存在数据库里面，项目启动：

1. 查询数据库开始走定时任务
2. 新增数据到定时器表里面， 开始走定时任务
3. 启动类上面

public static void main(String[] args) {  
 ConfigurableApplicationContext configurableApplicationContext = SpringApplication.*run*(CronApplication.class, args);  
 configurableApplicationContext.getBean(MyLampTask.class).startLampTask();  
 }

上面的MyLampTask类在配置文件里面的类名

startLampTask是MyLampTask类里面的方法名

1. 两个配置文件

MyLampRunable 和 MyLampTask

MyLampTask相当于controller层， 注入service层直接查询数据库里面的定时器任务，

Sql语句就是查询出cron字段 select cron from ‘表名’ group by cron

MyLampTask类

@Component  
@EnableScheduling  
public class MyLampTask implements SchedulingConfigurer {  
  
 @Autowired  
 private TestService testService;  
  
 private final String FIELD\_SCHEDULED\_FUTURES = "scheduledFutures";  
 private Set<ScheduledFuture<?>> scheduledFutures = new HashSet<>();  
 private Map<String, ScheduledFuture<?>> taskFutures = new ConcurrentHashMap<String, ScheduledFuture<?>>();  
 private ScheduledTaskRegistrar taskRegistrar;  
  
 @Override  
 public void configureTasks(ScheduledTaskRegistrar scheduledTaskRegistrar) {  
 this.taskRegistrar = scheduledTaskRegistrar;  
 }  
  
  
 */\*\*  
 \* 启动类里面加配置 applicationContext.getBean(MyLampTask.class).startLampTask(); 项目起来就查数据库里面的定时器放在次配置类里面  
 \* 的 scheduledFutures set集合中  
 \** ***@param*** *\** ***@return*** *\*/* public void startLampTask() {  
 List<Map<String,Object>> timelist = testService.queryLampTime();  
 for(int i=0;i<timelist.size();i++) {  
 String cron = timelist.get(i).get("cron").toString();  
 if(!taskFutures.containsKey(cron)) {  
 //taskRegistrar.setScheduler(null);  
 TaskScheduler scheduler = taskRegistrar.getScheduler();  
 ScheduledFuture<?> future = scheduler.schedule(new MyLampRunnable(cron, testService), getTrigger(cron));  
 scheduledFutures.add(future);  
 taskFutures.put(cron, future);  
 }  
 }  
 }  
  
 */\*\*  
 \* 动态的添加定时器 业务service层里面调用此方法  
 \** ***@param*** *map  
 \*/* public void addLamptimetask(Map<String,Object> map) {  
 String cron = map.get("cron").toString();  
 if(!taskFutures.containsKey(cron)) {  
 // int i = commonService.addLampTime(map);  
 TaskScheduler scheduler = taskRegistrar.getScheduler();  
 ScheduledFuture<?> future = scheduler.schedule(new MyLampRunnable(cron,testService), getTrigger(cron));  
 getScheduledFutures().add(future);  
 taskFutures.put(cron, future);  
 }  
 }  
  
 */\*\*  
 \* 全部删除定时器  
 \*/* public void clearLamptimetask() {  
 Iterator<ScheduledFuture<?>> it = scheduledFutures.iterator();  
 while (it.hasNext()) {  
 ScheduledFuture<?> future = it.next();  
 future.cancel(true);  
 it.remove();  
 //String str = it.next();  
 //.out.println(str);  
 }  
 }  
  
 */\*\*  
 \* 根据传入的参数删除定时器  
 \** ***@param*** *list  
 \*/* public void clearLamptimetask(List<Map<String,Object>> list) {  
 for(int i=0;i<list.size();i++) {  
 String cron = list.get(i).get("cron").toString();  
 ScheduledFuture<?> future = taskFutures.get(cron);  
 if (future != null) {  
 future.cancel(true);  
 }  
 taskFutures.remove(cron);  
 getScheduledFutures().remove(future);  
 }  
 }  
  
  
  
 private Trigger getTrigger(String cron) {  
 return new Trigger() {  
 @Override  
 public Date nextExecutionTime(TriggerContext triggerContext) {  
 // 触发器  
 CronTrigger trigger = new CronTrigger(cron);  
 return trigger.nextExecutionTime(triggerContext);  
 }  
 };  
 }  
 private Set<ScheduledFuture<?>> getScheduledFutures() {  
 if (scheduledFutures == null) {  
 try {  
 scheduledFutures = (Set<ScheduledFuture<?>>) *getProperty*(taskRegistrar, FIELD\_SCHEDULED\_FUTURES);  
 }catch (NoSuchFieldException e){  
 //throw new SchedulingException("not found scheduledFutures field.");  
 }  
 }  
 return scheduledFutures;  
 }  
 public static Object getProperty(Object obj, String name) throws NoSuchFieldException {  
 Object value = null;  
 Field field = *findField*(obj.getClass(), name);  
 if (field == null) {  
 throw new NoSuchFieldException("no such field [" + name + "]");  
 }  
 boolean accessible = field.isAccessible();  
 field.setAccessible(true);  
 try {  
 value = field.get(obj);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 field.setAccessible(accessible);  
 return value;  
 }  
  
 public static Field findField(Class<?> clazz, String name) {  
 try {  
 return clazz.getField(name);  
 } catch (NoSuchFieldException ex) {  
 return *findDeclaredField*(clazz, name);  
 }  
 }  
  
 public static Field findDeclaredField(Class<?> clazz, String name) {  
 try {  
 return clazz.getDeclaredField(name);  
 } catch (NoSuchFieldException ex) {  
 if (clazz.getSuperclass() != null) {  
 return *findDeclaredField*(clazz.getSuperclass(), name);  
 }  
 return null;  
 }  
 }  
}

MyLampRunable 类：

public class MyLampRunnable implements Runnable{  
 private TestService testService;  
  
 private String cron;  
  
 public MyLampRunnable(String time, TestService service) {  
 this.cron = time;  
 this.testService = service;  
 }  
  
 */\*\*  
 \* 开启线程, 有定时器就开始走此方法, 根据上面的time这个cron定时器表达式,查出需要控制的设备,进行控制...  
 \*/* @Override  
 public void run() {  
 List<Map<String, Object>> list = testService.selectTaskList(cron);  
 for (Map<String, Object> obj: list) {  
 String name = obj.get("name").toString();  
 String cron = obj.get("cron").toString();  
 System.*err*.println("定时器的name是" + name + "，定时表达式：" + cron);  
 }  
 }  
}

MyLampRunable就是监听定时器任务

1. 写接口增加定时器， 删除定时器

3-1）增加

在serviceImpl层中既要注入dao层，又要注入MyLampTask，既要增加定时器的数据库里面的数据，又要增加MyLampTask里面的集合

testDao.addTask(map); // 增加数据库里面的数据  
myLampTask.addLamptimetask(map); // 增加定时器

// map为数据库（定时器表）里面的字段

3-2）删除

既要删除数据库里面的数据，又要删除MyLampTask里面的集合

List<Map<String, Object>> list = testDao.selectById(id); // 根据id查询数据库  
myLampTask.clearLamptimetask(list); // 将查到的定时器数据传到MyLampTask进行删除定时器的数据  
testDao.deleteCron(id); // 删除数据库