Circuit Breaker

Circuit Breaker é um conceito vital em arquiteturas de microsserviços e sistemas distribuídos, desempenhando um papel crucial na manutenção da resiliência e estabilidade das aplicações. Inspirado no funcionamento de um disjuntor elétrico, o Circuit Breaker interrompe automaticamente chamadas repetitivas a serviços que estão falhando, prevenindo a sobrecarga e a propagação de erros. Este padrão de design é especialmente útil em ambientes onde diferentes serviços dependem uns dos outros, garantindo que uma falha em um serviço não cause um efeito cascata prejudicando toda a aplicação.

A implementação eficaz de um Circuit Breaker envolve a configuração de estados, como aberto, fechado e meio-aberto, e o uso de práticas recomendadas, como monitoramento em métricas, configuração adequada de timeouts, definição de limiares e implementação de fallbacks resilientes. Com essas técnicas, é possível detectar e reagir a falhas de maneira eficiente, permitindo que o sistema se adapte dinamicamente às condições dos serviços monitorados. Nesta aula, exploraremos em profundidade o que é um Circuit Breaker, seu funcionamento interno, melhores práticas, casos de uso real e a implementação prática, proporcionando uma compreensão completa deste importante padrão de design.

Introdução ao Circuit Breaker

Circuit Breaker é um conceito essencial na tecnologia, especialmente em arquiteturas distribuídas e microsserviços. Ele funciona como um “disjuntor” em sistemas, interrompendo automaticamente a comunicação entre serviços quando detecta falhas, prevenindo assim a propagação de erros e a sobrecarga de serviços. Similar ao disjuntor elétrico que desliga um

circuito para evitar danos maiores, o Circuit Breaker impede chamadas repetitivas a serviços que estão falhando, garantindo a robustez e a tolerância a falhas do sistema.

O principal objetivo do Circuit Breaker é evitar que uma aplicação faça chamadas repetitivas a um serviço que pode estar fora do ar ou com problemas, melhorando assim a resiliência da arquitetura. Em um ambiente de microsserviços, onde diferentes serviços dependem uns dos outros, a falha de um único serviço pode ter um efeito cascata, afetando vários outros serviços. O Circuit Breaker ajuda a mitigar esse risco, bloqueando as chamadas para o serviço com falha e permitindo que os outros serviços continuem operando normalmente.

Para ilustrar o conceito, imagine um projeto em Java com diversos serviços interconectados. Sem o Circuit Breaker, uma falha em um serviço de autenticação, por exemplo, poderia resultar em inúmeras tentativas de conexão, sobrecarregando ainda mais o serviço problemático e causando falhas em outros serviços dependentes. Com o Circuit Breaker implementado, após um número predefinido de falhas ou um tempo de resposta excessivo, o circuito se "abre", interrompendo novas tentativas de conexão até que o serviço se recupere, evitando assim a propagação de falhas e permitindo uma recuperação mais eficiente.

Funcionamento Interno do Circuit Breaker

Internamente, um Circuit Breaker possui uma estrutura baseada em estados e transições de estados. Ele pode estar em um dos seguintes estados: fechado, aberto ou meio-aberto. Quando está fechado, as chamadas são permitidas normalmente. Se um número excessivo de falhas for detectado, o circuito se abre, bloqueando todas as novas chamadas para o serviço problemático. Após um período de tempo, o circuito entra em um estado meio-aberto, permitindo algumas chamadas de teste para verificar se o serviço já se recuperou.

A transição entre esses estados é controlada por critérios predefinidos, como o número de falhas consecutivas, o tempo de resposta (timeout) ou a taxa de falhas. Por exemplo, se um serviço de autenticação não responder dentro de 4 segundos, o Circuit Breaker pode abrir o circuito após três tentativas falhas consecutivas. Essas configurações são críticas para o funcionamento eficiente do Circuit Breaker, garantindo que ele não abra o circuito desnecessariamente, mas também que ele reaja rapidamente a problemas reais.

Além das transições de estado, o Circuit Breaker deve implementar um mecanismo de fallback, ou seja, uma resposta alternativa para as chamadas que não puderem ser completadas devido ao circuito estar aberto. Esse fallback pode ser uma mensagem de erro amigável ao usuário, uma resposta em cache ou uma funcionalidade reduzida que permita ao sistema continuar operando de forma degradada até que o serviço original se recupere. Esse comportamento ajuda a melhorar a experiência do usuário e a resiliência do sistema.

Melhores Práticas e Caso de Uso Real

Implementar um Circuit Breaker de maneira eficaz envolve seguir algumas melhores práticas essenciais. Entre elas, destacam-se o monitoramento em métricas, a configuração adequada de timeouts, a definição de limiares (thresholds) e a implementação de fallbacks resilientes. O monitoramento em métricas é crucial para observar o comportamento do Circuit Breaker, como a frequência de transições de estado e a eficácia dos fallbacks. Ferramentas de monitoramento ajudam a identificar padrões e ajustar as configurações do Circuit Breaker para otimizar seu desempenho.

A configuração de timeouts adequados é outra prática fundamental. Um timeout muito curto pode resultar em falsos positivos, abrindo o circuito desnecessariamente, enquanto um timeout muito longo pode atrasar a detecção de falhas reais. Portanto, é essencial encontrar um equilíbrio,

configurando o timeout com base na média de tempo de resposta esperada dos serviços monitorados.

