

Содержание

Предисловие	14
Глава 1. Почему появился Rust?	18
Типобезопасность	19
Глава 2. Краткий обзор Rust	23
Скачивание и установка Rust	23
Простая функция	25
Написание и выполнение автономных тестов	27
Обработка аргументов командной строки	28
Простой веб-сервер	32
Конкурентность	37
Что такое множество Мандельброта	38
Разбор пары аргументов командной строки	42
Отображение пикселей на комплексные числа	44
Рисование множества	46
Запись файла изображения	47
Конкурентная программа рисования множества Мандельброта	48
Выполнение программы рисования множества Мандельброта	52
Невидимая безопасность	54
Глава 3. Базовые типы	55
Машинные типы	58
Целые типы	58
Типы с плавающей точкой	60
Тип <code>bool</code>	62
Символы	62
Кортежи	64
Указательные типы	65
Ссылки	66
Боксы	66
Простые указатели	66
Массивы, векторы и срезы	67
Массивы	67
Вектор	68
Поэлементное построение векторов	71
Срезы	71
Строковые типы	73
Строковые литералы	73
Байтовые строки	74
Строки в памяти	74
Тип <code>String</code>	75
Использование строк	76
Другие типы, похожие на строки	77
Более сложные типы	77

Глава 4. Владение	78
Владение	79
Передача владения	84
Другие операции с передачей	88
Передача владения и поток управления	89
Передача владения и индексированное содержимое	90
Копируемые типы: исключения из правила передачи владения	92
Rc и Arc: совместное владение	95
Глава 5. Ссылки	98
Ссылки как значения	101
Сравнение ссылок в Rust и в C++	101
Присваивание ссылкам	102
Ссылки на ссылки	103
Сравнение ссылок	103
Ссылки не бывают нулевыми	104
Заимствование ссылок на произвольные выражения	104
Ссылки на срезы и объекты характеристик	105
Безопасность ссылок	105
Заимствование локальной переменной	105
Получение ссылок в качестве параметров	108
Передача ссылок в качестве аргументов	110
Возврат ссылок	111
Структуры, содержащие ссылки	112
Различные параметрические времена жизни	114
Опускание параметрического времени жизни	115
Разделяемость и изменяемость	117
Оружие против моря объектов	123
Глава 6. Выражения	126
Язык выражений	126
Блоки и точки с запятой	127
Объявления	128
if и match	130
Циклы	132
Выражение return	134
Зачем в Rust цикл loop	135
Вызовы функций и методов	136
Поля и элементы	137
Операторы ссылки	139
Арифметические, поразрядные, логические операторы и операторы сравнения	139
Присваивание	140
Приведение типов	140
Замыкания	141
Приоритеты и ассоциативность	142
Что дальше	144
Глава 7. Обработка ошибок	145
Паника	145
Раскрутка стека	146

Снятие процесса	147
Тип Result	147
Обнаружение ошибок	148
Псевдонимы типа Result	149
Печать информации об ошибках	150
Распространение ошибок	151
Работа с ошибками нескольких типов	152
Ошибки, которых «не может быть»	153
Игнорирование ошибок	155
Обработка ошибок в main()	155
Обявление пользовательского типа ошибки	156
Почему именно тип Result?	157
Глава 8. Крейты и модули	158
Крейты	158
Сборочные профили	161
Модули	161
Модули в отдельных файлах	162
Пути и импорт	164
Стандартная прелюдия	166
Артикулы – строительные блоки в Rust	166
Превращение программы в библиотеку	168
Каталог src/bin	170
Атрибуты	171
Тесты и документация	173
Интеграционные тесты	175
Документация	176
Doc-тесты	178
Задание зависимостей	180
Версии	181
Cargo.lock	182
Публикация крейтов на сайте crates.io	183
Рабочие пространства	185
Прочие вкусности	185
Глава 9. Структуры	187
Структуры с именованными полями	187
Кортежеподобные структуры	190
Безэлементные структуры	191
Размещение структуры в памяти	191
Определение методов с помощью ключевого слова impl	192
Универсальные структуры	195
Структуры с параметрическим временем жизни	196
Выведение стандартных характеристик для структурных типов	197
Внутренняя изменяемость	198
Глава 10. Перечисления и образцы	202
Перечисления	203
Перечисления, содержащие данные	205
Перечисления в памяти	206
Обогащенные структуры данных на основе перечислений	206

