



Ву! ИИ готовит к ЕГЭ по информатике

[Попробовать](#)

Новый конкурс! Нарисуйте [комикс](#) или [инфографику](#) на любую тему из области информатики. Приём работ до 17 марта.

Урок Спринт 2.2

## io.Reader

В программировании, часто необходимо читать данные из различных источников (например, файла или сети) и записывать данные в различные места назначения (например, файл или базу данных). Для этого в Go существуют специальные интерфейсы `io.Reader` и `io.Writer`, предоставляемые стандартной библиотекой.

Напомню, что интерфейс подобен набору правил или контракту, которому могут следовать разные объекты. Реализуя эти интерфейсы, объекты могут вести себя как `io.Reader` или `io.Writer`.

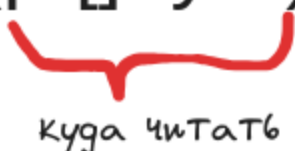

Эти интерфейсы абстрагируют реализации и предоставляют единый способ выполнения операций чтения и записи в различных источниках данных, таких как файлы, сетевые подключения или буферы памяти.

### io.Reader

Интерфейс `Reader` в Go определён следующим образом:

```
type Reader interface {  
    Read(p []byte) (n int, err error)  
}
```

`Read(p []byte) (n int, err error)`

куда читать      сколько прочитали

`Read` считывает до `len(p)` байт из источника в заданный байтовый слайс `p`, возвращая количество

прочитанных байт и ошибку (если она возникла). Важно проверить возвращаемое значение `n`, так как оно может быть меньше длины `p`, если источник содержит меньше данных.

Интерфейс `Reader` используется различными типами пакета `io`, такими как `File`, `Buffer` и `Net.Conn`, для чтения данных из разных источников.

Допустим, мы хотим прочитать локальный файл и обработать данные из этого файла. На первый взгляд, можно написать следующую функцию:

```
func ProcessData(ctx context.Context, fileName string) error{
    // откроем файл
    file, err := os.Open(fileName)
    // здесь обрабатываем данные
    ...
}
```

Но что, если через какое-то время понадобится прочитать файлы из файлового хранилища, например из `Google Cloud Storage`? Мы будем вынуждены вносить изменения в функцию `ProcessData`, потому что `os.Open` использовать для этого нельзя:

```
func ProcessData(ctx context.Context, bucket, fileName string) error{
    // клиент для работы с хранилищем
    client, err := storage.NewClient(ctx)
    if err != nil {
        // TODO: handle error.
    }
    // откроем файл в Google Cloud Storage
    file, err := client.Bucket(bucket).Object(filename).NewReader(ctx)
    if err != nil {
        // TODO: handle error.
    }
    // здесь обрабатываем данные
    ...
}
```

*Если что-то не понятно по работе с `Google Cloud Storage` — это нормально. Здесь приведён код, как пример, что могут быть различные источники данных и, как следствие, различная логика открытия файлов.*

Чтобы избежать таких ненужных изменений, вынесем код открытия файлов в отдельные функции:

```
func OpenLocalFile(filename string) (*os.File, error){
    // открытие локального файла
}
func OpenGCSFile(
    ctx context.Context,
    bucket, fileName string,
) (*storage.Reader, error){
    // открытие файла из Google Cloud Storage
}
```

А в функции `ProcessData` оставим только обработку данных (без логики открытия файла):

```
// обратите внимание, что теперь мы не передаем имя файла
func ProcessData(ctx context.Context, reader io.Reader) error{
    // здесь обрабатываем данные
    ...
}
```

`OpenLocalFile` и `OpenGCSFile` возвращают разные объекты, но все они реализуют интерфейс `io.Reader`, поэтому мы можем сделать так:

```
readerLocal, err := OpenLocalFile("myfile.txt")
if err != nil {
    // TODO: handle error.
}
err = ProcessData(ctx, readerLocal)
```

или так:

```
readerGCS, err := OpenGCSFile(ctx, "bucketname", "myfile.txt")
if err != nil {
    // TODO: handle error.
}
err = ProcessData(ctx, readerGCS)
```

Теперь `ProcessData` не зависит от источника данных и не требует внесения изменений в случае его изменения.

Рассмотрим, что может быть внутри `ProcessData`:

```
func ProcessData(ctx context.Context, reader io.Reader) error {
    data := make([]byte, 1024) // Создадим буфер для чтения в него данных
    bytesRead, err := reader.Read(data) // Прочитаем данные в буфер
    if err != nil {
        // TODO: handle error.
    }
    // сколько прочитали байт и сам контент
    fmt.Printf("Прочитано %d байт: %s", bytesRead, string(data[:bytesRead]))
    return nil
}
```

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»