

06、数据访问

1、SQL

1、数据源的自动配置-HikariDataSource

1、导入JDBC场景

XML | Copy

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-jdbc</artifactId>
4 </dependency>
5
```

org.springframework.boot:spring-boot-starter-jdbc:2.4.0

org.springframework.boot:spring-boot-starter:2.4.0 (omitted for duplicate)

com.zaxxer:HikariCP:3.4.5 数据源

org.springframework:spring-jdbc:5.3.1 jdbc

org.springframework:spring-tx:5.3.1 事务

数据库驱动？
为什么导入JDBC场景，官方不导入驱动？官方不知道我们接下要操作什么数据库。
数据库版本和驱动版本对应

XML | Copy

```
1 默认版本: <mysql.version>8.0.22</mysql.version>
2
3 <dependency>
4   <groupId>mysql</groupId>
5   <artifactId>mysql-connector-java</artifactId>
6   <!-- <version>5.1.49</version> -->
7 </dependency>
8 想要修改版本
9 1、直接依赖引入具体版本 (maven的就近依赖原则)
10 2、重新声明版本 (maven的属性的就近优先原则)
11 <properties>
12   <java.version>1.8</java.version>
13   <mysql.version>5.1.49</mysql.version>
14 </properties>
```

2、分析自动配置

1、自动配置的类

- DataSourceAutoConfiguration：数据源的自动配置
 - 修改数据源相关的配置：spring.datasource

- 数据库连接池的配置，是自己容器中没有DataSource才自动配置的
- 底层配置好的连接池是：**HikariDataSource**

```
1 @Configuration(proxyBeanMethods = false)
2 @Conditional(PooledDataSourceCondition.class)
3 @ConditionalOnMissingBean({ DataSource.class, XADataSource.class })
4 @Import({ DataSourceConfiguration.Hikari.class, DataSourceConfiguration.Tomcat.class,
5         DataSourceConfiguration.Dbcp2.class, DataSourceConfiguration.OracleUcp.class,
6         DataSourceConfiguration.Generic.class, DataSourceJmxConfiguration.class })
7 protected static class PooledDataSourceConfiguration
```

- DataSourceTransactionManagerAutoConfiguration：事务管理器的自动配置
- JdbcTemplateAutoConfiguration：**JdbcTemplate的自动配置，可以来对数据库进行crud**
 - 可以修改这个配置项@ConfigurationProperties(prefix = "spring.jdbc") 来修改JdbcTemplate
 - @Bean@Primary JdbcTemplate；容器中有这个组件
- JndiDataSourceAutoConfiguration：jndi的自动配置
- XADataSourceAutoConfiguration：分布式事务相关的

3、修改配置项

```
1 spring:
2   datasource:
3     url: jdbc:mysql://localhost:3306/db_account
4     username: root
5     password: 123456
6     driver-class-name: com.mysql.jdbc.Driver
```

4、测试

```
1 @Slf4j
2 @SpringBootTest
3 class Boot05WebAdminApplicationTests {
4
5     @Autowired
6     JdbcTemplate jdbcTemplate;
7
8
9     @Test
10    void contextLoads() {
11
12        //      jdbcTemplate.queryForObject("select * from account_tbl")
13        //      jdbcTemplate.queryForList("select * from account_tbl",)
14        Long aLong = jdbcTemplate.queryForObject("select count(*) from account_tbl", Long.class);
15        log.info("记录总数: {}", aLong);
16    }
17
18 }
```

2、使用Druid数据源

1、druid官方github地址

<https://github.com/alibaba/druid> <<https://github.com/alibaba/druid>>

整合第三方技术的两种方式

- 自定义
- 找starter

2、自定义方式

1、创建数据源

XML | Copy

```
1      <dependency>
2          <groupId>com.alibaba</groupId>
3          <artifactId>druid</artifactId>
4          <version>1.1.17</version>
5      </dependency>
6
7      <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
8          destroy-method="close">
9          <property name="url" value="${jdbc.url}" />
10         <property name="username" value="${jdbc.username}" />
11         <property name="password" value="${jdbc.password}" />
12         <property name="maxActive" value="20" />
13         <property name="initialSize" value="1" />
14         <property name="maxWait" value="60000" />
15         <property name="minIdle" value="1" />
16         <property name="timeBetweenEvictionRunsMillis" value="60000" />
17         <property name="minEvictableIdleTimeMillis" value="300000" />
18         <property name="testWhileIdle" value="true" />
19         <property name="testOnBorrow" value="false" />
20         <property name="testOnReturn" value="false" />
21         <property name="poolPreparedStatements" value="true" />
22         <property name="maxOpenPreparedStatements" value="20" />
23     </bean>
```

2、StatViewServlet

StatViewServlet的用途包括：

- 提供监控信息展示的html页面
- 提供监控信息的JSON API

XML | Copy

