链滴

Spring Boot Mybatis 优雅解决敏感信息加解密问题

作者：NekoChips

原文链接：https://ld246.com/article/1587991687126

来源网站：链滴

前言：在一些特定的应用场景下，需要对用户的敏感信息做加密处理，同样的在读取用户信息的时候需要解密处理。

1. 常见解决方案

针对这种应用场景，通常我们的做法是：**在数据入库之前，对敏感数据进行加密。执行查询 sql 返回果后，再对结果进行解密。**

我们很容易想到如下代码：

```
@Override
public int saveOne(VipCard vipCard) {
    vipCard.setName(encrypt(vipCard.getName()));
    vipCard.setCardNo(encrypt(vipCard.getCardNo()));
    vipCard.setIdNumber(encrypt(vipCard.getIdNumber()));
    vipCard.setPhoneNumber(encrypt(vipCard.getPhoneNumber()));
    return vipCardMapper.saveOne(vipCard);
}
@Override
public VipCard findById(Integer id) {
    VipCard vipCard = vipCardMapper.findById(id);
    vipCard.setName(decrypt(vipCard.getName()));
    vipCard.setCardNo(decrypt(vipCard.getCardNo()));
    vipCard.setIdNumber(decrypt(vipCard.getIdNumber()));
    vipCard.setPhoneNumber(decrypt(vipCard.getPhoneNumber()));
    return vipCard;
}
```

如果在设计阶段就提出来了这加密的需求，那这种方案是没有问题的。而实际的应用场景可能并不是么简单。

紧接着上面的解决方案，通过一问一答的方式了解这种方案的弊端：

Q：如果是后期业务更改，数据库中已经存在了大量明文敏感信息的情况怎么处理呢？

A：这个简单，对读取出来的内容做个判断再进行处理：

```
@Override
public VipCard findById(Integer id) {
    VipCard vipCard = vipCardMapper.findById(id);

    // ......
    String cardNo = vipCard.getCardNo();
    vipCard.setCardNo(isEncrypt(cardNo) ? decrypt(vipCard.getCardNo()) : cardNo);
    // ......
    return vipCard;
}
```

Q：如果业务中大量存在这种逻辑代码呢？难道每一个都去手动修改吗？

A：这个也难不倒我，使用 Spring AOP 搭配自定义注解的方式对类似的操作进行统一处理。

Q：现在都是分布式系统架构，如果很多模块中都存在这种逻辑呢？

A：把上一步的代码写到通用模块中，其他应用模块使用就可以了。（心中暗暗自喜）

Q：嗯，这种方案可行是可行，但对已有的代码还是有较大的侵入性，后期维护也不是那么方便。还更好的办法吗？

A：（纳尼？！还有更好的办法？）...... 应该有，但是目前不太了解。

2. 优雅解决方案

相信大部分朋友（包括我）都是使用上述方案来解决敏感信息加解密问题的，那么到底有没有更好的决方案呢？

直到发现了 github 上的一个开源项目 typehandlers-encrypt ，我仿佛打开了新世界的大门。该项中通过 Mybatis 的 TypeHandler 来实现通用的加解密解决方案。

2.1 TypeHandler 介绍

想要使用它，当然是要先了解它，我们先来看看它的源码：

```
public interface TypeHandler<T> {

  void setParameter(PreparedStatement ps, int i, T parameter, JdbcType jdbcType) throws SQL
xception;

  T getResult(ResultSet rs, String columnName) throws SQLException;

  T getResult(ResultSet rs, int columnIndex) throws SQLException;

  T getResult(CallableStatement cs, int columnIndex) throws SQLException;

}
```

重点看下 setParameter 方法中的参数：

- **PreparedStatement ps**：预编译 SQL 对象
- **int i**：参数中所在位置下标
- **T parameter**：参数的 Java 类型
- **JdbcType jdbcType**：数据库中的字段类型

不难推断出：**该接口在指定 实体类——数据类型映射中起关键作用。**

TypeHandler 有一个子类 BaseTypeHandler，源码这里就不贴了。 BaseTypeHandler 是一个抽象，它实现了 TypeHandler 方法去调用它自己的抽象方法，抽象方法通过具体的子类实现（这里使用是 **模板方法模式**）。

