УДК 004.92

Пєчкурова О. М., Ахмедзянов П. А.

АНАЛІЗ ФРЕЙМВОРКІВ РЕАЛІЗАЦІЇ МОДЕЛІ АКТОРІВ

У статті розглянуто питання щодо сучасного стану розвитку фреймворків реалізації моделі акторів Akka, Quasar, GPars, їхні основні функціональні можливості та особливості роботи з ними. Наведено результати порівняння та тестування обраних фреймворків.

Ключові слова: фреймворк, модель акторів, Akka, Quasar, GPars.

Вступ

Нині потреба в максимально ефективному використанні нових апаратних можливостей стимулює розвиток паралелізму.

Традиційною моделлю паралелізму є модель зі спільною пам'яттю. Проте в останні роки набувають популярності альтернативні моделі, причому як створюються деякі нові, так і відроджується інтерес до старих моделей. Саме серед таких моделей і є модель акторів, яка була створена ще у 1977 р. Карлом Х'юїтом [3]. Вона дає змогу частково позбутися звичайних проблем паралелізму, очікувань закінчення операцій, нескінченних м'ютексів і синхронізацій та істотно полегшує розпаралелення коду.

Створено значну кількість фреймворків та бібліотек для реалізації моделі акторів Akka, Quasar, GPars, Actor Framework, Orbit, Libactor, Orleans. Проаналізуємо фреймворки, які виконуються на базі JVM.

Модель акторів

Модель акторів – математична модель обчислень, що розглядає «акторів» як універсальні примітиви паралельних обчислень [3].

Актор – це обчислювальна сутність, яка може надсилати повідомлення тільки тим акторам, адреси яких знає. Після того як актор отримує повідомлення, він може одночасно: надсилати повідомлення на адреси інших акторів; створювати нових акторів; визначати, яким чином оброблювати наступні повідомлення.

Визначальною властивістю моделі акторів є паралельність обчислень як усередині акторів, так і поміж ними, динамічне створення акторів, включення адрес акторів до повідомлень, взаємодія лише через прямі асинхронні повідомлення без будь-яких обмежень на порядок їх отримання.

Найпоширеніший варіант реалізації моделі запропоновано у 1985 р. Ага [5].

У кожного актора є унікальне ім'я і поведінка, яка визначає його реакцію на отримані повідомлення. Щоб надіслати актору повідомлення, використовується його ім'я. Коли у стані очікування актору надходить бажане повідомлення, він виконує обчислення, які визначені його поведінкою на випадок отримання саме цього повідомлення. Актор може виконувати три типи дій: відправляти повідомлення, створювати нових акторів і оновлювати свій локальний стан. Поведінка актора може бути змінена зі зміною локального стану. Актори не мають спільного стану: потрібно явно надіслати повідомлення до іншого актора, щоб вплинути на його поведінку. Кожен актор виконує свої обчислення одночасно (й асинхронно) з іншими акторами. Маршрут проходження повідомлення, як і затримки в мережі, не визначаються. Тому порядок прибуття повідомлень є невизначеним.

Відмітимо і хорошу реалізацію визначальних семантичних характеристик стандартної моделі акторів: інкапсуляція стану та атомарне виконання методу у відповідь на повідомлення, прозорість у плануванні акторів і в доставці повідомлень та прозорість розташування, яка сприяє розподіленню обчислень та мобільності [6]. У роботі [5] їх називають: прозорість (fairness), мобільність (mobility), прозорість розташування (location transparency), інкапсуляція (encapsulation).

Інкапсуляція в контексті програмування полягає в інкапсуляції стану акторів та безпечних повідомленнях. Актор не може напряму доступитися до внутрішнього стану іншого актора, а лише може впливати на його стан актора за допомогою відправки повідомлень. У разі порушення цієї властивості два актори одночасно можуть доступитися до критичної секції іншого актора (один напряму, інший через відправку повідомлення) і таким чином порушити програмну семантику [5].

Щодо безпечних повідомлень, то не має бути спільного стану (sharedstate) між акторами. Семантика передачі повідомлень – «за значенням»,

що може вимагати копіювання об'єктів повідомлень. Але в багатьох фреймворках (наприклад, на JVM) повідомлення несуть у собі посилання на їхній вміст, створюючи спільний стан між акторами. Використання таких фреймворків вимагає від програмістів використовувати незмінні об'єкти або їхні копії.

