Spring Reactive Mongodb Jpa Auditing 审计

作者：lizhongyue248
原文链接：https://ld246.com/article/1586446760594
来源网站：链滴

相信很多人都知道 Jpa 的一个非常强大的功能：审计。

简单地说就是你提供一些审计的元数据，Jpa 会给你自动根据这些元数据去填充你相应的信息。在它实现中，其元数据就是我们的注解或者接口。通过注解或者接口，Jpa 可以更好的自动填充你的实体信息。

举个栗子：

```
class Customer {

  /**
   * 创建时间
   */
  @CreatedDate
  private LocalDateTime createTime = LocalDateTime.now();

  /**
   * 创建用户
   */
  @CreatedBy
  private String createUser;

  /**
   * 最后修改时间
   */
  @LastModifiedDate
  private LocalDateTime modifyTime = LocalDateTime.now();

  /**
   * 最后修改用户
   */
  @LastModifiedBy
```

```
  private String modifyUser;

  // ..... 其他字段
}
```

这四个字段如果在每个实体类创建、修改的时候手动设置无疑是非常麻烦的事情。但是我们通过上面注解就可以实现 Jpa 的审计，让他帮我们去填充。

一般情况

在传统模式中，配合 Spring Security 去实现这一个过程是很简单的，只需要如下几步：

1. 为实体类添加审计注解
    1. @CreatedDate 创建时间
    2. @CreatedBy 创建用户
    3. @LastModifiedDate 最后修改时间
    4. @LastModifiedBy 最后修改用户
2. 启动/配置类添加 @EnableJpaAuditing 开启审计
3. 实体类添加 @EntityListeners(AuditingEntityListener.class) 注解（在后面的版本中可以省略）

这样就可以自动审计了，如果是自定义用户实体了的，需要自定义一下获取用户的方式，例如：

```
@Configuration
public class UserAuditorHandle implements AuditorAware<String> {
    @NotNull
    @Override
    public Optional<String> getCurrentAuditor() {
        return Optional.ofNullable(SecurityContextHolder.getContext())
                .map(SecurityContext::getAuthentication)
                .map(Principal::getName);
    }
}
```

但是这是在传统模式下，使用 servlet 的阻塞式情况下去完成的。

Reactive Mongodb Auditing

使用依赖：

```
org.springframework.boot:spring-boot-starter-data-mongodb-reactive
```

最近在实践响应式微服务的时候就发现代码审计是存在问题的，在 Reactive 的环境下，需要下面几：

1. 同传统模式，为实体类添加审计注解
2. 启动类添加 @EnableMongoAuditing 开启审计

这样我们开启了部分的代码审计，这种模式下只会自动添加 时间 类型的代码审计。

但是我们如果需要用户的审计如何使用呢？AuditorAware 是不存在 Spring Security Reactive 版本。对于时间的审计，它存在一个 ReactiveAuditingEntityCallback 进行审计。官方也在 jira 中提出出了 Auditing should support reactive security context，但是至今为止快三年了，都没有去做。过他倒是给出了一个解决方案，使用 EntityCallbacks 来完成相应的 Reactive 审计。

官网中给出了三个 EntityCallbacks：

注： Jpa 在执行任何操作之前所有的实体类都会转化为 org.bson.Document

Callback	Method	Description	rder
Reactive/BeforeConvertCallback	onBeforeConvert(T entity, String collection)	在实体类转化为org.bson.Document 之前进行调用。	rdered.LOWEST_PRECEDENCE
Reactive/AuditingEntityCallback	onBeforeConvert(Object entity, String collection)	标记审计的实体是 创建 还是 修改	100
Reactive/BeforeSaveCallback	onBeforeSave(T entity, org.bson.Document target, String collection)	在保存实体之前调用。可以修改要持久化的目 Document，其中包含所有映射的实体信息。	Ordered.LOWEST_PRECED

其中第一、三个为接口，第二个是一个类，就是它来完成时间的审计的。如何选择呢？总结了一下他的适用场景

Callback	适用场景
Reactive/BeforeConvertCallback	所有的操作之都会调用。可以对目标进行统一处理，只能获取到转化前的实体类。
Reactive/AuditingEntityCallback	他是第一种的口的一个实现，在没有明确的字段能够识别当前实体是新建还是修改的情况下，可以参照这个实现类行区分和修改。
Reactive/BeforeSaveCallback	只有使用 save 法之前会调用到这个方法，同时他可以对转化后的 Document 进行操作。

很明显，我们需要填充用户的审计信息但是不需要操作 Document，那么就是需要统一处理，同时我以根据 id 去判断是否是新增的实体，所以就直接使用第一个了。

Kotlin 版本

```
@Bean
@Order(99)    // 可选
// 因为强转了会有警告，抑制一下，又因为实体类是 Java 的，他会要求我使用 = 访问属性
// 但是 Java 代码是不可以的，所以同样抑制下
@Suppress("UsePropertyAccessSyntax", "USELESS_CAST")
fun userAuditingHandler() =
  ReactiveBeforeConvertCallback { entity: Any, _: String ->
    ReactiveSecurityContextHolder.getContext()
      .map { context -> context.authentication.principal }
      // 这里根据你的授权机制去修改，强转为对应的授权对象
      .map { it as Jwt }
      .map { it.claims["user_id"].toString() }
      .map {
        val id = (entity as BaseEntity).getId()
```

```
            if (id == null) entity.setCreateUser(it)
            else entity.setModifyUser(it)
            // 一定要强转回去，kotlin 里面 as 以后，下面代码的所有类型就全改变了
            // 不然子类实体的字段会丢失
            entity as Any
        }.defaultIfEmpty(entity)
    }
```

Java 版本

```
@Bean
public ReactiveBeforeConvertCallback<Objects> userAuditingHandler() {
    return (entity, _) ->
        ReactiveSecurityContextHolder.getContext()
            .map(securityContext -> {
                // 这里根据你的授权机制去修改，强转为对应的授权对象并获取用户信息
                Jwt jwt = (Jwt) securityContext.getAuthentication().getPrincipal();
                return jwt.getClaims().get("user_id").toString();
            })
            .map(userId -> {
                String id = ((BaseEntity) entity).getId();
                if (id == null) {
                    ((BaseEntity) entity).setCreateUser(id);
                } else {
                    ((BaseEntity) entity).setModifyUser(id);
                }
                return entity;
            }).defaultIfEmpty(entity);
}
```

值得注意的是，对于 ReactiveBeforeConvertCallback，其 entity 的类型是什么，他就会拦截什类型的实体。这里是 Any/Object，那么他就会拦截所有的实体。同理 Reactive/BeforeSaveCallback 接口也是一样的。

总结

代码审计能够帮我节省不少麻烦事儿的。这次看了他审计的一些源码，其实一开始准备使用第二种他供的实例方式来实现的，但是发现有几个难点。

ReactiveAuditingEntityCallback 中核心就是 auditingHandlerFactory 里面的 IsNewAwareAuditin Handler，但是他需要一个 PersistentEntities 传入，他需要一个 MappingContext 上下文对象去创 。这就很麻烦了（我不会告诉你我找不到这个上下文怎么用。。。），所以在自己能够区分是新增还编辑的情况下大可以自己实现一个简单版本的。