BaseTypeHandler 的子类几乎囊括了我们常用的 Java 类型处理器，这里就拿 StringTypeHandler 为示例：

```
public class StringTypeHandler extends BaseTypeHandler<String> {

  @Override
  public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType jd
cType)
      throws SQLException {
    ps.setString(i, parameter);
  }

  @Override
  public String getNullableResult(ResultSet rs, String columnName)
      throws SQLException {
    return rs.getString(columnName);
  }

  @Override
  public String getNullableResult(ResultSet rs, int columnIndex)
      throws SQLException {
    return rs.getString(columnIndex);
  }

  @Override
  public String getNullableResult(CallableStatement cs, int columnIndex)
      throws SQLException {
    return cs.getString(columnIndex);
  }
}
```

2.2 自定义 CryptTypeHandler

参考 StringTypeHandler 自定义一个 TypeHandler 用于处理字段的加解密。

```
@Slf4j
public class CryptTypeHandler extends BaseTypeHandler<String> {
```

```
    static final int INPUT_MAXIMUM_LENGTH = 18;

    static RSAPublicKey publicKey;

    static RSAPrivateKey privateKey;

    static {
        try {
            KeyPair keyPair = RsaUtil.getKeyPair();
            publicKey = (RSAPublicKey) keyPair.getPublic();
            privateKey = (RSAPrivateKey) keyPair.getPrivate();
        } catch (NoSuchAlgorithmException e) {
            log.error("RsaUtil create keyPair fail", e);
        }
    }

    @Override
    public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType j
bcType) throws SQLException {
        try {
            String encrypt = RsaUtil.encrypt(parameter, publicKey);
            ps.setString(i, encrypt);
        } catch (Exception e) {
            log.error("encrypt message failed", e);
            // 加密失败，使用原始数据
            ps.setString(i, parameter);
        }
    }

    @Override
    public String getNullableResult(ResultSet rs, String columnName) {
        try {
            String parameter = rs.getString(columnName);
            if (isEncrypt(parameter)) {
                return RsaUtil.decrypt(parameter, privateKey);
            }
            // 不是密文直接返回
            return parameter;
        } catch (Exception e) {
            log.error("decrypt message failed", e);
        }
        return null;
    }

    @Override
    public String getNullableResult(ResultSet rs, int columnIndex) {
        try {
            String parameter = rs.getString(columnIndex);
            if (isEncrypt(parameter)) {
                return RsaUtil.decrypt(parameter, privateKey);
            }
            return parameter;
        } catch (Exception e) {
```

```
            log.error("decrypt message failed", e);
        }
        return null;
    }

    @Override
    public String getNullableResult(CallableStatement cs, int columnIndex) {
        try {
            String parameter = cs.getString(columnIndex);
            if (isEncrypt(parameter)) {
                return RsaUtil.decrypt(parameter, privateKey);
            }
            return parameter;
        } catch (Exception e) {
            log.error("decrypt message failed", e);
        }
        return null;
    }

    /**
     * 判断是否为加密过的内容
     */
    private boolean isEncrypt(String parameter) {
        // 这里只是粗略地对密文长度进行判断，不一定准确，实际开发中可从但不限于以下维度进行
断：
        // 1. 密文长度
        // 2. 是否包含中文
        // 3. 特殊字符
        // 4. 特定格式（正则表达式）
        // 5. ....
        return parameter.length() > INPUT_MAXIMUM_LENGTH;
    }
}
```

这里使用 RSA 加密算法对敏感信息进行加解密处理，**这种方式有弊端，后面会说到。**如需更换加密法只需更改加解密代码即可，**但更推荐使用模板方法模式编写通用加解密工具适配各种加密算法。**

2.3 使用类型别名

虽然我们自定义了 CryptTypeHandler ，但如何使用它呢?

我们在 mapper.xml 文件中通常这样写：

```
<resultMap id="VipCardMap" type="com.demo.crypt.bean.VipCard">
        <result property="id" column="id" javaType="Integer"/>
        <result property="cardNo" column="card_no" javaType="String"/>
    ......
</resultMap>
```

而我们要使用 CryptTypeHandler， 就得这样写：

```
<resultMap id="VipCardMap" type="com.demo.crypt.bean.VipCard">
        <result property="id" column="id" javaType="Integer"/>
        <result property="cardNo" column="card_no" typeHandler="com.demo.crypt.core.hand
```

```
er.CryptTypeHandler"/>
    ......
</resultMap>
```

