太强了！一个注解解决数据脱敏问题

飘渺Jam JAVA日知录 2023-05-31 08:33 Posted on 安徽

本文主要分享什么是数据脱敏，如何优雅的在项目中运用一个注解实现数据脱敏，为项目进行赋能。希望能给你们带来帮助。

什么是数据脱敏

数据脱敏是一种通过去除或替换敏感数据中的部分信息，以保护数据隐私和安全的技术。其主要目的是确保数据仍然可以在各种场景中使用，同时保护敏感信息，防止数据泄露和滥用。数据脱敏通常用于处理包含个人身份信息和其他敏感信息的数据集，如手机号、姓名、地址、银行卡、身份证号、车牌号等等。

在数据脱敏过程中，通常会采用不同的算法和技术，以根据不同的需求和场景对数据进行处理。例如，对于身份证号码，可以使用掩码算法（masking）将前几位数字保留，其他位用“X”或"*"代替；对于姓名，可以使用伪造（pseudonymization）算法，将真实姓名替换成随机生成的假名。

下面我讲为大家带来数据脱敏掩码操作，让我们一起学起来吧。

开胃菜

下面给大家介绍的是使用两种不同的工具类进行数据脱敏，而我们今天的主题使用一个注解解决数据脱敏问题的主要两个工具类。来跟着我学习吧。

使用 Hutool 工具类实现数据掩码

比喻说我们现在要对手机号进行数据脱敏，前三后四不掩码，其他全部用 * 进行掩码
如下图代码所示，

我们定义了一个手机号：17677772345，需要进行数据脱敏。

调用的 Hutool 的信息脱敏工具类。

```
@SpringBootTest
class ToolDesensitizationApplicationTests {

    @Test
    void contextLoads() {
        String phoneNumber = "17677772345";
        System.out.println(DesensitizedUtil.mobilePhone(phoneNumber));
    }
```

@稀土掘金技术社区

我们运行一下看看结果。一个简单的数据脱敏就实现了。

```
    @Test
    void contextLoads() {
        String phoneNumber = "17677772345";
        System.out.println(DesensitizedUtil.mobilePhone(phoneNumber));
    }
}
```

Tests passed: 1 of 1 test – 331 ms

```
2023-04-17 21:16:41.384  INFO 17344 --- [           main] .j.t.ToolDesensitizationApplicatio
176****2345
Disconnected from the target VM, address: '127.0.0.1:51671', transport: 'socket'
```

@稀土掘金技术社区

Hutool 信息脱敏工具类

根据上面的一个 Demo，大家可以看到我使用了 Hutool 的信息脱敏工具类进行对手机号掩码脱敏。那么让我们一起看看 Hutool 信息脱敏的工具类吧。

官网文档:

https://hutool.cn/docs/#/core/工具类/信息脱敏工具-DesensitizedUtil

看一下官网的介绍，支持多种脱敏数据类型，满足我们大部分需求，如果需要自定义还提供了自定义的方法实现。

介绍

在数据处理或清洗中，可能涉及到很多隐私信息的脱敏工作，因此Hutool针对常用的信息封装了一些脱敏方法。

现阶段支持的脱敏数据类型包括：

1. 用户id
2. 中文姓名
3. 身份证号
4. 座机号
5. 手机号
6. 地址
7. 电子邮件
8. 密码
9. 中国大陆车牌，包含普通车辆、新能源车辆
10. 银行卡

整体来说，所谓脱敏就是隐藏掉信息中的一部分关键信息，用 * 代替，自定义隐藏可以使用 `StrUtil.hide` 方法完成。

下面是里面定义号的脱敏规则，直接调用就可以实现简单的数据脱敏，这里给大家介绍是因为我们今天要给大家带来的注解实现数据脱敏核心就是利用我们的 Hutool 提供的工具类实现，支持自定义隐藏。

使用

我们以身份证号码为例：

```java
// 5***************1X
DesensitizedUtil.idCardNum("51343620000320711X", 1, 2);
```

