Java Cluster

implementacja,wydajność rozwiązania i zastosowanie

Piotr Pawłowski

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

czerwiec 2008

Założenia

- Efektywne narzędzie do obliczeń równoległych

Założenia

- Efektywne narzędzie do obliczeń równoległych
- Łatwe, znane każdemu programiście Javy API wbudowane w maszynę wirtualną

Założenia

- Efektywne narzędzie do obliczeń równoległych
- Łatwe, znane każdemu programiście Javy API wbudowane w maszynę wirtualną
- Zgodność ze specyfikacją

Założenia

- Efektywne narzędzie do obliczeń równoległych
- Łatwe, znane każdemu programiście Javy API wbudowane w maszynę wirtualną
- Zgodność ze specyfikacją
- Programista może się skupić na algorytmie

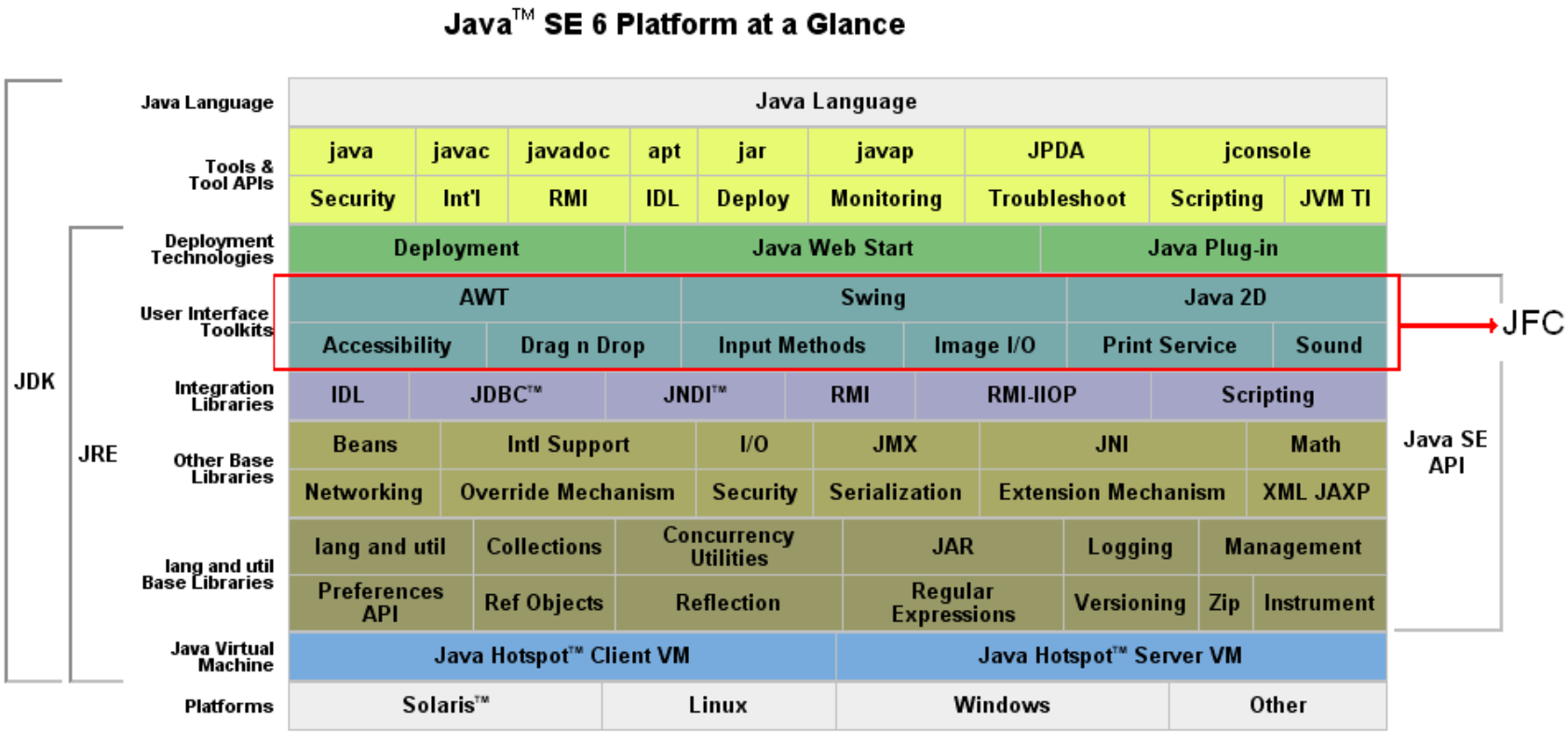
Założenia

- Efektywne narzędzie do obliczeń równoległych
- Łatwe, znane każdemu programiście Javy API wbudowane w maszynę wirtualną
- Zgodność ze specyfikacją
- Programista może się skupić na algorytmie
- Niezawodność

Założenia

- Efektywne narzędzie do obliczeń równoległych
- Łatwe, znane każdemu programiście Javy API wbudowane w maszynę wirtualną
- Zgodność ze specyfikacją
- Programista może się skupić na algorytmie
- Niezawodność
- Przenośność

Struktura Java SE



Koncepcja rozwiązania

- Wykorzystanie gotowych mechanizmów Java SE

Koncepcja rozwiązania

- Wykorzystanie gotowych mechanizmów Java SE
- Replikacja danych globalnych

Koncepcja rozwiązania

- Wykorzystanie gotowych mechanizmów Java SE
- Replikacja danych globalnych
- Synchronizacja jako podstawowy element wymiany danych

