CircuitBreaker e Microinvader

By Glauco Reis (gsreis@br.ibm.com.br) WW Competitive Migration Team

Quando falamos de microserviços, estamos discutindo mais do que simplesmente uma fragmentação dos tradicionais WebServices e um maior isolamento utilizando containers. Existe um conjunto de novos paradigmas e práticas de programação que devem ser aplicados, caso contrário os problemas do passado irão retornar amplificados.

Uma destas quebras de paradigma está relacionada à disponibilidade dos microserviços.

Em qualquer arquitetura distribuída, diversos componentes executarão em vários servidores distintos, e se comunicarão com alguma frequência. É impossível garantir que todos serviços estarão em execução o tempo todo. Esta é consequência de uma falácia de computação distribuída - https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing.

Uma das características desejáveis de um microserviço é a capacidade de se adequar, mesmo na falha de algum componente de que ele necessite, e ainda se possível deve tentar se recuperar e continuar a execução da melhor forma possível.

Quando um MicroServiço não está em execução, ou porque está em manutenção ou porque alguma latência da rede causou sua queda, outros Microserviços que tentarem acessá-lo irão falhar também. Isto é esperado. O problema é que existe um timeout para que o chamador descubra que isto ocorreu. Como os MicroServiços são baseados usualmente em REST e HTTP, que são requisições síncronas, o restante do código somente continuará após o período de timeout ser identificado. Durante este período de espera a thread em execução ficará em estado de espera e isto gera uma lentidão que pode refletir em todo o sistema que acessa este Microserviço indisponível.

No passado, utilizávamos o conceito do “tudo ou nada”, ou seja, se um serviço não estivesse disponível, o melhor seria o restante do sistema travar completamente e tomar medidas drásticas. Hoje se tem um conceito de que devemos executar da melhor forma possível, mesmo em caso de falha. Erros devem ser tratados sem que o usuário tenha a percepção de que algo não está normal, e de preferência fora do fluxo normal de navegação.

Isto pode ser exemplificado da seguinte forma: Imagine um cliente que entra em uma loja virtual, apenas para consultar produtos. Não terá a intenção de comprar o produto naquele momento, entretanto deseja colocar produtos no carrinho de compra para exercitar os custos finais de envio. O sistema foi projetado de forma a reservar no estoque o item quando é colocado no carrinho de compras, utilizando o mesmo serviço que será posteriormente executado na efetivação da compra. Neste caso, diversas chamadas ao serviço de reserva podem ser feitas, mas como ele está fora do ar irá acarretar em timeouts que o usuário poderá até perceber durante a navegação. Mesmo neste cenário onde não haverá uma compra, o usuário terá uma percepção negativa do site, porque todo o site estará lento.

Mas supondo que um serviço necessário no momento está fora do ar, e entendendo que as chamadas REST serão síncronas, como podemos tornar todo o processo mais agradável sob a ótica do usuário?

Aqui entra um padrão de desenvolvimento Interessante, aparentemente apresentado pela primeira vez no excelente livro “Release It!” de Michael Niggard.

Ele se chama Circuit Breaker, que em livre tradução seria um disjuntor ou interruptor de circuito, como os disponíveis nas caixas de energia centrais das residências. Basicamente, quando uma

corrente muito alta passa pelos fios (talvez devido a um curto circuito) o disjuntor desliga, mantendo outros cômodos da casa com energia.
Para fazer o paralelo com o Microserviço, como não há como medir a corrente, podemos efetuar contagens de falhas. Por exemplo, se um microserviço for chamado um certo número de vezes e falhar em todas elas, provavelmente ele está com problemas e não deveria mais ser chamado por algum tempo. Neste caso um componente de software "desliga" as próximas requisições, retornando imediatamente quando o serviço for chamado.
Claro que neste caso providências devem ser feitas, como talvez enviar notificações a algum administrador ou painel de controle que permita uma tomada de ações. Mas o fato é que após o disjuntor desligar, todo o sistema irá continuar com a mesma velocidade, sem timeouts causados por latência na resposta, e a percepção geral do usuário pode ser melhor.
Uma questão que surge com o uso deste padrão é como "religar" o disjuntor? Como estamos lidando com sistemas que podem estar sendo utilizados por milhares de usuários, uma ação manual está fora de cogitação.
O ideal é que após um certo tempo o circuito se conecte novamente, podendo desligar novamente no futuro caso outro problema ocorra. Isto acelera o sistema em casos de falhas, e não exige atuação manual para que o sistema volte a funcionar novamente.
No primeiro artigo sobre o MicroInvader, comentei que alguns padrões de desenvolvimento seriam mais facilmente visualizáveis com um jogo dinâmico e visual, e este é um dos padrões de desenvolvimento que podem ser visualizados.