对于约定俗成的脱敏，我们可以不用指定隐藏位数，比如手机号：

```java
// 188****1999
DesensitizedUtil.mobilePhone("18049531999");
```

当然还有一些简单粗暴的脱敏，比如密码，只保留了位数信息：

```java
// **********
DesensitizedUtil.password("1234567890");
```


使用 Jackson 进行数据序列化脱敏

首先创建一个实体类，此实体类只有一个测试的手机号。

注解的讲解：

- @Data：lombok 的注解生成 get,set 等等方法。
- @JsonSerialize(using = TestJacksonSerialize.class)：该注解的作用就是可自定义序列化，可以用在注解上，方法上，字段上，类上，运行时生效等等，根据提供的序列化类里面的重写方法实现自定义序列化。可以看下下面的源码，有兴趣的朋友可以去了解一下，也能解决我们日常开发中很多场景。

```java
@Target({ElementType.ANNOTATION_TYPE, ElementType.METHOD, ElementType.FIELD, ElementType.TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@JacksonAnnotation
public @interface JsonSerialize {
    Class<? extends JsonSerializer> using() default None.class;

    Class<? extends JsonSerializer> contentUsing() default None.class;

    Class<? extends JsonSerializer> keyUsing() default None.class;

    Class<? extends JsonSerializer> nullsUsing() default None.class;

    Class<?> as() default Void.class;
```



```java
@Data
public class TestDTO implements Serializable {
    /**
     * 手机号
     */
    @JsonSerialize(using = TestJacksonSerialize.class)
    private String phone;
}
```

然后创建一个 TestJacksonSerialize 类实现自定义序列化。

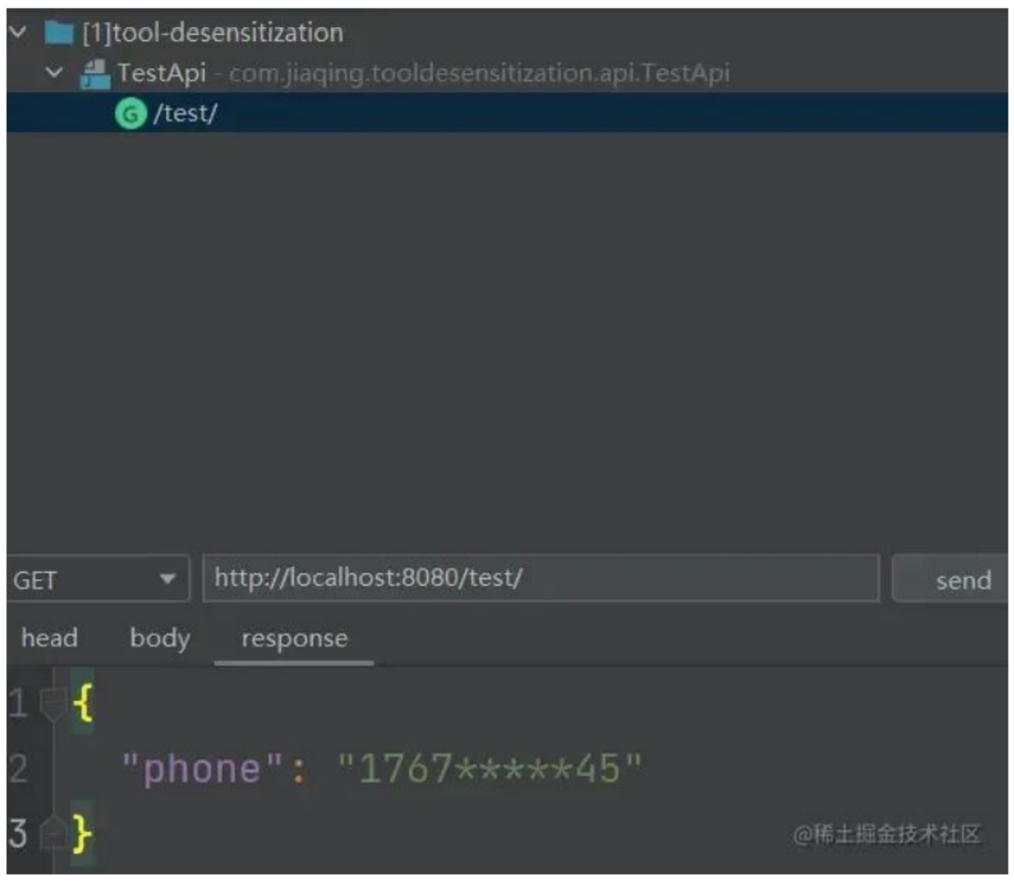
此类主要继承 JsonSerializer，因为我们这里需要序列化的类型是 String 泛型就选择 String。注意如果你使用此注解作用在类上的话，这里就是你要序列化的类。