Koncepcja rozwiązania

- Wykorzystanie gotowych mechanizmów Java SE
- Replikacja danych globalnych
- Synchronizacja jako podstawowy element wymiany danych
- Klient jako punkt synchronizacyjny

Koncepcja rozwiązania

- Wykorzystanie gotowych mechanizmów Java SE
- Replikacja danych globalnych
- Synchronizacja jako podstawowy element wymiany danych
- Klient jako punkt synchronizacyjny
- Realizacja często wykonywanych operacji, wymagających wydajności na poziomie JVM

Punkt wyjścia

- Java SE 6, źródła dostępne na licencji JRL (Java Research License)

Punkt wyjścia

- Java SE 6, źródła dostępne na licencji JRL (Java Research License)
- Microsoft Visual Studio .NET 2003 jako kompilator oraz edytor kodu JVM

Punkt wyjścia

- Java SE 6, źródła dostępne na licencji JRL (Java Research License)
- Microsoft Visual Studio .NET 2003 jako kompilator oraz edytor kodu JVM
- cygwin jako środowisko narzędziowe

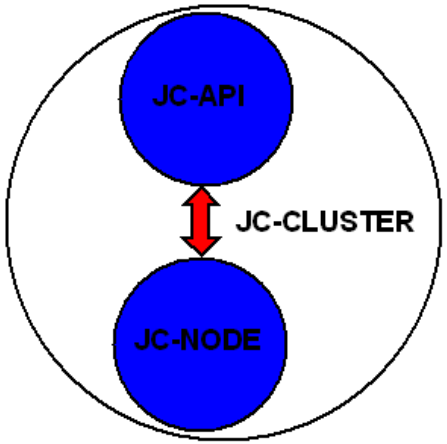
Punkt wyjścia

- Java SE 6, źródła dostępne na licencji JRL (Java Research License)
- Microsoft Visual Studio .NET 2003 jako kompilator oraz edytor kodu JVM
- cygwin jako środowisko narzędziowe
- Eclipse jako IDE dla części pisanej w Javie

Punkt wyjścia

- Java SE 6, źródła dostępne na licencji JRL (Java Research License)
- Microsoft Visual Studio .NET 2003 jako kompilator oraz edytor kodu JVM
- cygwin jako środowisko narzędziowe
- Eclipse jako IDE dla części pisanej w Javie
- Czas kopilacji około 4 godziny na komputerze średniej klasy

Elementy systemu



- Java Cluster API (JC-API)
- Java Cluster Node (JC-Node)