```
1      <servlet>
2          <servlet-name>DruidStatView</servlet-name>
3          <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>
4      </servlet>
5      <servlet-mapping>
6          <servlet-name>DruidStatView</servlet-name>
7          <url-pattern>/druid/*</url-pattern>
8      </servlet-mapping>
```

3、StatFilter

用于统计监控信息；如SQL监控、URI监控

XML | Copy

```
1      需要给数据源中配置如下属性；可以允许多个filter，多个用，分割；如：
2
3      <property name="filters" value="stat,slf4j" />
```

系统中所有filter:

别名	Filter类名
default	com.alibaba.druid.filter.stat.StatFilter
stat	com.alibaba.druid.filter.stat.StatFilter
mergeStat	com.alibaba.druid.filter.stat.MergeStatFilter
encoding	com.alibaba.druid.filter.encoding.EncodingConvertFilter
log4j	com.alibaba.druid.filter.logging.Log4jFilter
log4j2	com.alibaba.druid.filter.logging.Log4j2Filter
slf4j	com.alibaba.druid.filter.logging.Slf4jLogFilter
commonlogging	com.alibaba.druid.filter.logging.CommonsLogFilter

慢SQL记录配置

XML | Copy

```
1 <bean id="stat-filter" class="com.alibaba.druid.filter.stat.StatFilter">
2   <property name="slowSqlMillis" value="10000" />
3   <property name="logSlowSql" value="true" />
4 </bean>
5
6 使用 slowSqlMillis 定义慢SQL的时长
```

3、使用官方starter方式

1、引入druid-starter

XML | Copy

```
1 <dependency>
2   <groupId>com.alibaba</groupId>
3   <artifactId>druid-spring-boot-starter</artifactId>
4   <version>1.1.17</version>
5 </dependency>
```

2、分析自动配置

- 扩展配置项 **spring.datasource.druid**
- DruidSpringAopConfiguration.class, 监控SpringBean的; 配置项: **spring.datasource.druid.aop-patterns**
- DruidStatViewServletConfiguration.class, 监控页的配置: **spring.datasource.druid.stat-view-servlet**; 默认开启
- DruidWebStatFilterConfiguration.class, web监控配置; **spring.datasource.druid.web-stat-filter**; 默认开启
- DruidFilterConfiguration.class)) 所有Druid自己filter的配置

Java | Copy

```
1 private static final String FILTER_STAT_PREFIX = "spring.datasource.druid.filter.stat";
2 private static final String FILTER_CONFIG_PREFIX = "spring.datasource.druid.filter.config";
3 private static final String FILTER_ENCODING_PREFIX = "spring.datasource.druid.filter.encoding";
4 private static final String FILTER_SLF4J_PREFIX = "spring.datasource.druid.filter.slf4j";
5 private static final String FILTER_LOG4J_PREFIX = "spring.datasource.druid.filter.log4j";
6 private static final String FILTER_LOG4J2_PREFIX = "spring.datasource.druid.filter.log4j2";
7 private static final String FILTER_COMMONS_LOG_PREFIX = "spring.datasource.druid.filter.common-log";
8 private static final String FILTER_WALL_PREFIX = "spring.datasource.druid.filter.wall";
```

3、配置示例

