Asynchronous, Concurrent and Distributed Processing in Java EE

W przykładach z Oracle WebLogic Server, Oracle Coherence i Oracle TopLink Grid

Waldek Kot
waldemar.kot@oracle.com
Oracle Polska

ORACLE

Agenda

- Współbieżność - po co ?
- Dlaczego w Java EE jest to problem ?
- Rozwiązania
  - Co można użyć dzisiaj ?
    - Work Manager API
      - z przykładami w Oracle WebLogic Server
  - Co pojawi się w Java EE 6 ?
- Technologie Data Grid
  - Z przykładami w Oracle Coherence
  - Połączenie z Java Persistence API

Po co współbieżność ?

- Zwiększenie wydajności
- Skrócenie czasu tworzenia odpowiedzi systemu
  - Możliwość podziału większego zadania na mniejsze i ich równoległego wykonania
  - Lepsza „responsywność” aplikacji (szczególnie dla użytkowników)
- Prostsza obsługa operacji wykonywanych asynchronicznie i długotrwających
  - np. notyfikacje, batch processing, server maintenance
- Lepsze wykorzystanie dzisiejszego sprzętu
  - multi-server, multi-CPU, multi-core, multi-thread per core, …
- Java/JVM od początku oferowała tu dobre wsparcie

Dlaczego jest to trudne w J2EE/Java EE ?

- Specyfikacja EJB (21.1.2 – Programming Restrictions):
  - An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances.
    - Synchronization would not work if the EJB container distributed enterprise bean's instances across multiple JVMs.
  - The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread, or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups.
    - These functions are reserved for the EJB container. Allowing the enterprise bean to manage threads would decrease the container's ability to properly manage the runtime environment.
- Mit: komponenty Java EE (servlet/EJB) nie mogą samodzielnie zarządzać wątkami
  - W rzeczywistości: mogą ☺, choć **nie powinno** się tego robić z bezpośrednim użyciem mechanizmów Java SE (np. klasy *java.lang.Thread*)

Dlaczego ?

- Brak kontroli nad takimi wątkami ze strony kontenera (serwera aplikacyjnego)
- Brak kontekstów (metadanych) w takich wątkach
  - Konteksty: security, class loading, transaction, naming, locale, własnych konteksty (?), …
- W Java EE trzeba także zadbać o:
  - Skoordynowanie cyklu życia kontenera/komponentów i powoływanych przez nich wątków
  - Wysoki poziom izolacji aplikacji
- Tematy godne rozważenia w Java EE
  - Quality of Service
  - Overload protection
  - Rozdział zadań pomiędzy instancje serwerów aplikacyjnych
  - Wersjonowanie
- Fajnie, gdyby programista także miał nad tym kontrolę
  - Application-scoped

Jakie są więc opcje dla programisty Java EE?

- Bezpośrednie użycie wątków z Java SE (java.lang.Thread)
- Typowa odpowiedź: użyj JMS, systemu kolejkowego, itd.
  - ale JMS nie stworzono do tego celu. Nawet z JMS wychodzą nadmiernie skomplikowane rozwiązania
- Od 2003 (i na pewno jeszcze przez kilka lat ☺):
  - CommonJ Work Manager API
    - De facto standard
    - Implementowany przez liczące się serwery aplikacyjne
      - Oracle WebLogic, IBM WebSphere
- JCA (technicznie możliwe, ale nie polecam)
- Java EE 6 (?)
  - Concurrency Utilities for Java EE
    - JSR-236, JSR-237
    - możliwość skorzystania z dobrodziejstw Java SE (5, 6, …)
- Technologie Data Grid
  - przetwarzanie rozproszone „sterowane” danymi
  - np. Oracle Coherence

Work Manager API

- Kluczowe interfejsy:
  - Work, WorkItem, WorkManager, WorkEvent, WorkListener
  - Work rozszerza Runnable
- Operacja wykonywana przez WorkManager:
  - schedule (Work)
    - jest także opcja z listenerem
- Konstrukcje synchronizacyjna obiektów WorkItem
  - waitForAll
  - waitForAny
  - obie mają także wersje z timeout’em

