MyBatis拦截器在实际项目中的应用

MyBatis 是一个流行的 Java 持久层框架，它简化了数据库访问的复杂性，为开发者提供了强大的功能。其中，MyBatis 拦截器是一个非常有用的特性，可以帮助开发者灵活地解决各种问题。

本文将探讨 MyBatis 拦截器在实际项目中的应用场景和具体实现方法。

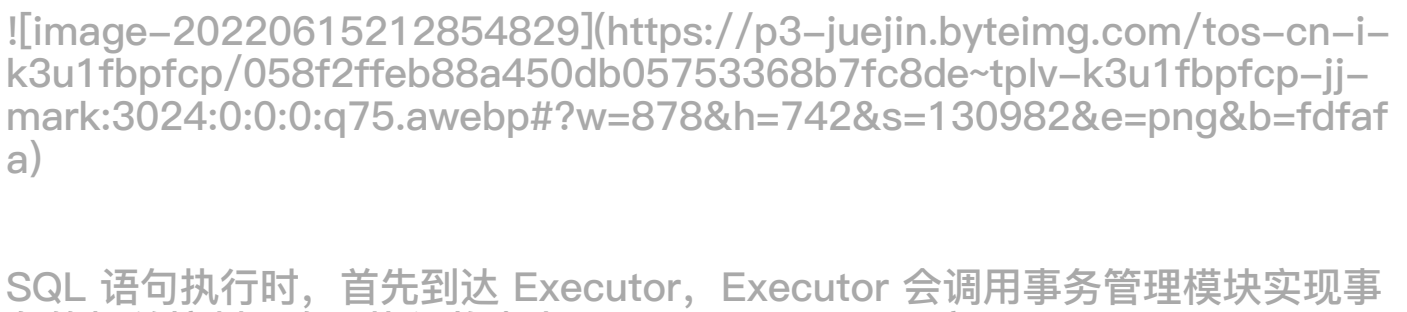
> 文中代码：[github.com/studeyang/m...](http://cxyroad.com/”https://github.com/studeyang/mybatis-interceptor-demo”)

首先我们需要认识 MyBatis 拦截器。

一、MyBatis 拦截器

1.1 从执行 SQL 语句的核心流程说起

在 MyBatis 中，要执行一条 SQL 语句，会涉及非常多的组件，比较核心的有：Executor、StatementHandler、ParameterHandler 和 ResultSetHandler。下图展示了 MyBatis 执行一条 SQL 语句的核心过程：



SQL 语句执行时，首先到达 Executor，Executor 会调用事务管理模块实现事务的相关控制。真正执行将会由 StatementHandler 实现，StatementHandler 会先依赖 ParameterHandler 进行 SQL 模板的实参绑定，然后由 java.sql.Statement 对象将 SQL 语句以及绑定好的实参传到数据库执行。

数据库执行后，从中拿到 ResultSet，最后，由 ResultSetHandler 将 ResultSet 映射成 Java 对象返回给调用方，这就是 SQL 执行模块的核心。

MyBatis 允许开发者拦截这些核心组件的关键方法，从而实现对 SQL 执行过程的自定义控制。

1.2 MyBatis 拦截器

MyBatis 允许我们自定义 Interceptor，拦截 SQL 语句执行过程中的某些关键逻辑，允许拦截的方法有：

* Executor 类中的 update()、query()、flushStatements()、commit()、rollback()、getTransaction()、close()、isClosed() 方法；
* ParameterHandler 中的 setParameters()、getParameterObject() 方法；
* ResultSetHandler中的 handleOutputParameters()、handleResultSets() 方法；
* StatementHandler 中的 parameterize()、prepare()、batch()、update()、query() 方法。

