

Язык программирования



и система разработки

расскажет Михаил В. Шохирев

Клуб программистов
Шадринск
2025-2026

Программная система — состоит из:

- ▶ **программный код** (*версии текстов программ*)
 - ▷ исходные файлы в каталогах (*на ПК разработчика и серверах*)
 - ▷ собственные и внешние библиотеки
- ▶ **разработчики** (*люди*)
 - ▷ разной квалификации и опыта
 - ▷ модифицируют разные части системы
- ▶ **средства разработки + процесс** (*программы + организация работ*)
 - ▷ инструментарий
 - ▷ тех. процесс (*этапы, процедуры, правила, ...*)
 - ▷ документация (*сгенерированная, рукописная*)
- ▶ **серверы** (*узлы в сети*)
 - ▷ железо: компьютеры (*ЦП, память, диски, ...*) и коммуникационные средства
 - ▷ системное ПО (*ОС, утилиты, библиотеки, драйверы, ...*)
- ▶ **работающая система** (*на серверах*)
 - ▷ исполняемые файлы и динамические библиотеки
 - ▷ данные редко изменяемые (*конфигурации, шаблоны, ...*)
 - ▷ данные часто изменяемые (*файлы, БД, сообщения, потоки, log-и*)
- ▶ **клиенты**
 - ▷ клиентские устройства, клиентское ПО
 - ▷ данные
 - ▷ пользователи (*люди и программы*)

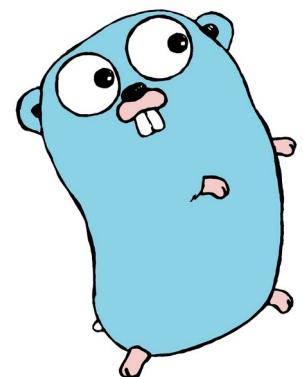


Большое ПО



Значимая часть современного ПО выполняется на серверах. И почти всегда оно большое, очень большое. Причём в разных измерениях:

- ▶ состоит из большого количества **исходных текстов**,
 - ▷ которые расположены во множестве **файлов и каталогов**;
 - ▷ и должны изменяться параллельно **разными людьми**;
- ▶ использует **много готовых программ**, собственных и сторонних,
 - ▷ которое обновляется;
- ▶ создаётся **большой командой** разработчиков,
 - ▷ состав которой **время от времени меняется**,
- ▶ обладает **широкой функциональностью**,
 - ▷ которая должна постоянно эволюционировать;
 - ▷ поскольку меняются требования;
 - ▷ и новые возможности должны встраиваться в существующую систему;
- ▶ представлено в **нескольких версиях и вариантах** (dev, test, prod);
- ▶ используется **длительное время**;
- ▶ выполняется **на многих ЦП, сетевых узлах**;
- ▶ к нему обращается **возрастающее количество клиентов**,
 - ▷ часто с разных устройств (с разной аппаратной архитектурой);
 - ▷ из под разных ОС;



Разработка больших программ

Большие программные системы: разработка

Изменяется ВСЁ!



Подход к разработке



Ещё в 1970-х годах Никлаус Вирт сформулировал принцип:

мощь языка программирования достигается не обилием функций, а минимальным набором хорошо сочетаемых элементов, которые могут произвольно комбинироваться.

Это перекликается с «**Unix philosophy**» 1970-х годов (Ken Thompson, Dennis Ritchie, Brian Kernighan, Rob Pike, Doug McIlroy), в которой подчёркивается важность простоты и минимализма:

- **Make it easy to write, test, and run programs.**
- **Economy and elegance of design due to size constraints ("salvation through suffering").**
- **Don't hesitate to throw away the clumsy parts and rebuild them.**
- **Write programs that do one thing and do it well.**
- **Write programs to work together.**
- **Expect the output of every program to become the input to another, as yet unknown, program.**

«Although that philosophy can't be written down in a single sentence, at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools.»

Проблемы (большого ПО) и решения (в Go)



Программисты совершают ошибки.

- все объявленные переменные имеют «нулевое» начальное значение, нет неинициализированных переменных;
- структуры языка простые и понятные, взаимно независимы в применении (ортогональны), хорошо сочетаются логичным образом;
- в языке со статической и строгой типизацией — жёсткая проверка типов при компиляции;
- программа не скомпилируется, если есть неиспользуемые переменные или ненужные импортированные пакеты;
- минимум «синтаксического сахара», практически всё надо объявлять и преобразовывать явно;
- средства языка дают возможность писать правильно (`defer`, `init()`, ...);



Проблемы и решения

Разные программисты пишут в собственных разных стилях (появляются персональные «диалекты языка»).



