

Язык программирования



Язык и технология

расскажет Михаил В. Шохирев

Клуб программистов
Шадринск
2025

Программная система — состоит из:

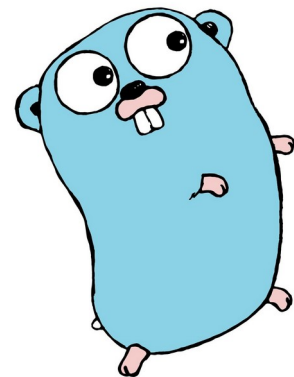


- **программный код** (версии)
 - — исходные файлы в каталогах
 - — внешние библиотеки
- **разработчики**
 - — разной квалификации
 - — модифицируют разные части системы
- **средства разработки + процесс**
 - — инструментарий
 - — тех. процесс (этапы)
 - — документация
- **сервер(ы)**
 - — железо: компьютеры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



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;
- использует много **стороннего ПО**;
 - которое часто 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;



Большие программные системы: разработка *Изменяется ВСЁ!*



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.



Люди совершают ошибки.



- все переменные имеют «нулевое» начальное значение, нет неинициализированных переменных;
- структуры языка простые и понятные, взаимно независимы в применении (ортогональны);
- 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()`, ...);

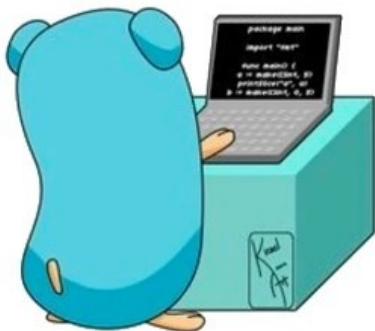
Разные программисты пишут в собственных разных стилях (появляются «диалекты языка»).

- строгий 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;



Существующие программы должны изменяться новыми разработчиками.

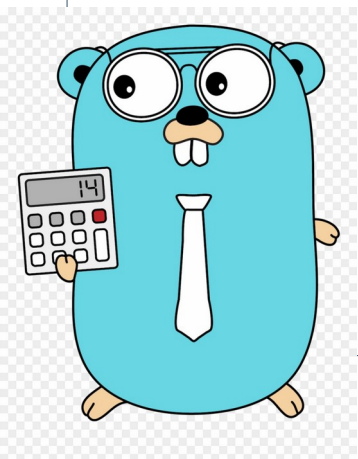
- ясный простой синтаксис обеспечивает читабельность и способствует хорошему пониманию программ*;
- правила видимости и области действия имён простые и понятные;
- все имена полностью определяются идентификаторами пакетов;
- синтаксис языка неизменен от версии к версии;
- спецификация языка совместима с предыдущими и последующими версиями (*Go Compatibility Promise*);



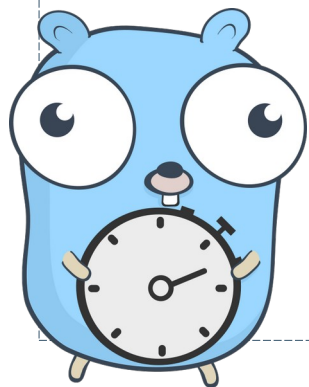
* «*Readable means reliable*» -- Rob Pike.

Программы должны постоянно развиваться (по мере изменения требований).

- 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;

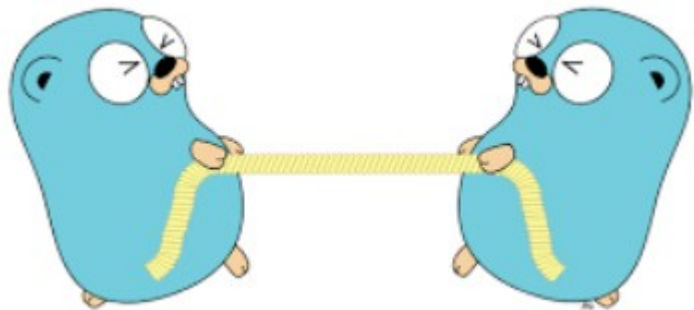


Программы должны эволюционировать в течение долгого времени.



- синтаксис совместим с предыдущими и последующими версиями языка;
- интерфейсы с неявным соответствием позволяют сочетать функциональность новых и существующих компонентов;
- система управления модулями (**go mod**) позволяет обновлять их до новых версий и использовать разные версии параллельно;
- есть инструменты для обнаружения ошибок и уязвимостей в модулях для их обновления (**go vet**, **go fix**);
- обновления внешних системных и сторонних модулей делаются легко (**go get**, **go install**);

Программы должны модифицироваться многими разработчиками (часто одновременно).



- исходные тексты свободно располагаются в разных файлах и каталогах проекта и 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 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;

Серверные программы имеют большой размер.

- система импортирования пакетов минимизирует включения при компоновке программ;
- эффективный 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;



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);

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



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 2 переменные окружения: GOARCH и GOOS);

— поддерживаются все основные ОС и платформы «железа»;

— есть инструмент для разработки программ, которые будут выполняться на «голом железе» (*bare metal*);

— есть TinyGo для разработки встроенных систем;

— можно компилировать в JavaScript или WASM для выполнения программ в браузерах;



Программы должны тщательно проверяться в ходе разработки.

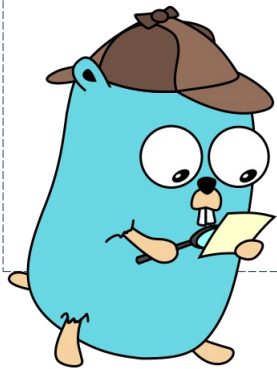
— в системе программирования Go поставляются стандартные средства тестирования (включая измерение покрытия тестами);

— тестовые программы располагаются рядом с исходниками (но не включаются в исполняемую программу);

— средства профилирования и benchmarking также стандартные;

— ещё больше средств тестирования разработано сообществом разработчиков Go;

— все средства тестирования, отладки, телеметрии можно удобно объединять в конвейеры для автоматизации разработки;



Разработка программ должна быть автоматизирована.

- в распоряжении разработчика — богатый набор стандартных инструментов (команды **go**, **go tools**, официальный языковой сервер **gopls**, ...);
- есть доступный способ улучшать и добавлять инструменты разработчика;
- синтаксис языка предусматривает простоту создания инструментария;
- у системы программирования Go открытые исходники;
- сообщество Go расширяет набор средств для автоматизации разработки;