A implementação de um CircuitBreaker pode ser feita através de um Adapter (GOF95), que envelopa as requisições e toma as atitudes necessárias.
As chamadas a outros MicroServiços sem CircuitBreaker foram feitas com o código (veja no artigo anterior como baixar do github os códigos):

```
private String  callRest(String urlin) {
        try {
            Client client = ClientBuilder.newClient();
            return client.target(urlin).request(MediaType.APPLICATION_JSON).get(String.class);
        }catch(Exception e) {}
        return "[]";
}
```

Podemos envelopar as requisições e definir dois parâmetros em uma classe para definir o número de tentativas antes de desligar, e o tempo até que seja religado novamente.
Estes parâmetros podem ser chamados por quem for utilizar o microserviço.

```
// restante da classe CircuitBreaker foi removido para reduzir espaço
public String callRest() {

               try {
                  if (!breaked) {
                       return client.target(urlin).
                        request(MediaType.APPLICATION_JSON).get(String.class);
                   }
                   else {
```

```
                        if (System.currentTimeMillis() - started_timeout >= timeout) {
                                breaked = false;
                                tries_count = 0;
                        }
                    }
                    }catch(Exception e) {
                        tries_count++;
                        if (tries_count >= tries) {
                           tries_count = 0;
                           breaked = true;
                           started_timeout = System.currentTimeMillis();
                    }
                  }
                  return "[]";
        }

}
```

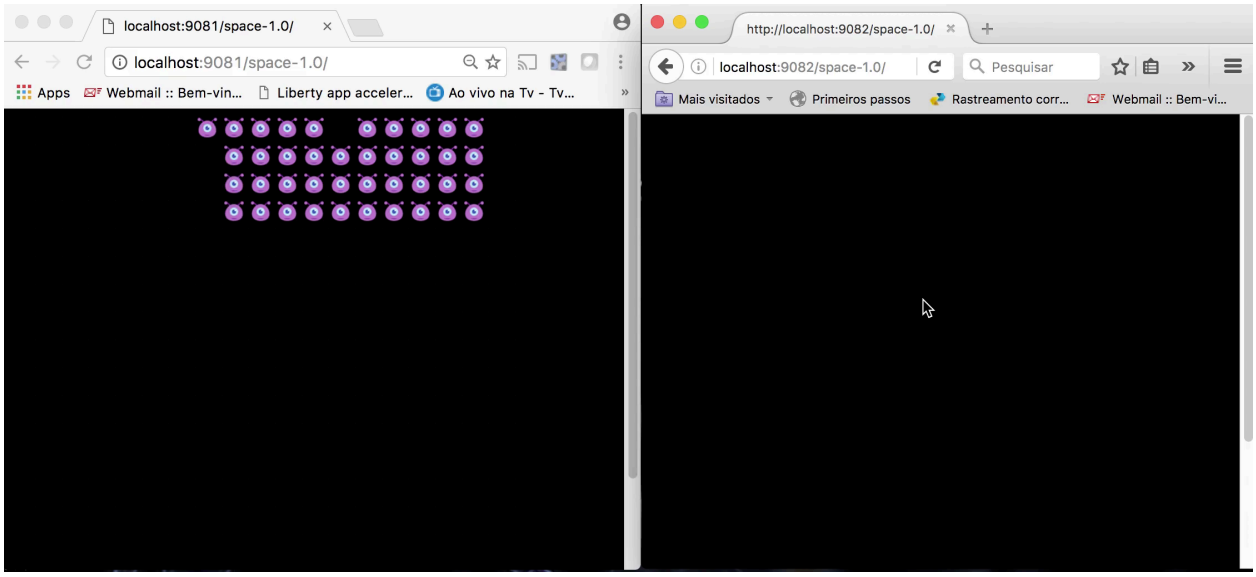
Com esta alteração, mesmo que um dos serviços seja removido, o tempo de resposta será na média menor porque quando o CircuitBreaker desligar o retorno será imediato.
Talvez a melhor implementação para um CB seja utilizando anotações Java, e muitos grupos de discussão sobre microserviços tem proposto esta forma de implementação.
Nesta implementação aplicada ao MicroInvader, foi criada uma coleção de CircuitBreakers, como uma caixa central de luz, uma para cada URL sendo chamada. Cada nova URL chamada irá gerar uma nova instância de CB, que irá conectar e desconectar em tempos distintos.

```
public class CollectionCircuitBreakers {

        private Hashtable<String, CircuitBreaker> vector = new Hashtable<String,
CircuitBreaker>();
        private int tries;
        private long timeout;
        public CollectionCircuitBreakers(int tries, long timeout) {
                this.tries = tries;
                this.timeout = timeout;
        }

        public String callRest(String urlrest) {
                CircuitBreaker cb = vector.get(urlrest);
                if (cb == null)
                {
                         cb = new CircuitBreaker(tries, timeout, urlrest);
                        vector.put(urlrest, cb);
                }
                return cb.callRest();
        }
}
```

O vídeo abaixo tem o mesmo código sem CB do lado esquerdo e com CB do lado direito. Ele tenta por 3 vezes se comunicar, e fica inativo por um período de 10 segundos para aquele serviço.



Para executar este código, basta baixar o arquivo microinvader_cb.zip. Dentro dele existem dois diretórios, um microinvader e outro microinvadercb. Basta chamar “mvn install” em cada um deles, e dois servidores serão criados (um na porta 9081 e outro na porta 9082). Pode-se iniciar cada um dos servidores em **/target/wlp/bin/server start Microinvader**, e depois abrir dois browsers, cada um apontando para uma URL distinta (http://localhost:9081/space-1.0 e http://localhost:9082/space-1.0).
Como no projeto anterior, todos os códigos estão disponíveis dentro do ZIP.