发布命令

什么是命令总线

命令总线是将命令分发给各自的命令Handler的机制。每个命令总是被精确发送到一个命令Handler。如果没有可用的命令Handler为已分发的命令，将会抛出NoHandlerForCommandException异常。对同一命令类型订阅多个命令Handler将导致订阅互相取代。在这种情况下，最后一个订阅获胜。

package org.axonframework.samples.trader.infra.config;

/\*

\* AxonFramework命令拦截器 Axon Framework支持基于JSR 303 Bean Validation的验证。

\* 这允许你用像@NotEmpty和@Pattern这样的注解，去注解命令上的字段。 你需要在你的类路径中include一个JSR

\* 303实现(比如hibernate-Validator)。

\* 然后，在命令总线上配置一个命令拦截器BeanValidationInterceptor，它会自动找到并配置你的验证器实现。

\* 虽然它使用合理的默认值，但你可以根据具体需要调整它。

\*/

// BeanValidationInterceptor类使用了 javax.validation中的接口

// 所以在.gradle中要引入 spring-boot-starter-validation

@Bean

public CommandBus commandBus() {

SimpleCommandBus commandBus = new SimpleCommandBus();

commandBus.registerDispatchInterceptor(new BeanValidationInterceptor<>());

return commandBus;

}

分发命令

CommandBus提供了两个方法去分发命令到它们各自的Handler：dispatch(commandMessage,callback)和dispatch(commandMessage)。第一个参数是一个包含要发送的实际命令的消息。第二个可选的参数接收一个回调，允许在命令处理完成时通知分发组件。这个回调有两个方法:onSuccess()和onFailure()，分别会在命令处理返回后被调用，或者当它抛出一个异常时调用。

调用组件可能不采取在分发命令的同一线程中调用回调。如果调用线程在继续之前依赖于结果，你可以使用FutureCallback。这是一个Future(在java.concurrent包中定义)和Axon的CommandCallback的组合。或者，考虑使用命令网关。

如果一个应用程序不直接对命令的结果感兴趣，可以使用dispatch(commandMessage) 方法。

import static org.axonframework.commandhandling.GenericCommandMessage.asCommandMessage;

command = new CreateCompanyCommand(new CompanyId(), userIdentifier, "Shell", 500, 5000);

commandBus.dispatch(new GenericCommandMessage<>(command));

commandBus.dispatch(asCommandMessage (command));

commandBus.dispatch("I got the magic words", new GenericCommandMessage<>(command) {

@Override protected void onSuccess() { }

@Override public void onFailure(Throwable cause) {cause.printStackTrace();}

});

commandGateway.send(command, new CommandCallback<CreateProductCommand, Object>() {  
 @Override  
 public void onSuccess(CommandMessage<? extends CreateProductCommand> commandMessage, Object o) {  
 *log*.info("commandGateway.send 成功");  
 createSendSuccess(id);  
 }  
 @Override  
 public void onFailure(CommandMessage<? extends CreateProductCommand> commandMessage, Throwable throwable) {  
 *log*.info("onFailureonFailureonFailureonFailureonFailureonFailure");  
 createSendFailure("sfdafdsdfs");  
 }

命令总线的类型

SimpleCommandBus AsynchronousCommandBus DisruptorCommandBus

命令处理器

建议在包含处理状态命令的聚合中直接定义命令处理器，因为命令处理器有可能需要该集合的状态来执行其任务。

要在一个聚合上定义一个命令处理器,只需用@CommandHandler注解命令处理方法即可。带@CommandHandler注解方法的规则和其他处理方法都是一样的。然而,命令不仅通过他们的有效载荷(payload)进行路由。命令消息携带一个名字,该名称默认为命令对象的完全限定类名。

一、聚合的构造函数上时

注意

当@CommandHandler注解放在一个聚合的构造函数上时，相应的命令将创建一个新的聚合实例，并将它添加到存储库。这些命令不需要针对特定聚合的实例。因此，这些命令不需要任何@TargetAggregateIdentifier或@TargetAggregateVersion注解，也不会调用自定义CommandTargetResolver。

当一个命令创建一个聚合实例时，该命令的回调函数在命令执行成功执行后，将得到聚合标识符。

@Aggregate

public class BankAccount {

@AggregateIdentifier

private String id;

@SuppressWarnings("unused")

private BankAccount() {

}

*@CommandHandler*

public BankAccount(CreateBankAccountCommand command) {

apply(new BankAccountCreatedEvent(command.getBankAccountId(), command.getOverdraftLimit()));

}

｝

二、**外部命令处理器**   
在某些情况下,想要直接向一个聚合实例路由命令是不可能。在这种情况下,可以注册一个命令处理器对象。命令处理器对象是一个简单的(常规的)对象，是带@CommandHandle注解的方法。与集合的情况不同，命令处理器对象只有单个实例，该对象处理其方法中声明的所有命令类型。

public class CompanyCommandHandler {

private Repository<Company> repository;

@CommandHandler

public void handleCreateCompany(CreateCompanyCommand command) throws Exception {

repository.newInstance(() -> {

return new Company(command.getCompanyId(), command.getCompanyName(),

command.getCompanyValue(), command.getAmountOfShares());

});

}

｝