Приступаем к изучению.

Введение.

Как и с любым другим языком программирования, вы пройдете через несколько стадий на вашем пути к мастерству в Эрланге. Давайте посмотрим на эти стадии, которые мы охватываем в данной книге и на то что вы изучите по мере своего продвижения.

Стадия 1: Я не уверен...

Когда вы новичок, вам надо научиться, с начала, запускать систему, выполнять команды в оболочке Эрланга, компилировать простые программы и, вообще, познакомиться с этим языком. (Эрланг - это маленький язык, так что это не займет много времени.)

Давайте разделим это на более мелкие куски. Как новичок, вы сделаете следующее

- Убедитесь, что на вашем компьютере установлена работающая система Эрланг.
- Научитесь запускать и останавливать командную оболочку Эрланга.
- Узнаете как набирать и выполнять различные выражения Эрланга в его командной оболочке, а также понимать результаты такого выполнения.
- Увидите как создавать и модифицировать программы на Эрланге с помощью вашего любимого текстового редактора.
- Поэкспериментируете с компиляцией и выполнением ваших программ в командной оболочке Эрланг

Стадия 2: Мне комфортно с Эрлангом.

Итак, вы уже немного научились работать с языком Эрланг. Поскольку вы уже познакомились с этим языком, то вы готовы изучать Главу 5. Углубленное последовательное программирование.

На этой стадии вы окончательно познакомитесь с Эрлангом и мы сможем перейти к его более интересным темам:

Вы узнаете о более хитрых техниках использования оболочки Эрланга. Оболочка может делать гораздо больше, чем мы будем себе представлять после первого знакомства с ней. (Например, вы можете в ней вызывать заново и редактировать ваши прошлые выражения и команды. Об этом рассказывается в разделе 6.5

Редактирование команд в оболочке Эрланга.)

Вы начнете изучение библиотек (называемых в Эрланге модулями). Большинство программ, из числа тех что я написал, могут быть написаны с использованием всего пяти модулей: lists, io, file, dict и gen_tcp . Следовательно мы будем активно пользоваться этими модулями на протяжении всей книги.

По мере того, как ваши программы будут становиться все больше, вам потребуется знать, как автоматизировать их компиляцию и запуск. Наилучшим решением для этого является утилита make. Мы научимся как можно контролировать этот процесс с помощью написания make-файлов. Об этом рассказывается в Разделе 6.4 *Автоматизация компиляции с помощью Make-файлов.*

В большом мире программирования на Эрланге активно используется большая коллекция библиотек, называемая OTP (от ее названия - Open Telecom Platform - Открытая Платформа для телекоммуникационных приложений). По мере накопления вами опыта работы с Эрлангом, вы начнете понимать, что владение OTP сбережет вам множество времени и сил при написании серьезных приложений. В конце-концов, зачем заново разрабатывать колесо, когда кто-то уже реализовал ту функциональность, которая вам нужна? Мы изучим основу OTP - *поведения*, в частности gen_server.

Одно из основных применений Эрланг - это написание распределенных программ, так что теперь настало время поэкспериментировать с этим. Начать можно с примеров приведенных в Главе 10 *Распределенное программирование*, а потом эту тему можно расширить насколько вы это пожелаете.

Стадия 2.5: Я могу изучить дополнительные темы.

Вам не надо изучать каждую главу этой книги при первом ее прочтении.

В отличии от большинства других языков, с которыми вы встречались ранее, Эрланг это параллельный язык программирования, и это делает его особенно удобным для написания распределенных программ, а также для программирования современных многоядерных и SMP (Symmetric multiprocessing - Симметрично-мультипроцессорных) компьютеров. Множество Эрланг программ начинают просто работать быстрее, будучи запущенными на многоядерных или на SMP машинах.

Программирование на Эрланге основывается на новой парадигме программирования, которую я называю *параллельно-ориентированное программирование* (ПОП, POP - parallel-oriented programming).

Когда вы используете ПОП, вы разбиваете проблему на множество мелких процессов и определяете естественный параллелизм в ее решении. Это

Стадия 3: Я - Эрланг Мастер.

Теперь вы являетесь мастером в языке можете написать полезные распределенные программы. Но чтобы достичь истинного мастерства вы должны изучить еще больше:

- Mnesia. Дистрибутив Эрланга поставляется всем бесплатно вместе с встроенной быстрой, реплицируемой базой данных называемой *Mnesia*. Она была изначально разработана для телекоммуникационных приложений где производительность и отказоустойчивость являются ключевыми критериями. Сейчас она широко используется и в различных не-телекоммуникационных приложениях.
- Интерфейсы с программами написанными на других языках и использование *встроенных драйверов.* Это рассматривается в разделе 12.4 *Встроенные драйверы*.
- Свободное использование основанных на поведении деревьев супервизоров, сценариев старта и так далее. Об этом рассказывается в Главе 18 *Разработка систем при помощи OTP*.
- Как запустить и оптимизировать вашу программу для многоядерных компьютеров. Об этом рассказывается в Главе 20 *Программирование многоядерных процессоров*.

Самый главный урок

Есть одно правило которое вы должны помнить на протяжении всей этой книги: программирование - это здорово, приятно, весело и интересно. И лично я считаю, что программирование распределенных приложений, таких как программа чата или обмена мгновенными сообщениями - это гораздо более приятно и весело, чем программирование обычных последовательных приложений. То что вы можете сделать на одном компьютере - ограничено его возможностями, но возможности сети компьютеров - практически безграничны. И Эрланг обеспечивает вам идеальную среду для экспериментов с сетевыми приложениями и для построения промышленных систем.

Чтобы помочь вам освоиться со всем этим я перемешал главы посвященные приложениям реального мира с техническими главами по системе Эрланг. Вы должны рассматривать эти приложения как отправные точки для ваших собственных экспериментов. Возьмите их, доработайте их и используйте их так, как я и предположить того не мог и я буду от этого очень счастлив.

