# Spring Boot

## Spring Boot集成 Mybatis

yml配置

|  |
| --- |
| **spring**:  **datasource**:  **url**: jdbc:mysql://10.58.241.10:3306/testspringboot?useunicode=true&characterEncoding=utf8  **username**: root  **password**: Wopt54321  **driver-class-name**: com.mysql.jdbc.Driver |

## Spring Boot 集成 Druid

### Pom配置

|  |
| --- |
| *<!--阿里druid数据库连接池 -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.1.10</**version**> </**dependency**> |

### Yml配置

|  |
| --- |
| **spring**:   **datasource**:  **url**: jdbc:mysql://10.58.241.10:3306/testspringboot?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8&useSSL=false  **username**: root  **password**: Wopt54321  **driver-class-name**: com.mysql.jdbc.Driver  **type**: com.alibaba.druid.pool.DruidDataSource  **druid**:  *#初始化大小* **initialSize**: 5  *#最小值* **minIdle**: 5  *#最大值* **maxActive**: 20  *#最大等待时间，配置获取连接等待超时，时间单位都是毫秒ms* **maxWait**: 60000  *#配置间隔多久才进行一次检测，检测需要关闭的空闲连接* **timeBetweenEvictionRunsMillis**: 60000  *#配置一个连接在池中最小生存的时间* **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，SpringBoot中没有log4j，我改成了log4j2* **filters**: stat,wall,log4j2  *#最大PSCache连接* **maxPoolPreparedStatementPerConnectionSize**: 20  **useGlobalDataSourceStat**: true  *# 通过connectProperties属性来打开mergeSql功能；慢SQL记录* **connectionProperties**: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500  *# 配置StatFilter* **web-stat-filter**:  *#默认为false，设置为true启动* **enabled**: true  **url-pattern**: **"/\*"  exclusions**: **"\*.js,\*.gif,\*.jpg,\*.bmp,\*.png,\*.css,\*.ico,/druid/\*"** *#配置StatViewServlet* **stat-view-servlet**:  **url-pattern**: **"/druid/\*"** *#允许那些ip* **allow**: 127.0.0.1  **login-username**: admin  **login-password**: 123456  *#禁止那些ip* **deny**: 192.168.1.102  *#是否可以重置* **reset-enable**: true  *#启用* **enabled**: true |

### DruidConfig类

|  |
| --- |
| **package** com.liay.common.config;  **import** javax.sql.DataSource;  **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** com.alibaba.druid.pool.DruidDataSource; **import** com.alibaba.druid.support.http.StatViewServlet; **import** com.alibaba.druid.support.http.WebStatFilter;  */\*\*  \* 线程池监控信息查看配置  \*/* @Configuration **public class** DruidConfig {   @Bean  **public** ServletRegistrationBean druidServlet() { *// 主要实现WEB监控的配置处理* ServletRegistrationBean servletRegistrationBean = **new** ServletRegistrationBean(  **new** StatViewServlet(), **"/druid/\*"**); *// 现在要进行druid监控的配置处理操作* servletRegistrationBean.addInitParameter(**"allow"**, **"127.0.0.1,10.1.1.1"**); *// 白名单  // servletRegistrationBean.addInitParameter("deny", "192.168.1.200"); // 黑名单* servletRegistrationBean.addInitParameter(**"loginUsername"**, **"admin"**); *// 用户名* servletRegistrationBean.addInitParameter(**"loginPassword"**, **"admin"**); *// 密码* servletRegistrationBean.addInitParameter(**"resetEnable"**, **"false"**); *// 是否可以重置数据源* **return** servletRegistrationBean ;  }   @Bean  **public** FilterRegistrationBean filterRegistrationBean() {  FilterRegistrationBean filterRegistrationBean = **new** FilterRegistrationBean() ;  filterRegistrationBean.setFilter(**new** WebStatFilter());  filterRegistrationBean.addUrlPatterns(**"/\*"**); *// 所有请求进行监控处理* filterRegistrationBean.addInitParameter(**"exclusions"**, **"\*.js,\*.gif,\*.jpg,\*.css,/druid/\*"**);  **return** filterRegistrationBean ;  }   @Bean  @ConfigurationProperties(prefix = **"spring.datasource"**)  **public** DataSource druidDataSource() {  **return new** DruidDataSource();  } } |

## Spring Boot 集成 Redis

### Pom配置

|  |
| --- |
| *!-- redis相关包 -->*  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-redis</**artifactId**> </**dependency**> |