Założenia

- Wykorzystanie istniejących elementów języka, słów kluczowych

Założenia

- Wykorzystanie istniejących elementów języka, słów kluczowych
- Niewielka ilość klas dodatkowych z punktu widzenia programisty

Założenia

- Wykorzystanie istniejących elementów języka, słów kluczowych
- Niewielka ilość klas dodatkowych z punktu widzenia programisty
- Przezroczystość rozwiązania

Słowa kluczowe

- synchronized
- transient

Nowe klasy oraz modyfikacje JVM

- RemoteThread, klasa dziedzicząca po Thread

Nowe klasy oraz modyfikacje JVM

- RemoteThread, klasa dziedzicząca po Thread
- Thread, modyfikacja na poziomie JVM i Java SE

Nowe klasy oraz modyfikacje JVM

- RemoteThread, klasa dziedzicząca po Thread
- Thread, modyfikacja na poziomie JVM i Java SE
- Java Virtual Machine, podmiana metod, implementacja synchornizacji poprzez monitorenter, monitorexit

Przykładowy kod

```
public class DummyCalculationThread extends RemoteThread
{
                private long result;

                public void run()
                {
                        int x = 0;
                        for ( int i = 0 ; i < 100000000 ; i++ )
                        {
                                x+=1;
                        }
                        synchronized(this)
                        {
                                result = x;
                        }
                }
}
```

Architektura

- java.lang.Thread i java.lang.RemoteThread jako klient

Architektura

- java.lang.Thread i java.lang.RemoteThread jako klient
- java.cluster.ThreadServer jako serwer

Architektura

- java.lang.Thread i java.lang.RemoteThread jako klient
- java.cluster.ThreadServer jako serwer
- Java Virtual Machine oraz JNI

Konfiguracja

- java.util.Properties, konfiguracja zapisana w .properties

Konfiguracja

- java.util.Properties, konfiguracja zapisana w .properties
- Każdy ThreadServer musi znać przynajmniej jednego sąsiada

Konfiguracja

- java.util.Properties, konfiguracja zapisana w .properties
- Każdy ThreadServer musi znać przynajmniej jednego sąsiada
- Klient musi znać przynajmniej jeden serwer

Konfiguracja

- java.util.Properties, konfiguracja zapisana w .properties
- Każdy ThreadServer musi znać przynajmniej jednego sąsiada
- Klient musi znać przynajmniej jeden serwer
- Konfiguracja aktualizowana dynamicznie w czasie działania systemu

Delegacja wątków

- Wątek migrowany jest tylko raz

Delegacja wątków

- Wątek migrowany jest tylko raz
- Każda migracja może wiązać się z delegacją wykonania wątku do innego serwera

Delegacja wątków

- Wątek migrowany jest tylko raz
- Każda migracja może wiązać się z delegacją wykonania wątku do innego serwera
- Klient wie o tym że wątek został delegowany

Synchronizacja i replikacja

- Dane klasy są replikowane

Synchronizacja i replikacja

- Dane klasy są replikowane
- Aktualizacja danych odbywa się tylko podczas monitorenter i monitorexit

Synchronizacja i replikacja

- Dane klasy są replikowane
- Aktualizacja danych odbywa się tylko podczas monitorenter i monitorexit
- Klient zawsze ma aktualne dane, zapewnia to sekcja krytyczna

Sytuacje wyjątkowe

- Śmierć klienta oznacza przerwanie wszystkich obliczeń

Sytuacje wyjątkowe

- Śmierć klienta oznacza przerwanie wszystkich obliczeń
- Klient zostaje uznany za martwego jeżeli nie nawiąże z serwerem połączenia w odpowiednio długim czasie

Sytuacje wyjątkowe

- Śmierć klienta oznacza przerwanie wszystkich obliczeń
- Klient zostaje uznany za martwego jeżeli nie nawiąże z serwerem połączenia w odpowiednio długim czasie
- Śmierć jednego serwera oznacza rozpoczęcie obliczeń od początku (na tym etapie)