Инсталляция Эрланга

Прежде чем вы сможете что-либо сделать, вам надо убедиться, что у вас установлена работающая версия Эрланг на вашей системе. Идите в командную консоль и наберите там erl:

```
$ erl
    Erlang (BEAM) emulator version 5.5.2 [source] ... [kernel-poll:false]
    Eshell V5.5.2 (abort with ^G)
1>
```

В Windows erl из консоли сработает только если он установлен и переменная окружения PATH указывает на его исполняемый файл тоже. Если вы установили Эрланг в Windows стандартным путем, то вы сможете его вызвать через меню Старт > Все Программы > Erlang OTP . В приложении B я рассказываю как я настроил Эрланг в Windows на совместную работу с

> *Примечание:* В этой книге я буду показывать приветственное обращение Эрланга (приведенное чуть выше) только изредка. Эта информация полезна только если вы хотите сообщить об ошибке. Я ее показал тут только для того, чтобы вы не волновались, увидев подобное. В большинстве примеров я его показывать не буду, если, конечно, это не будет необходимо.

Если вы увидели приветствие его командной оболочки, значит Эрланг установлен на вашем компьютере. Выйдите из него - нажмите **Ctrl+G** а потом еще **Q** и Ввод (Enter или Return). (Другой вариант - выполните команду **q().** в оболочке.) Теперь вы можете прямо перейти к разделу

Код программ в данной книге.

Если же, вместо этого, вы получили ошибку о том, что erl это неизвестная команда, вам надо установить Эрланг на ваш компьютер. А это означает, что вам придется принять решение - хотите ли вы установить готовый бинарный дистрибутив для вашейь машины, воспользоваться пакетным дистрибутивом (на OS X), собрать Эрланг из исходных кодов, или использовать Comprehensive Erlang Archive Network (CEAN) (\~"Полный сетевой архив Эрланга")?

Бинарные дистрибутивы

Бинарные дистрибутивы Эрланга доступны для Windows и Linux операционных систем. Инструкции по их установки значительно зависят от конкретной операционной системы. Так что мы пройдемся по им обоим.

Windows

Список релизов вы найдете по адресу http://www.erlang.org/download.html. Выбирайте

самый последний релиз и кликайте на линке к бинарному дистрибутиву для Windows - он указывает на исполнимый файл. Далее следует стандартная инсталляция для Windows, которая не должна у вас вызвать никаких проблем.

Linux

Бинарный пакет существует, например, для Debian варианта Linux. На Debian системе наберите следующую команду:

```
> apt-get install erlang
```

Инсталляция для Mac OS X

Будучи пользователем Mac вы можете инсталлировать готовую версию Эрланга используя систему MacPorts, либо же вы можете собрать Эрланг из исходных кодов. Использовать MacPorts немного проще и она следит за новыми версиями ПО. С другой стороны, MacPorts может несколько запаздывать с релизами Эрланга. Например, во время написания данной книги версия Эрланга в MacPorts отставала на два релиза от его текущей версии. По этой причине, я рекомендую вам стиснуть зубы и установить Эрланг из его исходного кода, как это описано в следующем разделе. Для этого вам надо убедиться, что у вас установлены средства разработчика (они есть на DVD с ПО, который приходит вместе с вашей машиной).

Сборка Эрланга из исходного кода

Альтернативным способом к инсталляции готовых бинарных дистрибутивов является сборка Эрланга из исходных кодов. Для Windows в этом нет особого смысла, так как каждая версия Эрланга выходит с полными бинарными дистрибутивами для этой ОС, включающими в себя, также, и исходные коды.

Но для пользователей Маков и Linux возможны задержки между официальным релизами Эрланга и готовностью бинарных дистрибутивов для данных систем. Для всех Unix-подобных ОС инструкции по инсталляции одни и те же:

Загрузите последние исходники Эрланга (с адреса http://www.erlang.org/download.html). Они будут находиться в файле с названием подобным otp_src_R11B-4.tar.gz (конкретно этот файл содержит четвертый релиз 11-ой версии Эрланга).

Распакуйте, сконфигурируйте, соберите и инсталлируйте согласно следующему:

```
$ tar -xzf otp_src_R11B-4.tar.gz
$ cd otp_src_R11B-4
$ ./configure
$ make
$ sudo make install
```

> *Примечание:* Вы можете использовать команду ./configure - -help чтобы ознакомиться со всеми опциями конфигурации перед построением системы.

Использование CEAN

Полный сетевой архив Эрланга (CEAN) - это попытка собрать вместе основные приложения Эрланга в одном месте с одним инсталлятором. Преимущество в использовании CEAN состоит в том, что там содержатся не только базовые системы Эрланга но и большое число программных пакетов написанных на Эрланге. Это означает, что не только ваша версия системы Эрланг будет современной, но вы также сможете управлять и своими пакетами.

CEAN содержит готовые бинарные дистрибутивы для большого числа операционных систем и процессорных архитектур. Чтобы инсталлировать систему при помощи CEAN, зайдите на http://cean.process-one.net/download/ и следуйте инструкциям. (Отметим, что некоторые пользователи сообщали, что CEAN не всегда инсталлирует компилятор Эрланга. Если это произойдет у вас, то запустите оболочку Эрланга и дайте там команду `cean:install(compiler)` - она инсталлирует компилятор.)

Код программ в данной книге

Большинство примеров кода, которые мы будем приводить в данной книге, вы можете свободно загрузить из сети (по адресу http://pragmaticprogrammer.com/titles/jaerlang/code.html). Чтобы помочь вам в этом, некоторые примеры будут снабжены специальной ссылкой вверху, подобно следующему: http://media.pragprog.com/titles/jaerlang/code/shop1.

```
-module(shop1).
-export([total/1]).

total([{What, N}|T]) -> shop:cost(What) * N + total(T);

total([]) -> 0.
```

Эта ссылка будет содержать адрес к исходному коду, который можно загрузить себе. Если вы читаете PDF версию данной книги и ваша программа для чтения PDF-файлов поддерживает гиперлинки, то вы можете кликнуть по этой ссылке и указанный код должен появиться в окне вашего браузера.