Универсальные перечисления.....	208
Образцы.....	211
Литералы, переменные и метасимволы в образцах	213
Кортежные и структурные образцы	215
Ссылочные образцы	216
Сопоставление с несколькими возможностями	218
Охранные выражения	219
@-образцы.....	219
Где еще используются образцы	220
Построение двоичного дерева	221
Общая картина.....	222
Глава 11. Характеристики и универсальные типы	224
Использование характеристик	226
Объекты характеристик	227
Размещение объекта характеристики в памяти	228
Универсальные функции	229
Что использовать.....	232
Определение и реализация характеристик	233
Методы по умолчанию.....	234
Характеристики и сторонние типы	235
Употребление Self в характеристиках.....	237
Подхарактеристики	238
Статические методы	239
Полностью квалифицированные вызовы методов	240
Характеристики, определяющие связи между типами	241
Ассоциированные типы, или Как работают итераторы	242
Универсальные характеристики, или Как работает перегрузка операторов	245
Парные характеристики, или Как работает rand::random()	245
Обратное конструирование ограничений	247
Заключение	250
Глава 12. Перегрузка операторов	251
Арифметические и поразрядные операторы	252
Унарные операторы	254
Бинарные операторы	255
Составные операторы присваивания	255
Сравнение на равенство.....	257
Сравнение на больше-меньше	260
Index и IndexMut	261
Прочие операторы.....	264
Глава 13. Вспомогательные характеристики	265
Характеристика Drop.....	266
Характеристика Sized	268
Характеристика Clone	271
Характеристика Copy.....	272
Характеристики Deref и DerefMut	273
Характеристика Default.....	276
Характеристики AsRef и AsMut.....	277
Характеристики Borrow и BorrowMut.....	279

Характеристики From и Into	280
Характеристика ToOwned	282
Borrow и ToOwned за работой: скромное копирование при записи	283
Глава 14. Замыкания	285
Захват переменных	286
Замыкания с заимствованием	287
Замыкания с кражей	287
Типы функций и замыканий	289
Производительность замыканий	291
Замыкания и безопасность	292
Замыкания, которые убивают	293
FnOnce	293
FnMut	295
Обратные вызовы	297
Эффективное использование замыканий	299
Глава 15. Итераторы	302
Характеристики Iterator и IntoIterator	303
Создание итераторов	305
Методы iter и iter_mut	305
Реализации характеристики IntoIterator	305
Метод drain	307
Другие источники итераторов	308
Адаптеры итераторов	309
map и filter	309
filter_map и flat_map	311
scan	313
take и take_while	314
skip и skip_while	315
peekable	316
fuse	317
Обратимые итераторы и rev	317
inspect	318
chain	319
enumerate	319
zip	320
by_ref	321
cloned	322
cycle	322
Потребление итераторов	323
Простое аккумулирование: count, sum, product	323
max, min	324
max_by, min_by	324
max_by_key, min_by_key	324
Сравнение последовательностей	325
any и all	326
position, rposition и ExactSizeIterator	326
fold	327
nth	327
last	328

find	328
Построение коллекций: collect и FromIterator	328
Характеристика Extend	330
partition	331
Реализация собственных итераторов	332
Глава 16. Коллекции	336
Обзор	337
Тип Vec<T>.....	338
Доступ к элементам.....	338
Итерирование	340
Увеличение и уменьшение вектора	340
Соединение	343
Расщепление.....	343
Перестановка элементов	345
Сортировка и поиск.....	345
Сравнение срезок	347
Случайные элементы	347
В Rust отсутствуют ошибки недействительности	347
Тип VecDeque<T>	348
Тип LinkedList<T>	350
Тип BinaryHeap<T>	350
Типы HashMap<K, V> и BTreeMap<K, V>	351
Записи	354
Обход отображения	356
Типы HashSet<T> и BTreeSet<T>	356
Обход множества.....	357
Когда равные значения различны	357
Операции над множествами как единым целым	358
Хеширование	359
Применение пользовательского алгоритма хеширования	360
За пределами стандартных коллекций	361
Глава 17. Строки и текст	362
Общие сведения о Юникоде	362
ASCII, Latin-1 и Юникод.....	362
UTF-8.....	363
Направление текста.....	365
Символы (char).....	365
Классификация символов	365
Работа с цифрами.....	366
Преобразование регистра символов	366
Преобразование в целое число и обратно	367
Типы String и str.....	367
Создание значений типа String	368
Простая инспекция	369
Дописывание и вставка текста	369
Удаление текста	371
Соглашения о поиске и итерировании	371
Образцы для поиска текста	372
Поиск и замена	372