使用 typeHandler 属性就得写 CryptTypeHandler 权限定名。Mybatis 提供的 @Alias 注解可以帮我们使用它的别名来指定字段的类型处理器。

创建 CryptType 类，添加别名：

```
@Alias("Crypt")
public class CryptType {
}
```

在 CryptTypeHandler 中指定处理类型：

```
@Slf4j
@MappedTypes(CryptType.class)
public class CryptTypeHandler extends BaseTypeHandler<String> {
    // ......
}
```

然后 mapper.xml 文件就可以这样写：

```
<resultMap id="VipCardMap" type="com.demo.crypt.bean.VipCard">
        <result property="id" column="id" javaType="Integer"/>
        <result property="cardNo" column="card_no" javaType="Crypt"/>
    ......
</resultMap>
```

2.4 注册类型以及类型处理器

以为这样就可以用了？

flushed 你都没注册这些组件，怎么用嘛！

Mybatis 在 Spring Boot 中配置起来就很方便了，只需这样：

```
mybatis:
  mapper-locations: classpath:mapper/*.xml
  type-aliases-package: com.demo.crypt.core.alias
  type-handlers-package: com.demo.crypt.core.handler
```

3. 测试

建表语句：

```
drop table if exists `vip_card`;
create table `vip_card`
(
    `id`           int          not null auto_increment comment 'id',
    `card_no`      varchar(2048) not null comment '卡号',
    `name`         varchar(2048) not null comment '用户名',
    `id_number`    varchar(2048) not null comment '身份证号',
    `phone_number` varchar(2048) not null comment '手机号',
```

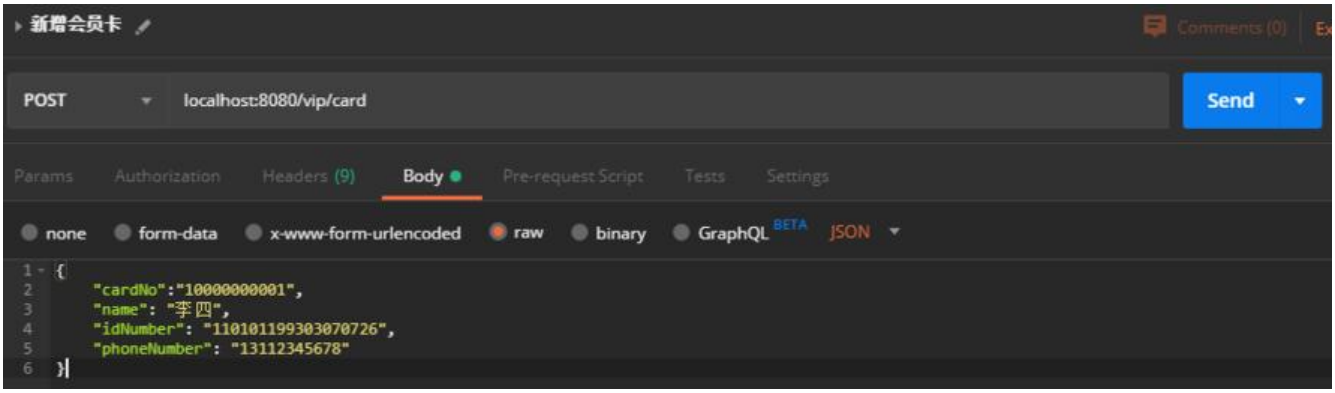
```
    `create_time`  datetime default current_timestamp comment '创建时间',
    `update_time`  datetime comment '更新时间',
    primary key (`id`)
) comment '会员卡表'
    engine = InnoDB
    default charset = utf8;

insert into vip_card
(card_no, name, id_number, phone_number)
values ('10000000000', '张三', '11010119900307221X', '13812345678');
```

为了模拟数据表中已经存在明文数据，这里直接插入一条数据。

为了方便观察执行的 sql，开启 debug：

```
logging:
  level:
    com.demo.crypt.mapper: debug
```

