun		不再使用，一个DruidDataSource只支持一个EvictionRun
minEvictableIdleTimeMillis		
connectionInitSqls		物理连接初始化的时候执行的sql
exceptionSorter	根据dbType自动识别	当数据库抛出一些不可恢复的异常时，抛弃连接
filters		属性类型是字符串，通过别名的方式配置扩展插件，常用的插件有： 监控统计用的filter:stat日志用的filter:log4j防御sql注入的filter:wall
proxyFilters		类型是List<com.alibaba.druid.filter.Filter>，如果同时配置了filters和proxyFilters，是组合关系，并非替换关系

4.3 基本使用

4.3.1 引入数据源（添加依赖）

```
<!-- https://mvnrepository.com/artifact/com.alibaba/druid -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.21</version>
</dependency>
```

4.3.2 配置文件设置Type

用type切换数据源产品。

```
spring:
  datasource:
    username: root
    password: mynewroot
    # 假如时区报错了，就添加时区的配置：serverTimezone=UTC
    url: jdbc:mysql://localhost:3306/mybatis?
serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
```

测试一下：

```
package com.kuang;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

@SpringBootTest
class SpringbootO5DataApplicationTests {

    @Autowired
    DataSource dataSource;

    @Test
    void contextLoads() throws SQLException {
        //查看默认的数据源：class com.zaxxer.hikari.HikariDataSource
        System.out.println(dataSource.getClass());

        //获得数据库的连接
        Connection connection = dataSource.getConnection();
        System.out.println(connection);

        //关闭
        connection.close();
    }

}
```

```
class com.alibaba.druid.pool.DruidDataSource
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new dr
 the SPI and manual loading of the driver class is generally unneces
2020-02-24 15:41:26.240  INFO 15552 --- [           main] com.alibaba
com.mysql.cj.jdbc.ConnectionImpl@44e93c1f
```

一些常用的配置：

```yaml
spring:
  datasource:
    username: root
    password: mynewroot
    #?serverTimezone=UTC解决时区的报错
    url: jdbc:mysql://localhost:3306/mybatis?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.jdbc.Driver
    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
```

因为使用了log4j，所以记得导入依赖：

```
<!--Log4j-->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

4.3.3 自定义配置类

```
package com.kuang.config;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.druid.support.http.StatViewServlet;
import com.alibaba.druid.support.http.WebStatFilter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;
import java.util.HashMap;
import java.util.Map;

@Configuration
public class DruidConfig {
    //绑定配置文件
    @ConfigurationProperties(prefix = "spring.datasource")
    @Bean
    public DataSource druidDataSource() {
        return new DruidDataSource();
    }

    //后台监控===》相当于web.xml
    //因为SpringBoot内置了servlet容器，所以没有web.xml
    //要想使用web.xml，我们就用ServletRegistrationBean进行替代。
    @Bean
    public ServletRegistrationBean statViewServlet() {
```

```
        ServletRegistrationBean<StatViewServlet> bean = new
ServletRegistrationBean<>(new StatViewServlet(), "/druid/*");

        //后台需要登录管理====》账户、密码配置
        HashMap<String, String> initParameters = new HashMap<>();

        //增加配置
        //key是固定的loginUsername和loginPassword
        initParameters.put("loginUsername", "admin");
        initParameters.put("loginPassword", "123");

        //允许谁可以进行访问,值为空代表所有人都可以访问
        //localhost：本机可以访问
        initParameters.put("allow", "");

        //禁止谁不能访问 可以直接禁止ip
        //initParameters.put("Geekst", "192.168.11.123");


        bean.setInitParameters(initParameters);//设置初始化参数

        return bean;
    }

    ///如果需要使用filter，就用FilterRegistrationBean
    @Bean
    public FilterRegistrationBean webStatFilter() {
        FilterRegistrationBean bean = new FilterRegistrationBean();

        bean.setFilter(new WebStatFilter());

        //可以过滤哪些请求
        Map<String, String> initParameters = new HashMap<>();