重写序列化方法，里面的实现很简单就是调用我们的 Hutool 工具类进行手机号数据脱敏。

```java
public class TestJacksonSerialize extends JsonSerializer<String> {

    @Override
    @SneakyThrows
    public void serialize(String str, JsonGenerator jsonGenerator, SerializerP
        // 使用我们的hutool工具类进行手机号脱敏
        jsonGenerator.writeString(DesensitizedUtil.fixedPhone(String.valueOf(s
    }
}
```

让我们测试一下吧，因为此注解是运行时生效，我们定义一个接口来测试。

```
@RestController
@RequestMapping("/test")
public class TestApi {

    @GetMapping
    public TestDTO test(){
        TestDTO testDTO = new TestDTO();
        testDTO.setPhone("17677772345");
        return testDTO;
    }
}
```



可以看到测试成功，经过上面的两个工具类的介绍，联想一下我们怎么通过两个工具类定义一个自己的注解实现数据脱敏呢。

注解实现数据脱敏

我们考虑一下，工具类现在有了，那么我们怎么去实现一个注解优雅的解决数据脱敏呢?

请看下文，让我带大家一起学习。

1、定义一个注解

定义一个 Desensitization 注解。

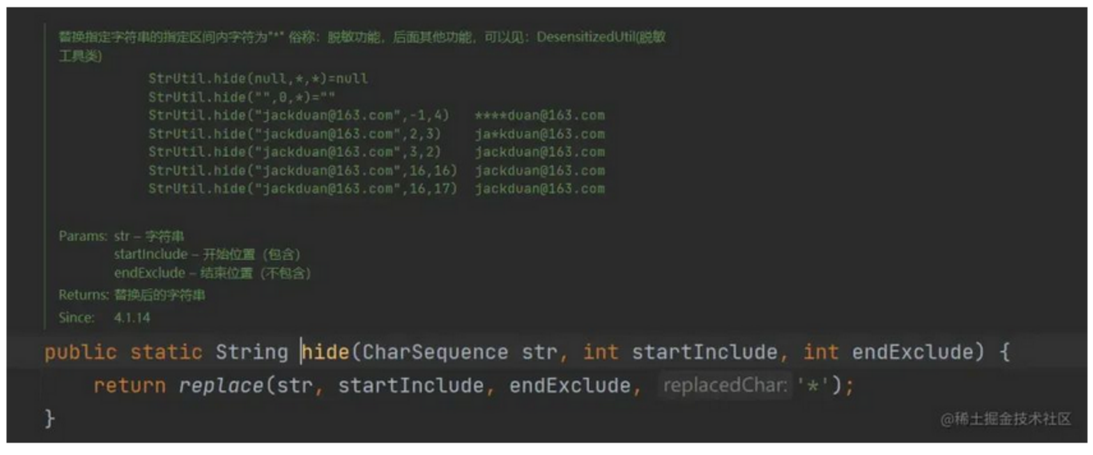
- @Retention(RetentionPolicy.RUNTIME)：运行时生效。
- @Target(ElementType.FIELD)：可用在字段上。
- @JacksonAnnotationsInside：此注解可以点进去看一下是一个元注解，主要是用户打包其他注解一起使用。
- @JsonSerialize：上面说到过，该注解的作用就是可自定义序列化，可以用在注解上，方法上，字段上，类上，运行时生效等等，根据提供的序列化类里面的重写方法实现自定义序列化。