Обход текста.....	373
Усечение	375
Преобразование регистра.....	376
Создание значений других типов из строк	376
Преобразование других типов в строки	376
Заимствование в виде других текстообразных типов	377
Доступ к байтам текста в кодировке UTF-8	378
Порождение текста из данных в кодировке UTF-8	378
Откладывание выделения памяти.....	379
Строки как универсальные коллекции	381
Форматирование значений	381
Форматирование текстовых значений	383
Форматирование чисел.....	384
Форматирование прочих типов	385
Форматирование значений для отладки	386
Форматирование указателей для отладки.....	387
Ссылка на аргументы по индексу или по имени	387
Динамическая ширина и точность.....	388
Форматирование пользовательских типов	389
Применение языка форматирования в своем коде	391
Регулярные выражения	392
Основы работы с Regex	392
Ленивое построение значений типа Regex.....	393
Нормализация.....	394
Формы нормализации	395
Крейт unicode-normalization.....	396
Глава 18. Ввод и вывод.....	398
Читатели и писатели	399
Читатели.....	400
Буферизованные читатели	401
Чтение строк	402
Собирание строк	405
Писатели	405
Файлы	406
Поиск	407
Другие типы читателей и писателей.....	407
Двоичные данные, сжатие и сериализация.....	409
Файлы и каталоги	410
Типы OsStr и Path	410
Методы типов Path и PathBuf	412
Функции доступа к файловой системе	413
Чтение каталогов.....	414
Платформенно-зависимые средства	416
Средства сетевого программирования	417
Глава 19. Конкурентность.....	420
Вилочный параллелизм	421
Функции spawn и join.....	423
Обработка ошибок в потоках	425
Разделение неизменяемых данных между потоками	426

Rayon	428
И снова о множестве Мандельброта	430
Каналы.....	431
Отправка значений	433
Получение значений	436
Выполнение конвейера.....	437
Возможности и производительность каналов	438
Потокобезопасность: Send и Sync	440
Отправка объектов почти любого итератора по каналу.....	442
За пределами конвейеров.....	443
Разделяемое изменяемое состояние.....	444
Что такое мьютекс?	444
Мьютексы в Rust	446
mut и Mutex.....	448
Почему мьютексы – не всегда хорошая идея	448
Взаимоблокировка	449
Отравленные мьютексы.....	450
Каналы с несколькими производителями и мьютексом	450
Блокировки чтения-записи (RwLock)	451
Условные переменные (Condvar).....	452
Атомарные типы	453
Глобальные переменные.....	455
Как выглядит написание конкурентного кода на Rust	457
Глава 20. Макросы.....	458
Основы макросов.....	459
Основы макрорасширения	460
Непредвиденные последствия	461
Повторение	463
Встроенные макросы.....	465
Отладка макросов	466
Макрос json!	467
Типы фрагментов	468
Рекурсия в макросах.....	471
Использование характеристик совместно с макросами	471
Области видимости и гигиена.....	473
Импорт и экспорт макросов	476
Предотвращение синтаксических ошибок при сопоставлении.....	477
За пределами macro_rules!	478
Глава 21. Небезопасный код.....	480
Небезопасность от чего?	481
Unsafe-блоки	482
Пример: эффективный тип ASCII-строки	483
Unsafe-функции	485
Unsafe-блок или unsafe-функция?.....	487
Неопределенное поведение.....	488
Небезопасные характеристики.....	490
Простые указатели	492
Безопасное разыменование простых указателей	494
Пример: RefWithFlag	495

Нулевые указатели	498
Размеры и выравнивание типов	498
Арифметика указателей	499
Передача в память и из памяти.....	500
Пример: GapBuffer	503
Безопасность паники в небезопасном коде	510
Иноязычные функции: вызов функций на С и С++ из Rust.....	511
Поиск общего представления данных	511
Объявление иноязычных функций и переменных.....	514
Использование библиотечных функций	515
Низкоуровневый интерфейс с libgit2.....	519
Безопасный интерфейс к libgit2.....	524
Заключение	535
Предметный указатель	536
Об авторах	548
Колофон	549

Предисловие

Язык Rust предназначен для системного программирования.