现在我们使用 **postman 添加一条数据**：

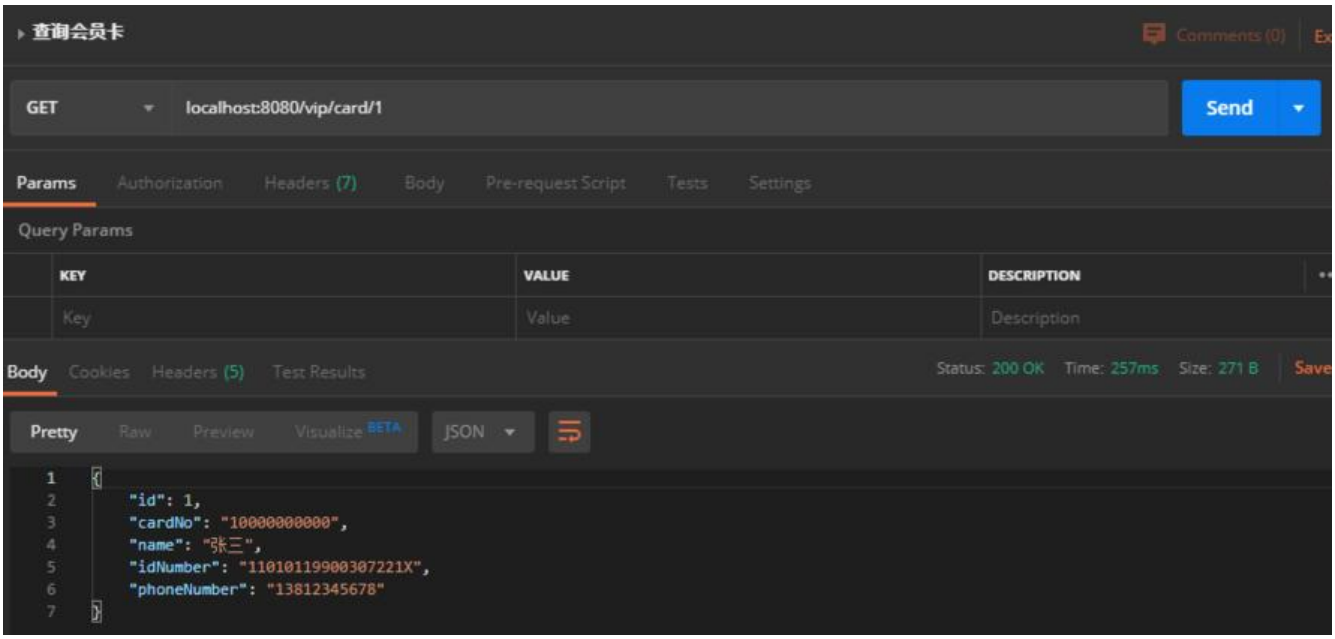


控制台输出：

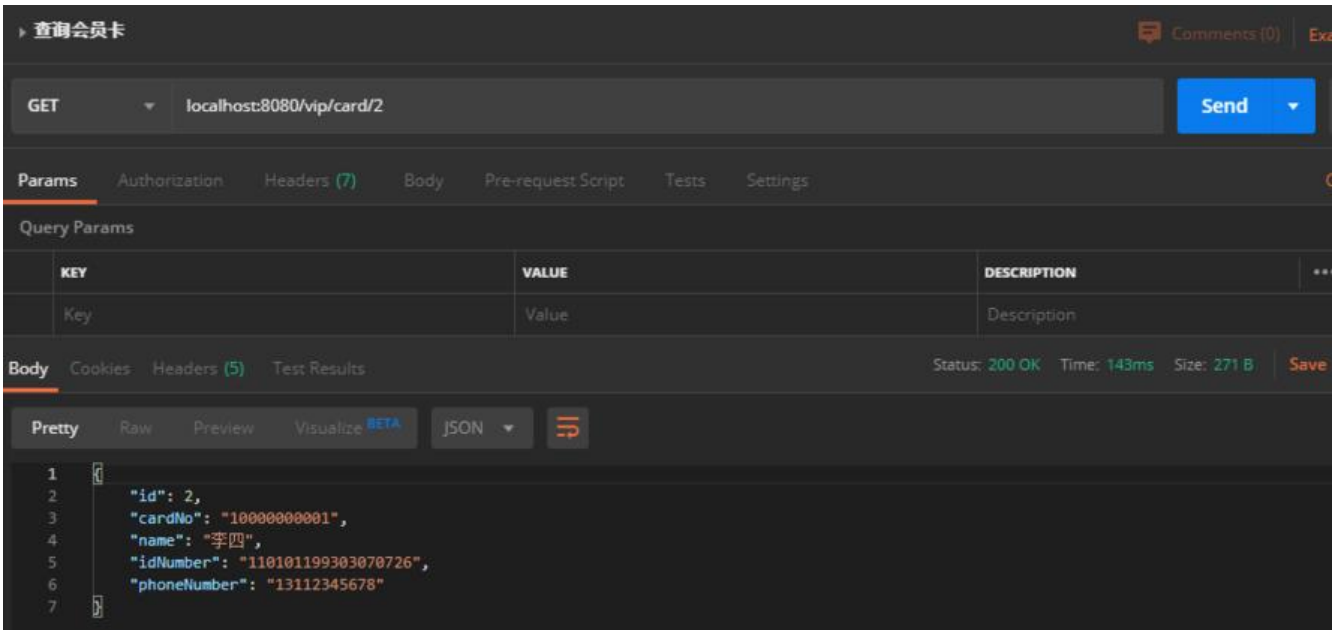
```
c.d.crypt.mapper.VipCardMapper.saveOne   : ==>  Preparing: insert into vip_card (card_no, n
me, id_number, phone_number) values (?, ?, ?, ?)
c.d.crypt.mapper.VipCardMapper.saveOne   : ==> Parameters: S7S56NhMCr3d ..... ElcORxQ==
String), KDzAkeBY512Lilo ..... 5rdHA==(String), hT4QGXpe ..... o8UBA==(String), RICq0gSk .....
 MU1ig7Q==(String)
c.d.crypt.mapper.VipCardMapper.saveOne   : <==    Updates: 1
```

