GO-07 05: Кодогенерация. Враппинг

Описание:

В прошлом задании мы с вами научились при помощи кодогенерации создавать целые структуры со своими методами и полями, а также использовать их в своем коде. Теперь давайте разберем еще одну проблему, которую можно решить при помощи кодогенерации. Эта проблема заключается в том, что для многих задач в нашем приложении мы часто пишем много шаблонного кода, например:

- метрики,
- логирование,
- валидаторы для структур,
- многое другое.

Хотелось бы при добавлении новой структуры (если брать пример из прошлого задания, то при добавлении нового монитора) не писать для нее методы, отвечающие за такой функционал. И в этом нам может помочь такой инструмент, как wrapping.
Враппинг - это своего рода обертка над структурой, которая добавляет к уже имеющемуся функционалу новый, тем самым наделяя структуру новыми свойствами. Например, при помощи врапперов можно сделать логи внутри методов структуры более информативными, обогащая их различной информацией, или для методов интерфейса сразу ввести метрики, которые будут отдаваться, например, в prometheus. В ООП такой паттерн еще называют Декоратор.
Давайте представим, что у нас есть некий воркер, который выполняет какую-нибудь работу. Выглядеть это может следующим образом:

```
// Интерфейс для всех воркеров.
type Worker interface {
  Start()
}

type worker struct {
  name string
}

func newWorker(name string) Worker {
  return &worker{name: name}
}

func (w *worker) Start() error {
  fmt.Printf("woekr %s is start\n", w.name)
  return nil
```

```
}
```

Таких воркеров может быть много, они могут делать разные вещи и выглядеть по-разному, главное, чтобы все они подчинялись интерфейсу Worker. А теперь нам прилетает новая задача: нам нужно, чтобы каждый воркер считал и выводил время своей работы (то есть, за какое количество времени он отработал).

- Самый простой и наивный способ это реализовать - в каждом воркере добавить функционал подсчета времени выполнения. Но проблема в том, что воркеров может быть большое количество - это раз, а два - это то, что при каждом изменении одного воркера надо будет идти и в остальные и менять там изменившейся функционал тоже.
- Второй способ - создать обертку над воркерами, которая будет считать время выполнения, а затем обернуть все наши воркеры. Таким образом, изменения придется вносить только в обертку и можно не трогать код самих воркеров.

Выглядеть такая обертка может так:

```
type workerWithTime struct {
	worker Worker
}

func newWorkerWithTime(worker Worker) Worker {
	return &workerWithTime{worker: worker}
}

func (wwt *workerWithTime) Start() error {
	start := time.Now()
	defer func() {
		fmt.Println("worker done: time -", time.Since(start))
	}()

	return wwt.worker.Start()
}
```

Таким образом мы можем обернуть либо все воркеры, либо только те, что нам нужно, и получить для них новый функционал, а интерфейс при этом остается тем же.

```
func main() {
	w := newWorkerWithTime(newWorker("simple worker"))
	w.Start()
}
```

Gowrap

Но где же тут кодогенерация, спросите вы? А она именно здесь! Сейчас объясню. Представьте, что такой базовый функционал, как время работы, логирование или метрики, нужен нам не только для одной сущности, мы также захотим добавить весь этот функционал и в другие части программмы. И если мы оставим это в ручном режиме, то нам придется при добавлении новой функциональности писать к ней дополнительно и обертки. Но в go мы можем сгенерировать такие обертки при помощи шаблонов либо самостоятельно, либо при помощи сторонних библиотек.
Генерировать что-то по шаблону руками мы уже умеем, давайте теперь воспользуемся специальной библиотекой, которая как раз и нужна для того, чтобы генерировать обертки (декораторы) для наших программ.
Называется эта библиотека gowrap. Скорее это даже не библиотека, а CLI (Command Line Interface) для генерации оберток на основе интерфейсов ваших программ и шаблонов.
После простейшей установки мы уже можем начать генерировать. Для этого нужно ввести в терминале следующую команду:

```
gowrap gen -p worker -i Worker -t log -o ./worker_with_log.go
```

- -p - это аргумент, в который мы передаем пакет, для которого нужно генерировать обертку.
- -i - это аргумент, в который мы передаем имя интерфейса, для которого нужно генерировать обертку.
- -t - это аргумент, в который мы передаем шаблон (можно передать путь до собственного шаблона или же передать одно из стандартных имен шаблонов такое, как log, например).
- -o - это аргумент, в который мы передаем путь до файла, в который будет сгенерирована обертка.

Но что если пакетов, для которых нужно сгенерировать обертки, много? Не будем же мы каждый раз писать в терминале кучу команд? Для этого в go предусмотрен специальный тег для команд генерации, который можно положить в файл в неограниченном количестве, а затем вызывать все сразу одной командой. Выглядит такой файл так:

```
package generated

//go:generate gowrap gen -p worker -i Worker -t log -o
./worker_with_log.go
```

Имя пакета может быть любым, а вот содержание подчиняется определенному формату. Сначала указывается тег //go:generate, а затем - любая shell команда. И таких строк мы можем писать, сколько угодно. А для того чтобы запустить все команды из файла, нам достаточно ввести go generate и передать в нее либо путь до пакета с тегами генерации, либо ./..., для того чтобы запустить генерацию для всех пакетов, в которых go generate найдет комментарий с тегом //go:generate.

Полезные ссылки:

- Github repository - gowrap
- Generating code
- Golang Templates Cheatsheet

Задание:

В этом задании мы предлагаем вам поиграть с библиотекой gowrap и погенерировать различные шаблоны для пакета module07/internal/monitors.
Условия:

- Генерировать нужно при помощи пакета генерации (с использованием тегов //go:generate).
- Генерировать нужно только те обертки, которые вы сможете запустить и показать (например, если вы хотите сгенерировать шаблон для метрик prometheus, тогда вам нужно обеспечить запуск сервера с ручкой /metrics, при переходе на которую можно проверить, что все работает, как надо).
- Можно генерировать не одну обертку, а несколько. Главное, чтобы их все можно было проверить на работоспособность.
- Нужно написать в Makefile проекта команду make generate, которая будет создавать нужные папки в проекте и запускать go generate.

Задание со звездочкой:

Попробуйте написать собственный шаблон, опираясь на описание и примеры из репозитория custom-templates. Шаблон может быть любым на ваше усмотрение, главное - приложите описание того, что ваша обертка будет делать, в сообщение к merge request. Это позволит нам быстрее проверить корректность написанного вами решения.
Порядок действий:

1. В вашем проекте module07 сделайте новую ветку module07_05.
2. Создайте пакет module07/internal/generated/wrappers и положите в него файл для генерации (файл, в котором будут находится теги //go:generate).
3. Установите gowrap и ознакомьтесь с тем, как работает тулза.
4. Сгенерируйте обертки к пакету module07/internal/monitors, а конкретно - к интерфейсу Monitor, на основе стандартных шаблонов gowrap (например, log).
5. Если вы будете выполнять задание со звездочкой и писать собственный шаблон, создайте файл шаблона в пакете module07/internal/templates.
6. В пакете module07/cmd/app в функции Task05 напишите код, который будет инициализировать монитор, навешивать на него все сгенерированные обертки и запускать функции, для того чтобы убедиться, что все работает корректно. В функции main сделайте вызов функции Task05.

7. Напишите команду make generate в Makefile проекта.
8. Проверьте, что все работает, как надо (запустите make generate, затем make run, ошибок быть не должно).
9. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки module07_05.