Під прозорістю, або прозорим плануванням, розуміють обов'язковість отримання повідомлення адресатом. Повідомлення може не дійти тільки в тому разі, якщо актор-адресат не є активним. Інша властивість прозорості полягає в тому, що актор має стати активним, інакше цей актор ніколи не зможе отримати повідомлення [5]. Прозорість розташування акторів надає можливість розробникам програмувати без врахування фізичного розташування акторів. Прозоре йменування полегшує автоматичну міграцію в середовищі виконання або на інші вершини в мережі. Така мобільність дозволяє використовувати балансування навантажень та відмовостійкість [5].

Мобільність – це можливість переміщення обчислення між різними вузлами в мережі. Через підтримку інкапсуляції об'єктно-орієнтовані мови можуть підтримувати мобільність на рівні об'єктів, але всі стеки ниток, що виконуються, повинні отримати інформацію про віддалений об'єкт. Більше того, коли стеки ниток повинні отримати доступ до такого об'єкта, стек виконання має бути переміщений на віддалений вузол для завершення обчислень, а потім переміщений назад [5].

Опис фреймворків тестування

Akka – це подійно-орієнтований підпрограмний JVM фреймворк для побудови продуктивних та надійних програмних застосунків. Akka дещо схожий на бібліотеку Scala Actors, яка запозичила деякі елементи синтаксису з Erlang. Також Akka є частиноюTypesafeStack.

Основними функціями, які забезпечує Akka, є такі [7]: масштабованість – існує можливість масштабованості застосунків на багатоядерних серверах; паралелізм – фреймворк дозволяє розробникам абстрагуватися від примітивів паралелізму та сфокусуватися на бізнес-логіці; відмовостійкість – Akka запозичило з Erlang модель обробки помилок «Let It Crash»; прозорість розташування – забезпечується уніфікована програмна модель для багатоядерних та розподілених обчислень.

Будь-яка система, яка потребує високої пропускної спроможності і короткого часу відповіді, є хорошим кандидатом на використання Akka для розв'язання цих задач [7].

Мови, доступні для розробки за допомогою Akka: Java, Scala. Для тестування ми використовували мову Scala 2.3.9.

Quasar – це Java бібліотека, яка надає можливість роботи з легковісними потоками, каналами, схожими на канали в Go, та акторами, схожими на акторів в Erlang. Quasar – частина стеку технологій Parallel Universe.

Визначальною особливістю Quasar є легковісні потоки, так звані фібри (fibers). Фібри забезпечують функціональність, аналогічну потокам, але вони не керуються ОС. Фібри легкі в плані використання оперативної пам'яті (в режимі очікування фібра займає ~400 байтів оперативної пам'яті) і набагато менше навантажують процесор при переключенні задач. Можна створювати мільйони фібр у програмних застосунках. Планування виконання завдань фібр у Quasar відбувається за допомогою одного або декількох Fork Join Pool [8].

На базі фібр Quasar побудований фреймворк моделі акторів, що сильно нагадує Erlang. Актори Quasar – це простий і природний спосіб реалізації масштабованої і відмовостійкої бізнес-логіки [8].

Розробники зазначають такі переваги бібліотеки Quasar [8]: надзвичайна простота; зіставлення з шаблонами (pattern matching); вибіркове отримання повідомлень; справжні легковісні потоки (фібри); ненав'язлива інтеграція – можливість використовувати тільки те, що потрібно; масштабованість та продуктивність, без складного коду.

Мови, доступні для розробки за допомогою Quasar: Java. Версія, що використовувалась для тестування: 0.6.2.

GPars (GroovyParallelSystems) – бібліотека паралелізму, яка надає високорівневі абстракції паралелізму на Groovy, такі як map/reduce, fork/join, asynchronous closures, actors, agents, dataflow concurrency та інші.

Основні концепти GPars можна розділити на три групи: помічники на рівні коду, концепти на рівні архітектури, захист від спільного стану.

Помічники на рівні коду включають конструкції, які можуть бути застосовані до невеликої частини коду, як-от індивідуальний алгоритм або структура даних, без будь-яких серйозних змін для загальної архітектури проекту (паралельні колекції; асинхронна обробка; Fork/Join (Divide/Conquer)). Концепти на рівні архітектури підтримують конструкції, які потрібно враховувати під час розробки структури проекту (актори; Communicating sequential processes; Dataflow; Data parallelism). Захист від спільного стану забезпечують концепти: агенти; Software Transactional Memory.

