07

Executando serviços Assincronamente

Transcrição

Enfim! Veremos como deixar o service executando de forma assíncrona! Para isso usaremos a API de assíncrono do Java EE 7. Ele já disponibiliza um objeto chamado *Executor Service* que nos deixará executar por meio de um *pool* de conexões.

Em `PagamentoService.java` fazemos:

```
private static ExecutorService executor = Executors.newFixedThreadPool(50);
```

O `50` é a quantidade de *Threads* que queremos que sejam criadas pelo executor. Ele permite criar esse Pool de threads autogerenciável pelo Java.

Porém, por si só, o executor não faz toda essa mágica. Precisamos chamá-lo dentro do método `pagar()` com um método `submit()` com um *Runnable*:

```
public Response pagar(@QueryParam("uuid") String uuid) {
    //...

    executor.submit(new Runnable() {

        @Override
        public void run() {

        }
    });

    //...
}
```

Foi criado automaticamente o método `run()` através da classe anônima. No java 8 podemos fazer tudo isso através de *Lambda Expressions*:

```
executor.submit(() -> {

});
```

O primeiro passo é chamar o Gateway, então pegamos a linha correspondente dentro do método `pagar()` e passamos para dentro:

```
executor.submit(() -> {
    pagamentoGateway.pagar(compra.getTotal());

});
```

Perceba que agora dentro de uma thread, também deveremos passar a resposta do Gateway para dentro dela:

```
executor.submit(() -> {
    pagamentoGateway.pagar(compra.getTotal());

    URI responseUri= UriBuilder.fromPath("http://localhost:8080" + context.getContextPath() + "/
Response response = Response.seeOther(responseUri).build();

});
```

O `return response` podemos apagar e o método `pagar()`, consequentemente vira `void`:

```
public void pagar(@QueryParam("uuid") String uuid)
```

Agora que essa execução está sendo feita de forma assíncrona, precisamos notificar o servidor de que a requisição acabou. Fazemos isso atrave´s de um novo objeto que o método `pagar()` recebe como parâmetro:

```
public void pagar(AsyncResponse ar, ...)
```

Estamos integrando, de forma automática, a API de ocorrência do Java com o JAX-RS. Temos que dizer também que a Response pode controlar a suspenção:

```
public void pagar(@Suspended final AsyncResponse ar, ...)
```

E podemos falar qual será a reposta:

```
public void pagar(@Suspended final AsyncResponse ar, ...) {
    //...
    ar.resume(response);
}
```

O `@Suspended` notifica o sevidor que tota a execução desse método deve ser feita em um contexto assíncrono, ou seja, vai liberar o servidor para executar outras tarefas. Depois de ser feita toda a requisição é chamado o `ar.resume(response)`. Podemos colocar toda a requisição dentro de um `try` também:

```
executor.submit(() -> {
    try {
        //...
        ar.resume(response);
    } catch (Exception e) {
        ar.resume(new WebApplicationException(e));
    }
});
```

Dessa forma teremos uma resposta de sucesso e uma de erro.