Запуск оболочки Эрланга

Теперь давайте начнем. Мы можем взаимодействовать с Эрлангом используя интерактивное средство называемое *Оболочка (Shell)*. Если мы ее запустим, мы можем набирать в ней выражения, а оболочка будет показывать их значения.

Если вы уже инсталлировали Эрланг на вашей машине (раздел 2.2 *Инсталляция Эрланга*), то значит и его оболочка, `erl`, также инсталлирована. Чтобы ее запустить, откройте стандартную командную консоль вашей операционной системы (cmd в Windows или что-то вроде bash в Unix-подобных системах). И запустите оболочку Эрланга, набрав команду erl:

```
(1) $ erl
    Erlang (BEAM) emulator version 5.5.1 [source] [async-threads:0] [hipe]
    Eshell V5.5.1 (abort with ^G)
(2) 1> % I'm going to enter some expressions in the shell ..
(3) 1> 20 + 30.
(4) 50
(5) 2>
```

Давайте посмотрим, что мы только что сделали:

- (1) Эта Unix команда запускает оболочку Эрланга. Оболочка отвечает стандартным приветствием, где говорится какую именно версию Эрланга вы запустили.
- (2) Оболочка вывела приглашение 1> и мы напечатали комментарий. Знак процента (%) означает начало комментария в языке Эрланг. Любой текст от знака процента до конца строки считается комментарием и игнорируется оболочкой или компилятором Эрланга.
- (3) Оболочка повторяет приглашение 1> поскольку мы не ввели полной команды. На этот раз мы вводим выражение 20+30 с точкой и возвратом каретки. (Начинающие изучение Эрланга часто забывают ставить точку. Но без нее Эрланг не может определить, что закончили мы наше выражение и хотим увидеть его результат.)
- (4) Оболочка вычисляет введенное выражение и печатает его результат - 50, в данном случае.
- (5) Оболочка выводит следующее приглашение, на этот раз, для команды номер 2 (поскольку номер команды увеличивается каждый раз, когда вводится очередная команда).

Вы уже попробовали запустить оболочку Эрланга на вашей машине? Если нет, то, пожалуйста, остановитесь и попробуйте это сделать сейчас. Если вы будете просто

читать текст без опробования команд, вы может и будете думать, что все понимаете, но вы не будете переносить ваши знания из вашего мозга в ваши пальцы - программирование это не спорт для зрителей. Как и в любом виде атлетики, вам нужно очень много практиковаться.

Введите выражения из примеров в точности так, как они приведены в тексте книги, а потом немного измените их. Если они не сработают, остановитесь и спросите себя, что пошло не так. Даже опытные программисты на Эрланге проводят множество часов работая с его оболочкой.

По мере накопления вами опыта, вы узнаете, что оболочка - это, на самом деле, очень мощное средство для работы. Предыдущие введенные команды могут быть заменены (с помощью Ctrl+P и Ctrl+N) и отредактированы (командами подобными Emacs-совским). Подробнее об этом рассказано в разделе 6.5 *Редактирование команд в оболочке Эрланга.* А лучше всего вы это поймете, когда начнете писать распределенные программы и узнаете, что можно присоединить вашу оболочку к другой, работающей Эрланг системе, на другом Эрланг узле в кластере или, даже, организовать зашифрованное взаимодействие (ssh) с Эрланг системой работающей на другом, удаленном компьютере. Используя ее вы можете взаимодействовать с любой Эрланг программой на любом Эрланг узле в сообществе таких узлов.

> *Предупреждение:* Вы не можете напечатать в оболочку все, что вы увидите в этой книге. В частности, вы не можете напечатать в оболочку код из примеров Эрланг программ. Синтаксическая форма .erl файла это вовсе не выражение и оболочкой они не воспринимаются. Оболочка может выполнять только Эрланг выражения и не понимает больше ничего другого. В частности нельзя ввести в оболочку заголовок модуля, это те его части, которые начинаются на тире (такие как -module, -export и так далее).

Остальная часть данной главы построена в форме коротких диалогов с оболочкой Эрланга. Часто я не буду рассказывать в мельчайших подробностях, что там происходит, поскольку это помешает изложению материала в книге. В разделе 5.4 *Многочисленные короткие заметки* я дополню некоторые детали.

Простая арифметика целых чисел

Давайте вычислим несколько простых арифметических выражений:

```
1> 2 + 3 * 4.
14
2> (2 + 3) * 4.
20
```

> *Важно:* Вы видите что диалог начинается с приглашения с номером 1 (то есть оболочка напечатала 1>). Это означает, что мы запустили новую оболочку Эрланга. Всякий раз, когда вы будете видеть, что диалог начинается с 1> вам нужно будет запускать новую оболочку если вы хотите *в точности* воспроизвести пример из книги. А когда пример начинается с приглашения, номер которого больше чем единица, это означает что работа оболочки продолжена с предыдущего примера и вам не надо запускать ее заново.

Оболочка Эрланг не отвечает?

Если оболочка не отвечает после того, как вы ввели команду, тогда, возможно, вы забыли набрать в конце точку и возврат каретки (что еще называют *точка-конец-строки*).

Также возможно, что вы начали вводить что-то в кавычках (то есть открыли одиночные двойные кавычки (")), но пока не ввели соответствующий им второй символ кавычек, который должен быть точно таким же как и первый.

Если подобное возможно, то лучшее, что вы можете сделать - это ввести закрывающие кавычки и точку-конец-строки.

Если же все серьезно испортилось и система вообще вам не отвечает, тогда нажмите Ctrl+C (в Windows - Ctrl+Break). Вы увидите следующее:

```
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
(v)ersion (k)ill (D)b-tables (d)istribution
```

А теперь просто нажмите **A** чтобы прервать текущую сессию работы с Эрлангом.