可见插入数据库的数据都经过加密处理。

查询之前已存在的明文数据：



查询之后逻辑添加的密文数据：



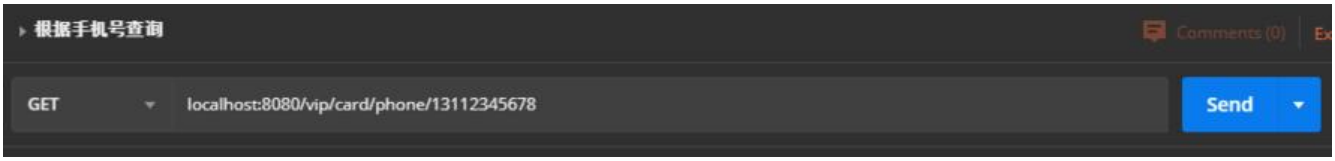
看见了吧，无论是之前存在的**明文数据**还是之后的**密文数据**，对用户而言总是能得到**想要的结果**。

4. 问题

上面提到过当前使用 RsaUtil 加密敏感信息会存在问题。

4.1 问题一

尝试通过敏感字段进行查询，比如 手机号：



由于 Controller 没有做**空值判断**我们直接看控制台：

```
c.d.c.mapper.VipCardMapper.findByPhone  : ==>  Preparing: select id, card_no, name, id_nu
ber, phone_number from vip_card where card_no in (?, ?) limit 1
c.d.c.mapper.VipCardMapper.findByPhone  : ==> Parameters: DSX5oiqHy ..... fDe1fnztw==(S
ring), 13112345678(String)
c.d.c.mapper.VipCardMapper.findByPhone  : <==      Total: 0
```

发现通过手机号查不出会员卡信息，为啥呢?

查阅资料后发现 RSA 加密算法对同样的数据**每次加密的结果不一样**！不仅仅是 RSA ，还有其他的一主流加密算法，如 AES 、DES 等等也是这样的。

当然这种问题也是可以解决的，解决方案不限于 **更换加密算法。**

4.2 问题二

由于这个项目中 RSA 的密钥对是通过工具类随机生成的，所以在每次重启项目后获取的密钥对不一，从而导致上一次启动后新增的数据无法在本次启动中正确解密，也就拿不到数据。这可是一个大 Bu！

解决方案不限于：

1. 自行创建 RSA 密钥对，以文件的形式存放在项目中。
2. 项目首次启动创建密钥对，并将密钥对以文件形式存放在项目中。
3. 使用缓存，项目首次启动创建密钥对，并将密钥对存放在缓存中。

5. 总结

这里虽然只是提到如何通过 Mybatis 的 TypeHandler 实现通用的敏感信息加解密方案，但其实是想过了解 TypeHandler ，然后在实际开发中使用它来实现更多功能。

源码地址：https://github.com/NekoChips/SpringDemo/tree/master/22.springboot-Mybatis-TpeHandler

参考项目：typehandlers-encrypt