Desenvolvimento Reativo

Capítulo 1. Introdução ao desenvolvimento reativo

Prof. Raphael Gomide

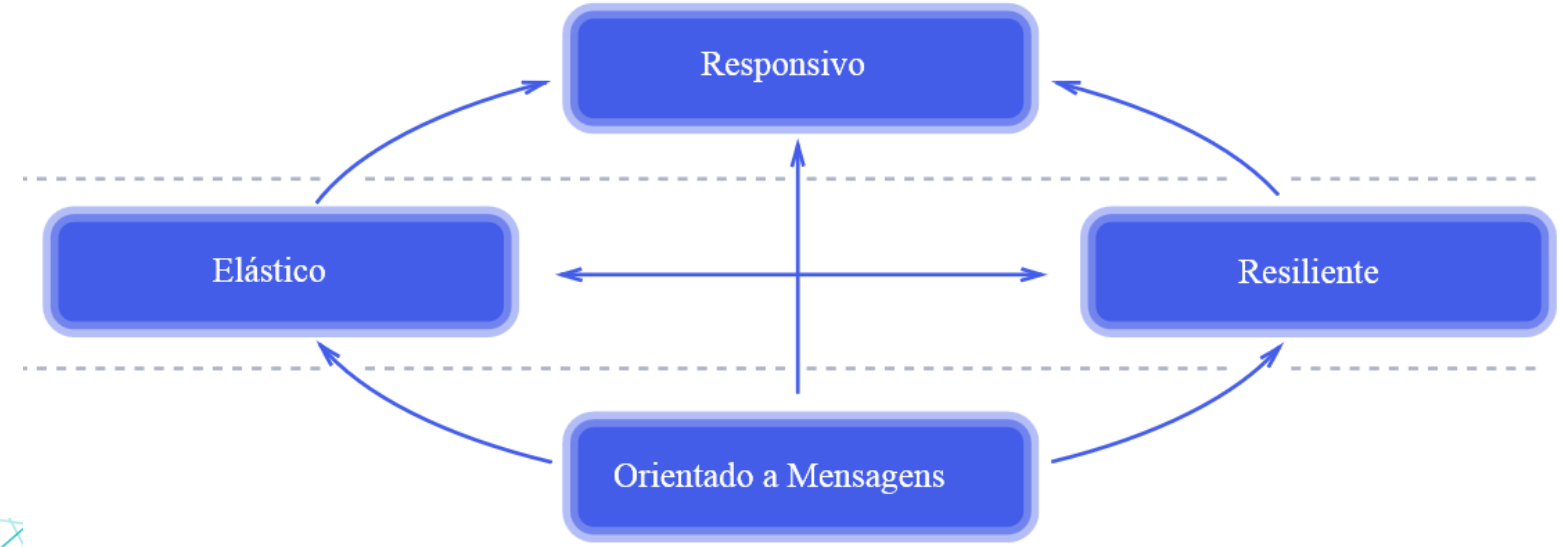
Aula 1.1. Desenvolvimento Reativo

Nesta aula

- Sistemas Reativos (*Reactive Systems*).
- Programação Reativa (*Reactive Programming*).
- Demonstrações com JavaScript.

Sistemas Reativos (*Reactive Systems*)

- Conceito mais amplo.
- Envolve todo o Sistema de Informação.
- Manifesto Reativo (2014).



Programação Reativa (*Reactive Programming*)

- Comportamento **assíncrono**.
- Orientada a **eventos** (*quando/when*).
- Os dados *"chegam"* a qualquer momento, ou seja, são **imprevisíveis**.
- Elementos que "*escutam"* esses dados *"reagem"* às alterações.
- Dados "monitorados" podem representar o **estado** da aplicação (***state***).
- A escrita passa a ser mais declarativa: **o que fazer** (*what*)...
- ... e não mais tão imperativa (clássica): **como fazer** *(how)***.**
- Características funcionais: *Functional Reactive Programming* (FRP):
  - Imutabilidade.
  - *Funções puras.*
  - Funções que recebem outras funções como parâmetros (*Callbacks* ou *Higher Order Functions).*
  - Encadeamento de funções.

Programação Reativa (*Reactive Programming*)

- Exemplos de reatividade:
  - Planilhas;
  - Linguagem SQL;
  - *Web* – eventos.
- JavaScript assíncrono:
  - Utilização de *call-backs;*
  - *Promises;*
  - *Async/await (ES6+).*

```
function somar() {
  var a = 3;
  var b = 8;
  var c = a + b;

  a = 4;
  b = 9;

  //c continua sendo 11
}
```

Não reativo

	A	B
1	A	3
2	B	8
3	C	=B1+B2

Reativo

Demonstração

- Acompanhe o professor:
  - **Reatividade** em JavaScript com formulários;
  - **Execução assíncrona** com utilização de *callbacks;*
  - **Execução assíncrona** com *promise;*
  - **Execução assíncrona** com *async/await.*
- Repositório:
  - https://gitlab.com/rrgomide-rxjs/rxjs-examples.git

Conclusão

- ☑ Sistemas Reativos:
  - Conceito mais amplo (Sistemas de Informação);
  - Responsivo, elástico, resiliente e orientado a mensagens.
- ☑ Programação Reativa:
  - Comportamento assíncrono;
  - Orientada a eventos (quando/when);
  - Reage às mudanças;
  - Escrita declarativa;
  - Foca em o que fazer (what) e não em como fazer (how);
  - Características funcionais.