> *Подробнее*: Вы можете запускать и останавливать множество оболочек. Смотрите подробности в разделе 6.7 *Если оболочка не отвечает*.
>
> *Примечание:* Если вы хотите опробовать все приводимые примеры из книги в оболочке Эрланга сразу по мере чтения (что является *абсолютно* наилучшим способом обучения) тогда вам возможно стоит быстро ознакомиться с разделом 6.5 *Редактирование команд в оболочке Эрланга*.

Вы увидите, что Эрланг следует нормальным правилам арифметических выражений, и 2+3*4 означает 2+(3*4) а не (2+3)*4 .

Эрланг использует целые числа произвольного размера для целочисленных вычислений. Целочисленная арифметика в Эрланге точная, так что вам не надо волноваться насчет переполнения или неспособности представить целое большой длинны.

Почему бы это не попробовать? Вы можете удивить своих друзей вычисляя очень большие числа:

```
3> 123456789 * 987654321 * 112233445566778899 * 998877665544332211.
1366956026032180998596619889892576169661342790993534 1
```

Вы можете вводить целые множеством различных способов (подробнее см. в разделе 5.4 *Целые*) Вот вам пример записи чисел с использованием оснований 16 и 32:

```
4> 16#cafe * 32#sugar.
1577682511434
```

Переменные

Нотация для переменных

Часто нам придется говорить о значении какой-то конкретной переменной. Для этого я буду использовать запись вида Var -> Value . То есть, например, запись A -> 42 будет означать что переменная A имеет значение 42. Когда будет несколько переменных я буду писать {A -> 42, B -> true...} , что означает что A это 42, B это true и так далее.

Как можно сохранить результат выражения, чтобы использовать его в дальнейшем? Для этого предназначены переменные. Вот пример:

```
1> X = 123456789.
123456789
```

Что здесь произошло? Во-первых, мы присвоили значение переменной X . Потом оболочка Эрланг напечатала ее значение.

> *Примечание*: Все имена переменных *должны* начинаться с большой буквы.

Если вы хотите узнать значение переменной - просто введите ее имя:

```
2> X.
123456789
```

Теперь переменная X имеет значение и им можно воспользоваться:

```
3> X*X*X*X.
232305722798259244150093798251441
```

Тем не менее, если вы попытаетесь присвоить переменной X другое значение, вы получите довольно большое сообщение об ошибке:

```
4> X = 1234.
=ERROR REPORT==== 11-Sep-2006::20:32:49 ===
Error in process <0.31.0> with exit value:
{{badmatch,1234},[{erl_eval,expr,3}]}
    exited: {{badmatch,1234},[{erl_eval,expr,3}]}
```

Что сейчас произошло с Эрлангом? Чтобы объяснить это, мне придется разрушить два ваших внутренних предположения относительно такой простой записи, как X=1234 :

- Во-первых, X это не переменная, по крайней мере не такая переменная, к которым вы привыкли в таких языках как C и Java.
- Во-вторых, = - это вовсе не оператор присвоения.

Возможно это самая сложная область для освоения для всех новичков в Эрланге, так что давайте потратим на нее несколько страниц, чтобы во всем досконально разобраться.

Переменные не изменяются

В Эрланге переменным можно присваивать значения *только один раз*. Другими словами Эрланг - это язык с *однократно присваиваемыми переменными*. Если вы попытаетесь изменить значение переменной, после того как оно было установлено, вы получите сообщение об ошибке (на самом деле, это сообщение об ошибке несоответствия, как вы могли видеть). Переменная, которой уже назначено значение называется *связанной* переменной. В противном случае она называется *свободной* переменной. Изначально, все переменные свободны.

Когда Эрланг видит выражение типа X=1234 он пытается привязать переменной X значение 1234. До такого связывания переменная X может принять любое значение, она - это просто пустая ячейка, которую нужно заполнить. Но, как только, она получает свое значение, оно останется там неизменным навечно.

Одиночное присвоение подобно школьной Алгебре

Когда я ходил в среднюю школу мой учитель математики говорил нам: "Если есть несколько X в разных частях одного уравнения, все эти X означают одно и то же". Именно поэтому мы можем решать уравнения: если мы знаем, что X+У=10 а X-Y=2, то X будет 6, а У будет 4 в обоих этих уравнениях.

Но когда я стал изучать мой первый язык программирования мы увидели вот такую

фигню:

```
X=X+1
```

Мы все протестовали, говорили "так нельзя делать!", но учитель программирования сказал, что мы неправы и должны забыть все, что мы изучили на математике. X теперь это вовсе не математическая переменная. Она теперь такая ячейка, подобная маленькой коробочке...

В Эрланге переменные точно такие же, как и в математике. Когда вы присваиваете им значение, вы делаете утверждение - заявляете о непреложном факте. Эта переменная имеет именно это значение. И никак иначе.

Возможно вы теперь удивитесь, а почему мы тогда используем, вообще, слово переменная? На это есть две причины:

Они действительно переменные, но их значение можно изменить только один раз (тогда они переходят из свободного состояния в связанное со значением).

Они выглядят как переменные обычных языков программирования, поэтому когда вы видите строку вида X=... наш мозг говорит нам: "Ага! Я знаю, что это такое. X - это переменная, а = - это оператор присваивания". И наш мозг почти что прав: X - это почти переменная, а = - это почти что оператор присваивания. (*Примечание*: Использование троеточия в коде Эрланга означает просто "код, который я не показываю".)

На самом деле, = - это оператор сопоставления по образцу, который ведет себя как оператор присвоения, в случае, когда X - это несвязанная (свободная) переменная.

И, наконец, *область видимости* переменной - это всегда лексический раздел кода, в котором она была определена. Так что, если X использовалась внутри определения варианта функции, ее значение никак "не убежит" из этих границ. В Эрланге не существует таких вещей, как глобальные или приватные переменные используемые многими вариантами одной функции. Если X появляется в разных функциях, это все разные X.

Соответствие Образцу

В большинстве языков символ = означает оператор присваивания. Тем не менее в Эрланге символ = означает операцию *сопоставления по образцу*. Выражение Л = П означает на самом деле следующее: вычисляется правая сторона (П) и, потом, она сопоставляется образцу в левой стороне выражения (Л).