A definição de limiares envolve configurar quantas falhas consecutivas ou qual taxa de falhas é aceitável antes de abrir o circuito. Esses limiares devem ser baseados no comportamento esperado do sistema e ajustados conforme necessário para evitar interrupções desnecessárias. Um fallback resiliente, por outro lado, garante que o sistema responda de forma adequada durante as falhas, informando os usuários sobre a indisponibilidade temporária e sugerindo ações alternativas.

Um exemplo real de uso do Circuit Breaker pode ser visto em sistemas de comércio eletrônico. Se um serviço de pagamento estiver inoperante, o Circuit Breaker pode abrir o circuito, evitando que novas tentativas de pagamento sobrecarreguem o serviço falho. Em vez disso, os usuários podem receber uma mensagem informando sobre a indisponibilidade temporária do serviço e sugerindo tentar novamente mais tarde. Isso não só protege o serviço de pagamento de sobrecarga, mas também melhora a experiência do usuário, evitando frustrações.

Detecção de Falhas e Ativação do Circuit Breaker

A detecção de falhas é um aspecto crucial do funcionamento de um Circuit Breaker. Essa detecção pode ser baseada em vários critérios, como timeouts, erros e exceções, número de falhas consecutivas e rate limiting. O timeout é uma forma simples e comum de detectar falhas, onde se uma chamada de serviço exceder um tempo de espera aceitável, é considerada uma falha. Já erros e exceções monitoram o comportamento do sistema, identificando problemas específicos que podem exigir a abertura do circuito.

O número de falhas consecutivas é outra métrica importante. Após um número predefinido de falhas seguidas, o Circuit Breaker pode abrir o circuito para impedir novas tentativas de conexão que poderiam

sobrecarregar ainda mais o serviço problemático. O rate limiting, por sua vez, envolve a limitação da taxa de chamadas que podem falhar antes de o circuito ser aberto. Por exemplo, se mais de 50% das chamadas falharem em um determinado período, o Circuit Breaker é ativado para prevenir falhas adicionais.

A ativação do Circuit Breaker segue um processo claro: ao detectar falhas conforme os critérios estabelecidos, ele transita para o estado aberto, bloqueando novas chamadas. Durante este período, ele pode retornar um fallback previamente definido. Após um tempo de abertura, o Circuit Breaker testa o serviço novamente para verificar se ele se recuperou, transitando para o estado meio-aberto. Se os testes forem bem-sucedidos, o circuito se fecha novamente, permitindo chamadas normais. Esse ciclo de abertura e fechamento garante que o sistema se adapte dinamicamente às condições do serviço, mantendo a resiliência e a estabilidade.

Implementação de Circuit Breaker

Para implementar um Circuit Breaker em um projeto, é essencial seguir alguns passos básicos. Primeiro, é necessário adicionar a dependência correta no arquivo de configuração do projeto, como o POM.xml em projetos Spring Boot. No caso do Spring Boot, a dependência do Resilience4J é uma escolha comum. Depois de adicionar a dependência, é preciso configurar as propriedades da aplicação, especificando os parâmetros do Circuit Breaker, como o timeout, o limiar de falhas e as janelas deslizantes.

Após configurar as propriedades, deve-se criar um bean de configuração no projeto, que define o comportamento do Circuit Breaker. Esse bean utiliza um builder para estabelecer as configurações, como o tempo limite e as regras de fallback. Em seguida, é necessário identificar os pontos críticos na aplicação onde o Circuit Breaker será implementado, geralmente em

métodos que fazem chamadas a serviços externos ou operações que podem falhar.

Finalmente, é importante anotar esses métodos com a anotação específica do Circuit Breaker, garantindo que ele seja aplicado corretamente. Durante o teste da aplicação, é fundamental verificar se o Circuit Breaker está funcionando conforme o esperado, acionando em caso de falhas e retornando os fallbacks apropriados. Esse processo garante que o Circuit Breaker seja integrado de maneira eficaz ao sistema, melhorando a resiliência e a capacidade de recuperação diante de falhas.

GitHub da Disciplina

Você pode acessar o repositório da disciplina no GitHub a partir do seguinte link:

https://github.com/FaculdadeDescomplica/Framework. Esse espaço é o seu portal para mergulhar fundo no universo da aprendizagem interativa. Nele, você encontrará todos os códigos, além dos links para os arquivos e dados.

Conteúdo Bônus

Título: Design pattern para microservices — Circuit Breaker

Autor: Evandro F. Souza

Plataforma: Medium

Este artigo, publicado no Medium pelo Training Center, fornece uma excelente introdução ao padrão de design Circuit Breaker aplicado em microservices. Ele explica de forma clara como implementar este padrão

para prevenir falhas em cascata em sistemas distribuídos, garantindo maior resiliência e estabilidade aos serviços.

Referência Bibliográfica

CARDOSO, L. da C. **Design de aplicativos**. Intersaberes: 2022

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7.ed. Pearson: 2018.

JOÃO, B. do N. **Usabilidade e interface homem-máquina**. Pearson: 2017

LEAL, G. C. L. **Linguagem, programação e banco de dados**: guia prático de aprendizagem. Intersaberes: 2015.

MEDEIROS, L. F. de. **Banco de dados**: princípios e prática. Intersaberes: 2013.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. Pearson: 2013.

SETZER, V. W.; SILVA, F. S. C. **Bancos de dados**. Blucher: 2005.

VICCI, C. (Org.). **Banco de dados**. Pearson: 2014.

Ir para exercício