Erlang как язык, направленный на создание параллельных приложений

А.А. Рыбаков, *ЗАО «МЦСТ», старший научный сотрудник, rybakov.aax@gmail.com.*

Введение

С момента появления первых языков программирования их развитие шло по пути уменьшения затрат на разработку, отладку и сопровождение программ. Это естественным образом вело к повышению уровня абстракции, что позволило в процессе написания программы оперировать не архитектурно-зависимыми понятиями, а сущностями, максимально приближенными к области постановки задачи [1].

Появление языков программирования высокого уровня (таких как Fortran, Algol) позволило разрабатывать переносимые приложения, переводимые с помощью различных компиляторов в машинные коды для конкретных архитектур. Разработка программ стала более универсальной. Кроме того, появились типы данных с определенными над ними операциями, наглядные управляющие конструкции, возможность выделения частей кода в подпрограммы, что позволило разработчикам приблизить текст программы к логике реализуемого алгоритма.

Позднее появление парадигмы объектно-ориентированного программирования (Smalltalk, C++) дало возможность рассматривать программу не как совокупность подпрограмм, вызываемых друг из друга, а как множество объектов, обладающих определенным поведением и обменивающихся между собой сообщениями. Такой подход позволил максимально приблизить друг к другу процессы проектирования и разработки приложения путем реализации программных объектов, соответствующих объектам из области постановки задачи.

Отдельно следует выделить возникновение декларативных языков программирования. До их появления все языки являлись императивными, то есть рассматривали программу как последовательность инструкций, меняющий ее состояние. Декларативные языки (Erlang, Haskell) рассматривают программу как совокупность описаний входных данных и результата вычислений. Примером декларативных языков программирования могут послужить функциональные языки. В парадигме функционального программирования программа представлена совокупностью

определений функций в математическом смысле. Все вычисления сводятся к получению значений без явного хранения состояния программы. Достоинством функционального подхода является то, что функции для вычисления результата могут использовать только свои аргументы и не меняют состояние программы. То есть побочные эффекты отсутствуют, и значение функции всегда зависит только от ее аргументов.

Параллельность в современных приложениях

Параллельное исполнение играет важную роль в разработке современных приложений. С помощью распараллеливания вычислений достигается существенное повышение производительности вычислительных систем. Это стало особенно актуально с развитием многоядерных процессоров и многопроцессорных суперкомпьютеров.

Другим аспектом параллельных вычислений является моделирование сложных систем, состоящих из множества независимых процессов, выполняющихся параллельно. Обеспечение параллельного выполнения моделей этих процессов в разработанной программной системе порождает более точную общую модель исходной системы.

Создание параллельного приложения может достигаться путем распараллеливания уже готовой программы. Для выполнения этой задачи существует множество технологий (одними из самых известных можно считать OpenMP и MPI), подробно о них можно прочитать в [2]. В процессе распараллеливания одной из самых серьезных проблем является синхронизация различных процессов или потоков при работе с общей памятью. Для этих целей используется механизм семафоров, которые в определенные моменты могут разрешить или запретить доступ к конкретным областям памяти. Выявление возможных конфликтов между различными процессами по доступу в память в общем случае является очень непростой задачей, и отладка распараллеленного приложения может занять длительное время.

Другой подход к разработке параллельных приложений предусматривает использование специализированного языка программирования. В данной статье мы рассмотрим один из таких языков — Erlang.

Особенности языка Erlang

Язык Erlang [3, 4, 5] позволяет создавать параллельные приложения, описывая их как совокупность отдельных процессов, которые могут обмениваться между собой сообщениями. Это хорошо соотносится с представлениями об окружающем нас мире, объекты которого также общаются, обмениваясь информацией.

Процессы в Erlang являются примитивной базовой сущностью, их создание настолько молниеносно, что в программе могут создаваться десятки и даже сотни тысяч процессов без ущерба производительности. Обмен сообщениями между процессами также является быстрой операцией.

Кроме того, между процессами могут устанавливаться связи, так что гибель одного процесса не пройдет незамеченной для остальных, связанных с ним.

Таким образом, отпадает необходимость искусственно моделировать сложную систему взаимоотношений между параллельно живущими объектами реального мира — параллельность является естественным качеством языка Erlang.

При использовании языка Erlang нет необходимости дополнительно модифицировать код программы, чтобы получить преимущества, связанные с использованием многопоточности. Для многопоточного распараллеливания программы, написанного на стандартном языке программирования, необходимо использовать специальные технологии, о которых было сказано выше. Это нужно для предотвращения конфликтов доступа к разделяемой памяти. В языке Erlang не нужно использовать блокировки для доступа к разделяемой памяти, так как разделяемая память отсутствует. Каждый процесс, как и в реальном мире, обладает своей памятью, доступ к которой другие процессы получить не могут. В рамках одного процесса конфликтов по доступу в память также нет из-за отсутствия побочных эффектов вычислений.

В классических языках программирования, таких как C или Pascal, переменная представляет собой некоторую сущность, которая в процессе вычислений может изменять свое значение. В языке Erlang, как в функциональном языке, переменная имеет другой смысл, ей можно присвоить значение, но изменить его уже нельзя. Таким образом, мы можем говорить о переменной однократного присваивания (single assignment variable). Переменная, которой еще не было присвоено значение, называется несвязанной, а после получения значения — связанной.