Przykład implementacji Work Manager

- **Oracle WebLogic Server**
- Definicje work manager'ów są używane przez scheduler serwera
  - Dostęp do zasobów, priorytetyzacja, ochrona zasobów, self-tuning, itp.
  - Różne klasy przetwarzania (np. minimize response time)
  - Zaawansowana priorytetyzowana kolejka wykonawcza
- Work Manager to zasób kontenera
  - podobnie jak JMS quue, czy connection pool
  - może mieć przypisane polityki wykonania
    - QoS (Quality of Service)
    - np. min/max liczba wątków
      - nawet można zgrać z konfiguracją innych komponentów
        - np. JDBC connection pool
  - przypisuje się go do komponentów
    - aplikacji, komponentów aplikacji, nawet metod EJB
    - poprzez deployment descriptor
      - lub poprzez DI: adnotacja @Resource
- Aplikacje także mogą tworzyć swoje work manager'y
  - application-scoped
- Programista aplikacji ma dostęp do work manager'ów poprzez CommonJ Work Manager API
  - @Resource

Demonstracja użycia zasobów typu Work Manager i interfejsu programistycznego CommonJ Work Manager API w Oracle WebLogic Server

DEMO (1/3)

1. Prosta aplikacja webowa
   - 3 servlet'y (serial, parallel, custom)
   - servlet'y wyświetlają wyniki i mierzą czas wykonania
2. Pojedyncze zadanie do wykonania (klasa *MySubTaskImpl*)
   - parametr *delay* określa w milisekundach czas trwania zadania
     - najprostsza realizacja, czyli z wykorzystaniem *Thread.sleep()* ☺
   - parametr *input* zawiera dane do przetworzenia (tekst)
   - wynik wykonania zadania zamiana tekstu w *input* na duże litery ☺
   - możliwość przerwania wykonywanego zadania
     - wtedy zadanie zwraca wynik: „##cancelled##" (hardcoded ☺)

DEMO (2/3)

3. Wykonanie szeregowe
   - servlet **SerialServlet**
   - czas wykonania mniej więcej równy sumie czasów pojedynczych zadań
4. Wykonanie zrównoleglone
   1. servlet **ParallelServlet**
   2. czas wykonania równy czasowi wykonania najdłuższego z zadań
   3. utworzenie w serwerze aplikacyjnym zasobu work manager (**MyWorkManager**)
   4. wstrzelenie (DI) poprzez *@Resource* zasobu **MyWorkManager**
      - możliwe także zdefiniowanie poprzez deployment descriptor
   5. przekazywanie pracy do wykonania: metoda *submit*
      - dodatkowo wersja z listener'em
   6. konstrukcje synchronizacji zadań
      - *waitForAll* (w tym wersja z timeout'em)
      - *waitForAny* (w tym wersja z timeout'em)
      - wyświetlanie aktualnego statusu zadań
        - *ACCEPTED, STARTED, COMPLETED, ABORTED*
   7. przerywanie wykonywania zadań
   8. określanie polityk wykonania i Quality of Service dla zasobów Work Manager
      - określenie, że work manager **MyWorkManager** ma ograniczenie typu *Max Threads*
        - ustawienie tego ograniczenia na wartość 1, spowoduje, że ParallelServlet wykona się tak samo jak SerialServlet, czyli zadania będą wykonywane szeregowo

DEMO (3/3)

3. Work Manager definiowany przez aplikację (application-scoped)
   - zasób **MyWorkManager** jest globalny
     - tzn. może z niego skorzystać każda aplikacja uruchomiona na tym serwerze/klastrze
     - nie tylko aplikacja, ale także komponenty aplikacyjne (np.. servlet'y, obiekty EJB, metody EJB, itd.)
   - aplikacyjny („lokalny") work manager można zdefiniować poprzez specyficzne dla kontenera deployment descriptor'y
     - w WebLogic Server: np. *weblogic.xml*, *application-weblogic.xml*, …
   - servlet **CustomServlet** używa lokalnego work manager'a (**MyComponentLevelWM)**
     - logika taka sama, jak w *ParallelServlet*
       - Ale *@Resource* wstrzykuje (DI) **MyComponentLevelWM**
     - dla lokalnych work manager'ów także można określać ograniczenia (constraints) i klasy wykonania
       - jeśli **MyWorkManager** ma ustawioną maksymalną liczbę wątków np. na „20", a **MyComponentLevelWM** ma to ograniczenie na „1", to:
         - czas wykonania **ParallelServlet** równy czasowi najdłuższego zadania
         - czas wykonania **CustomServlet** równy sumie czasów poszczególnych zadań
4. Poprzez JMX (czyli m.in. w konsoli administracyjnej WebLogic Server, a także poprzez WebLogic Scripting Tool) można wyświetlić statystyki work manager'ów, zrobić notyfikacje, itp.