В наши дни эта фраза требует пояснений, потому что многие современные программисты не знают, что такое системное программирование. Однако же оно лежит в основе всего, что мы делаем.

Вы опускаете крышку ноутбука. Операционная система замечает это, приостанавливает все работающие программы, выключает экран и переводит компьютер в режим ожидания. Позже вы поднимаете крышку: на экран и на все остальные компоненты снова подается питание, и все программы продолжают работу с того места, где остановились. Мы считаем это само собой разумеющимся. Однако системные программисты написали немало кода, для того чтобы все происходило именно так.

Системное программирование предназначено для разработки:

- операционных систем;
- драйверов;
- файловых систем;
- баз данных;
- кода, работающего в очень дешевых устройствах или устройствах, требующих повышенной надежности;
- криптографических приложений;
- мультимедийных кодеков (программ, которые читают и записывают аудио, видео и графические файлы);
- мультимедийных приложений (например, для распознавания речи и редактирования фотографий);
- управления памятью (например, для реализации сборщика мусора);
- отрисовки текста (преобразования совокупности текста и шрифтов в набор пикселей);
- высокоуровневых языков программирования (например, JavaScript и Python);
- средств сетевого программирования;
- средств виртуализации и контейнеров программ;
- математических моделей;
- игр.

Короче говоря, системное программирование – это программирование в условиях *ограниченности ресурсов*, когда каждый байт и каждый такт процессора имеют значение.

Объем системного кода, участвующего в поддержке самого простого приложения, ошеломляет.

Эта книга – не учебник по системному программированию. В ней рассматриваются многочисленные детали управления памятью, которые могут показаться

излишне заумными тому, кто раньше не занимался системным программированием. Но профессиональный системный программист найдет в языке Rust нечто удивительное: новый инструмент, который устраняет серьезные и давно знакомые проблемы, преследовавшие нашу отрасль в течение многих десятилетий.

Для кого написана эта книга

Если вы – системный программист, созревший для поиска альтернативы C++, то эта книга для вас.

Если у вас есть опыт разработки на каком-нибудь языке программирования, будь то C#, Java, Python, JavaScript или еще что-то, то эта книга будет полезна и вам. Однако изучить Rust недостаточно. Чтобы взять от языка максимум, необходимо иметь хоть небольшой опыт системного программирования. Мы рекомендуем читать эту книгу параллельно с реализацией каких-то побочных проектов системного программирования на Rust. Займитесь чем-то таким, чего никогда не делали раньше, чем-то, где могут в полной мере проявиться быстродействие, конкурентность и безопасность Rust. На какие-то идеи может навести приведенный выше перечень.

Зачем мы написали эту книгу

Мы решили написать книгу, которой нам недоставало, когда мы взялись за изучение Rust. Наша цель – четко и ясно представить новые важные идеи Rust во всей полноте, не пренебрегая деталями, чтобы свести к минимуму познание методом проб и ошибок.

Обзор содержания книги

Первые пять глав содержат введение в Rust и знакомят с фундаментальными типами данных, а также с базовыми понятиями владения и ссылки. Эти главы рекомендуется читать последовательно.

В главах 6–10 рассматриваются основные конструкции языка: выражения, обработка ошибок, крейты и модули, структуры, перечисления и образцы. Все читать необязательно, но главу, посвященную обработке ошибок, пропускать не стоит – поверьте нам.

В главе 11 рассматриваются характеристики и универсальные типы – последние две концепции, знать о которых необходимо. Характеристики (trait) похожи на интерфейсы в Java или C#. Кроме того, в Rust это основной способ включения пользовательских типов в язык. В главе 12 показано, как с помощью характеристик поддерживается перегрузка операторов, а в главе 13 рассматриваются многочисленные служебные характеристики.

Понимание характеристик и универсальных типов – ключ к остальной части книги. В главах 14 и 15 описываются замыкания и итераторы – два важнейших средства, без которых вам не обойтись. Прочие главы можно читать в любом порядке или лишь по мере необходимости. В них рассматриваются коллекции, строчки и обработка текста, ввод-вывод, конкурентность, макросы и небезопасный код.

ГРАФИЧЕСКИЕ ВЫДЕЛЕНИЯ

В книге применяются следующие графические выделения:

Курсив

Новые термины, имена и расширения имен файлов.

Моноширинный

Листинги программ, а также элементы кода в основном тексте: имена переменных и функций, базы данных, типы данных, переменные окружения, предложения и ключевые слова языка.