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
@JacksonAnnotationsInside
@JsonSerialize(using = DesensitizationSerialize.class)
public @interface Desensitization {
    /**
     * 脱敏数据类型，只要在CUSTOMER的时候，startInclude和endExclude生效
     */
    DesensitizationTypeEnum type() default DesensitizationTypeEnum.CUSTOMER;

    /**
     * 开始位置（包含）
     */
    int startInclude() default 0;

    /**
     * 结束位置（不包含）
     */
    int endExclude() default 0;
}
```

可以看到此注解有三个值，一个是枚举类定义了我们的脱敏数据类型。一个开始位置，一个结束位置。

枚举类待会给大家讲解，如果选择了自定义类型，下面的开始位置，结束位置才生效。

开始结束位置是我们 Hutool 工具提供的自定义脱敏实现需要的参数。可以看此方法，需要提出一点的是此方法硬编码了掩码值。如果我们的场景需要其他掩码值的话实现也很简单，把 Hutool 的源码拷出来，代替他的硬编码，就可以实现。



2、创建一个枚举类

此枚举类是我们数据脱敏的类型，包括了大部分场景。以及可以满足我们日常开发咯。

```
public enum DesensitizationTypeEnum {
    //自定义
    CUSTOMER,
    //用户id
    USER_ID,
    //中文名
    CHINESE_NAME,
    //身份证号
    ID_CARD,
    //座机号
    FIXED_PHONE,
    //手机号
    MOBILE_PHONE,
    //地址
    ADDRESS,
    //电子邮件
    EMAIL,
    //密码
    PASSWORD,
    //中国大陆车牌，包含普通车辆、新能源车辆
    CAR_LICENSE,
    //银行卡
    BANK_CARD
}
```

3、创建我们的自定义序列化类

此类是我们数据脱敏的关键。主要是继承了我们的 JsonSerializer，实现了我的 ContextualSerializer。重写了它俩的方法。

- @NoArgsConstructor：Lombok 无参构造生成。
- @AllArgsConstructor：Lombok 有参生成。
- ContextualSerializer：这个类是序列化上下文类，主要是解决我们这个地方获取字段的一些信息，可以看一下源码，他的实现类有很多，Jackson 提供的 @JsonFormat 注解也是实现此类，获取字段的一些信息进行序列化的。有兴趣的朋友可以看一下，多看源码，才能学到 Jackson 的实现方法，才能有今天我们的实现。

两个重写的方法解读：

- serialize：重写，实现我们的序列化自定义。
- createContextual：序列化上下文方法重写，获取我们的字段一些信息进行判断，然后返回实例。具体代码可以看下面代码，都有注释噢。

```java
@NoArgsConstructor
@AllArgsConstructor
public class DesensitizationSerialize extends JsonSerializer<String> implemer
    private DesensitizationTypeEnum type;

    private Integer startInclude;

    private Integer endExclude;
    @Override
    public void serialize(String str, JsonGenerator jsonGenerator, Serializer
        switch (type) {
            // 自定义类型脱敏
            case CUSTOMER:
                jsonGenerator.writeString(CharSequenceUtil.hide(str,startIncl
                break;
            // userId脱敏
            case USER_ID:
                jsonGenerator.writeString(String.valueOf(DesensitizedUtil.use
                break;
            // 中文姓名脱敏
            case CHINESE_NAME:
                jsonGenerator.writeString(DesensitizedUtil.chineseName(String
                break;
            // 身份证脱敏
            case ID_CARD:
                jsonGenerator.writeString(DesensitizedUtil.idCardNum(String.v
                break;
            // 固定电话脱敏
            case FIXED_PHONE:
                jsonGenerator.writeString(DesensitizedUtil.fixedPhone(String.
                break;
            // 手机号脱敏
            case MOBILE_PHONE:
                jsonGenerator.writeString(DesensitizedUtil.mobilePhone(String
                break;
            // 地址脱敏
            case ADDRESS:
                jsonGenerator.writeString(DesensitizedUtil.address(String.val
                break;
            // 邮箱脱敏
            case EMAIL:
                jsonGenerator.writeString(DesensitizedUtil.email(String.value
                break;
            // 密码脱敏
            case PASSWORD:
                jsonGenerator.writeString(DesensitizedUtil.password(String.va
                break;
            // 中国车牌脱敏
            case CAR_LICENSE:
                jsonGenerator.writeString(DesensitizedUtil.carLicense(String.
                break;
            // 银行卡脱敏
            case BANK_CARD:
                jsonGenerator.writeString(DesensitizedUtil.bankCard(String.va
                break;
            default:
        }