### Yml配置

|  |
| --- |
| **spring**:  **redis**:  *# Redis服务器地址* **host**: 127.0.0.1  *# Redis服务器连接端口* **port**: 6379  *# 连接超时时间（毫秒）* **timeout**: 1000 |

### RedisConfig配置类

|  |
| --- |
| **package** com.liay.common.config;  **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.data.redis.connection.RedisConnectionFactory; **import** org.springframework.data.redis.core.RedisTemplate; **import** org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer; **import** org.springframework.data.redis.serializer.StringRedisSerializer; **import** com.fasterxml.jackson.annotation.JsonAutoDetect; **import** com.fasterxml.jackson.annotation.PropertyAccessor; **import** com.fasterxml.jackson.databind.ObjectMapper;  */\*\*  \* redis配置类  \*/* @Configuration **public class** RedisConfig {   @Bean  @SuppressWarnings(**"all"**)  **public** RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {   RedisTemplate<String, Object> template = **new** RedisTemplate<String, Object>();  template.setConnectionFactory(factory);  Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);  ObjectMapper om = **new** ObjectMapper();  om.setVisibility(PropertyAccessor.***ALL***, JsonAutoDetect.Visibility.***ANY***);  om.enableDefaultTyping(ObjectMapper.DefaultTyping.***NON\_FINAL***);  jackson2JsonRedisSerializer.setObjectMapper(om);  StringRedisSerializer stringRedisSerializer = **new** StringRedisSerializer();  *//key采用String的序列化方式* template.setKeySerializer(stringRedisSerializer);  *//hash的key也采用String的序列化方式* template.setHashKeySerializer(stringRedisSerializer);  *//value序列化方式采用jackson* template.setValueSerializer(jackson2JsonRedisSerializer);  *//hash的value序列化方式采用jackson* template.setHashValueSerializer(jackson2JsonRedisSerializer);  template.afterPropertiesSet();  **return** template;  }  } |