下面，我们就从实际出发，看看 MyBatis 拦截器的具体使用场景。

二、使用场景

2.1 数据加密

多数公司出于信息安全等考虑，会将个人信息等敏感数据在存储时进行加密，在数据读取时进行解密。这种场景就适合使用 MyBatis 拦截器实现了，具体来说：

写入数据时，拦截 insert 和 update 语句，通过自定义注解获取到加密字段，并对其进行加密后再写入数据库。

读取数据时，拦截 select 语句，通过自定义注解获取到加密字段，对密文进行解密，然后返回给上层调用。

这样就能够在不修改业务代码的情况下，自动完成数据的加解密处理了。我们来看具体的代码实现。

在实体属性上添加`@EncryptField`注解：

```
public class UserEntity {
    /**
     * 身份证
     */
    @EncryptField
    private String idCard;
    //其它属性 包括get, set方法
}
```

接着执行下面插入操作：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest extends TestCase {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void insert() {
        UserEntity user = new UserEntity();
        user.setName(”张三”);
        user.setIdCard(”442222111233322210”);
        user.setSex(”男”);
        user.setAge(0);
        user.setCreateTime(new Date());
        user.setUpdateTime(new Date());
        user.setStatus(0);

        userMapper.insert(user);
    }

}
```

数据库里`id_card`就是密文了。

在读取数据时，执行下面查询操作：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserMapperTest extends TestCase {

    @Autowired
    private UserMapper userMapper;

    @Test
    public void selectByPrimaryKey() {
        UserEntity user =
userMapper.selectByPrimaryKey(6822304809682247 68L);
        System.out.println(user);
    }

}
```

返回结果如下：

```
{
  "id": 6822304809682247 68,
  "name": "张三",
  "idCard": "44222211123332 2210",
  "sex": "男",
  "age": 0,
  "createTime": "2024-07-05 10:16:56",
  "updateTime": "2024-07-05 10:16:56",
  "status": 0
}
```

可以看到，拦截器实现了对数据自动解密。拦截器的具体实现请点击：

* [读拦截器](http://cxyroad.com/”https://github.com/studeyang/mybatis-interceptor-demo/blob/master/src/main/java/com/oujiong/plugin/encrypt/WriteEncryptInterceptor.java”)
* [写拦截器](http://cxyroad.com/”https://github.com/studeyang/mybatis-interceptor-demo/blob/master/src/main/java/com/oujiong/plugin/encrypt/ReadEncryptInterceptor.java”)

接着我们来看第二个场景的应用。

2.2 生成ID主键

在生成表主键 ID 时，我们通常会考虑主键自增或者 UUID，但它们都有很明显的缺点。

* 对于自增 ID 来说，第一，容易被爬虫遍历数据；第二，分表分库会有 ID 冲突。
* 对于 UUID 来说，数据太长，且有索引碎片、过多占用索引空间的问题。

雪花算法就很适合在分布式场景下生成唯一 ID，它既可以保证唯一又可以保证有序。通过 MyBatis 拦截器，我们可以实现在插入数据时自动生成全局唯一且有序的雪花 ID。

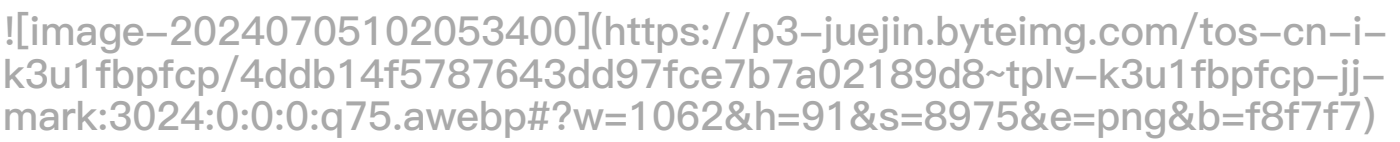
具体做法是：拦截 insert 语句，通过自定义注解获取主键字段，然后为其赋值雪花 ID 后再写入数据库。我们来看具体的代码实现。

