05

Registrando o cliente

Você se lembra do Retrofit? Vimos que o Retrofit abstrai para nós todo o trabalho com o protocolo HTTP. Não foi preciso se preocupar com os detalhes do protocolo, mas para isso funcionar foi preciso definir *Callbacks*. Aqueles *Callbacks* são classes/objetos nossos que o Retrofit executa quando, por exemplo, receberá a resposta de servidor. Como tudo isso executa fora da *thread* principal do Android, foi necessário criar os *Callbacks*.

De certa forma, o mesmo princípio é aplicado no servidor e podemos ver os *Callbacks* no método `ouvirMensagem` :

```
@ResponseBody
@RequestMapping(method=RequestMethod.GET)
public DeferredResult<Message> ouvirMensagem()  {

    long timeout = 20 * 1000L; //20s
    final DeferredResult<Message> client = new DeferredResult<>(timeout);

    //callbacks
    TimeoutCallback timeoutCallback = new TimeoutCallback(client, clients);
    ClientCallback clientCallback = new ClientCallback(client, clients);

    //passando os callbacks
    client.onTimeout(timeoutCallback);
    client.onCompletion(clientCallback);

    //adicionado o cliente na fila
    clients.offer(client);
    return client;
}
```

Através desse método registramos um cliente na fila de clientes. Isso acontece na linha:

```
clients.offer(client);
```

Agora imagina que você conecte dois clientes, mas nenhum deles enviou uma mensagem ainda. Devemos notificar os clientes apenas quando recebemos a mensagem, e para tal existe a classe `ClienteCallback` . O Spring, como o Retrofit, usa uma nova *thread* para tal e por isso criamos esse *callback*, que será executado pelo Spring.

Como os nossos clientes não podem ficar conectados para sempre, existe um segundo *Callback*, o `TimeoutCallback` . Esse será executado pelo Spring depois do `Timeout` , nesse código, de 20 segundos.

Agora abra a classe `ClienteCallback` e verifique o código. Qual é o objetivo?