06

Enviando e-mail assíncrono

Transcrição

Vamos tentar melhorar a parte assíncrona da aplicação. O `AsyncResponse` foi feito com o propósito de deixara aplicação mais rápida, porém o usuário ainda não terá essa sensação. No vídeo anterior percebemos que o envio de e-mail, toda a operação de pagamento e a requisição REST levaram um tempo considerável para processar e a aplicação ficou sem responder ao usuário.

Será que, de fato, precisamos fazer o usuário esperar pelo envio do e-mail? Ou seja, o e-mail não poderia ser enviado em segundo plano enquanto é confirmado o pagamento para o usuário? Assim que é confirmado o pagamento, segue-se o fluxo normal da aplicação. Parece fazer sentido.

Primeiramente retiramos o código de e-mails:

```
String messageBody = "Sua compra foi realizada com sucesso, com o número de pedido " + compra.ge

mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compra na CD
```

de dentro do `executor` e criamos uma nova Classe `EnviaEmailCompra` dentro do pacote de *services*. Nela teremos o método `enviar()` onde passaremos o código retirado anteriormente:

```
package br.com.casadocodigo.loja.service;

public class EnviaEmailCompra {

    public void enviar() {
        String messageBody = "Sua compra foi realizada com sucesso, com o número de pedido " + 

        mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compr
    }
}
```

Podemos perceber que o Eclipse acusará erro em ambas as linhas copiadas. Falta passar o `mailSender` e o `@Inject` também para a nova Classe:

```
package br.com.casadocodigo.loja.service;

import javax.inject.Inject;
import br.com.casadocodigo.loja.infra.MailSender;

public class EnviaEmailCompra {

    @Inject
    private MailSender mailSender;

    public void enviar() {
        String messageBody = "Sua compra foi realizada com sucesso, com o número de pedido " + 
```

```
        mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compr
    }
}
```

Além disso temos a compra que será obtida através do `buscaPorUuid()` e injetamos o `compraDao` :

```
package br.com.casadocodigo.loja.service;

import javax.inject.Inject;
import br.com.casadocodigo.loja.infra.MailSender;
import br.com.casadocodigo.loja.models.Compra;

public class EnviaEmailCompra {

    @Inject
    private CompraDao compraDao;

    @Inject
    private MailSender mailSender;

    public void enviar() {
        Compra compra = compraDao.buscaPorUuid(uuid);
        String messageBody = "Sua compra foi realizada com sucesso, com o número de pedido " + c

        mailSender.send("compras@casacodigo.com.br", compra.getUsuario().getEmail(), "Nova Compr
    }
}
```

Faltando agora apenas passar o *uuid* como parâmetro do método `enviar()`

```
public void enviar(String uuid) {
    //...
}
```

O e-mail já está em uma classe separada. Como faremos para que o envio do e-mail seja feito em background. É o que chamamos de processo assíncrono. Mas já não vimos isso nos cursos anteriores? Existe uma diferença. O que o `AsyncResponse` está fazendo é abrir uma thread que roda por traz da aplicação de modo assíncrono no qual novos usuários ficarão executando uma thread nova. É um assíncrono no servidor para novos usuários.

Porém o que queremos agora é um assíncrono mais aparente para o usuário. Diremos para o sistema que depois de termos a aprovação da resposta enviaremos o e-mail e liberamos o usuário:

```
executor.submit(() -> {
    try {
        String resposta = pagamentoGateway.pagar(compra.getTotal());

        enviaEmailELiberaOMeuUsuario();

        //...
```

```
    }
});
```

O Java tem que entender que esse método `enviaEmailELiberaOMeuUsuario()` deve trabalhar em background.

O assíncrono é muito importante, mas deve ser usado com muita cautela. Lembre-se que o usuário sempre precisa de uma resposta. Não vamos nos preocupar com isso porque estamos dando uma resposta com o `ar.resume(response)`. Se tivéssemos apenas o envio do e-mail e o usuário não visse mais nada na tela seria estranho.

Em geral usamos o JMS (Java Messaging Service) para fazer o assíncrono no Java. O JMS é uma especificação do Java EE com a qual temos a capacidade de enviar uma mensagem assíncrona. Para isso precisamos que o JMS passe por algumas configurações via *annotations* e um contexto do JMS dentro de `pagamentoService.java`:

```
@Inject
private JMSContext jmsContext;
```

O `JMSContext` é um objeto que tem toda a comunicação com o servidor. Com o contexto do JMS podemos criar a mensagem. Para isso precisaremos de um produtor e de um "ouvidor" (*listener*) dessa mensagem. Então, dentro do método `pagar()` fazemos:

```
JMSProducer producer = jmsContext.creatProducer();
```

E dentro do `executor`:

```
producer.send(destination, compra.getUuid());
```

O `destination` é criado e configurado pelo servidor, mas em tempo de execução é criado pelo próprio servidor e o servidor precisa colocá-lo para nós, logo:

```
private Destination destination;
```

O `destination` é um recurso JMS encontrado através do JNDI, então anotamos com `@Resource`:

```
@Resource(name="")
private Destination destination;
```

Dentro do JMS temos dois tipos de mensagens assíncronas que podemos enviar:

- Tópico (topic): lista de discussão, como grupo de e-mail ou whatsapp, onde todos recebem a mesma mensagem.
- Fila (queue): envia um e-mail para o primeiro da fila, o segundo poderá receber outra mensagem, diferente do Tópico.

No nosso caso usaremos um tópico:

```
@Resource(name="java:/jms/topics/CarrinhoComprasTopico")
private Destination destination;
```

O `/jms` abre o contexto JMS no Wildfly, mas o `/topics` é aleatório, podendo ser substituído por, por exemplo, `/topic` ou `/topico`.

Por enquanto só queremos o envio do e-mail, em uma lista de discussão. Mais para frente podemos colocar outros recursos dentro desse grupo, como geração de nota fiscal, notificação para o departamento de embalagem, etc. Podem haver diversas pessoam ouvindo o mesmo tópico. Por isso é mais interessante usar tópico em vez de fila.

Temos então o `Destination` sendo injetado e o envio da mensagem pelo `producer.send(destination, compra.getUuid())`. O próximo passo configuraremos quem irá ouvir (listener) a mensagem.