Доступною мовою для розробки за допомогою GPars є Groovy. Для тестування ми використовували версію 1.2.1.

Опис тестових задач

У цій роботі ми розглядали три тестові задачі: знаходження числа пі, наближений метод розв'язання систем лінійних алгебраїчних рівнянь, просте рівняння теплопровідності.

Елементарну задачу зі ***знаходження числа π*** [2] було обрано за можливість знаходження наближення з будь-якою кількістю знаків після коми та скільки завгодно довго, що є гарним показником для тестування потужності фреймворків (дасть змогу протестувати на різних розмірностях, для тестування різних варіантів навантаження на фреймворк).

Елементарна задача зі знаходження числа π за формулою [2]:

$$\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots=\frac{\pi}{4}. \qquad (1)$$

Як бачимо з формули, наближення можна знаходити з будь-якою кількістю знаків після коми та скільки завгодно довго, що також добре для тестування потужності фреймворків (дасть змогу протестувати на різних розмірностях, для тестування різних варіантів навантаження на фреймворк).

Задача належить до приголомшливо паралельних, оскільки не потребує зв'язку між окремими завданнями (кожний доданок суми може бути обрахований паралельно, без будь-якої комунікації між ними). А в кінці обрахування кожного завдання результат його роботи просто додається до загальної суми.

Щодо логіки реалізації цієї задачі в моделі акторів, то актор-майстер поступово надсилає акторам-робітникам їхні завдання (доданки ряду, які потрібно обрахувати). Кількість робітників, кількість завдань та кількість членів ряду, які обраховуються в кожному завданні, є параметрами алгоритму, які задаються заздалегідь. У кінці обробки кожного завдання актор-робітник у повідомленні надсилає результат своєї роботи актору-майстру, який просто сумує всі отримані результати від робітників. При цьому не важливий порядок отримання таких повідомлень, і система може обраховувати окремі завдання в довільному порядку.

На Quasar задача була реалізована повністю, а у випадку з GPars та Akka були модифіковані старі приклади цієї задачі. Зазначимо, що швидкість виконання цієї задачі особливо залежить від балансування навантажень між усіма доступними ресурсами, а загальна кількість акторів є дуже маленькою (тестувалися системи акторів з кількістю не більше десяти).

Метод ітерації – числовий наближений метод розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) [1].

Ми використовували алгоритм пошуку розв'язку СЛАР методом ітерацій з [1].

Нехай дано рівняння вигляду $AX=B$.

Припустимо, що $a_{ii} \neq 0$, для $i = 1\ldots n$, та перепишемо початкове рівняння. Виразимо x_1 через перше рівняння, x_2 – через друге і так далі. Отримаємо рівняння вигляду (1):

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \cdots \frac{a_{1n}}{a_{11}}x_n. \end{cases} \qquad (1)$$

Позначимо $\frac{b_i}{a_{ii}} = \beta_i$, а також $\frac{a_{ij}}{a_{ii}} = \alpha_i$. Отже, отримаємо нове матричне рівняння вигляду $X = \beta + \alpha x$ [1].

На початку роботи алгоритму приймаємо за перше наближення вектор β, формула (2).

$$\begin{bmatrix} X_1^{(0)} \\ \ldots \\ X_n^{(0)} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \cdots \\ \beta_1 \end{bmatrix} \qquad (2)$$

На наступних кроках роботи алгоритму обраховуємо наближення результату таким чином: $X^{(k)} = \beta + \alpha X^{(k-1)}$, де k – крок роботи алгоритму. k-те наближення в матричному вигляді.

Алгоритм застосовується і дає нормальні результати лише за деякої умови збіжності. Але в рамках статті не важливий результат роботи алгоритму, тому не будемо брати цей факт до уваги, а також опустимо перевірку на збіжність, адже завданням цієї статті є перевірка і порівняння потужностей фреймворків моделі акторів.

При побудові системи акторів для розв'язання цієї задачі розпаралелення відбувається при множенні матриці α на вектор X.

