palavra-chave synchronized em Java

Documents

A palavra-chave `synchronized` em Java controla o acesso a recursos compartilhados entre vários threads. Ele garante que somente um thread possa executar um bloco ou método sincronizado por vez, evitando a interferência de threads e garantindo a consistência da memória.

Uso

A palavra-chave `synchronized` pode ser aplicada a métodos ou blocos dentro de métodos. Quando um método ou bloco é declarado como sincronizado, um bloqueio é obtido no objeto especificado, e outros threads são impedidos de acessar o código sincronizado até que o bloqueio seja liberado.

Sintaxe

Método sincronizado

```
public synchronized void synchronizedMethod() {
    // method code
}
```

Bloco sincronizado

```
public void method() {
    synchronized (object) {
        // synchronized code
    }
}
```


- `object` : O objeto no qual você deseja sincronizar. Normalmente, essa é a instância atual (`this`) ou um objeto específico.

Exemplos

Exemplo 1: Método sincronizado

```
public class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }

    public static void main(String[] args) {
        Counter counter = new Counter();

        // Create multiple threads to increment the counter
        Thread t1 = new Thread(() -> counter.increment());
        Thread t2 = new Thread(() -> counter.increment());

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
```

```
        }

        System.out.println("Final count: " + counter.getCount());
    }
}
```

POWERED BY datalab

Neste exemplo, o método `increment` é sincronizado, garantindo que somente um thread possa incrementar o contador por vez, evitando condições de corrida.

Exemplo 2: Bloco sincronizado

```
public class SynchronizedBlockExample {
    private final Object lock = new Object();
    private int count = 0;

    public void increment() {
        synchronized (lock) {
            count++;
        }
    }

    public int getCount() {
        return count;
    }

    public static void main(String[] args) {
        SynchronizedBlockExample example = new SynchronizedBlockExample();

        // Create multiple threads to increment the counter
        Thread t1 = new Thread(() -> example.increment());
        Thread t2 = new Thread(() -> example.increment());

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Final count: " + example.getCount());
    }
}
```

Este exemplo usa um bloco sincronizado para controlar o acesso à variável `count`. O objeto de bloqueio garante que somente um thread possa executar o bloco sincronizado por vez.

Dicas e práticas recomendadas

- **Minimizar o código sincronizado**: Mantenha os blocos sincronizados tão curtos quanto possível para reduzir a contenção e melhorar o desempenho.
- **Use Final Objects**: Ao usar blocos sincronizados, use objetos finais como bloqueios para garantir a consistência.
- **Evite a sincronização aninhada**: Os blocos sincronizados aninhados podem levar a deadlocks. Evite-os ou use-os com cautela.
- **Use coleções simultâneas**: Para uma sincronização complexa, considere o uso de coleções simultâneas do pacote `java.util.concurrent`, como `ConcurrentHashMap`, que oferece sincronização integrada.
- **Volatile Keyword**: Para variáveis que são acessadas por vários threads, mas que não exigem sincronização complexa, considere usar a palavra-chave `volatile` para garantir a visibilidade.

```
private volatile boolean flag;
```


- **Segurança da linha**: Certifique-se sempre de que os recursos compartilhados estejam devidamente sincronizados para manter a segurança do thread e evitar a corrupção de dados.
- **Métodos estáticos sincronizados**: Sincronize métodos estáticos para impor o acesso sincronizado no nível da classe, garantindo que apenas um thread possa executar o método estático em todas as instâncias.

```
public static synchronized void staticMethod() {
    // method code
}
```

- **Reentrada:** Os bloqueios intrínsecos do Java são reentrantes, o que significa que, se um thread mantiver um bloqueio, ele poderá adquiri-lo novamente sem entrar em um deadlock.

SQL Cursos

Power BI Cursos

Tableau Cursos

Alteryx Cursos

Azure Cursos

AWS Cursos

Google Sheets Cursos

Excel Cursos

IA Cursos

Análise de dados Cursos

Visualização de dados Cursos

Machine learning Cursos

Engenharia de dados Cursos

Probabilidade e estatística Cursos

DATALAB

Comecar

Preços

Segurança

Documentacao

CERTIFICACAO

Certificacoes

Cientista de dados

Analista de dados

Engenheiro de dados

EMPRESAS

Preços de negócios

Teams Plan

Data & AI Plano Ilimitado

Histórias

Programa de Parceiros

SOBRE

Quem somos

Histórias de alunos

Carreiras

Torne-se um instrutor

Imprensa

Liderança

Fale Conosco

DataCamp English

DataCamp Español

DataCamp Deutsch

DataCamp Français

SUPORTE

Central de Ajuda

Torne-se um Afiliado