    }

    @Override
    public JsonSerializer<?> createContextual(SerializerProvider serializerPr
        if (beanProperty != null) {
            // 判断数据类型是否为String类型
            if (Objects.equals(beanProperty.getType().getRawClass(), String.c
                // 获取定义的注解
                Desensitization desensitization = beanProperty.getAnnotation(
                // 为null
                if (desensitization == null) {
                    desensitization = beanProperty.getContextAnnotation(Deser
                }
                // 不为null
                if (desensitization != null) {
                    // 创建定义的序列化类的实例并且返回，入参为注解定义的type,开始位
                    return new DesensitizationSerialize(desensitization.type(
                            desensitization.endExclude());
                }
            }

            return serializerProvider.findValueSerializer(beanProperty.getTyp
        }
        return serializerProvider.findNullValueSerializer(null);
```

```
    }
}
```

4、测试

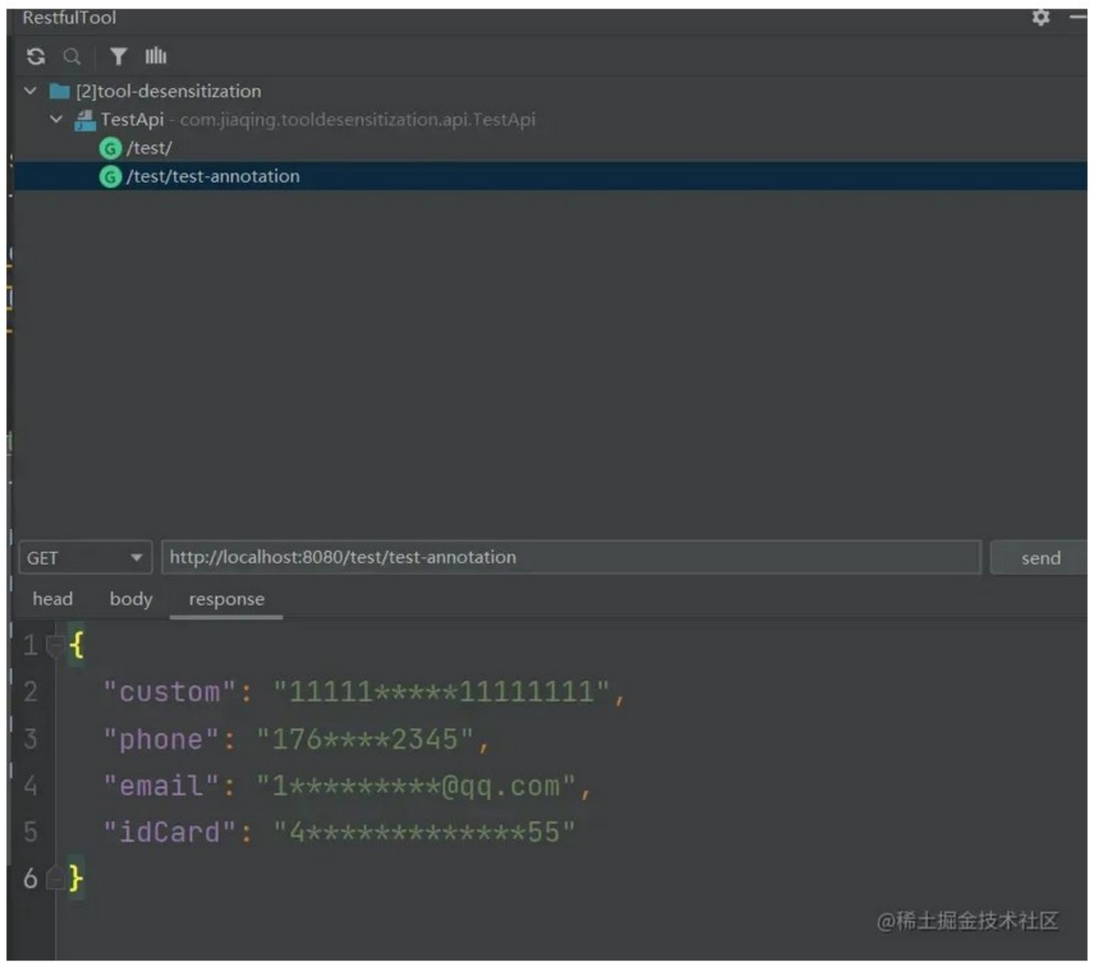
创建一个测试注解的 DTO，此测试如下。