Моноширинный полужирный

Команды и иной текст, который пользователь должен вводить точно в указанном виде.

Моноширинный курсив

Текст, вместо которого следует подставить значения, заданные пользователем или определяемые контекстом.

СКАЧИВАНИЕ ИСХОДНОГО КОДА ПРИМЕРОВ

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.dmk.ru на странице с описанием соответствующей книги.

Мы высоко ценим, хотя и не требуем, ссылки на наши издания. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «Программирование на языке Rust» Джима Блэнди, Джейсона Орендорфф (O'Reilly, ДМК Пресс). Copyright © 2018 Jim Blandy and Jason Orendorff, 978-1-491-92728-1 (англ.), 978-5-97060-236-2 (рус.).

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу dmkpress@gmail.com.

ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной

из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

БЛАГОДАРНОСТИ

Книга, которую вы держите в руках, стала значительно лучше благодаря усилиям официальных технических рецензентов: Брайана Андерсона (Brian Anderson), Мэтта Брабека (Matt Brubeck), Дж. Дэвида Эйзенберга (J. David Eisenberg) и Джека Моффита (Jack Moffit). Было также много неофициальных рецензентов, которые читали ранние варианты и давали ценные советы: Джек Уолден (Jeff Walden), Джо Уолкер (Joe Walker), Николас Пьеррон (Nicolas Pierron), Эдди Брюэл (Eddy Bruel), Ярослав Шнайдр (Jaroslav Šnajdr), Джейфри Лим (Jeffrey Lim) и Джан-Карло Паскутто (Gian-Carlo Pascutto). Книга по программированию, как и любое другое предпринятие в этой области, много выигрывает от сообщений об ошибках. Спасибо.

Мы благодарны корпорации Mozilla и нашим непосредственным руководителям – без них эта работа не состоялась бы.

Спасибо сотрудникам издательства O'Reilly, помогавшим довести этот проект до конца, а особенно нашим редакторам Брайану Макдональду и Джеку Блэйелу (Jeff Bleiel).

И никакими словами не выразить горячую благодарность нашим женам и детям за их непоколебимую любовь, энтузиазм, терпение, снисходительность и готовность прощать.

Глава 1

Почему появился Rust?

Есть контексты – например, те, на которые ориентирован Rust, – когда десяти- или хотя бы двукратное превосходство в скорости решает все. От этого зависит судьба программной системы на рынке – точно так же, как на рынке оборудования.

— Грейдон Хоар

В наши дни все компьютеры стали параллельными... Параллельное программирование – это и есть программирование.

— Майкл Маккул и др.
«Структурное параллельное программирование»

Дефект в анализаторе TrueType использовался противниками для шпионажа; безопасность должна быть свойственна любой программе.

— Энди Уинго

За те пятьдесят лет, что мы используем языки высокого уровня для написания операционных систем, языки системного программирования прошли долгий путь, но две проблемы оказались особенно трудными.

- Трудно написать безопасный код. Особенно трудно корректно управлять памятью в программах на С и С++. С последствиями – брешами в системе защиты – пользователи сталкиваются на протяжении десятков лет. Началось это с червя Морриса в 1988 году.
- Очень трудно писать многопоточный код, а ведь это единственный способ задействовать возможности современных компьютеров. Даже опытные программисты подходят к многопоточному коду с опаской, поскольку конкурентность может стать причиной новых классов ошибок, а хорошо знакомые ошибки становятся гораздо труднее воспроизвести.

Для того и придуман Rust: безопасный конкурентный язык, не уступающий по производительности С и С++.

Rust – новый язык системного программирования, разработанный Mozilla и сообществом. Подобно С и С++, Rust предоставляет средства точного контроля над использованием памяти и поддерживает близкое соответствие между примитивными операциями языка и машинными командами, что позволяет заранее оценить быстродействие написанного кода. Rust ставит перед собой те же цели,

которые Бъярн Страуструп сформулировал для C++ в статье «Abstraction and the C++ machine model»:

Вообще говоря, реализации C++ придерживаются принципа нулевых издержек: не платить за то, чем не пользуешься. И более того: то, чем пользуешься, нельзя закодировать вручную более эффективно.

К этому Rust добавляет еще две цели: безопасная работа с памятью и надежная конкурентность.