Строго говоря, в языке Erlang отсутствует оператор присваивания. Символ `'='` означает оператор проверки на соответствие шаблону. Если в тексте программы встречается запись `X = 5`, и переменная `X` еще не связана ни с каким значением, то осуществляется связывание, в этом смысле данное действие условно можно считать присваиванием. Если после этого встречается строка `X = 5`, то компилятор подставляет значение переменной и проверяет верное тождество `5 = 5`, программа продолжает работу. Но если встречается строка `X = 3`, то после подстановки значения проверяется равенство `5 = 3`, которое не

выполняется, и возникает исключение. Таким образом, переменные в языке Erlang можно рассматривать как переменные в математическом смысле, а проверки на соответствие шаблону — как математические уравнения.

Проверка на соответствие шаблону (pattern matching) является мощным инструментом, который повсеместно используется в конструкциях Erlang: при доступе к полям структуры, передаче параметров в функции, обработке строк и бинарных данных, в условных конструкциях и других местах. Данная техника позволяет упростить код и повысить его наглядность.

Использование переменных однократного присваивания в языке Erlang является его сильной стороной, так как позволяет избавиться от побочных эффектов при проведении вычислений, что крайне важно для параллельных приложений.

Так как значение никакой переменной не может быть изменено, то в программе отсутствуют глобальные данные, а значит, вычисление любой функции зависит только от значений ее аргументов, то есть функции являются детерминированными.

Эти базовые принципы действительно облегчают процесс создания параллельных приложений, так как нет необходимости прибегать к механизмам разрешения межпотоковых конфликтов — такие конфликты не могут возникнуть.

Язык Erlang обладает мощным инструментарием для создания распределенных масштабируемых отказоустойчивых систем, известным как OTP (Open Telecom Platform) [3]. Обсуждение возможностей OTP выходит за рамки данной статьи, в которой мы лишь коснемся базовых примитивов создания параллельного приложения и рассмотрим простой пример, иллюстрирующий управление взаимодействием процессов.

Всего в языке Erlang есть три базовых примитива параллельного программирования:

- Создание параллельного процесса. Выполняется, например, с помощью вызова функции `Pid = spawn(Fun)`. Она возвращает идентификатор процесса, в котором запущена функция `Fun`. На самом деле существует семейство функций для создания процесса, отличающихся между собой деталями взаимодействия родительского и дочернего процессов.
- Отправление сообщения процессу. Для этого действия в языке предусмотрен специальный оператор: `Pid ! Message`.
- Конструкция приема сообщения, соответствующего одному из предусмотренных шаблонов. Синтаксически ожидание сообщения реализуется следующим образом:

```
receive
    Pattern1 [when Guard1] ->
        Expressions1;
    Pattern2 [when Guard2] ->
        Expressions2;
   …
end
```

Этих трех примитивов достаточно для создания параллельного приложения. Рассмотрим простой пример взаимодействия процессов на языке Erlang.

В следующем примере всего два процесса. Один из этих процессов создается из другого и постоянно находится в состоянии ожидания сообщения. Как только он получает сообщение, он в зависимости от вида этого сообщения выполняет некоторые действия (складывает или перемножает числа и выводит результат на экран). Сообщения же этому процессу посылает процесс, его создавший. Конечно, этот пример не очень интересен с практической точки зрения, однако на нем видна простота применения примитивов параллельного программирования в языке Erlang.

```
-module(exm_parallel).
-export([start/0, loop/0]).

loop() ->
  receive
    {sum, A, B} ->
      io:format("~w + ~w = ~w~n", [A, B, A + B]);
    {mult, A, B} ->
      io:format("~w * ~w = ~w~n", [A, B, A * B]);
    Other ->
      io:format("unknown command ~w~n", [Other])
  end,
  loop().

start() ->
  Pid = spawn(fun exm_parallel:loop/0),
  Pid ! {sum, 5, 6},
  Pid ! {mult, 3, 5},
  Pid ! {sub, 3, 5},
  ok.
```

При запуске данного модуля получим примерно следующую выдачу на экран:

```
(Erlang@host)1> exm_parallel:start().
5 + 6 = 11
ok
3 * 5 = 15
unknown command {sub,3,5}
```

При этом заметим, что печать `ok` соответствует окончанию работы процесса, в котором выполнялась функция `start`, в то время как второй процесс продолжает свою работу и обрабатывает все посланные ему сообщения.

Заключение

Язык программирования Erlang обладает простым и в то же время мощным инструментарием для создания полноценных параллельных приложений, прекрасно подходит для обучения концепции параллельного программирования. Он не отягощен конструкциями синхронизации конкурирующих процессов, так как, являясь функциональным языком программирования, исключает появление побочных эффектов вычислений.

Из-за интуитивно-понятного синтаксиса и богатого набора вспомогательных библиотек Erlang обладает низким порогом вхождения, что позволяет проектировать и создавать распределенные системы уже с первых дней практики.

Литература

1. Роберт У. Себеста. Основные концепции языков программирования. // Вильямс, 2001.
2. В. В. Воеводин, Вл. В. Воеводин. Параллельные вычисления. // БХВ-Петербург, 2004.
3. Joe Armstrong. Programming Erlang: Software for a Concurrent World. // Pragmatic Bookshelf, 2007.
4. Joe Armstrong, Robert Virding, Claes Wikstrom, Mike Williams. Concurrent Programming in Erlang. // Prentice Hall, 1996.
5. Francesco Cesarini, Simon Thompson. Erlang Programming. // O'Reilly, 2009.