Sromek Mateusz

3 Is(s) P3

Wykonaj sprawozdanie, opisz wymienione poniżej funkcje/klasy/interfejsy paczki java.util.concurrency odpowiadające za zrównoleglenie, użyte w przykładach.

- interfejs Runnable – ma na celu zapewnienia wspólnego protokołu dla obiektów, które są aktywne i chcą wykonać kod. Oznacza to że wątek został uruchomiony i nie został jeszcze zatrzymany. . Klasa, która implementuje Runnable, może działać bez tworzenia podklas Thread poprzez utworzenie wystąpienia Thread i przekazanie go jako elementu docelowego. W większości przypadków interfejs Runnable powinien być używany tylko wtedy, gdy planuje się zastąpić metodę run (), a nie inne metody Thread.

- interfejs Callable<T> - jest podobny do interfejsu Runnable ponieważ są zaprojektowane dla klas, których wystąpienia sa potencjalnie wykonywane przez inny wątek z tą różnica że interfejs Runnable nie zwraca wyniku i nie może zgłosić sprawdzonego wyjątku, a interfejs Callable zwraca wynik i może zgłosić wyjątek.

- klasa Executor (w tym metoda newFixedThreadPool()) – tą metodą można uzyskać stałą pulę wątków. Wprowadzona dopuszczona pula maksymalna po wysłaniu większej ilości wątków przechowywane są w kolejce dopóki wątki nie staną się dostępne

- klasa ExecutorService (szczególnie metody shutdown(), shutdownNow() , awaitTermination(), isTerminated())

Shutdown() - metoda nie powoduje natychmiastowego zniszczenia ExecutorService. Spowoduje to, że ExecutorService przestanie akceptować nowe zadania i zostanie zamknięty po zakończeniu bieżącej pracy przez wszystkie uruchomione wątki

shutdownNow() metoda próbuje zniszczyć ExecutorService natychmiast, ale nie gwarantuje, że wszystkie uruchomione wątki zostaną zatrzymane w tym samym czasie

awaitTerminator() - połączenie metody shutdown() oraz shutdownNow() co powoduje najpierw przestanie wykonywania nowego zadania w ExecutorService, a następnie odczeka do określonego czasu na wykonanie wszystkich zadań. Jeśli ten czas wygaśnie, wykonanie zostanie natychmiast zatrzymane.

- klasa FutureTask<T> i jej metoda T – Ta klasa zapewnia podstawową implementację Future, z metodami do rozpoczynania i anulowania obliczeń, zapytań w celu sprawdzenia, czy obliczenia zostały zakończone, i pobrania wyników obliczeń. Wynik można pobrać dopiero po zakończeniu

obliczeń; metody zostaną zablokowane, jeśli obliczenia nie zostały jeszcze zakończone , metoda która może zwrócić każdy rodzaj obiektu

- metody Thread.sleep() - Thread.sleep powoduje, że bieżący wątek zawiesić wykonanie na określony czas. Jest to skuteczny sposób udostępniania czasu procesora innym wątkom aplikacji lub innym aplikacjom, które mogą być uruchomione w systemie komputerowym.

Thread.yield() - wskazuje, że wątek nie robi nic szczególnie ważnego i jeśli trzeba uruchomić jakiekolwiek inne wątki lub procesy, mogą. W przeciwnym razie bieżący wątek będzie nadal działać.

<<Thread Object>>.join() - służy do łączenia początku wykonywania wątku z końcem wykonywania innego wątku w taki sposób, że wątek nie zaczyna działać, dopóki nie zakończy się inny wątek

- funkcje System.currentTimeMillis() - zwraca bieżący czas w milisekundach. Jednostką czasu zwracanej wartości jest milisekunda, ziarnistość wartości zależy od podstawowego systemu operacyjnego i może być większa.

- czym różni się Catch(Exception e) od Catch(InterruptException e)

Catch(Exception e) – próbuje wyłapać błędy (błędy kodowania popełniane przez programistę, błędy spowodowane nieprawidłowym wprowadzaniem danych) podczas wykonywania kodu

Catch(InterruptException e) – wyłapuje wątek i go zatrzymuje