Майстер-актор розділяє матрицю α на окремі рядки та відсилає як одне окреме завдання два вектори – зворотний рядок з матриці α та поточний вектор X. Актор-майстер очікує, поки не отримає повний вектор результатів множення, і тільки потім продовжує роботу. Особливістю цього тесту є те, що для кожного завдання створюється спеціальний актор-робітник, який ліквідується після пересилання результату своєї роботи актору-майстру. Таким чином планується зробити перевірку на швидкість створення та видалення акторів.

Цю задачу реалізовано на всіх обраних фреймворках з однаковою архітектурою системи акторів та логікою їх взаємодії між собою.

Рівняння теплопровідності – рівняння, яке визначає закон зміни температури з часом при теплопередачі.

Розглянемо задачу обрахування температури на обмеженій прямокутній ділянці, яка складається з окремих клітин. Початкова температура на межах – нульова і поступово збільшується до центру решітки. На межах утримується нульова температура протягом усього часу роботи алгоритму [4].

Для явного окреслення проблеми використовується алгоритм покрокового обчислення температури з часом. Елементи 2-вимірного масиву представляють значення температури в окремих точках на площині (рис. 1) [4].

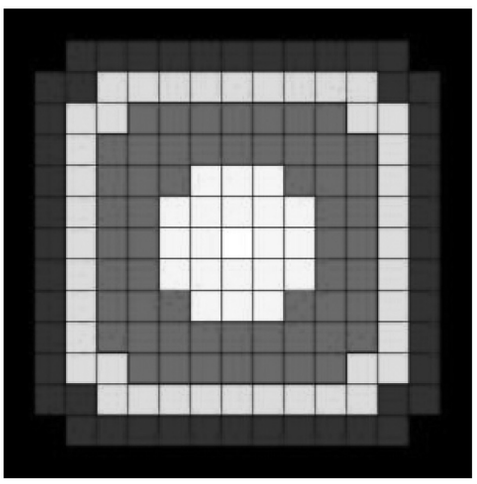

Рис. 1. Решітка температур

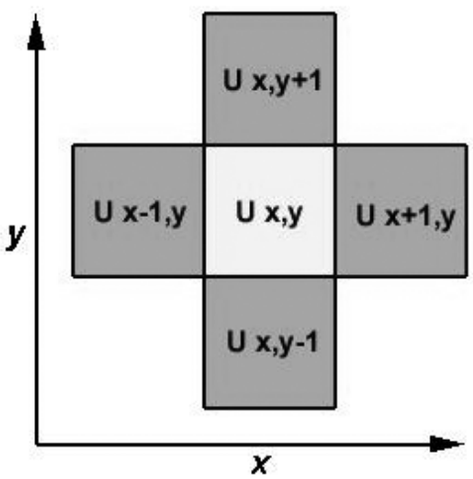

Рис. 2. Окіл сусідів

На кожному наступному кроці роботи алгоритму значення температури обчислюється за формулою (3) [4]:

$$U_{x,y}^{(k)} = U_{x,y}^{(k-1)} + C_x\left(U_{x+1,y}^{(k-1)} + U_{x-1,y}^{(k-1)} - 2U_{x,y}^{(k-1)}\right) + \\ + C_y\left(U_{x,y+1}^{(k-1)} + U_{x,-1y}^{(k-1)} - 2U_{x,y}^{(k-1)}\right). \quad (3)$$

де k – номер кроку роботи алгоритму; C_x, C_y – константи, які регулюють силу залежності температури від температур сусідів; $U_{x,y}^{(k)}$ – значення температури клітини з індексами x, y на кроці k (на рис. 2 можна побачити порядок індексів та розташування сусідів для окремої клітини).

Інколи існує потреба комунікації між акторами-робітниками. Реалізація цієї задачі є гарним прикладом необхідності комунікації між акторами-робітниками. Щодо будови системи акторів, то для збільшення навантаження на фреймворки для кожної окремої клітини створено свого актора-робітника (клітини на межі решітки завжди мають залишатися з нульовою температурою, тому для них немає сенсу створювати акторів), який відповідає лише за окрему клітину та значення температури саме в цій клітині. У кожного актора-робітника є адреси своїх сусідів для надсилання інформації про температуру цього актора. Процес керується актором-майстром, який після ініціалізації початкової решітки створює робітників та надсилає кожному адреси його сусідів.