Kod demonstracji Work Managerów w Oracle WebLogic Server

- Będzie opublikowany na moim blogu:
  - http://jdn.pl/blog/88

Concurrency Utilities for Java EE

- Chris Johnson (IBM), Naresh Revanuru (BEA/Oracle), Doug Lea (m.in. JSR-166, java.util.concurrent), Cameron Purdy (Tangosol/Oracle) i inni
- Status: Early draft (04.2009)
  - Java EE 6 ???
- Możliwość użycia, w tym w rozszerzony sposób java.util.concurrent i innych mechanizmów Java SE 5+
  - javax.util.concurrent
- Kluczowe konstrukcje
  - ManagedThreadFactory
  - ManagedExecutorService
  - ManagedScheduledExecutorService
  - ContextService
  - Identyfikowalność poprzez interfejs Identifiable
  - Zarządzanie i monitorowanie poprzez JMX/MBeans

Concurrency Utilities for Java EE

- ManagedThreadFactory
  - Fabryka „wątków” (prac do wykonania)
  - Dzięki temu możliwe będzie skorzystanie z API wyższego poziomu (np. java.util.concurrent czy Spring framework)
- ContextService
  - Określa sposób przekazywania metadanych z wątku rodzica do wątku dziecka
  - Zarządza kontekstami
- ManagedExecutorService
  - Przypomina WorkManager z Work Manager API
  - Operacje: submit, invokeAll, invokeAny
  - Listeners
  - Zwykle komunikacja poprzez Future
  - Specyfikacja wspomina także o wersji Distributable
- ManagedScheduledExecutorService
  - Timer + Trigger

Data Grid

- Rozproszenie danych na wiele współpracujących ze sobą węzłów
  - Węzły tworzą klastry
- Często przeniesienie danych do pamięci
  - Dodatkowe wymagania na niezawodność i bezpieczeństwo tych danych
- Przetwarzanie równoległe „sterowane danymi”

Technologia: In-memory Data Grid

- **Oracle Coherence**
- coherence.jar
  - Zarówno klient, jak i serwer (węzeł) klastra
  - standalone, integracja z serwerami aplikacyjnymi, integracja z Spring, JPA/TopLink
- Buforowanie danych
  - jeden lub więcej nazwany bufor (NamedCache)
    - Przechowuje **obiekty** (para: klucz – obiekt)
    - Obiekty: Java, C++, .NET
  - Każdy może mieć indywidualną konfigurację odnośnie:
    - sposobu działania (np. jeśli jako cache – polityki wygasania danych: LRU, LFU)
    - implementacji (Java heap, NIO buffers, inne)
    - przechowywania danych (pamięć, baza danych, inny storage)
    - topologii (m.in. zarządzanie kopiami danych, near-cache)
  - Interfejs java.util.Map
    - Operacje: put/get, remove, clear, entrySet, values, containsKey, containsValue, …
- Rozproszone mechanizmy przetwarzania danych w buforach
  - zapytania, zdarzenia, monitorowanie zmian, modyfikacja danych in-place,
- Mocno rozszerzalny (bo to komponent middleware ☺)

Demonstracja technologii data grid z użyciem technologii Oracle Coherence

DEMO (1/4)

1. Jeden plik: **coherence.jar**
   - zarówno klient, jak i serwer
2. Uruchomienie serwera (instancji) Coherence: *\bin\cache-server.cmd*
3. Uruchomienie konsoli (shell) Coherence: *\bin\coherence.cmd*
   - Przydatne do podglądania stanu środowiska Coherence i poszczególnych buforów
     - Dostęp do konkretnego buforu: polecenie *cache* (tu: *cache jdd09*)
     - Aktualna liczba obiektów w buforze: polecenie *size*
     - Dostęp do konkretnego obiektu: polecenie *get*
     - Wstawienie obiektu: polecenie *put*
     - Czyszczenie buforu: polecenie *clear*
4. Prosty projekt Java – klient wstawiający i odczytujący obiekty do/z konkretnego buforu (tu bufor nazywa się *jdd09*)
   - Dodać coherence.jar do classpath
   - Fabryka dająca dostęp do konkretnego buforu: CacheFactory
   - Bufor implementuje interfejs java.util.Map
     - metody: put, get, size, …
5. Utworzyć drugi projekt Java
   - Demonstracja komunikacji z buforem z innego procesu Java

DEMO (2/4)