Ключом к выполнению обещанного является новаторская система проверяемых на этапе компиляции владения, передачи и заимствования, тщательно спроектированная так, что дополняет гибкую систему статической типизации в Rust. Система владения устанавливает время жизни каждого значения, что делает не нужным сборку мусора в ядре языка и обеспечивает надежные, но вместе с тем гибкие интерфейсы для управления такими ресурсами, как сокеты и описатели файлов. Передача (move) позволяет передать значение от одного владельца другому, а заимствование (borrowing) – использовать значение временно, не изменяя владельца. Поскольку многие программисты раньше не встречались с подобными механизмами, мы подробно остановимся на них в главах 4 и 5.

Те же самые правила владения лежат в основе модели надежной конкурентности в Rust. В большинстве языков связь между мьютексом и данными, которые он защищает, описывается в комментариях; Rust может на этапе компиляции проверить, что программа удерживает мьютекс в течение всего времени, пока обращается к данным. Как правило, язык лишь рекомендует не использовать структуру данных, после того как она передана другому потоку; Rust проверяет, что структура действительно не используется. Rust способен предотвратить гонки за данные на этапе компиляции.

Rust не является настоящим объектно-ориентированным языком, хотя некоторые объектно-ориентированные черты в нем присутствуют. Rust не является функциональным языком, хотя стремится сделать результат вычисления более явно выраженным, как в функциональных языках. В определенной степени Rust напоминает С и C++, но многие идиомы этих языков к Rust неприменимы, так что типичный код на Rust имеет лишь поверхностное сходство с кодом на С или C++. Лучше отложить суждение о том, что же такое Rust, на потом – когда вы освоитесь с языком.

Чтобы оценить дизайн языка в реальных условиях, корпорация Mozilla разработала на Rust новый движок браузера, Servo. Потребности Servo и цели Rust отлично согласуются: браузер должен работать быстро и обрабатывать данные из ненадежных источников безопасно. Servo использует безопасную конкурентность Rust, чтобы положить все ресурсы машины на службу задачам, которые было бы непрактично распараллеливать на С или C++. На самом деле Servo и Rust разрабатывались вместе: в Servo использовались самые последние возможности языка, а Rust эволюционировал с учетом пожеланий разработчиков Servo.

Типобезопасность

Rust – типобезопасный язык. Но что понимается под «типобезопасностью»? Безопасность – это, конечно, хорошо, но от чего именно мы стараемся обезопасить себя?

Ниже приведено определение «неопределенного поведения» из стандарта языка С 1999 года, известного под названием «C99»:

неопределенное поведение

поведение, являющееся следствием использования непереносимой или некорректной программной конструкции либо некорректных данных, для которого в настоящем Международном стандарте нет никаких требований.

Рассмотрим следующую программу на С:

```
int main(int argc, char **argv) {
    unsigned long a[1];
    a[3] = 0x7ffff7b36cebUL;
    return 0;
}
```

Эта программа обращается к элементу за концом массива `a`, поэтому согласно С99 ее поведение не определено, т. е. она может делать все что угодно. Сегодня утром запуск этой программы на ноутбуке Джима закончился печатью сообщения

```
undef: Error: .netrc file is readable by others.
undef: Remove password or make file unreadable by others.
```

После чего программа «грохнулась». На компьютере Джима даже нет файла `.netrc`. Если вы попробуете запустить ее сами, то, возможно, результат будет совсем иным.

В генерированном компилятором С машинном коде функции `main` массив `a` размещен в стеке на три слова раньше адреса возврата, поэтому запись значения `0x7ffff7b36cebUL` в `a[3]` изменяет адрес возврата из `main`, так что он указывает на какой-то код из стандартной библиотеки С, который пытается прочитать пароль из файла `.netrc`. После возврата из `main` выполнение возобновляется не с команды, следующей за вызовом `main`, а с машинного кода, соответствующего таким строкам из библиотеки:

```
warnx(_("Error: .netrc file is readable by others."));
warnx(_("Remove password or make file unreadable by others."));
goto bad;
```

Но из того, что обращение к элементу массива влияет на поведение последующего предложения `return`, вовсе не следует, что компилятор С не отвечает стандарту. «Неопределенная» операция не просто возвращает неопределенный результат, она дает программе карт-бланш на *произвольное* поведение.

Стандарт С99 предоставляет компилятору такое право, чтобы он мог генерировать более быстрый код. Чем возлагать на компилятор ответственность за обнаружение и обработку странного поведения вроде выхода за конец массива, стандарт предполагает, что программист должен позаботиться о том, чтобы такие ситуации никогда не возникали.