- минималистичный синтаксис принуждает записывать (никакого ~~TIME TO WRITE!~~);
 - программа `go fmt` форматирует исходники одинаковым для всех способом, приводит к единому виду;
 - исходники в единственной кодировке UTF-8;
 - просто и понятно написанные исходные тексты стандартных библиотек служат наглядным примером хорошего стиля;
 - очень богатая стандартная библиотека предлагает единообразные унифицированные решения;
- ▼ (*также см. следующие разделы*)

«*Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.*»
~ Go proverbs

Проблемы и решения

Существующие программы должны изменяться новыми разработчиками (которые должны сначала понять их).



- ясный, логичный и компактный синтаксис обеспечивает читабельность и способствует хорошему пониманию программ*;
- простой язык Go легко изучить тем, кто не знал его раньше;
- правила видимости и области действия имён: их мало, они простые и понятные;
- все имена полностью определяются идентификаторами пакетов;
- синтаксис языка стабилен от версии к версии (минимум изменений); спецификация языка совместима с предыдущими и последующими версиями (*Go Compatibility Promise*);
- `go doc` показывает документацию по пакетам и функциям (стандартным и пользовательским);

* «*Readable means reliable.*» ~ Rob Pike.

Проблемы и решения

Программы должны постоянно развиваться (по мере изменения требований).

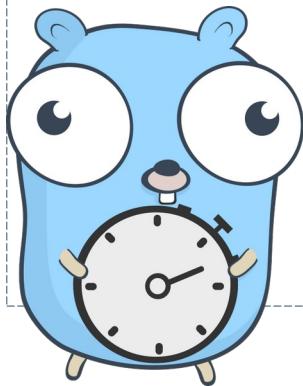
- легче развивать программу, когда нет необходимости зависеть от жёсткой иерархии классов;
- составление (composition) вместо наследования (inheritance) позволяет программным компонентам эволюционировать независимо друг от друга;
- интерфейсы с неявным соответствием позволяют сочетать новые компоненты с уже существующими, а также легко применять функциональность старых компонентов в новых;
- функции как полноценные типы данных обеспечивает гибкость при взаимодействии компонентов;
- очень богатая стандартная библиотека предоставляет решения при добавлении новой функциональности;



Проблемы и решения



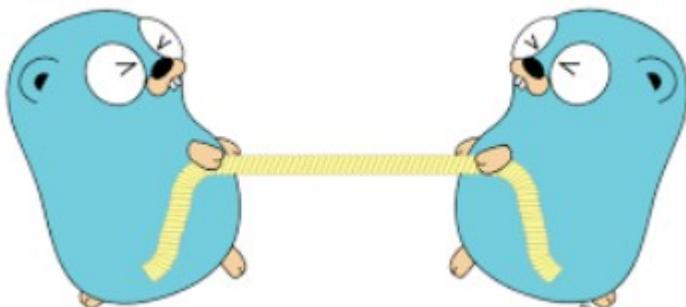
Программы должны эволюционировать в течение долгого времени.



- синтаксис совместим с предыдущими и последующими версиями языка (*Go Compatibility Promise*);
- интерфейсы с неявным соответствием позволяют сочетать функциональность новых и существующих компонентов;
- система управления модулями (`go mod`) обеспечивает компоновку и обновление модулей, управление их версиями;
- стандартные инструменты для статического анализа (`go vet`) и обнаружения изменений в API модулей для их обновления (`go fix`), информации о загруженных модулях (`go list`);
- обновления версий языка, системных и внешних модулей легко делаются стандартными средствами (`go get`, `go install`);

Проблемы и решения

Программы должны модифицироваться многими программистами в одно и то же время.