Próxima aula

- O padrão de projeto *Observer*.

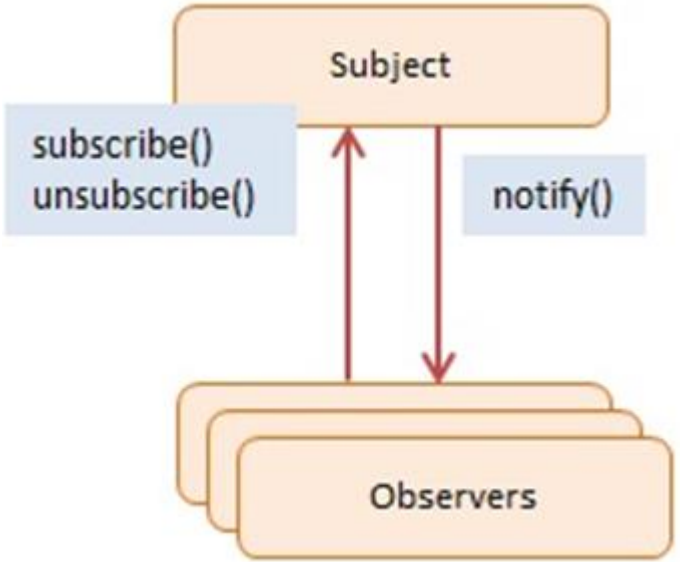
Aula 1.2. O padrão de projeto *Observer*

Nesta aula

- O padrão de projeto Observer.
- Demonstração com JavaScript.

O padrão de projeto *Observer*

- Muito utilizado com manipulação de **eventos.**
- Analogia: modelo de assinaturas de jornais/revistas.
- Funcionamento:
  - Objeto a ser observado – ***Subject****;*
  - ***Observers*** – Objetos interessados em observar o ***Subject****;*
  - Para *observar*, é necessário se inscrever *(**subscribe**);*
  - **Quando** há alteração no ***Subject:***
    - Emitidas notificações (***notify***) para todos os ***observers*** **inscritos**, que podem, então, ***reagir****;*
  - Caso um ***observer*** não mais se interesse no ***Subject***, basta se desinscrever (***unsubscribe***).



Demonstração

- Acompanhe o professor:
  - Exemplo de aplicação com o padrão Observer em JavaScript.
- Repositório:
  - https://gitlab.com/rrgomide-rxjs/rxjs-examples.git

Conclusão

☑ O padrão de projeto Observer:

- Objeto a ser observado: Subject.
- Quem observa são os observers.
- Para observar é necessária a inscrição (subscribe).
- Quando não há mais interesse em observar, basta se desinscrever (unsubscribe).
- O Subject pode notificar os seus observers a cada mudança de estado (notify).

Próxima aula

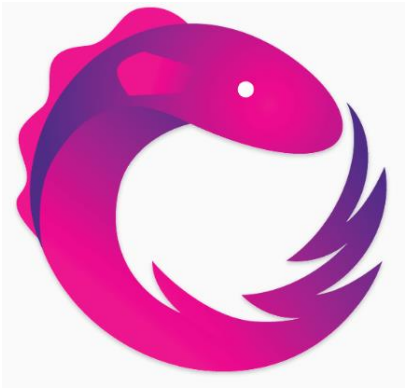
- Introdução à biblioteca ReactiveX for JavaScript (RxJS).

Aula 1.3. RxJS (ReactiveX for JavaScript)

Nesta aula

- Introdução.
- Conceitos importantes.
- Demonstrações.

RxJS – Introdução



- Biblioteca JavaScript.
- Assíncrona.
- Baseadas em eventos.
- Dependência obrigatória do Angular.
- Agrupa, basicamente:
  - Padrão Observer;
  - Padrão Iterator;
  - Programação Funcional.

Conceitos importantes

- ***Observable***: **coleção** invocável de **valores** ou **eventos futuros**.
- ***Observer***: coleção de ***callbacks*** que sabem **reagir** aos **valores** emitidos pelo ***Observable***.
- ***Subscription***: representa a execução de um *Observable.*
- ***Operators****:* funções puras para manipular os dados de forma funcional.

Demonstração

- Acompanhe o professor:
  - RxJS com cliques do mouse.
  - RxJS com busca dinâmica em API.
  - RxJS com streams de dados.
- Repositório:
  - https://gitlab.com/rrgomide-rxjs/rxjs-examples.git

Conclusão

- ☑ RxJS:
  - Biblioteca extremamente útil para:
    - Programação assíncrona;
    - Programação reativa;
    - Programação funcional.
  - Auxilia na construção de software com menos bugs graças a:
    - Imutabilidade;
    - Funções puras;
    - Abstração de implementações complexas.
      - Timeout;
      - Iterações;
      - Transformações de dados.

Próxima aula

- Capítulo 2 – SPAs (*Single Page Applications*).