DEV-J130. Java SE. Разработка многоуровневых приложений. Введение в многопоточное программирование

© Ю.Д.Заковряшин, 2007-2022. E-mail: ydz@mail.ru

Вводные замечания

- Платформа Java разработана со встроенной поддержкой параллельного (многопоточного) программирования.
- Язык Java и стандартные библиотеки классов обеспечивают базовую поддержку параллелизма.
- Начиная с версии 5.0, платформа Java также включает высокоуровневые API-интерфейсы параллелизма.

Типичные задачи многопоточного приложения

- Разделение потоков ввода и потоков вывода для сокращения «простоев» программы.
- Отделение обработки команд пользовательского интерфейса от основной логики работы программы.
- Обеспечение асинхронной обработки команд.
- Выполнение фоновых и служебных задач.
- Разделение трудоёмких задач на подзадачи, которые могут выполняться параллельно.

Понятие о процессе

- Процесс – совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие (ISO 9000:2000).
- Программа по своей сути представляет собой последовательность пассивных инструкций, предназначенных для выполнения процессором компьютера, а процесс – достаточно изолированную среду, в которой выполняются эти инструкции.

Понятие о процессе

- Процесс в системе имеет:
  - отдельную среду выполнения;
  - автономный набор системных ресурсов, в частности отдельное адресное пространство;
  - может взаимодействовать с другими процессами посредством механизмов межпроцессного взаимодействия (IPC - Inter Process Communications).
- Виртуальная машина Java, как правило, работает как отдельный процесс.

Основные ресурсы процесса

- Образ исполняемого машинного кода программы.
- Изолированная область памяти, которая, как правило, разделяется между:
  - исполняемым кодом программы,
  - областью статических данных процесса,
  - стеками вызовов,
  - областью для временных данных (heap).
- Входные и выходные данные процесса.
- Дескрипторы ресурсов, которые ОС выделяет процессу.
- Набор полномочий/разрешений процесса.
- Контекст процессора, который включает в себя содержимое регистров и адреса объектов в памяти.

Многопоточность

- Многопото́чность (англ. Multithreading) — свойство платформы (операционной системы/ виртуальной машины/приложения), которое позволяет разделять процесс на отдельные потоки, выполняющиеся параллельно и, возможно, независимо друг от друга.
- Каждый процесс включает хотя бы один поток.
- Каждый поток принадлежит ровно одному и только одному процессу и не может существовать отдельно от процесса.

Понятие о потоке

- Поток – сегмент/часть процесса, который также представляет среду выполнения, но выполняется в адресном пространстве родительского процесса.
- Поток — это основная единица, которой операционная система выделяет время процессора. Поток имеет собственный программный счетчик, который отслеживает исполняемые инструкции, а также собственные системные регистры для обработки переменных и стек вызовов.

Основные преимущества потоков

- Поток требует меньше ресурсов, чем процесс.
- Переключение между потоками происходит без сложного взаимодействия с ОС.
- Потоки могут совместно использовать все данные и ресурсы процесса, например, открытые файлы и соединения с базами данных.
- Потоки могут легко взаимодействовать между собой.
- Потоки могут выполняться независимо друг от друга.

Определение потока в Java

- Явно определить отдельный поток можно двумя способами:
  - создать класс, наследующий интерфейс Runnable;
  - создать подкласс класса Thread с переопределением метода run().

Интерфейс Runnable

- Интерфейс java.lang.Runnable должны наследовать классы, назначением которых является определение того, что должно выполняться в отдельном потоке.
- Интерфейс определяет единственный метод
  - `void run()`,

с которого начинается выполнение потока, т.е. по своему назначению данный метод аналогичен методу main ().

Класс Thread

- Класс java.lang.Thread представляет в приложении отдельный поток.
- Для потока определены следующие основные свойства:
  - идентификатор потока, который присваивается потоку системой и доступен только на чтение;
  - имя потока (по умолчанию Thread??);
  - приоритет потока;
  - группа потока (по умолчанию main);
  - текущее состояние.

Класс Thread

- Методы класса позволяют:
  - получать информацию о текущих свойствах потока, который связан с данным объектом Thread;
  - изменять некоторые свойства потока;
  - управлять выполнением потока.
- При создании потока необходимо определить код метода run(), который описывает что должно выполняться в данном потоке.

Класс Thread

- Методы чтения и установки основных свойств потока:
  - `long getId();`
  - `final String getName();`
  - `final void setName(String name);`
  - `final int  getPriority();`
  - `final void setPriority(int newPriority);`
  - `final ThreadGroup getThreadGroup();`
  - `Thread.State getState();`
  - `final void checkAccess().`

Перечисление Thread.State

- Поток может находиться в одном из следующих состояний:
  - NEW,
  - RUNNABLE,
  - BLOCKED,
  - WAITING,
  - TIMED_WAITING,
  - TERMINATED.

Класс Thread

- Методы управления потоком:
  - `void run();`
  - `void interrupt();`
  - `final void join() throws InterruptedException;`
  - `final void join(long millis) throws InterruptedException;`
  - `final void join(long millis, int nanos) throws InterruptedException;`
  - `static void sleep(long millis) throws InterruptedException;`
  - `void start();`
  - `static void yield().`

Класс Thread

- Некоторые вспомогательные методы класса:
  - `static int activeCount();`
  - `static Thread currentThread();`
  - `final boolean isAlive().`

ThreadGroup

- Класс java.lang.ThreadGroup представляет группу потоков.
- Каждый поток относится к какой-то группе. По умолчанию все потоки относятся к группе «main».
- Класс предоставляет методы:
  - чтения/записи основных свойств группы;
  - управления потоками группы.

Класс ThreadGroup

- Основные методы класса:
  - `final String getName(),`
  - `final ThreadGroup getParent(),`
  - `final int getMaxPriority(),`
  - `final void setMaxPriority(int i),`
  - `final void checkAccess(),`
  - `int activeCount(),`
  - `final void interrupt().`

Синхронизация и блокировка потоков

- Ключевое слово synchronized используется в случаях, когда нескольким потокам необходимо выполнять один и тот же код, но в каждый конкретный момент времени его должен выполнять один и только один из потоков.
- Ключевое слово synchronized может использоваться в двух вариантах:
  - в заголовке метода – в этом случае синхронизируется код всего метода;
  - в заголовке специального блока синхронизации.

Пример использования ключевого слова synchronized

- Объявление синхронизированного метода:
  - `public synchronized String getName () {`
    `… }.`
- Определение синхронизированного блока кода:
  - `synchronized (lockObject) { … }`

Методы класса Object

- Методы класса java.lang.Object, используемые для синхронизации потоков:
  - `final native void wait() throws InterruptedException;`
  - `final native void wait(long timeout) throws InterruptedException;`
  - `final native void wait(long timeout, int nanos) throws InterruptedException;`
  - `final native void notify();`
  - `final native void notifyAll().`

Пример приложения

- Приложение Ping-Pong имитирует игру в настольный теннис.
- В приложении запускается три потока:
  - главный поток определяет общий сценарий выполнения программы;
  - каждый из двух других потоков имитирует одного игрока;
  - кроме этого в приложении используется вспомогательный класс, имитирующий теннисный мячик.
- Приложение должно демонстрировать следующее поведение:
  - игроки строго последовательно должны наносить «удар» (вызов специального метода этого класса) по «мячу»;
  - игра должна заканчиваться после строго определённого обмена ударами.

Заключение

- Обзор рассмотренных тем.
- Вопросы?