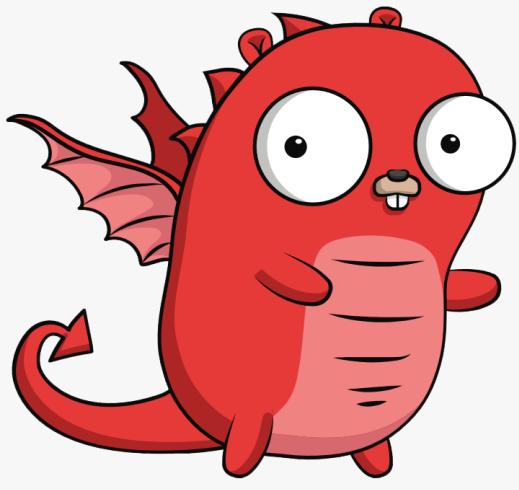
- исходные тексты свободно располагаются в разных файлах и каталогах проекта, но легко объединяются через файл `go.mod`;
- система модулей поощряет разделять код на небольшие пакеты, где каждый отвечает за свою задачу, а `go mod` управляет зависимостями и сборкой;
- один модуль состоит из множества пакетов в разных каталогах, которые могут независимо изменяться разными людьми;
- программный код одного пакета можно располагать в одном или в разных файлах в каталоге (*не изменять, а добавлять*);
- унифицированное представление всех исходников (с помощью `go fmt ./...`) упрощает выявление изменений в репозитории;

Проблемы и решения



Серверные программы имеют большой размер.

- система импортирования пакетов эффективно обрабатывает пакеты и модули при компоновке программ;
- высокопроизводительный компилятор очень быстро обрабатывает большую кодовую базу;
- сборка в исполняемый файл тоже реализована эффективно;
- для инициализации пакетов в языке предусмотрены специальные функции `init()`;
- распределённая система модулей с идентификацией по URL упрощает автоматизацию и масштабирование;
- при сборке программы `go build` может забирать исходники прямо из систем управления версиями;



Проблемы и решения

Серверные программы должны эффективно использовать аппаратные ресурсы.

- начиная с 1-й версии многозадачность (*concurrency*) реализована как встроенный в язык механизм* (эффективно использующий ядра ЦП), поэтому её легко использовать понятным образом;
- сборщик мусора (GC) эффективно управляет распределением и освобождением оперативной памяти;
- исходники компилируются в быстро исполняемые двоичные программы (без зависимостей);
- пакеты из стандартной библиотеки реализованы очень эффективно (и улучшаются в новых версиях);
- при кросс-компиляции учитываются особенности аппаратной платформы;
- есть возможность подключать библиотеки на С (`cgo`);

* конструкции языка (`go`, `select`, `chan`), а не библиотечные функции.



Проблемы и решения

Программы должны иметь возможность выполняться на разных аппаратных платформах под разными ОС.

- очень легко выполнить компиляцию исходной программы на новую архитектуру и ОС (для настройки задаются всего 2 переменные окружения: GOARCH и GOOS);
- поддерживаются все основные ОС и очень много платформ «железа»;
- проект TamaGo позволяет разрабатывать программы, которые выполняются на «голом железе» (*bare metal*);
- проект TinyGo для разработки программ для встраиваемых систем (микропроцессоров и микроконтроллеров);
- можно компилировать Go в JavaScript или WASM для выполнения программ в браузере или внешней runtime;
- есть (нестандартный) интерпретатор для выполнения программ без компиляции;



Проблемы и решения



Программы должны тщательно проверяться в ходе разработки.

- в системе программирования Go поставляются стандартные средства тестирования (white/black box, *fuzzing*), измерение покрытия тестами (`go test -cover`), инструмент обнаружения гонок (`go test -race`) при одновременных вычислениях;
- примеры и тестовые программы располагаются рядом с исходниками (но не включаются в исполняемую программу);
- средства профилирования (`go test + go tool pprof`) и измерения производительности (`go test -bench`) также стандартные;
- есть средства обнаружения уязвимостей (`govulncheck ./...`) в исходниках и зависимостях;
- все средства тестирования, отладки, телеметрии можно удобно объединять в конвейеры для автоматизации разработки;



Проблемы и решения

Разработка программ должна быть автоматизирована (в т. ч. с помощью ИИ).

- в распоряжении разработчика целый *toolchain* — богатый набор стандартных инструментов (команды **go command**, **go tools**, официальный языковой сервер **gopls**, ...);
- стандартные команды удобно применять в скриптах;
- есть стандартные средства, чтобы улучшать и добавлять инструменты разработчика (**assembler**, **ast**, **parser**, **scanner**, **token**, ...);
- имеются стандартные средства кодо-генерации (**go generate**);
- синтаксис языка облегчает создание инструментария;
- у системы программирования Go открытые исходники;
- сообщество Go расширяет набор средств для автоматизации разработки;