```

1  spring:
2    datasource:
3      url: jdbc:mysql://localhost:3306/db_account
4      username: root
5      password: 123456
6      driver-class-name: com.mysql.jdbc.Driver
7
8    druid:
9      aop-patterns: com.atguigu.admin.* #监控SpringBean
10     filters: stat,wall # 底层开启功能, stat (sql监控), wall (防火墙)
11
12     stat-view-servlet: # 配置监控页功能
13       enabled: true
14       login-username: admin
15       login-password: admin
16       resetEnable: false
17
18     web-stat-filter: # 监控web
19       enabled: true
20       urlPattern: /*
21       exclusions: '*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*'
22
23
24     filter:
25       stat: # 对上面filters里面的stat的详细配置
26         slow-sql-millis: 1000
27         logSlowSql: true
28         enabled: true
29       wall:
30         enabled: true
31         config:
32           drop-table-allow: false
33

```

SpringBoot配置示例

<https://github.com/alibaba/druid/tree/master/druid-spring-boot-starter> <<https://github.com/alibaba/druid/tree/master/druid-spring-boot-starter>>

配置项列表<https://github.com/alibaba/druid/wiki/DruidDataSource%E9%85%8D%E7%BD%AE%E5%B1%9E%E6%80%A7%E5%88%97%E8%A1%A8>
<<https://github.com/alibaba/druid/wiki/DruidDataSource%E9%85%8D%E7%BD%AE%E5%B1%9E%E6%80%A7%E5%88%97%E8%A1%A8>>

3、整合MyBatis操作

<https://github.com/mybatis> <<https://github.com/mybatis>>

starter







SpringBoot官方的Starter: spring-boot-starter-*

第三方的: *-spring-boot-starter

```

1  <dependency>
2    <groupId>org.mybatis.spring.boot</groupId>
3    <artifactId>mybatis-spring-boot-starter</artifactId>
4    <version>2.1.4</version>
5  </dependency>

```

- ▼  org.mybatis.spring.boot:mybatis-spring-boot-starter:2.1.4
 -  org.springframework.boot:spring-boot-starter:2.4.0 (omitted for duplicate)
 - >  org.springframework.boot:spring-boot-starter-jdbc:2.4.0
 - >  org.mybatis.spring.boot:mybatis-spring-boot-autoconfigure:2.1.4
 -  org.mybatis:mybatis:3.5.6
 -  org.mybatis:mybatis-spring:2.0.6

1、配置模式

- 全局配置文件
- SqlSessionFactory: 自动配置好了
- SqlSession: 自动配置了 **SqlSessionTemplate** 组合了**SqlSession**
- **@Import(AutoConfiguredMapperScannerRegistrar.class)** ;
- Mapper: 只要我们写的操作MyBatis的接口标准了 **@Mapper** 就会被自动扫描进来

```
Java | Copy
1 @EnableConfigurationProperties(MybatisProperties.class) : MyBatis配置项绑定类。
2 @AutoConfigureAfter({ DataSourceAutoConfiguration.class, MybatisLanguageDriverAutoConfiguration.class })
3 public class MybatisAutoConfiguration{}
4
5 @ConfigurationProperties(prefix = "mybatis")
6 public class MybatisProperties
```

可以修改配置文件中 mybatis 开始的所有;

```
YAML | Copy
1 # 配置mybatis规则
2
3 mybatis:
4   config-location: classpath:mybatis/mybatis-config.xml #全局配置文件位置
5   mapper-locations: classpath:mybatis/mapper/*.xml #sql映射文件位置
6
7
8 Mapper接口--->绑定Xml
9
10 <?xml version="1.0" encoding="UTF-8" ?>
11 <!DOCTYPE mapper
12     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
13     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
14
15 <mapper namespace="com.atguigu.admin.mapper.AccountMapper">
16   <!-- public Account getAcct(Long id); -->
17   <select id="getAcct" resultType="com.atguigu.admin.bean.Account">
18     select * from account_tbl where id=#{id}
19   </select>
20 </mapper>
```

配置 **private** Configuration **configuration**; mybatis.configuration下面的所有, 就是相当于改mybatis全局配置文件中的值

```
YAML | Copy
1 # 配置mybatis规则
2
3 mybatis:
4   # config-location: classpath:mybatis/mybatis-config.xml
5   mapper-locations: classpath:mybatis/mapper/*.xml
6   configuration:
7     map-underscore-to-camel-case: true
8
9
10 可以不写全局; 配置文件, 所有全局配置文件的配置都放在configuration配置项中即可
```

- 导入mybatis官方starter
- 编写mapper接口。标准@Mapper注解
- 编写sql映射文件并绑定mapper接口
- 在application.yaml中指定Mapper配置文件的位置, 以及指定全局配置文件的信息 (建议; 配置在mybatis.configuration)

2、注解模式

```

1  @Mapper
2  public interface CityMapper {
3
4
5      @Select("select * from city where id=#{id}")
6      public City getById(Long id);
7
8
9      public void insert(City city);
10
11 }

```

3、混合模式

