

# **Язык программирования**



## **Язык и технология**

*расскажет Михаил В. Шохирев*

Клуб программистов  
Шадринск  
2025

# Программная система — состоит из:

- **программный код** (версии текста)
  - — исходные файлы в каталогах (на ПК и серверах)
  - — внешние библиотеки
- **разработчики** (люди)
  - — разной квалификации
  - — модифицируют разные части системы
- **средства разработки + процесс** (программы + организация)
  - — инструментарий
  - — тех. процесс (этапы)
  - — документация
- **серверы** (узлы в сети)
  - — железо: компьютеры (ЦП, память, диски, ...) и коммуникационные средства
  - — системное ПО
- **работающая система** (на серверах)
  - — исполняемые файлы и библиотеки
  - — конфигурации, шаблоны (~неизменяемые)
  - — данные (изменяемые), в т. ч. мониторинга
- **пользователи**
  - — клиентское ПО, устройства
  - — данные
  - — люди

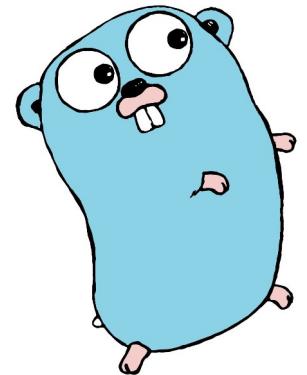


# Большое ПО



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 **стороннего ПО**,
- 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** (prod, test, dev);
  
- 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!*



# Язык программирования



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.



# Проблемы и решения

Программисты совершают ошибки.

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



# Проблемы и решения

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

- минималистичный синтаксис принуждает записывать алгоритмы единообразными конструкциями (*никакого TIMTOWTDI*);
  - 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;
- ▼ (*также см. следующие разделы*)



# Проблемы и решения

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

- ясный логичный синтаксис обеспечивает читабельность и способствует хорошему пониманию программ\*;
- правила видимости и области действия имён 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*);



\* «*Readable means reliable*» -- Rob Pike.

# Проблемы и решения

Программы должны постоянно развиваться (по мере изменения требований).

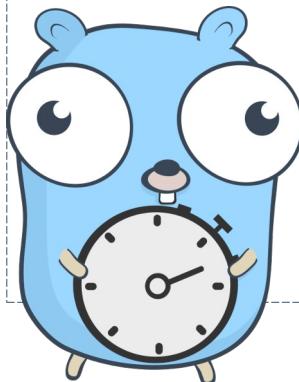
- легче развивать программу, когда нет необходимости зависеть от жёсткой иерархии классов;
- объединение (composition) вместо наследования позволяет программным компонентам эволюционировать независимо друг от друга;
- интерфейсы с неявным соответствием позволяют сочетать новые компоненты с уже существующими, а также легко применять функциональность старых компонентов в новых;
- функции как полноценные типы данных обеспечивает гибкость при взаимодействии компонентов;
- очень богатая стандартная библиотека упрощает добавление новой функциональности;
- `go doc` показывает документацию по пакетам и функциям (стандартным и пользовательским);



# Проблемы и решения

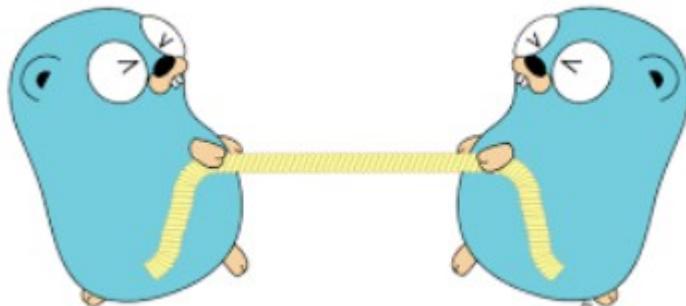
Программы должны эволюционировать в течение долгого времени.

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



# Проблемы и решения

Программы должны модифицироваться многими программистами одновременно (параллельно).



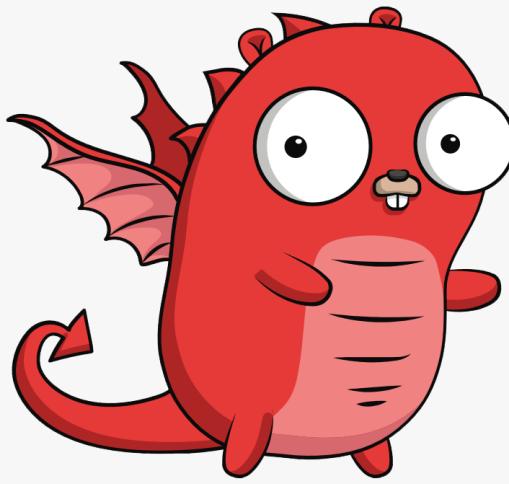
- исходные тексты свободно располагаются в разных файлах и каталогах проекта, но легко объединяются через `go.mod`;
- один модуль состоит из множества пакетов в разных каталогах, которые могут независимо изменяться разными людьми;
- программный код одного пакета можно располагать в одном или в разных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

# 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;
- есть возможность динамически подключать общие библиотеки;
- распределённая система модулей с идентификацией по URL упрощает автоматизацию и масштабирование;
- при сборке программы `go build` забирает исходники прямо из систем управления версиями;



# Проблемы и решения

Серверные программы должны эффективно использовать аппаратные ресурсы.

- начиная с 1-й версии одновременность (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;
- при кросс-компиляции учитываются возможности 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 платформы «железа»;
- есть проект TamaGo для разработки программ, которые будут выполняться на «голом железе» (*bare metal*);
- имеется TinyGo для разработки программ для встраиваемых систем (микропроцессоров и микроконтроллеров);
- можно компилировать в JavaScript или WASM для выполнения программ в браузерах;
- 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 распоряжении разработчика целый *toolchain* — богатый набор стандартных инструментов (команды `go command`, `go tools`, официальный языковой сервер `gopls`, ...);
- есть стандартные средства улучшать и добавлять инструменты разработчика (`assembler`, `ast`, `parser`, `scanner`, `token`, ...);
- синтаксис языка предусматривает простоту создания инструментария;
- у системы программирования Go открытые исходники;
- сообщество Go расширяет набор средств для автоматизации разработки;

