O que é promisse?

Construtos usados para sincronizar a execução de um programa em linguagens de programação concorrentes. Eles descrevem um objeto que atua como um proxy para um resultado que é, inicialmente, desconhecido devido a sua computação não estar completa no momento da chamada.

Fluxo Assíncrono/Síncrono

O JavaScript por si só é tido como uma linguagem que tem que lidar com várias chamadas e execuções que não acontecem no momento que o programador executou o código, por exemplo, a leitura de um arquivo no NodeJS de forma síncrona:

Esta função é uma função síncrona, ou seja, quando a chamarmos, vamos pausar o que quer que esteja sendo executado e vamos realizar este processamento, depois vamos retornar o valor final. Desta forma estamos fazendo uma operação completamente síncrona. No nosso caso, vamos parar a execução do programa para buscar e ler o arquivo e depois vamos retornar seu resultado ao fluxo normal do programa.

Como queremos que nossas operações e nosso código rodem o mais rápido possível, queremos paralelizar o máximo de ações que conseguirmos. Ações de leitura de arquivos são consideradas lentas porque I/O é sempre mais lento que processamento em memória, vamos paralelizar a nossa função dizendo que queremos ler o arquivo de forma assíncrona:

Agora o que estamos fazendo é passando um callback para a função readFile que deverá ser executado após a leitura do arquivo. Em essência — e abstraindo muito a funcionalidade — o que a função `readFile` faz é algo assim:

Basicamente estamos registrando uma ação que vai ser executada após uma outra ação ser concluída, mas não sabemos quando essa ação será concluída. O que sabemos é apenas que em um momento ela será concluída, então o JavaScript utiliza o EventLoop — que não vamos cobrir neste artigo, mas vocês podem pesquisar aqui e aqui — para registrar um callback, basicamente o que estamos dizendo é: "Quando função X acabar, execute Y e me dê o resultado". Então estamos delegando a resolução de uma computação para outro método.

RESUMINDO:Temos a mesma coisa, porém a qualidade de código ficar muit melhor.

Promises

Promises, como já dissemos, definem uma ação que vai ser executada no futuro, ou seja, ela pode ser resolvida (com sucesso) ou rejeitada (com erro).

Aqui podemos simular uma promisse em execução: http://bevacqua.github.io/promisees/

O que é um Observable?

Contrário do Subject, o Observable é unidirecional
Por definição é uma coleção que funciona de forma unidirecional, ou seja, ele emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disso podemos executar uma ação. Digamos que ele resolve o mesmo problema que a versão anterior do Angular havia resolvido com o $watch, porém sem usar força bruta. Enquanto no $watch verificamos todo nosso escopo por alterações após cada $digest cycle (o que tem um grande custo na performance), com Observable esta verificação não acontece, pois para cada evento é emitida uma notificação para nosso Observable e então tratamos os dados.

O que é o BehaviorSubject?

É meio complicado explicar o que é um Subject, mas um jeito muito prático é se você imaginar uma roda de conversa.

1.Imagine uma roda de conversa entre amigos.
2.Todos que estão na roda estão “inscritos” no Subject, portanto, recebem as informações que este Subject tem.
3.Quando alguém fala: “Subject, o próximo assunto é Comida”, o Subject recebe a alteração e envia para todo mundo qual é o próximo assunto.
4.Agora, todos sabem que o conteúdo do Subject é “Comida”
5.Quando alguém novo chega e se inscreve no Subject, ele automaticamente recebe o valor “Comida”
6.Alguém novamente diz: “Subject, o próximo assunto é Protecionismo Florestal”.
7.Agora, todos, inclusive o recém chegado, recebem que o novo conteúdo do Subject é “Protecionismo Florestal”.