Майстер також ініціює кожний крок оновлення всієї решітки, під час якого актори-робітники самостійно обмінюються інформацією та обраховують свої оновлені значення температур. Після чого надсилають нові значення актору-майстру для отримання повноцінної інформації про всю решітку.

Оскільки під час обрахування цієї задачі відсилається велика кількість повідомлень (наприклад, для решітки з рис. 1, розмір якої 15 на 15 клітин, на кожному кроці розсилається й обробляється приблизно 1000 повідомлень), то найкращі результати у виконанні цієї задачі повинен показати фреймворк, який найшвидше працює з повідомленнями (передача, обробка і т. д.).

Аналіз результатів тестів

Для всіх тестів ми використовували систему у складі IntelCore i5-4200M CPU; 6,0 GB RAM; Java Heap size: 1024M. Для кожного набору параметрів зроблено по десять запусків та порівняно середні та найкращі значення. Час роботи фреймворку виводиться в мілісекундах.

Параметрами задачі ***знаходження числа пі*** є кількість акторів-робітників, кількість повідомлень, які потрібно обробити, та кількість членів ряду, які будуть обраховані в кожному повідомленні.

Перший набір параметрів: кількість робітників – 4; кількість повідомлень – 5000; кількість елементів – 5000. Результати для першого набору параметрів: Akka – середній 85, мінімальний 61;

Quasar – середній 83, мінімальний 60; GPars – середній 75808.18, мінімальний 74374.

Другий набір параметрів: кількість робітників – 20; кількість повідомлень – 5000; кількість елементів – 50. Результати для другого набору параметрів: Akka – середній 9, мінімальний 3; Quasar – середній 11, мінімальний 5; GPars – середній 929, мінімальний 846.

За результатами тестів, проведених для двох наборів параметрів цієї задачі, можна зробити висновок, що фреймворк GPars значно відстає за потужністю від двох інших фреймворків. Мінімальну перевагу над фреймворком Akka для першого набору параметрів має Quasar. Але для другого набору параметрів ситуація змінюється, і вже Akka має мінімальну перевагу над Quasar. За результатом другого набору параметрів, де кількість акторів-робітників було збільшено до двадцяти, можна зробити висновок, що планувальник Akka показує себе мінімально краще – за рахунок кращого балансування навантажень за більшої кількості акторів-робітників.

Параметрами для реалізації ***другої задачі*** обрано розмір матриць та кількість ітерацій, які потрібно обрахувати.

Перший набір параметрів: розмір матриць – 100; кількість ітерацій наближення – 1000. Результати для першого набору параметрів: Akka – середній 526, мінімальний 361; Quasar – середній 971, мінімальний 448; GPars – середній 2210.54, мінімальний 1994.

Другий набір параметрів: розмір матриць – 250; кількість ітерацій наближення – 1000. Результати для другого набору параметрів: Akka – середній 1104, мінімальний 956; Quasar – середній 1384, мінімальний 948; GPars – середній 5578.27, мінімальний 5307.

Результати фреймворку GPars як за першим, так і за другим набором параметрів є значно гіршими за результати конкурентів. Більше того, збільшення розміру вхідних матриць у два з половиною рази призводить майже до аналогічного збільшення часу виконання цього тесту. Також можна помітити, що зі збільшенням розміру задачі час роботи Quasar зростає не сильно, а от час роботи Akka, аналогічно GPars, також збільшується майже пропорційно розміру задачі. Та все ж таки середній час Akka за результатами виконання тестів для другого набору параметрів є меншим за час Quasar, хоча й мінімальний час Quasar кращий, ніж мінімальний час роботи Akka.

Параметрами для розв'язку ***рівняння теплопровідності*** обрано значення C_x, C_y з формули оновлення температури, розміри решітки температур та кількість ітерацій (на час виконання завдання впливають лише розміри решітки та кількість ітерацій).

Перший набір параметрів: розмір решітки по осі X – 50; розмір решітки по осі Y – 50; кількість ітерацій наближення – 1000. Результати для першого набору параметрів: Akka – середній 4701, мінімальний 3100; Quasar – середній 6607, мінімальний 6277; GPars – середній 19368.09, мінімальний 18817.