        //exclusions ===>这些东西不进行统计
        initParameters.put("exclusions", "*.js,*.css,/druid/*");

        bean.setInitParameters(initParameters);
        return bean;
    }
}
```

localhost:8080/druid 进入后台监控页面。

注意点：

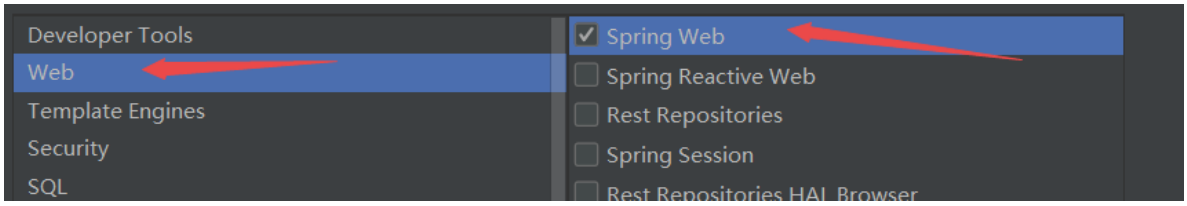
- @Configuration使其成为组件，配置类
- @Bean 注册到Spring的容器中进行管理

5 Mybatis

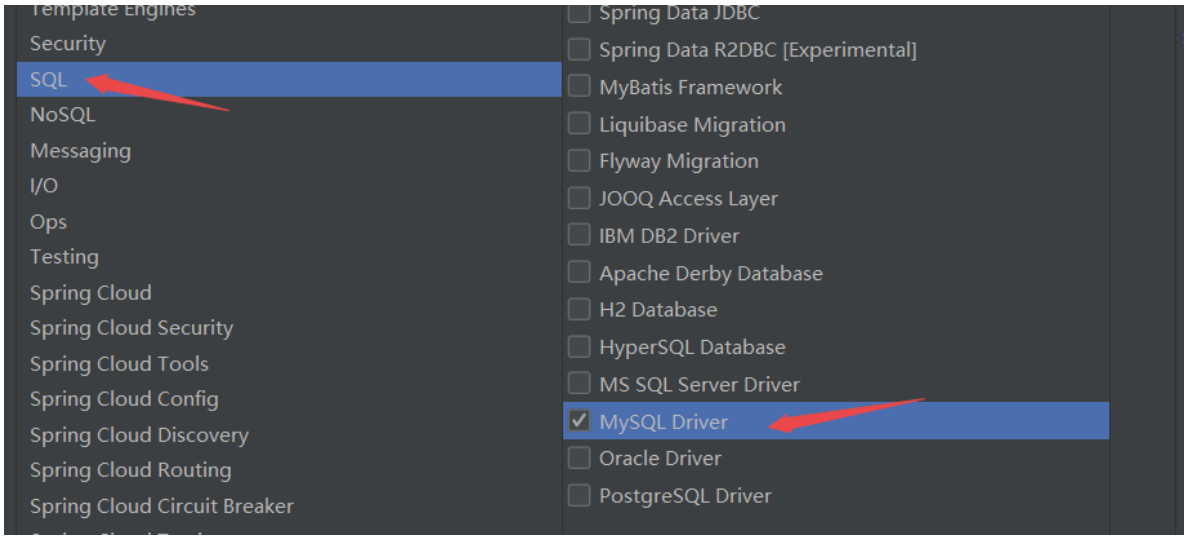
5.1 搭建环境

5.1.1 添加支持

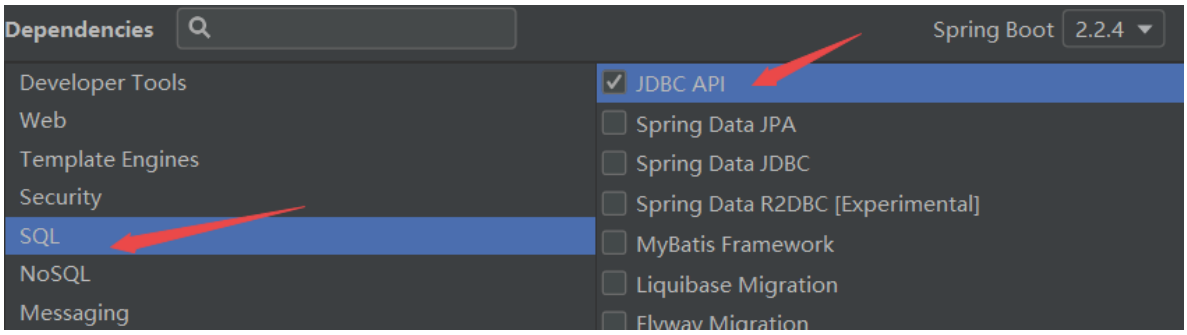
web：



mysql驱动：



jdbc：



5.1.2 添加依赖

