

Создание и управление потоками

`java.lang.Thread`

Александр Кораблин

- Плюсы и минусы многопоточного программирования



The Thread Class

- **static Thread currentThread()** Returns a reference to a Thread object that represents the invoking thread.
- **long getID()** Returns a thread's ID.
- **final String getName()** Obtains a thread's name.
- **final int getPriority()** Obtains a thread's priority.
- **Thread.State getState()** Returns the current state of the thread.
- **static boolean holdsLock(Object o)** *Returns true if the invoking thread holds the lock on obj.*
- **void interrupt()** Interrupts a thread.
- **static boolean interrupted()** Returns true if the invoking thread has been interrupted.
- **final boolean isAlive()** Determines whether a thread is still running.
- **final boolean isDaemon()** Returns true if the invoking thread is a daemon thread.
- **boolean isInterrupted()** Returns true if the thread on which it is called has been interrupted.
- **final void join()** Waits for a thread to terminate.
- **void run()** Entry point for the thread.
- **final void setDaemon(boolean how)** *If how is true, the invoking thread is set to daemon status.*
- **final void setName(String thrdName)** *Sets a thread's name to thrdName.*
- **final void setPriority(int level)** *Sets a thread's priority to level.*
- **static void sleep(long milliseconds)** *Suspends a thread for a specified period of milliseconds.*
- **void start()** Starts a thread by calling its run() method.
- **static void yield()** Yields the CPU to another thread.

Создание потока

```
class MyThread1 extends Thread {  
    MyThread1() {  
        super("name");  
        ..... }  
    public void run() {  
        System.out.println("starting....");  
        try {  
            ..... }  
        catch (InterruptedException exc) {  
            System.out.println("interrupted....");  
        } }  
}
```

```
Class Demo {  
    public static void main(String args[]) {  
        System.out.println("Main thread starting....");  
        MyThread1 thread = new MyThread1();  
        thread.start();  
        .....  
    }  
}
```


Создание потока (второй способ)

```
class MyThread2 implements Runnable {  
    MyThread2() {  
        // new Thread(this, "name") . start();  
        ..... }  
    public void run() {  
        System.out.println("starting....");  
        try { ..... }  
        catch (InterruptedException exc) {  
            System.out.println("interrupted....");  
        } }  
}
```

```
Class Demo {  
    public static void main(String args[]) {  
        System.out.println("Main thread starting.....");  
        Thread thread= new Thread(new MyThread2());  
        thread.start();  
        .....  
    }  
}
```

Анонимный класс

```
new Thread()  
{  
    public void run() {  
        System.out.println("starting....");  
        try { ..... }  
        catch (InterruptedException exc) {  
            System.out.println("interrupted....");  
        }  
    }  
}.start();  
// доступ только к полям "final"
```

Ожидание завершения потока

```
Class Demo {  
public static void main(String args[]) {  
    System.out.println("Main thread starting.....");  
    Thread thread= new Thread(new MyThread2());  
    thread.start();  
  
    .....  
    try {  
        thread.join();           // ждём – нет загрузки CPU  
        // thread.join(1000);  
        // while(thread.isAlive()) { ..... }      // загружаем CPU работой  
    }  
    catch (InterruptedException ex) {  
        System.out.println("main interrupted.....");  
    }  
}
```

смотреть **Demo1**

Завершение потока

- return
- Daemon thread

```
Thread thread= new Thread(new MyThread2());  
thread.setDaemon(true);  
thread.start();
```

.....

- suspend(), resume(), stop() - deprecated
- interrupt()

Interrupt a Thread

- Пример 1.
- создать класс «MyThread» реализ. инт. Runnable
- переопределить метод - run()
- в ЭТОМ методе :
 - получить имя потока
 - реализовать цикл с продолжительной работой
 - на каждой итерации проверять состояние потока
 - если поток прерван, то завершить цикл
- создать класс "Demo"
- реализовать метод "main"

Наблюдение за состоянием потока getState()

BLOCKED - The thread is blocked, which means that it is waiting for access to a synchronized code.

NEW - The thread has been created, but its start() method has not yet been called.

RUNNABLE - The thread is either currently executing or will execute as soon as it gets access to the CPU.

TERMINATED - The thread has ended. A thread ends when its run() method returns, or when the thread is stopped via a call to stop(). (Note that stop() is deprecated and should not be used.)

TIMED_WAITING - The thread is suspended, waiting on another thread for a specific period of time. This can occur because of a call to the timeout versions of sleep(), wait(), or join(), for example.

WAITING - The thread is suspended, waiting on another thread. This can occur because of a call to the non-timeout versions of wait() or join(), for example.

Локальная память потока

```
private static ThreadLocal<Integer> threadLocal =  
    new ThreadLocal<Integer>()  
{  
    protected Integer initialValue()  
    {  
        return new Integer(0);  
    }  
};
```

смотреть Demo2

Синхронизация потоков

```
synchronized type method(arg-list){  
    // synchronized method body  
}
```

```
synchronized(objref) {  
    // synchronized statements  
}
```

- делать пример 2

Взаимодействие потоков

```
class Test {  
    boolean ready = false;  
    synchronized void waitFor() {  
        try {  
            while(!ready) wait();  
        } catch (InterruptedException exc) {  
            System.out.println("Interrupted.....");  
        }  
    }  
  
    synchronized void goNow() {  
        ready = true;  
        notify();  
    }  
}
```

смотреть Demo3



Группы потоков

```
MyThread a = new MyThread();
```

```
MyThread b= new MyThread();
```

```
ThreadGroup gr = new ThreadGroup("name");
```

```
Thread t1= new Thread(gr, a, "Thread #1");
```

```
Thread t2= new Thread(gr, b, "Thread #2");
```

```
t1.start();
```

```
t2.start();
```

```
Thread threads[] = new Thread[gr.activeCount()];
```

```
gr.enumerate(threads);
```

```
for(Thread t : threads)
```

```
    System.out.println(t.getName());
```

```
gr.interrupt();
```

- что ещё нужно
 - читатель/писатель
 - код завершения потока
 - пул потоков
 - проверка доступности ресурса
- JDK 1.5
 - `java.util.concurrent.*;` смотреть Demo4
 - `java.util.concurrent.locks*;`

делать пример 3

```
ReadWriteLock lock = new ReentrantReadWriteLock();  
Lock rl = lock.readLock();  
Lock wl = lock.writeLock();
```

- `rl.lock(); rl.unlock();`
- `if (rl.tryLock()) { rl.unlock(); }`
- `if (rl.tryLock(5, TimeUnit.SECONDS)) {
rl.unlock(); }`