Переменная, такая, например, как X, является простейшей формой образца (или паттерна). Как мы уже говорили ранее, переменной можно присваивать ее значение только один раз. Когда мы *первый* раз говорим X=ПервоеВыражение, Эрланг спрашивает сам себя: "Что я могу такого сделать, чтобы это утверждение было истинным?" Поскольку X пока не имеет никакого значения, то можно привязать X к значению Первого Выражения, и тогда это утверждение все станет корректным, и все будут счастливы.

Если потом, когда нибудь, мы скажем X=ДругоеВыражение, то это будет корректным только если ПервоеВыражение и ДругоеВыражение идентичны. Вот вам пример всего этого:

```
1> X = (2+4).
6
2> Y = 10.
10
3> X = 6.
6
4> X = Y.
=ERROR REPORT==== 27-Oct-2006::17:25:25 ===
Error in process <0.32.0> with exit value:
{{badmatch,10},[{erl_eval,expr,3}]}
5> Y = 10.
10
6> Y = 4.
=ERROR REPORT==== 27-Oct-2006::17:25:46 ===
Error in process <0.37.0> with exit value:
{{badmatch,4},[{erl_eval,expr,3}]}
7> Y = X.
=ERROR REPORT==== 27-Oct-2006::17:25:57 ===
Error in process <0.40.0> with exit value:
{{badmatch,6},[{erl_eval,expr,3}]}
```

Вот что здесь произошло: В строке 1 система вычислила значение 2+4 и ответила что это будет 6. После этой строки оболочка имеет следующее множество связанных переменных: {X -> 6}. А после третьей строки мы будем иметь следующее множество связей: {X -> 6, Y -> 10}.

Теперь мы добрались до строки 5. Еще до вычисления выражения мы знали что X -> 6 и, значит, сопоставление по образцу X = 6 проходит вполне успешно.

Потом мы говорим, что X = Y в строке 7. Но множество наших связей это {X -> 6, Y -> 10}, а значит, это сопоставление образцов не проходит и нам выдается сообщение об ошибке.

Утверждения в строках с 4 по 7-ую либо заканчиваются успешно, либо нет, в зависимости от значений X и Y. Теперь самое время изучить их всех хорошенько и убедиться, что вы все их отлично понимаете, перед тем как читать что-то далее.

На этой стадии вам может показаться, что я вам втолковываю очевидное. Все паттерны слева от "=" - это просто переменные, либо связанные, либо нет. Но как мы увидим далее мы можем там поставить очень сложные паттерны и сопоставить их с помощью оператора "=" . Мы с вами вернемся к этой теме после того как познакомимся с кортежами и списками, которые используются для хранения сложных, составных структур данных.

Почему одиночное присвоение делает мои программы лучше?

В Эрланге переменная - это просто ссылка на ее значение. В имплементации Эрланга связанная переменная представлена указателем на область хранения, содержащую значение. И это значение нельзя изменить.

То, что никак нельзя изменить значение связанной переменной, это очень важно и очень необычно, по сравнению с императивными языками программирования, такими как Си и Java.

Давайте посмотрим, что бы могло произойти, если бы было разрешено менять переменные. Давайте определим переменную X следующим образом:

```
1> X = 23.
23
```

Теперь используем X в вычислениях:

```
2> Y = 4 * X + 3.
95
```

А теперь, допустим, что нам можно изменить значение X:

```
3> X = 19.
```

К счастью, Эрланг этого не позволяет. Его компилятор начинает орать как ненормальный и выдает что-то вроде вот этого:

```
=ERROR REPORT==== 27-Oct-2006::13:36:24 ===
Error in process <0.31.0> with exit value:
{{badmatch,19},[{erl_eval,expr,3}]}
```

Что просто означает, что X не может быть 19, раз уж мы определили, что она будет 23.

Но, давайте просто допустим, что мы можем сделать это. Тогда значение У становится тоже некорректным, в том смысле, что мы уже не можем полагать утверждение за номером 2 как уравнение. Более того, если Х может менять свое значение много раз в программе и что-то в ней пошло не так, может потребоваться много усилий, чтобы понять, какое именно значение Х привело к ошибке и в какой именно точке программы вдруг появилось это неверное значение.

В Эрланге значения переменных не могут быть изменены, после того как они были установлены. Это упрощает поиск ошибок (debuging). Чтобы понять, почему это так, мы должны спросить сами себя, а что такое ошибка, вообще, и как мы узнаем о ней.

Одним из наиболее распространенных способов определить, что программа работает не корректно, является обнаружение, что ее переменная приняла неверное значение. В этом случае, вам придется искать в программе то место, где эта переменная получает свое неправильное значение. Если эта переменная меняется много-много раз и в различных местах вашей программы, тогда такие поиски места некорректного изменения ее значения, могут оказаться весьма и весьма трудными.

В Эрланге таких проблем просто нет. Переменная может быть установлена только один раз и более никогда не меняется. Так что если мы обнаруживаем, что у переменной неправильное значение, мы *немедленно* переходим к месту где эта переменная стала связанной с этим значением, а значит именно там и произошла ошибка.

Возможно вы сейчас задаетесь вопросом, а как, вообще, возможно писать программы без переменных? Как можно выразить что-то подобное X=X+1 средствами Эрланга? Ответ очень прост. Придумайте новую переменную, чье имя еще не было использовано (например, Х1) и напишите Х1=Х+1.

Отсутствие побочных эффектов означает возможность распараллелить нашу программу.

Технически, области памяти, которые могут быть модифицированы, называются *изменяемым состоянием*. Эрланг - это язык программирования с неизменяемыми состояниями.

Гораздо позднее в этой книге мы будем рассматривать вопросы программирования многоядерных процессоров. Когда дело доходит до этого, последствия отсутствия изменяемых состояний, просто огромные.

Если вы используете обычный язык программирования, такой как Си или Java для программирования многоядерных процессоров, тогда вы столкнетесь в проблемой, называемой *разделяемой памятью*. Чтобы не испортить эту разделяемую память, она должна быть заблокирована во время ее использования. Поэтому программы,

пользующиеся ей, не должны зависать или падать, во время ее использования.