Но опыт показывает, что с этой задачей мы справляемся неважно. Будучи студентом университета штата Юта, исследователь Пень Ли (Peng Li) модифицировал компиляторы С и С++, так чтобы оттранслированные ими программы сообщали о некоторых видах неопределенного поведения. Обнаружилось, что этим грешат почти все программы, в т. ч. и весьма уважаемые, авторы которых стремились

соблюдать высочайшие стандарты. На практике неопределенное поведение часто ведет к брешам в системе безопасности, допускающим написание эксплойта. Червь Морриса распространялся с одной машины на другую, применяя вариант описанной выше техники, и такого рода эксплойты часто встречаются и по сей день.

Теперь определим некоторые термины. Если программа написана так, что ни на каком пути выполнения неопределенное поведение невозможно, то будем говорить, что программа *корректна* (well defined). Если встроенные в язык проверки гарантируют корректность программы, то будем называть язык *типобезопасным* (type safe).

Тщательно написанная программа на С или С++ может оказаться типобезопасной, но ни С, ни С++ не является типобезопасным языком: в приведенной выше программе нет ошибок типизации, и тем не менее она демонстрирует неопределенное поведение. С другой стороны, Python – типобезопасный язык, его интерпретатор тратит время на обнаружение выхода за границы массива и обрабатывает его лучше, чем компилятор С:

```
>>> a = [0]
>>> a[3] = 0x7fffff7b36ceb
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
```

Python возбудил исключение, а это уже не неопределенное поведение: в документации по Python сказано, что такое присваивание элементу `a[3]` должно приводить к исключению `IndexError`, что мы и видели. Разумеется, модуль `ctypes`, дающий неограниченный доступ к машине, может стать причиной неопределенного поведения в Python, но сам базовый язык типобезопасен. Таковы же языки Java, JavaScript, Ruby и Haskell.

Отметим, что типобезопасность не зависит от того, когда язык проверяет типы: на этапе компиляции или выполнения. Язык С делает это на этапе компиляции и не является типобезопасным, Python – на этапе выполнения и является таковым.

По иронии судьбы, основные языки системного программирования, С и С++, не типобезопасны, тогда как большинство других популярных языков типобезопасно. Учитывая, что С и С++ предназначены для реализации фундамента системы, что им доверено обеспечивать безопасность границ, на которых происходит контакт с ненадежными данными, типобезопасность была бы весьма важным свойством.

На решение этой проблемы, существующей много десятков лет, – получить типобезопасный язык системного программирования – и нацелен Rust. Он проектировался для реализации тех фундаментальных уровней системы, которым необходимы высокая производительность и точный контроль над ресурсами, но вместе с тем базовые гарантии предсказуемости, которые дает типобезопасность. Далее мы подробно рассмотрим, как Rust справляется с этой задачей.

Выбранная в Rust форма типобезопасности влечет удивительные последствия для многопоточного программирования. Известно, как трудно написать правиль-

ную многопоточную программу на С и С++, поэтому разработчики обращаются к многопоточности только тогда, когда однопоточная программа не способна достичь необходимой производительности. Но Rust гарантирует отсутствие гонок за данные и обнаруживает некорректное использование мьютексов и других примитивов синхронизации на этапе компиляции. В Rust можно пользоваться конкурентностью, не опасаясь сделать программу понятной только самым квалифицированным программистам.

В Rust имеется механизм обхода правил безопасности для тех случаев, когда использование простого указателя абсолютно необходимо. Такой код называется небезопасным, и, хотя в большинстве Rust-программ это не нужно, в главе 21 мы все же обсудим, как его писать и как он укладывается в общую схему безопасности Rust.

Как и в других статически типизированных языках, применение типов в Rust отнюдь не ограничено предотвращением неопределенного поведения. Опытный программист на Rust применяет типы, для того чтобы использование значений было не только безопасным, но и соответствовало целям приложения. В частности, характеристики и универсальные типы, описанные в главе 11, дают лаконичный, гибкий и эффективный способ описать общие свойства группы типов, а затем воспользоваться этой общностью.

В этой книге мы ставим перед собой цель научить вас не просто писать программы на Rust, а применять язык так, чтобы написанные вами программы были безопасны и правильны и обладали предсказуемой производительностью. Наш опыт показывает, что Rust – большой шаг вперед в области системного программирования, и мы хотим помочь вам извлечь из этого все преимущества.