Adminstracja

- Możliwość podglądu ile aktualnie wątków się wykonuje

Adminstracja

- Możliwość podglądu ile aktualnie wątków się wykonuje
- Możliwość ponowienia obliczeń

Adminstracja

- Możliwość podglądu ile aktualnie wątków się wykonuje
- Możliwość ponowienia obliczeń
- Możliwość ustawienia automatycznej delegacji wątku na inny serwer

Migracja wątków

- Migrujemy tak na prawdę obiekt, rozszerzający RemoteThread

Migracja wątków

- Migrujemy tak na prawdę obiekt, rozszerzający RemoteThread
- Serializacja binarna obiektu

Migracja wątków

- Migrujemy tak na prawdę obiekt, rozszerzający RemoteThread
- Serializacja binarna obiektu
- Możliwość sterowania co będzie migrowane (transient)

Synchronizacja

- Każde wejście do monitorenter powoduje odwołanie do serwera mutexów

Synchronizacja

- Każde wejście do monitorenter powoduje odwołanie do serwera mutexów
- Każdy obiekt ma unikalny GUID, nadawany przez klienta (ip:port:numer)

Synchronizacja

- Każde wejście do monitorenter powoduje odwołanie do serwera mutexów
- Każdy obiekt ma unikalny GUID, nadawany przez klienta (ip:port:numer)
- Serwer mutexów pozwala na reentrantność, zgodność ze specyfikacją

Problemy

- Czytanie ze strumieni, generalnie jest to do realizacji, kwestie wydajnościowe

Problemy

- Czytanie ze strumieni, generalnie jest to do realizacji, kwestie wydajnościowe
- Częsta synchronizacja

Problemy

- Czytanie ze strumieni, generalnie jest to do realizacji, kwestie wydajnościowe
- Częsta synchronizacja
- Duże złożone obiekty

Zastosowanie

- Duże problemy obliczeniowe, które da się zrównoleglić

Zastosowanie

- Duże problemy obliczeniowe, które da się zrównoleglić
- Najlepsza wydajność jeżeli bazuje się na stosunkowo prostych typach, mały graf zależności obiektów

Zastosowanie

- Duże problemy obliczeniowe, które da się zrównoleglić
- Najlepsza wydajność jeżeli bazuje się na stosunkowo prostych typach, mały graf zależności obiektów
- Założenie że czas obliczeń jest znacząco większy niż czas potrzebny na komunikację

Java i wydajność

- Java jest wolna ?

Java i wydajność

- Java jest wolna ?
- Tak na prawdę I/O w Javie jest wolne + operacje graficzne

Java i wydajność

- Java jest wolna ?
- Tak na prawdę I/O w Javie jest wolne + operacje graficzne
- obliczenia + alokacja i dealokacja pamięci nie odbiegają znacząco od języków niższego poziomu

Java Cluster - wydajność

- Małe problemy obliczeniowe, częsta komunikacja - niska wydajność

Java Cluster - wydajność

- Małe problemy obliczeniowe, częsta komunikacja - niska wydajność
- Optymalizację można uzyskać przenosząc część operacji do jądra JVM, tylko czy warto ?

Java Cluster - wydajność

- Małe problemy obliczeniowe, częsta komunikacja - niska wydajność
- Optymalizację można uzyskać przenosząc część operacji do jądra JVM, tylko czy warto ?
- Dla dużych danych, rzadko synchronizowanych sprawdza się bardzo dobrze, na razie proste testy.

Java Cluster - wydajność

- Małe problemy obliczeniowe, częsta komunikacja - niska wydajność
- Optymalizację można uzyskać przenosząc część operacji do jądra JVM, tylko czy warto ?
- Dla dużych danych, rzadko synchronizowanych sprawdza się bardzo dobrze, na razie proste testy.
- Złożone i duże dane wymagają świadomej ingerencji programisty

Co dalej ?

- Testowanie rozwiązania dla różnych algorytmów

Co dalej ?

- Testowanie rozwiązania dla różnych algorytmów
- Optymalizacja

Co dalej ?

- Testowanie rozwiązania dla różnych algorytmów
- Optymalizacja

Dziękuję za uwagę.