Другий набір параметрів: розмір решітки по осі X – 200; розмір решітки по осі Y – 200; кількість ітерацій наближення – 100. Результати для другого набору параметрів: Akka – середній 6041, мінімальний 4956; Quasar – середній 12232, мінімальний 11069; GPars – середній 31294.72, мінімальний 30323.

Зі збільшенням кількості повідомлень (між першим та другим наборами параметрів), які пересилаються в системі акторів на кожному кроці роботи алгоритму, в усіх фреймворках збільшується час роботи, але найгірший результат у Quasar – середній час роботи збільшується у два рази. Щодо загального результату цього тесту, то, аналогічно іншим задачам, найгірший середній час має GPars, а найкращий – Akka. Оскільки визначальною особливістю цього тесту є комунікація між акторами-робітниками, а отже, велика кількість повідомлень та адресатів для таких повідомлень, можна зробити висновок, що фреймворк Akka є найкращим для завдань з великою кількістю повідомлень та акторів, які надсилають ці повідомлення.

Висновки

У роботі протестовано реалізацію трьох задач з використанням основних фреймворків. Приклади для Akka було написано на Scala, для GPars – на Groovy, а для Quasar – на Java. Набір тестових прикладів використовувався для порівняння обраних фреймворків реалізації моделі акторів. На основі тестування цих задач і проведеного порівняльного аналізу результатів зроблено висновок про перевагу фреймворку Akka.

Підтверджено застосовність моделі акторів для розв'язання багатьох прикладних математичних задач. Підтверджено той факт, що потужні можливості інтеграції, величезні набори зручних програмних абстракцій, а також гарна відмовостійкість роблять фреймворки акторів Akka, GPars, Quasar корисним програмним інструментом для розв'язання найсучасніших прикладних задач.

Список літератури

1. Метод итерации [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8. – Загл. с экрана.
2. Число пі [Електронний ресурс]. – Режим доступу: http://uk.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_ %D0%BF%D1%96. – Назва з екрана.
3. Hewitt C. Actor Model of Computation for Scalable Robust Information Systems [Electronic resource] / Carl Hewitt. – Inconsistency Robustness, 2015. – 978-1-84890-159-9. – <hal-01163534v3>. – Mode of access: https://hal.archives-ouvertes.fr/hal-01163534/document. – Title from the screen.
4. Introduction to Parallel Computing. Simple Heat Equation [Electronic resource]. – Mode of access: https://computing.llnl.gov/tutorials/parallel_comp/#ExamplesHeat. – Title from the screen.
5. Karmani R. Actor Frameworks for the JVM Platform: A Comparative Analysis / Rajesh Karmani, Amin Shali, Gul Agha. – Department of Computer Science University of Illinois, Urbana-Champaign, 2009. – 10 p.
6. Karmani R. Actors / Rajesh K. Karmani and Gul Agha // Encyclopedia of Parallel Computing. – Springer, 2011. – P. 1–11.
7. Munish K. Gupta. Akka Essentials / K. Munish. – Packt Publishing, 2012. – 234 p.
8. Quasar. User Manual [Electronic resource]. – Mode of access: http://docs.paralleluniverse.co/quasar/. – Title from the screen.

O. Pyechkrova, P. Akhmedzyanov

ANALYSIS IMPLEMENTATION MODEL ACTORS FRAMEWORKS

In the article a question is considered in relation to modern development of frameworks, realization of model of actors status, they are considered functional possibilities, and also features of work, with them. The results of comparison and testing of select frameworks are analysed.

Keywords: framework, model of actors, Akka, Quasar, Gpars.

Матеріал надійшов 30.09.2015

УДК 004.42

Кирієнко О. В., Малькевич Б. І.

ВИКОРИСТАННЯ ФРЕЙМВОРКУ JAVA PLAY ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

У роботі розглянуто основні особливості Java фреймворку Play. Описано середовище та основні переваги цього фреймворку.

Ключові слова: Play Framework, Java фреймворк Play.

Вступ

Play – каркас розробки з відкритим кодом, написаний на Scala і Java, який імплементує патерн проектування модель-представлення-контролер (MVC).

Play 2 повністю базований на RESTful. У ньому інтегровано юніт тестування, JUnit та Selenium, вбудовані в ядро; API містить велику кількість різних компонентів. Усі контролери каркасу статичні, а архітектура модульна [4].