Глава 2

Краткий обзор Rust

Опыт каждого человека строится на базе его языка.
— Анри Делакруа

В этой главе мы рассмотрим несколько коротких программ, чтобы познакомиться с тем, как синтаксис, типы и семантика Rust в совокупности позволяют писать безопасный, конкурентный и эффективный код. Мы опишем процедуру скачивания и установки Rust, продемонстрируем простой код с математическими операциями, поэкспериментируем с веб-сервером на основе сторонней библиотеки и организуем несколько потоков для построения множества Мандельброта.

Скачивание и установка Rust

Для установки Rust проще всего воспользоваться установщиком `rustup`. Зайдите на сайт <https://rustup.rs> и следуйте приведенным там инструкциям.

Можно вместо этого зайти на сайт <https://www.rust-lang.org>, нажать кнопку «Downloads» и скачать готовый пакет для Linux, macOS или Windows. Rust также включен в состав некоторых дистрибутивов операционных систем. Мы предпочтаем `rustup`, потому что этот инструмент специально предназначен для управления установкой Rust, как RVM для Ruby или NVM для Node. Например, после выпуска очередной версии Rust для перехода на нее нужно будет всего лишь набрать `rustup update`.

Как бы то ни было, после завершения установки должны появиться три новые программы:

```
$ cargo --version
cargo 0.18.0 (fe7b0cdcf 2017-04-24)
$ rustc --version
rustc 1.17.0 (56124baa9 2017-04-24)
$ rustdoc --version
rustdoc 1.17.0 (56124baa9 2017-04-24)
$
```

Здесь `$` – приглашение к вводу команды; в Windows вместо него выводится `C:\>` или что-то в этом роде. Выше мы выполнили все три установленные команды с флагом печати версии.

- `cargo` – диспетчер компиляции Rust, менеджер пакетов и вообще мастер на все руки. Он позволяет создать новый проект, собрать и запустить программу и вести учет внешних библиотек, от которых зависит ваша программа.

- `rustc` – компилятор Rust. Обычно компилятор вызывает Cargo, но иногда полезно запускать его непосредственно.
- `rustdoc` – средство документирования для Rust. Если в исходный код включены комментарии в определенном формате, то `rustdoc` построит по ним красиво отформатированную HTML-документацию. Как и в случае `rustc`, для запуска `rustdoc` обычно используется Cargo.

Для удобства Cargo может создать новый Rust-пакет со стандартными метаданными:

```
$ cargo new --bin hello
Created binary (application) 'hello' project
```

Эта команда создает пакет с именем `hello`, а флаг `--bin` означает, что пакет будет представлять собой исполняемый файл, а не библиотеку. Вот как выглядит верхний уровень каталога пакета:

```
$ cd hello
$ ls -la
total 24
drwxrwxr-x. 4 jimb jimb 4096 Sep 22 21:09 .
drwx----- 62 jimb jimb 4096 Sep 22 21:09 ..
drwxrwxr-x. 6 jimb jimb 4096 Sep 22 21:09 .git
-rw-rw-r-- 1 jimb jimb 7 Sep 22 21:09 .gitignore
-rw-rw-r-- 1 jimb jimb 88 Sep 22 21:09 Cargo.toml
drwxrwxr-x. 2 jimb jimb 4096 Sep 22 21:09 src
$
```

Как видим, Cargo создал файл `Cargo.toml` для хранения метаданных пакета. Пока что в этом файле почти ничего нет:

```
[package]
name = "hello"
version = "0.1.0"
authors = ["You <you@example.com>"]

[dependencies]
```

Если впоследствии программе понадобятся сторонние библиотеки, то их можно будет прописать в этом файле, и тогда Cargo возьмет на себя скачивание, установку и обновление этих библиотек. Подробно файл `Cargo.toml` будет рассмотрен в главе 8.

Cargo подготовил пакет к работе с системой управления версиями `git`, для чего создал подкаталог `.git` и файл `.gitignore`. Этот шаг можно пропустить, задав в командной строке параметр `--vcs none`.

Каталог `src` содержит исходный код на Rust:

```
$ cd src
$ ls -l
total 4
-rw-rw-r-- 1 jimb jimb 45 Sep 22 21:09 main.rs
```

Выходит, что Cargo уже начал писать программу от нашего имени. Файл `main.rs` содержит такой код:

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине «Электронный универс»
(e-Univers.ru)