```
<!--整合Mybatis-->
<!-- https://mvnrepository.com/artifact/org.mybatis.spring.boot/mybatis-spring-boot-starter -->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.1.1</version>
</dependency>
```

官方文档：https://github.com/mybatis/spring-boot-starter/blob/master/mybatis-spring-boot-autoconfigure/src/site/markdown/index.md

5.1.3 配置文件

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

测试一下：

```java
package com.kuang;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.sql.SQLException;

@SpringBootTest
class SpringbootO6MybatisApplicationTests {

    @Autowired
    DataSource dataSource;
    @Test
    void contextLoads() throws SQLException {
        System.out.println(dataSource.getClass());
        System.out.println(dataSource.getConnection());

    }

}
```

```
class com.zaxxer.hikari.HikariDataSource
2020-02-24 16:42:18.941  INFO 17140 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2020-02-24 16:42:19.439  INFO 17140 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
HikariProxyConnection@18338606 wrapping com.mysql.cj.jdbc.ConnectionImpl@51d387d3
```

5.2 基本使用

- pojo：

```java
package com.kuang.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

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

- mapper接口：

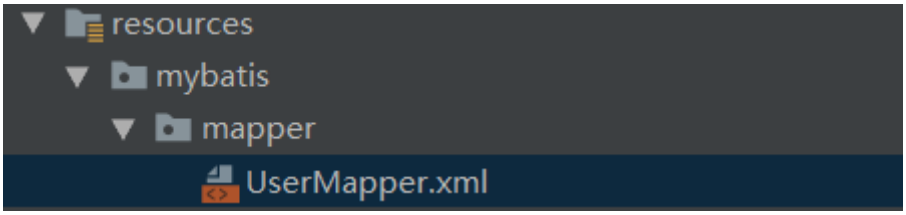
```java
package com.kuang.mapper;

import com.kuang.pojo.User;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;
```

```java

import java.util.List;

/**
 * @Mapper 这个注解表示这是一个Mybatis的Mapper接口
 * <p>
 * 或者在主启动类上加扫描包的注解 @MapperScan("com.kuang.mapper")
 */
@Mapper
@Repository //Dao层的注解，代表被Spring整合
public interface UserMapper {
    List<User> selectUser();

    User selectUserById(int id);

    int addUser(User user);

    int updateUser(User user);

    int deleteUser(int id);
}
```

- mapper.xml:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.kuang.mapper.UserMapper">
    <select id="selectUser" resultType="User">
    select * from user
  </select>

    <select id="selectUserById" resultType="User">
    select * from user where id = #{id}
</select>

    <insert id="addUser" parameterType="User">
    insert into user (id,name,pwd) values (#{id},#{name},#{pwd})
</insert>

    <update id="updateUser" parameterType="User">
    update user set name=#{name},pwd=#{pwd} where id = #{id}
</update>

    <delete id="deleteUser" parameterType="int">
    delete from user where id = #{id}
</delete>
</mapper>
```

xml放置在resources中。



- 配置文件整合

```
# 整合Mybatis
# 起别名，对应实体类的路径
mybatis.type-aliases-package=com.kuang.pojo
# mapper.xml的路径，指定myBatis的核心配置文件与Mapper映射文件
mybatis.mapper-locations=classpath:mybatis/mapper/*.xml
```

- Controller层

```
package com.kuang.controller;

import com.kuang.mapper.UserMapper;
import com.kuang.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.util.List;

@Controller
public class UserController {
    @Autowired
    private UserMapper userMapper;

    //查询全部用户
    @GetMapping("/selectUser")
    @ResponseBody
    public List<User> selectUser() {
        List<User> users = userMapper.selectUser();

        return users;
    }

    //根据id查询用户
    @ResponseBody
    @GetMapping("/selectUserById")
    public User selectUserById() {
        User user = userMapper.selectUserById(1);
        return user;
    }

    //添加一个用户
    @GetMapping("/addUser")
    public String addUser() {
        userMapper.addUser(new User(3, "aaa", "856b544"));
        return "redirect:/selectUser";
    }

```

```
41      //修改一个用户
42      @GetMapping("/updateUser")
43      public String updateUser() {
44          userMapper.updateUser(new User(3, "da", "421319"));
45          return "redirect:/selectUser";
46      }
47
48      //根据id删除用户
49      @GetMapping("/deleteUser")
50      public String deleteUser() {
51          userMapper.deleteUser(5);
52          return "redirect:/selectUser";
53      }
54  }
```

5.3 小结

- 整体开发流程和之前的差不多，不同的是mybatis的配置写在springboot中进行整合。
- 分层注解记得写。