```

1  @Mapper
2  public interface CityMapper {
3
4
5      @Select("select * from city where id=#{id}")
6      public City getById(Long id);
7
8
9      public void insert(City city);
10
11 }

```

最佳实践：

- 引入mybatis-starter
- 配置application.yaml中，指定mapper-location位置即可
- 编写Mapper接口并标注@Mapper注解
- 简单方法直接注解方式
- 复杂方法编写mapper.xml进行绑定映射
- @MapperScan("com.atguigu.admin.mapper") 简化，其他的接口就可以不用标注@Mapper注解

4、整合 MyBatis-Plus 完成CRUD

1、什么是MyBatis-Plus

MyBatis-Plus <<https://github.com/baomidou/mybatis-plus>>（简称 MP）是一个 MyBatis <<http://www.mybatis.org/mybatis-3/>> 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。

mybatis plus 官网 <<https://baomidou.com/>>

建议安装 **MybatisX** 插件

2、整合MyBatis-Plus

```

1  <dependency>
2      <groupId>com.baomidou</groupId>
3      <artifactId>mybatis-plus-boot-starter</artifactId>
4      <version>3.4.1</version>
5  </dependency>

```

自动配置

- MybatisPlusAutoConfiguration 配置类，MybatisPlusProperties 配置项绑定。**mybatis-plus: xxx 就是对mybatis-plus的定制**

- `SessionFactory` 自动配置好。底层是容器中默认的数据源
- `mapperLocations` 自动配置好的。有默认值。 `classpath*/mapper/**/*.xml`；任意包的类路径下的所有mapper文件夹下任意路径下的所有xml都是sql映射文件。 建议以后sql映射文件，放在 `mapper`下
- 容器中也自动配置好了 `SqlSessionTemplate`
- `@Mapper` 标注的接口也会被自动扫描；建议直接 `@MapperScan("com.atguigu.admin.mapper")` 批量扫描就行

优点:

- 只需要我们的Mapper继承 **BaseMapper** 就可以拥有crud能力

3、CRUD功能

```
1      @GetMapping("/user/delete/{id}")
2      public String deleteUser(@PathVariable("id") Long id,
3                               @RequestParam(value = "pn",defaultValue = "1")Integer pn,
4                               RedirectAttributes ra){
5
6          userService.removeById(id);
7
8          ra.addAttribute("pn",pn);
9          return "redirect:/dynamic_table";
10     }
11
12
13     @GetMapping("/dynamic_table")
14     public String dynamic_table(@RequestParam(value="pn",defaultValue = "1") Integer pn,Model model){
15         //表格内容的遍历
16         response.sendError
17         // List<User> users = Arrays.asList(new User("zhangsan", "123456"),
18         //     new User("lisi", "123444"),
19         //     new User("haha", "aaaaa"),
20         //     new User("hehe ", "aadd"));
21         //     model.addAttribute("users",users);
22         //
23         //     if(users.size(>3){
24         //         throw new UserTooManyException();
25         //     }
26         //从数据库中查出user表中的用户进行展示
27
28         //构造分页参数
29         Page<User> page = new Page<>(pn, 2);
30         //调用page进行分页
31         Page<User> userPage = userService.page(page, null);
32
33
34         //     userPage.getRecords()
35         //     userPage.getCurrent()
36         //     userPage.getPAGES()
37
38
39         model.addAttribute("users",userPage);
40
41         return "table/dynamic_table";
42     }
```