在主键的属性上添加`@AutoId`注解。

```
public class UserEntity {
    /**
     * id(添加自定义注解)
     */
    @AutoId
    private Long id;
```

```
    /**
     * 姓名
     */
    private String name;
    //其它属性 包括get，set方法
}
```

执行插入操作后，数据库里就已经有雪花ID了。



如果在正式环境中，由于只要涉及到插入数据的操作都被该插件拦截，并发量会很大。所以该插件代码既要保证线程安全又要保证高性能。

1、线程安全

产生雪花 ID 的时候必须是线程安全的，不能出现同一台服务器同一时刻出现了相同的雪花 ID，可以通过：

```
单例模式 + synchronized
```

来实现。

2、高性能

性能消耗比较大可能会出现在两个地方：

```
1）雪花算法生成雪花ID的过程。
2）通过类的反射机制找到哪些属性带有@AutoId注解的过程。
```

```

第一，生成雪花ID。简单测试过，生成20万条数据，大约在1.7秒，能满足我们实际开发中的需要。

第二，反射查找。可以在插件中添加缓存。

```
 /**
  * key值为Class对象 value可以理解成是该类带有AutoId注解的属性
  */
 private Map<Class, List<Handler>> handlerMap = new ConcurrentHashMap<>();
```

插件部分源码如下：

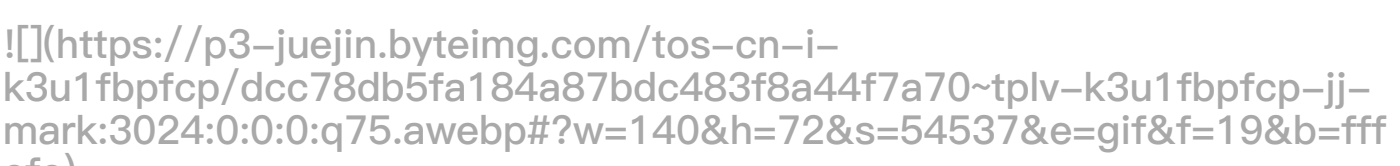
```
 public class AutoIdInterceptor implements Interceptor {
     /**
      * 处理器缓存
      */
     private Map<Class, List<Handler>> handlerMap = new ConcurrentHashMap<>();

     private void process(Object object) throws Throwable {
         Class handlerKey = object.getClass();
         List<Handler> handlerList = handlerMap.get(handlerKey);
         //先判断handlerMap是否已存在该class，不存在先找到该class有哪些属性带有@AutoId
         if (handlerList == null) {
             handlerMap.put(handlerKey, handlerList = new ArrayList<>());
             // 通过反射 获取带有AutoId注解的所有属性字段,并放入到handlerMap中
         }
          //为带有@AutoId赋值ID
         for (Handler handler : handlerList) {
             handler.accept(object);
         }
     }
 }
```
```

```

## 三、小结

MyBatis 拦截器是一个非常值得开发者深入学习和应用的技术。相信通过本文的介绍，您已经对 MyBatis 拦截器有了更加具体的认识。如果您还有任何疑问，欢迎与我交流探讨。



## 封面

## 更多文章

* [Kafka 位移提交的正确姿势](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzkwMTI4NTI1NA==&mid=2247485021&idx=1&sn=f8c8c38477afca017f023a788c1d485e&chksm=c0b652b4f7c1dba2cb568ce4c92fa69ff2e10ae2c628bf89baca2ea98a8b00cfe3d8c82a32d7&cur_album_id=2632990196359430146&scene=189#wechat_redirect”)
* [23种设计模式的必备结构图](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzkwMTI4NTI1NA==&mid=2247484703&idx=1&sn=d08dbd2acd1213934df3c9ac74b5303d&chksm=c0b651f6f7c1d8e005494b36132d1e20ff4ee163af37722bacdefe16cc8dc405b29101c3bc22&cur_album_id=2632990196359430146&scene=190#rd”)

原文链接: https://juejin.cn/post/7388056946120491045
```