6. Można wstawiać dowolne obiekty Java (i nie tylko Java)
   - Klasa *Employee*
     - Atrybuty: *id, imię, nazwisko, płeć, wiek, adres, pensja*
   - Warto skorzystać z własnego, wydajnego mechanizmu serializacji obiektów w Coherence
     - Interfejs **ExternalizableLite**
     - Klasa *EmployeeExternalizable* (*implements ExternalizableLite*)
       - metody: *readExternal* i *writeExternal*
   - Prosty klient wstawiający 20 000 obiektów *EmployeeExternalizable*
7. Uruchomienie drugiego (i kolejnych) serwerów (instancji) Coherence
   - *\bin\cache-server.cmd*
   - Dane w buforach są rozpraszane między instancje Coherence
     - Zgodnie z konfiguracją danego buforu
       - Będą tworzone repliki (kopie) obiektów
       - Coherence dba o poprawną synchronizację tych kopii
8. Demonstracja replikacji danych
   - Zatrzymanie dowolnego serwera (oprócz ostatniego ☺) nie powoduje utraty danych (obiektów) w buforze

DEMO (3/4)

9. Demonstracja mechanizmów rozproszonego, równoległego przetwarzania
   - „sterowane danymi” – operacje są wykonywane przez wszystkie instancje Coherence zarządzające danymi buforami
     - Jeśli 100 instancji zarządza buforem, to każda z nich będzie wykonywać przetwarzanie
       - Kompletnie transparentne dla aplikacji klienckich
10. Rozproszone zapytania (distributed queries)
   - średni wiek kobiet, średni wiek mężczyzn, maksymalny wiek, suma zarobków
   - Demonstracja czasu wykonania zapytań, gdy działa 1, 2, 3 instancje Coherence
     - wyniki zależą od liczby maszyn/CPU/core
   - Indeksowanie
     - *addIndex*() dla atrybutu *imię*
     - demo: wpływ indeksów na czas wykonania zapytań (jest znaczny ☺)
   - Własne klucze
     - asocjacje i możliwość „sterowania” jak obiekty będą rozmieszczane w klastrze Coherence
     - demo: wpływ na czas wykonania zapytań

DEMO (4/4)

11. Rozproszone modyfikacje danych w buforze
    - Podwyżka pensji o 20% dla wszystkich pracowników
    - Przy użyciu **get / put**
    - Blokowanie obiektów: **lock / unlock**
    - Demo: czas wykonania dla 1, 2, 3 instancji serwerów Coherence
12. Przetwarzanie danych in-place
    - Czyli zamiast transferować obiekty które będą modyfikowane, wyślij logikę modyfikacji
    - Klasa *AbstractProcessor*
        - Logika modyfikacji w metodzie *process*
        - Demo: klasa **RaiseSalary** (*extends AbstractProcessor*)
    - Metoda **invoke** w *NamedCache*
        - Można podać filtr, czyli na których obiektach w buforze dana modyfikacja będzie wykonana
    - Przyspieszenie wykonywania modyfikacji jest radykalne ☺
13. Możliwe jest także skorzystanie z trigger'ów i monitorowania zmian

Kod demonstracji Oracle Coherence

- Będzie opublikowany na moim blogu:
  - http://jdn.pl/blog/88

Coherence i JPA

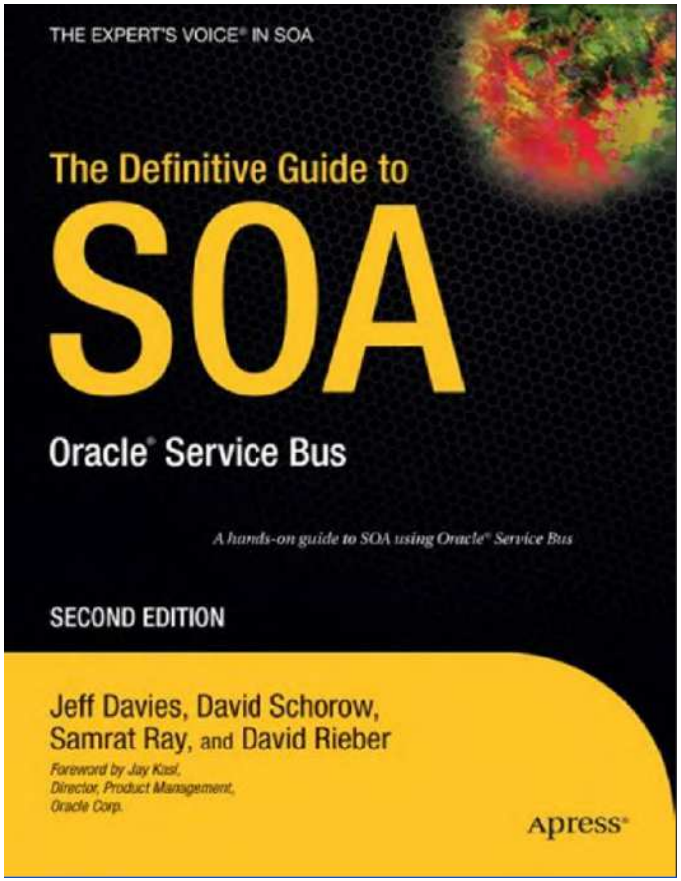
- **TopLink Grid**
  - JPA wzbogacone o rozproszone zarządzanie danymi
- Możliwe są 4 konfiguracje
  - Coherence korzysta z TopLink/JPA
    - Per bufor (NamedCache)
    - programista używa Coherence API
    - obiekty są utrwalane w bazie danych poprzez JPA
  - TopLink korzysta z Coherence
    - Per Entity
    - Wariant 1: Coherence jako rozproszony cache Level 2
    - Wariant 2: TopLink tylko odczytuje entity z buforów zarządzanych przez Coherence
      - zapis bezpośrednio do DB (wraz z uaktualnieniem buforu)
    - Wariant 3: TopLink odczytuje i zapisuje entity poprzez bufory Coherence
      - możliwość skorzystania z opcji opóźnionych, asynchronicznych zapisów zmian w entity do bazy danych