```
Java | Copy
1 @Service
2 public class UserServiceImpl extends ServiceImpl<UserMapper,User> implements UserService {
3
4
5
6 }
7
8
9 public interface UserService extends IService<User> {
10
11 }
```

2、NoSQL

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构，如 [字符串（strings）](http://www.redis.cn/topics/data-types-intro.html#strings) <http://www.redis.cn/topics/data-types-intro.html#strings>，[散列（hashes）](http://www.redis.cn/topics/data-types-intro.html#hashes) <http://www.redis.cn/topics/data-types-intro.html#hashes>，[列表（lists）](http://www.redis.cn/topics/data-types-intro.html#lists) <http://www.redis.cn/topics/data-types-intro.html#lists>，[集合（sets）](http://www.redis.cn/topics/data-types-intro.html#sets) <http://www.redis.cn/topics/data-types-intro.html#sets>，[有序集合（sorted sets）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) <http://www.redis.cn/topics/data-types-intro.html#sorted-sets> 与范围查询，[bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps) <http://www.redis.cn/topics/data-types-intro.html#bitmaps>，[hyperloglogs](http://www.redis.cn/topics/data-types-intro.html#hyperloglogs) <http://www.redis.cn/topics/data-types-intro.html#hyperloglogs> 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) <http://www.redis.cn/commands/geoadd.html> 索引半径查询。Redis 内置了 [复制（replication）](http://www.redis.cn/topics/replication.html) <http://www.redis.cn/topics/replication.html>，[Lua脚本（Lua scripting）](http://www.redis.cn/commands/eval.html) <http://www.redis.cn/commands/eval.html>，[LRU驱动事件（LRU eviction）](http://www.redis.cn/topics/lru-cache.html) <http://www.redis.cn/topics/lru-cache.html>，[事务（transactions）](http://www.redis.cn/topics/transactions.html) <http://www.redis.cn/topics/transactions.html> 和不同级别的 [磁盘持久化（persistence）](http://www.redis.cn/topics/persistence.html) <http://www.redis.cn/topics/persistence.html>，并通过 [Redis哨兵（Sentinel）](http://www.redis.cn/topics/sentinel.html) <http://www.redis.cn/topics/sentinel.html> 和自动 [分区（Cluster）](http://www.redis.cn/topics/cluster-tutorial.html) <http://www.redis.cn/topics/cluster-tutorial.html> 提供高可用性（high availability）。

1、Redis自动配置

```
XML | Copy
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-redis</artifactId>
4 </dependency>
```

org.springframework.boot:spring-boot-starter-data-redis:2.4.0

org.springframework.boot:spring-boot-starter:2.4.0 (omitted for

org.springframework.data:spring-data-redis:2.4.1

org.springframework.data:spring-data-keyvalue:2.4.1

org.springframework:spring-tx:5.3.1

org.springframework:spring-oxm:5.3.1

org.springframework:spring-aop:5.3.1 (omitted for duplicat

org.springframework:spring-context-support:5.3.1

org.slf4j:slf4j-api:1.7.30 (omitted for duplicate)

io.lettuce:lettuce-core:6.0.1.RELEASE

io.netty:netty-common:4.1.54.Final

io.netty:netty-handler:4.1.54.Final

io.netty:netty-transport:4.1.54.Final

io.projectreactor:reactor-core:3.4.0

atguigu.com 尚硅谷

自动配置：

- RedisAutoConfiguration 自动配置类。RedisProperties 属性类 --> **spring.redis.xxx是对redis的配置**
- 连接工厂是准备好的。**LettuceConnectionConfiguration**、**JedisConnectionConfiguration**
- **自动注入了RedisTemplate<Object, Object>：xxxTemplate；**
- **自动注入了StringRedisTemplate；k: v都是String**
- **key: value**

- 底层只要我们使用 `StringRedisTemplate`、`RedisTemplate`就可以操作redis

redis环境搭建

- 1、阿里云按量付费redis。经典网络
- 2、申请redis的公网连接地址
- 3、修改白名单 允许0.0.0.0/0 访问

2、RedisTemplate与Lettuce

Java | Copy

```
1  @Test
2  void testRedis(){
3
4      ValueOperations<String, String> operations = redisTemplate.opsForValue();
5
6
7      operations.set("hello","world");
8
9
10     String hello = operations.get("hello");
11     System.out.println(hello);
12 }
```

3、切换至jedis

XML | Copy

```
1  <dependency>
2      <groupId>org.springframework.boot</groupId>
3      <artifactId>spring-boot-starter-data-redis</artifactId>
4  </dependency>
5
6  <!-- 导入jedis-->
7  <dependency>
8      <groupId>redis.clients</groupId>
9      <artifactId>jedis</artifactId>
10 </dependency>
```

YAML | Copy

```
1  spring:
2    redis:
3      host: r-bp1nc7reqesxisgxpipd.redis.rds.aliyuncs.com
4      port: 6379
5      password: lfy:Lfy123456
6      client-type: jedis
7    jedis:
8      pool:
9        max-active: 10
```

%E2%80%8BHikariDataSource1%E3%80%81%E5%AF%BC%E5%85%A5JDBC%E5%9C%BA%E6%99%AF%20'