В Эрланге нет изменяемых состояний, нет разделяемой памяти и нет блокировок. Это позволяет гораздо легче распараллеливать наши программы.

Числа с плавающей точкой.

Давайте попробуем несколько арифметических примеров с вещественными числами:

```
1> 5/3.
1.66667
2> 4/2.
2.00000
3> 5 div 3.
1
4> 5 rem 3.
2
5> 4 div 2.
2
6> Pi = 3.14159.
3.14159
7> R = 5.
5
8> Pi * R * R.
78.5397
```

Обратите внимание. В строке 1 число в конце выражения - это целое число 3. Точка означает конец выражения, а не десятичную точку. Если бы я хотел ввести здесь число с плавающей точкой, то надо было ввести 3.0.

"/" всегда возвращает вещественное число. Поэтому 4/2 равно 2.0000 (в оболочке Эрланга). N div M и N rem M используются для целочисленного деления и остатка. Поэтому 5 div 3 дает 1, а 5 rem 3 дает 2.

Вещественные числа должны иметь десятичную точку и, по крайней мере, одну цифру после нее. Когда вы делите два целых числа с помощью "/" , то результат автоматически конвертируется в число с плавающей точкой.

Атомы.

В Эрланге атомы используются для представления различных не-числовых констант.

Если вы использовали перечислимые типы в Си или Java, то вы уже использовали что-то очень похожее на атомы, сами того не подозревая.

Программисты на Си должны быть знакомы с соглашением об использовании символических имен констант, чтобы программа была более само-документирована. В типичной Си-программе определяется множество глобальных констант во включаемом файле, который состоит из большого числа их определений. Например, некоторый файл glob.h может содержать в себе:

```
#define OP_READ 1
#define OP_WRITE 2
#define OP_SEEK 3
...
#define RET_SUCCESS 223
...
```

Типичный Си код, использующий эти символические константы, может выглядеть следующим образом:

```
#include "glob.h"

int ret;
ret = file_operation(OP_READ, buff);
if( ret == RET_SUCCESS ) { ... }
```

В Си программах сами значения этих констант часто совсем не интересны. От них важно только, чтобы они были все разные и их можно было сравнивать на равенство и неравенство.

В Эрланге эквивалент этой программы может выглядеть следующим образом:

```
Ret = file_operation(op_read, Buff),
if Ret == ret_success ->
...
```

В Эрланге все атомы глобальны и это достигается без определений макросов или включаемых файлов.

Предположим, вы хотите написать программу, которая манипулирует днями недели. Как вы представите дни недели в Эрланге. Конечно же вы используете атомы monday, tuesday, ...

Все атомы начинаются с прописной буквы, а далее может быть буквенно-числовая последовательность, включающая в себя подчеркивание (_) или знак @ . Например:

red, december, cat, meters, yards, joe@somehost, and a_long_name. (Вы можете обнаружить, что точка (.) тоже может использоваться в атомах - это одна из неофициальных возможностей Эрланга.)

Атомы также могут быть заключены в одиночные кавычки ('). Используя такую их форму мы можем создавать атомы, которые начинаются с большой буквы (которые, иначе, считались бы переменными) или содержащие не только буквенно-числовые символы. Например: 'Monday', 'Tuesday', '+', '*', 'an atom with spaces'. Вы также можете поместить в одиночные кавычки и обычный атом, при этом 'a' будет в точности тем же самым что и просто a.

Значением атома является сам атом. Так что если вы дадите команду, которая состоит только из одного атома, то Эрланг и выведет значение этого атома.

```
1> hello.
hello
```

Может показаться странным, что мы обсуждаем значения атомов или значения целых чисел. Но, надо не забывать, что Эрланг - это функциональный язык программирования, в котором любое выражение должно иметь свое значение. Это включает в себя и целые числа и атомы, которые являются просто крайне примитивными формами выражений.

Кортежи

Предположим вам нужна группа из фиксированного числа объектов как единая сущность. Для этого мы будем использовать *кортеж* (tuple). Вы можете легко создать кортеж просто заключив нужные вам значения в фигурные скобки и разделив их запятыми. Например, если вы хотите записать чье-либо имя и рост, вы можете использовать {joe,1.82}. Этот кортеж содержит атом и число с плавающей точкой.

Кортежи похожи на структуры в Си, но при условии, что они анонимные. В Си переменная P типа точка может быть определена следующим образом:

```
struct point {
    int x;
    int y;
} P;
```

Доступ к полям этой структуры осуществляется в Си с использованием оператора точка. Так что для установки значений x и y в y этой переменной вы можете написать:

```
P.x = 10; P.y = 45;
```

В Эрланге нет определения типов, чтобы создать "точку" и мы можем написать просто:

```
P = {10, 45}
```

Эта запись создает кортеж и привязывает его к переменной P. В отличии от Си поля кортежа не имеют имен. И поскольку кортеж содержит в себе только несколько целых чисел, мы должны сами помнить для чего они используются. Чтобы облегчить такое запоминание, обычной практикой является использование атома в качестве первого элемента кортежа, который описывает, что представляет собой этот кортеж. То есть мы лучше напишем {point, 10, 45}, а не {10, 45}, что сделает нашу программу намного более удобочитаемой. (Это, конечно, вовсе не требование языка, а просто рекомендуемый стиль программирования.)

Кортежи могут быть вложенными друг в друга. Предположим мы хотим представить некоторые факты о человеке - его имя, рост, размер ноги и цвет глаз. мы можем сделать это следующим способом:

```
1> Person = {person,
        {name, joe},
        {height, 1.82},
        {footsize, 42},
        {eyecolour, brown}}.
```

Заметьте, что для цвета глаз (в последней строчке) мы использовали атомы как для обозначения имени поля кортежа, так и для задания его значения.