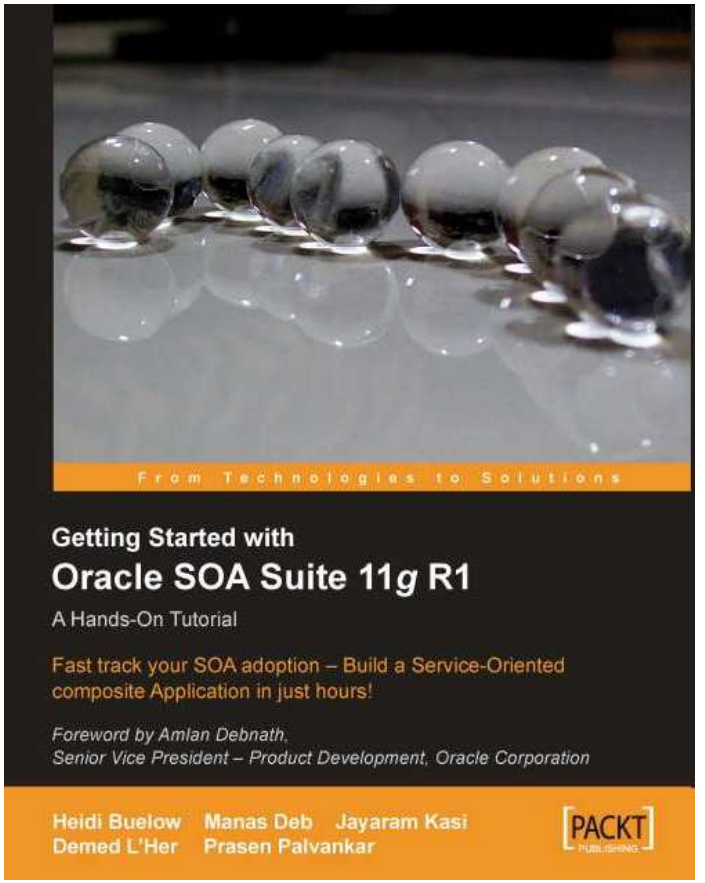
Podsumowanie

- Współbieżność - po co ?
- Dlaczego w Java EE jest to problem ?
- Rozwiązania
  - Co można użyć dzisiaj ?
    - Work Manager API
      - z przykładami w Oracle WebLogic Server
  - Co pojawi się w Java EE 6 ?
- Technologie Data Grid
  - Z przykładami w Oracle Coherence
  - Połączenie z Java Persistence API

Konkurs z nagrodami ☺

- **Do wygrania książki o technologiach Oracle !**
  - **Oracle Service Bus, Definitive Guide to SOA**
    - **http://apress.com/book/view/1430210575**
  - **Oracle SOA Suite 11g, Getting Started – A Hands-on Tutorial**
    - **http://www.packtpub.com/getting-started-with-oracle-soa-suite-11g-r1/book#**





Dziękuję ☺ !

Waldemar Kot
Principal Systems Engineer, Eastern Europe
Oracle Polska
Sienna 75, 00-833 Warsaw, Poland
Mobile: +48 660 78 55 78
Email: waldemar.kot@oracle.com

Private email: waldek.kot@googlemail.com
Blog: http://jdn.pl/blog/88