### Redis工具类

|  |
| --- |
| **package** com.liay.common.utils;  **import** java.util.List; **import** java.util.Map; **import** java.util.Set; **import** java.util.concurrent.TimeUnit; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.data.redis.core.RedisTemplate; **import** org.springframework.stereotype.Component; **import** org.springframework.util.CollectionUtils;  **import** javax.annotation.Resource;  */\*\*  \* Redis工具类  \*/* @Component **public class** RedisUtil {   @Autowired  **private** RedisTemplate<String, Object> **redisTemplate**;  *// =============================common============================  /\*\*  \* 指定缓存失效时间  \** ***@param key*** *键  \** ***@param time*** *时间(秒)  \** ***@return*** *\*/* **public boolean** expire(String key, **long** time) {  **try** {  **if** (time > 0) {  **redisTemplate**.expire(key, time, TimeUnit.***SECONDS***);  }  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 根据key 获取过期时间  \** ***@param key*** *键 不能为null  \** ***@return*** *时间(秒) 返回0代表为永久有效  \*/* **public long** getExpire(String key) {  **return redisTemplate**.getExpire(key, TimeUnit.***SECONDS***);  }  */\*\*  \* 判断key是否存在  \** ***@param key*** *键  \** ***@return*** *true 存在 false不存在  \*/* **public boolean** hasKey(String key) {  **try** {  **return redisTemplate**.hasKey(key);  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 删除缓存  \** ***@param key*** *可以传一个值 或多个  \*/* @SuppressWarnings(**"unchecked"**)  **public void** del(String... key) {  **if** (key != **null** && key.**length** > 0) {  **if** (key.**length** == 1) {  **redisTemplate**.delete(key[0]);  } **else** {  **redisTemplate**.delete(CollectionUtils.*arrayToList*(key));  }  }  }  *// ============================String=============================  /\*\*  \* 普通缓存获取  \** ***@param key*** *键  \** ***@return*** *值  \*/* **public** Object get(String key) {  **return** key == **null** ? **null** : **redisTemplate**.opsForValue().get(key);  }  */\*\*  \* 普通缓存放入  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@return*** *true成功 false失败  \*/* **public boolean** set(String key, Object value) {  **try** {  **redisTemplate**.opsForValue().set(key, value);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 普通缓存放入并设置时间  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@param time*** *时间(秒) time要大于0 如果time小于等于0 将设置无限期  \** ***@return*** *true成功 false 失败  \*/* **public boolean** set(String key, Object value, **long** time) {  **try** {  **if** (time > 0) {  **redisTemplate**.opsForValue().set(key, value, time, TimeUnit.***SECONDS***);  } **else** {  set(key, value);  }  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 递增  \** ***@param key*** *键  \** ***@param delta*** *要增加几(大于0)  \** ***@return*** *\*/* **public long** incr(String key, **long** delta) {  **if** (delta < 0) {  **throw new** RuntimeException(**"递增因子必须大于0"**);  }  **return redisTemplate**.opsForValue().increment(key, delta);  }  */\*\*  \* 递减  \** ***@param key*** *键  \** ***@param delta*** *要减少几(小于0)  \** ***@return*** *\*/* **public long** decr(String key, **long** delta) {  **if** (delta < 0) {  **throw new** RuntimeException(**"递减因子必须大于0"**);  }  **return redisTemplate**.opsForValue().increment(key, -delta);  }  *// ================================Map=================================  /\*\*  \* HashGet  \** ***@param key*** *键 不能为null  \** ***@param item*** *项 不能为null  \** ***@return*** *值  \*/* **public** Object hget(String key, String item) {  **return redisTemplate**.opsForHash().get(key, item);  }  */\*\*  \* 获取hashKey对应的所有键值  \** ***@param key*** *键  \** ***@return*** *对应的多个键值  \*/* **public** Map<Object, Object> hmget(String key) {  **return redisTemplate**.opsForHash().entries(key);  }  */\*\*  \* HashSet  \** ***@param key*** *键  \** ***@param map*** *对应多个键值  \** ***@return*** *true 成功 false 失败  \*/* **public boolean** hmset(String key, Map<String, Object> map) {  **try** {  **redisTemplate**.opsForHash().putAll(key, map);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* HashSet 并设置时间  \** ***@param key*** *键  \** ***@param map*** *对应多个键值  \** ***@param time*** *时间(秒)  \** ***@return*** *true成功 false失败  \*/* **public boolean** hmset(String key, Map<String, Object> map, **long** time) {  **try** {  **redisTemplate**.opsForHash().putAll(key, map);  **if** (time > 0) {  expire(key, time);  }  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 向一张hash表中放入数据,如果不存在将创建  \** ***@param key*** *键  \** ***@param item*** *项  \** ***@param value*** *值  \** ***@return*** *true 成功 false失败  \*/* **public boolean** hset(String key, String item, Object value) {  **try** {  **redisTemplate**.opsForHash().put(key, item, value);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 向一张hash表中放入数据,如果不存在将创建  \** ***@param key*** *键  \** ***@param item*** *项  \** ***@param value*** *值  \** ***@param time*** *时间(秒) 注意:如果已存在的hash表有时间,这里将会替换原有的时间  \** ***@return*** *true 成功 false失败  \*/* **public boolean** hset(String key, String item, Object value, **long** time) {  **try** {  **redisTemplate**.opsForHash().put(key, item, value);  **if** (time > 0) {  expire(key, time);  }  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 删除hash表中的值  \** ***@param key*** *键 不能为null  \** ***@param item*** *项 可以使多个 不能为null  \*/* **public void** hdel(String key, Object... item) {  **redisTemplate**.opsForHash().delete(key, item);  }  */\*\*  \* 判断hash表中是否有该项的值  \** ***@param key*** *键 不能为null  \** ***@param item*** *项 不能为null  \** ***@return*** *true 存在 false不存在  \*/* **public boolean** hHasKey(String key, String item) {  **return redisTemplate**.opsForHash().hasKey(key, item);  }  */\*\*  \* hash递增 如果不存在,就会创建一个 并把新增后的值返回  \** ***@param key*** *键  \** ***@param item*** *项  \** ***@param by*** *要增加几(大于0)  \** ***@return*** *\*/* **public double** hincr(String key, String item, **double** by) {  **return redisTemplate**.opsForHash().increment(key, item, by);  }  */\*\*  \* hash递减  \** ***@param key*** *键  \** ***@param item*** *项  \** ***@param by*** *要减少记(小于0)  \** ***@return*** *\*/* **public double** hdecr(String key, String item, **double** by) {  **return redisTemplate**.opsForHash().increment(key, item, -by);  }  *// ============================set=============================  /\*\*  \* 根据key获取Set中的所有值  \** ***@param key*** *键  \** ***@return*** *\*/* **public** Set<Object> sGet(String key) {  **try** {  **return redisTemplate**.opsForSet().members(key);  } **catch** (Exception e) {  e.printStackTrace();  **return null**;  }  }  */\*\*  \* 根据value从一个set中查询,是否存在  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@return*** *true 存在 false不存在  \*/* **public boolean** sHasKey(String key, Object value) {  **try** {  **return redisTemplate**.opsForSet().isMember(key, value);  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 将数据放入set缓存  \** ***@param key*** *键  \** ***@param values*** *值 可以是多个  \** ***@return*** *成功个数  \*/* **public long** sSet(String key, Object... values) {  **try** {  **return redisTemplate**.opsForSet().add(key, values);  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  }  */\*\*  \* 将set数据放入缓存  \** ***@param key*** *键  \** ***@param time*** *时间(秒)  \** ***@param values*** *值 可以是多个  \** ***@return*** *成功个数  \*/* **public long** sSetAndTime(String key, **long** time, Object... values) {  **try** {  Long count = **redisTemplate**.opsForSet().add(key, values);  **if** (time > 0)  expire(key, time);  **return** count;  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  }  */\*\*  \* 获取set缓存的长度  \** ***@param key*** *键  \** ***@return*** *\*/* **public long** sGetSetSize(String key) {  **try** {  **return redisTemplate**.opsForSet().size(key);  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  }  */\*\*  \* 移除值为value的  \** ***@param key*** *键  \** ***@param values*** *值 可以是多个  \** ***@return*** *移除的个数  \*/* **public long** setRemove(String key, Object... values) {  **try** {  Long count = **redisTemplate**.opsForSet().remove(key, values);  **return** count;  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  }  *// ===============================list=================================  /\*\*  \* 获取list缓存的内容  \** ***@param key*** *键  \** ***@param start*** *开始  \** ***@param end*** *结束 0 到 -1代表所有值  \** ***@return*** *\*/* **public** List<Object> lGet(String key, **long** start, **long** end) {  **try** {  **return redisTemplate**.opsForList().range(key, start, end);  } **catch** (Exception e) {  e.printStackTrace();  **return null**;  }  }  */\*\*  \* 获取list缓存的长度  \** ***@param key*** *键  \** ***@return*** *\*/* **public long** lGetListSize(String key) {  **try** {  **return redisTemplate**.opsForList().size(key);  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  }  */\*\*  \* 通过索引 获取list中的值  \** ***@param key*** *键  \** ***@param index*** *索引 index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，表尾，-2倒数第二个元素，依次类推  \** ***@return*** *\*/* **public** Object lGetIndex(String key, **long** index) {  **try** {  **return redisTemplate**.opsForList().index(key, index);  } **catch** (Exception e) {  e.printStackTrace();  **return null**;  }  }  */\*\*  \* 将list放入缓存  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@param time*** *时间(秒)  \** ***@return*** *\*/* **public boolean** lSet(String key, Object value) {  **try** {  **redisTemplate**.opsForList().rightPush(key, value);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 将list放入缓存  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@param time*** *时间(秒)  \** ***@return*** *\*/* **public boolean** lSet(String key, Object value, **long** time) {  **try** {  **redisTemplate**.opsForList().rightPush(key, value);  **if** (time > 0)  expire(key, time);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 将list放入缓存  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@param time*** *时间(秒)  \** ***@return*** *\*/* **public boolean** lSet(String key, List<Object> value) {  **try** {  **redisTemplate**.opsForList().rightPushAll(key, value);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 将list放入缓存  \*  \** ***@param key*** *键  \** ***@param value*** *值  \** ***@param time*** *时间(秒)  \** ***@return*** *\*/* **public boolean** lSet(String key, List<Object> value, **long** time) {  **try** {  **redisTemplate**.opsForList().rightPushAll(key, value);  **if** (time > 0)  expire(key, time);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 根据索引修改list中的某条数据  \** ***@param key*** *键  \** ***@param index*** *索引  \** ***@param value*** *值  \** ***@return*** *\*/* **public boolean** lUpdateIndex(String key, **long** index, Object value) {  **try** {  **redisTemplate**.opsForList().set(key, index, value);  **return true**;  } **catch** (Exception e) {  e.printStackTrace();  **return false**;  }  }  */\*\*  \* 移除N个值为value  \** ***@param key*** *键  \** ***@param count*** *移除多少个  \** ***@param value*** *值  \** ***@return*** *移除的个数  \*/* **public long** lRemove(String key, **long** count, Object value) {  **try** {  Long remove = **redisTemplate**.opsForList().remove(key, count, value);  **return** remove;  } **catch** (Exception e) {  e.printStackTrace();  **return** 0;  }  } } |