Создание кортежей

Кортежи создаются автоматически, когда их объявляют в программе и автоматически уничтожаются, если их больше не используют. В Эрланге используется сборщик мусора для освобождения неиспользуемой памяти, так что нам не надо беспокоиться насчет выделения и освобождения памяти.

Если при построении кортежа вы используете переменную, тогда он будет разделять с ней то же значение данных, на структуру которых она и указывает. Приведем пример:

```
2> F = {firstName, joe}.
{firstName,joe}
3> L = {lastName, armstrong}.
{lastName,armstrong}
4> P = {person, F, L}.
```

```
{person,{firstName,joe},{lastName,armstrong}}
```

Если вы попробуете создать структуру данных с неопределенной (т.е. несвязанной) переменной, вы получите ошибку. Рассмотрим это также на примере, используя неопределенную переменную Q:

```
5> {true, Q, 23, Costs}.
1: variable 'Q' is unbound
```

Это просто означает, что переменная Q не определена.

Извлечение данных из кортежа

Ранее мы говорили, что "=" , которое выглядит как оператор присваивания, на самом деле таковым не является, а представляет собой, на самом деле, оператор сопоставления по образцу. Вы тогда еще могли удивиться, и зачем это нам надо было быть такими педантичными. Ну, дело в том, что сопоставление по образцу — это фундаментальный механизм в Эрланге и он используется в нем для множества разных вещей. Он используется для извлечения данных из различных структур данных, а также для контроля порядка исполнения внутри функции и для выбора, какое из пришедших сообщений обработать в параллельных программах из числа посланных данному процессу.

Если нам нужно извлечь какие-то данные из кортежа, то мы также будем использовать оператор сопоставления "=" .

Давайте вернемся к нашему кортежу, который представляет собой точку на экране:

```
1> Point = {point, 10, 45}.
{point, 10, 45}.
```

Предположим мы хотим из него извлечь данные из его полей в две переменные X и У. Поступим следующим образом:

```
2> {point, X, Y} = Point.
{point,10,45}
3> X.
10
4> Y.
45
```

В команде под номером 2 переменная X привязывается к 10, а У к 45. Значением выражения Л=П является значение П (правой части) , поэтому оболочка печатает {point,10,45}.

Как можно видеть, кортежи с обоих сторон знака равенства должны иметь одинаковое число элементов и соответствующие элементы с обоих сторон должны иметь одинаковое значение.

А теперь предположим, что вы ввели что-то вроде вот этого:

```
5> {point, C, C} = Point.
=ERROR REPORT==== 28-Oct-2006::17:17:00 ===
Error in process <0.32.0> with exit value:
{{badmatch,{point,10,45}},[{erl_eval,expr,3}]}
```

Что же тут произошло? Образец {point, C, C} не может соответствовать {point, 10, 45} так как C не может быть одновременно и 10 и 45. Поэтому сопоставление по образцу не удалось и система выдала ошибку. (Для читателей знакомых с Прологом: Эрланг считает несоответствие образцов ошибкой и не имеет отката (backtrack).)

Если у вас имеется сложный кортеж и вы хотите извлечь из него данные, то вы можете написать кортеж такой же формы (структуры) как и ваш, но в тех местах откуда вы хотите извлечь данные поместите несвязанные переменные. (Этот метод извлечения данных путем сопоставления по образцу называется *унификацией* и используется во множестве функциональных и логических языков программирования.)

Чтобы продемонстрировать все это мы, для начала, определим переменную Person которая будет содержать в себе сложную структуру данных:

```
1> Person={person,{name,{first,joe},{last,armstrong}},{footsize,42}}.
{person,{name,{first,joe},{last,armstrong}},{footsize,42}}
```

Теперь мы напишем паттерн (образец) для извлечения имени этой персоны:

```
2> {_,{_,{_,Who},_},_} = Person.
{person,{name,{first,joe},{last,armstrong}},{footsize,42}}
```

И, наконец, мы распечатаем значение Who:

```
3> Who.
joe
```

Заметьте, что в предыдущем примере мы написали символ _ вместо переменных, которые нам сейчас не интересны. Символ _ называется *анонимной переменной*. В отличии от обычных переменных, несколько _ в одном паттерне, вовсе не привязываются к одному и тому-же значению.

Списки

Мы используем в жизни списки для хранения самых различных вещей: того, что надо купить в магазине, имен планет, результатов возвращенных вашей функцией простых чисел и так далее.

Мы создаем список в Эрланге путем заключения списка его элементов в квадратные скобки и разделяя их запятыми. Вот как мы можем создать список покупок:

```
1> ThingsToBuy = [{apples,10},{pears,6},{milk,3}].
[{apples,10},{pears,6},{milk,3}]
```

Сами элементы списка могут быть произвольного типа, так что мы можем, например, написать:

```
2> [1+7,hello,2-2,{cost, apple, 30-20},3].
[8,hello,0,{cost,apple,10},3]
```

Терминология

Мы называем первый элемент списка *головой* списка. А если удалить из списка его голову, то все оставшееся называется *хвостом* списка.

Например, если у нас есть список [1,2,3,4,5], то его головой будет целое число 1, а хвостом список [2,3,4,5]. Заметьте, что головой списка может быть все что угодно, но хвост списка это, обычно, тоже список.

Доступ к голове списка - это очень быстрая операция, так что чуть ли не все функции обработки списков начинают с выделения его головы, ее обработки, а потом переходят, аналогично к хвосту списка.

Определение списков

Если T - это список то [H|T] это тоже список в котором голова - это H, а хвост это T. Вертикальная черта отделяет голову списка от его хвоста. [] - это пустой список.

Когда бы мы не создавали список с использованием конструктора [...|T] , мы должны быть уверены, что T - это список. Если это так, то новый список будет "правильно сформирован". А если T это не список, тогда такой новый список называют "неправильный список". Большинство функций библиотечных функций полагают, что список для них был правильно сформирован и не будут работать с неправильными списками.

Мы можем добавить более одного элемента в начало T если напишем [E1,E2,..,En|T].