```
@Data
public class TestAnnotationDTO implements Serializable {
    /**
     * 自定义
     */
    @Desensitization(type = DesensitizationTypeEnum.CUSTOMER,startInclude = 5
    private String custom;
    /**
     * 手机号
     */
    @Desensitization(type = DesensitizationTypeEnum.MOBILE_PHONE)
    private String phone;
    /**
     * 邮箱
     */
    @Desensitization(type = DesensitizationTypeEnum.EMAIL)
    private String email;
    /**
     * 身份证
     */
    @Desensitization(type = DesensitizationTypeEnum.ID_CARD)
    private String idCard;
}
```

新增测试接口：

```
@GetMapping("/test-annotation")
public TestAnnotationDTO testAnnotation(){
    TestAnnotationDTO testAnnotationDTO = new TestAnnotationDTO();
    testAnnotationDTO.setPhone("17677772345");
    testAnnotationDTO.setCustom("11111111111111111");
    testAnnotationDTO.setEmail("1433926101@qq.com");
    testAnnotationDTO.setIdCard("4444199810015555");
    return testAnnotationDTO;
}
```

测试一下看看效果。如下图所示，完美！



项目 pom 文件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.c
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.10</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.jiaqing</groupId>
    <artifactId>tool-desensitization</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>tool-desensitization</name>
    <description>数据脱敏</description>
    <properties>
        <java.version>1.8</java.version>
        <hutool.version>5.8.5</hutool.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-core</artifactId>
            <version>${hutool.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--json模块-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-json</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

后记

今天给大家带来的是如何实现一个注解进行数据脱敏。

- 如何使用 hutool 工具类进行数据脱敏。
- 如何使用 @JsonSerialize 注解实现自定义序列化。
- 如何使用 hutool 工具+Jackson 实现自己的脱敏注解。

- End-

最后说一句（别白嫖，求关注）

我的每一篇文章都是精心输出，如果这篇文章对你有所帮助，或者有所启发的话，帮忙**点赞、在看、转发、收藏**，你的支持就是我坚持下去的最大动力！

另外我的 **知识星球** 开通了，公众号回复关键词 知识星球 获取限量30元优惠券加入，每天不到3毛钱。目前更新了**SpringCloud alibaba开发实战、Kubernetes云原生实战、分库分表实战、设计模式实战、架构实战、一起学DDD 、SpringBoot 老鸟等，还有每周的送书活动等着你....**

收录于合集 #开发实战43

‹ 上一篇
Spring Boot 实现多租户架构：支持应用多租户部署和管理

下一篇 ›
加密后的敏感字段还能进行模糊查询吗？该如何实现？