Например:

```
3> ThingsToBuy1 = [{oranges,4},{newspaper,1}|ThingsToBuy].
[{oranges,4},{newspaper,1},{apples,10},{pears,6},{milk,3}]
```

Выделение элементов из списка.

Как и из всего остального, мы можем извлекать элементы из списка с помощью оператора сопоставления по образцу. Если у нас имеется не пустой список L , тогда выражение [X|Y]=L , где X и У - это несвязанные переменные, поместит голову списка в X, а хвост списка - в У.

Итак, мы пришли в магазин и у нас собой наш список необходимых покупок - ThingsToBuy1. Первым делом нам надо распаковать этот список на голову и хвост:

```
4> [Buy1|ThingsToBuy2] = ThingsToBuy1.
[{oranges,4},{newspaper,1},{apples,10},{pears,6},{milk,3}]
```

Это успешно связывает

```
Buy1 -> {oranges,4}
```

и

```
ThingsToBuy2 -> [{newspaper,1}, {apples,10}, {pears,6}, {milk,3}].
```

Мы идем и покупаем апельсины, а потом мы хотим извлечь еще несколько позиций:

```
5> [Buy2,Buy3|ThingsToBuy3] = ThingsToBuy2.
{newspaper,1},{apples,10},{pears,6},{milk,3}]
```

Это приводит к тому, что Buy2 -> {newspaper,1}, Buy3 -> {apples,10}, и ThingsTo-

```
Buy3 -> [{pears,6},{milk,3}].
```

Строки

Строго говоря, в Эрланге нет строчек. *Строки*, на самом деле, это просто списки целых чисел. строки заключаются в двойные кавычки ("), так что мы можем, например, написать:

```
1> Name = "Hello".
```

```
"Hello"
```

> *Примечание*: В некоторых языках строки можно заключать и в двойные и в одиночные кавычки. В Эрланге вы можете использовать только двойные кавычки.

"Hello" - это просто краткая запись списка целых чисел, которые представляют отдельные буквы в этой строчке.

Когда оболочка распечатывает значение списка, она печатает список как строку, но только тогда, когда все целые в нем представляют собой печатные буквы:

```
2> [1,2,3].
[1,2,3]
3> [83,117,114,112,114,105,115,101].
"Surprise"
4> [1,83,117,114,112,114,105,115,101].
[1,83,117,114,112,114,105,115,101].
```

В выражении номер 2 список [1,2,3] напечатан без изменения. Это потому, что 1, 2 и 3 - это не печатные символы.

В выражении номер 3 все позиции в списке - это печатные символы, поэтому он печатается как строка.

Выражение 4 почти точно такое как и выражение 3, за исключением того что список начинается с 1, что не является печатным символом. Из-за этого весь список печатается без конверсии в строку.

нам совсем не нужно знать, какое целое число представляет конкретный символ. Для этого мы можем использовать "долларовый синтаксис". Например, $a - это на самом деле целое число, которое представляет собой символ "a" и так далее.

```
5> I = $s.
115
6> [I-32,$u,$r,$p,$r,$i,$s,$e].
"Surprise"
```

Набор символов используемый в строках

Буквы в строке представляются кодами из таблицы Latin-1 (ISO-8859-1). Например, строка содержащая шведское имя Håkan будет закодирована как [72,229,107,97,110].

> *Примечание*: Если вы ведете [72,229,107,97,110] как выражение в оболочке Эрланга, то, возможно, вы не получите то, что вы ожидали:

```
1> [72,229,107,97,110].
"H\\345kan"
```

Что случилось с “Håkan”, куда он подевался? На самом деле это никак не связано с Эрлангом, а только с локальной кодировочной таблицей для вашего терминала.

Эрланг просто считает строку списком целых чисел в какой-то кодировке и более его ничего не волнует. Если они вдруг печатные согласно кодам из Latin-1, тогда они должны корректно отображаться (если корректны установки на вашем терминале).

Снова о сопоставлении по образцу.

В завершении этой главы мы еще раз вернемся к сопоставлению по образцу.

В следующей таблице приведено несколько примеров паттернов (образцов) и терминов (структур данных Эрланга). Третья колонка этой таблицы, называемая *Результат*, показывает соответствует ли образец термину и, если это так, то и создаваемые, при этом, связи переменных. Посмотрите на все эти примеры и убедитесь, что вы все их действительно понимаете:

Паттерн Термин Результат

{X,abc} {123,abc} *Успех*. X -> 123 {X,Y,Z} {222,def,"cat"} *Успех*. X -> 222, Y -> def, Z -> "cat" {X,Y} {333,ghi,"cat"} *Провал*. У кортежей разная структура X true *Успех*. X -> true {X,Y,X} {{abc,12},42,{abc,12}} *Успех*. X -> {abc,12}, Y -> 42 {X,Y,X} {{abc,12},42,true} *Провал*. X не может быть одновременно и {abc,12} и true [H|T] [1,2,3,4,5] *Успех*. H -> 1, T -> [2,3,4,5] [H|T] "cat" *Успех*. H -> 99, T -> "at" [A,B,C|T] [a,b,c,d,e,f] *Успех*. A -> a, B -> b, C -> c, T -> [d,e,f]

Если вы не уверены в каком то из этих примеров, тогда попытайтесь ввести соответствующее выражение вида Паттерн = Термин в оболочку Эрланга и посмотреть что из этого выйдет.

Например:

```
1> {X, abc} = {123, abc}.
{123,abc}.
2> X.
123
3> f().
ok
4> {X,Y,Z} = {222,def,"cat"}.
```

```
{222,def,"cat"}.
5> X.
222
6> Y.
def
...
```

> *Примечание*: команда f(). говорит оболочке Эрланга забыть все связывания, которые у нее были до этого. После этой команды все переменные становятся свободными. Так что X в строке 4 никак не связана с X в строках 1 и 2.

Теперь, когда мы хорошо освоились с базовыми типами данных и с идеями одиночного присвоения и сопоставления по образцу, мы можем немного ускорить темп и рассмотреть как определять функции и модули. Давайте рассмотрим это в следующей главе.