Раздел 12. Ошибки

Ошибки в Go - это значения, которые представляют собой неудачное выполнение какой-либо операции. Они используются для обработки и возврата информации об ошибках, которые могут возникнуть при выполнении программы.

Например, если вы пытаетесь открыть файл, который не существует, функция `os.Open` вернет ошибку.

В Go, ошибки являются важной частью API. Функции, которые могут завершиться неудачей, обычно возвращают ошибку в качестве последнего возвращаемого значения.

```
f, err := os.Open("filename.ext")
if err != nil {
    // Файл не открыт, обрабатываем ошибку
    log.Fatal(err)
}
// Файл открыт успешно, продолжаем
```

Пример выше демонстрирует, как вы можете возвращать ошибку от функции `os.Open` и затем проверять, была ли ошибка, прежде чем продолжать.

В этом примере мы вызываем функцию `Open` и передаем ей имя файла, который нужно открыть. Функция возвращает два значения - файл и ошибку. Мы проверяем, не содержит ли переменная `err` значение `nil`, и выводим значение переменной в стандартный вывод.

Если в программе возникает ошибка, она может быть обработана при помощи конструкции `if` с проверкой переменной ошибки на значение `nil`. Если переменная содержит значение отличное от `nil`, это означает, что произошла ошибка, и нужно выполнить определенные действия для ее обработки.

В Go вы можете создавать собственные ошибки, используя функцию `errors.New` или `fmt.Errorf`. Это полезно, когда вам нужно предоставить

дополнительный контекст об ошибке или когда вам нужно определить свои собственные типы ошибок.

Пример создания простой ошибки:

```
var ErrInvalidInput = errors.New("invalid input")
```

В этом примере `ErrInvalidInput` является пользовательской ошибкой, которую можно сравнить с другими ошибками.

Вы также можете добавить дополнительный контекст к ошибке, используя `fmt.Errorf`:

```
func divide(x, y int) (int, error) {
    if y == 0 {
        return 0, fmt.Errorf("division by zero: %d/%d", x, y)
    }
    return x / y, nil
}
```

В этом примере, если при делении происходит деление на ноль, функция возвращает ошибку с дополнительной информацией.

Если вы хотите определить свой собственный тип ошибки, вы можете создать структуру, которая реализует интерфейс `error`. Это позволяет вам добавлять дополнительные поля в свои ошибки и предоставлять более подробную информацию о том, что пошло не так.

```
type MyError struct {
    Msg    string
    File   string
    Line   int
}

func (e *MyError) Error() string {
    return fmt.Sprintf("%s:%d: %s", e.File, e.Line, e.Msg)
}
```

В этом примере, `MyError` - это собственная ошибка, которая содержит информацию о файле и строке, где произошла ошибка, а также сообщение об ошибке.

Домашнее задание

1. **Возвращение ошибки из функции**: Напишите функцию, которая возвращает ошибку. Вызовите эту функцию и обработайте возвращенную ошибку.
2. **Создание новой ошибки**: Используйте функцию errors.New для создания новой ошибки. Верните эту ошибку из вашей функции и обработайте её.
3. **Проверка на наличие ошибки**: Напишите функцию, которая возвращает ошибку в некоторых случаях. Вызовите эту функцию и проверьте, есть ли ошибка, прежде чем продолжать выполнение кода.
4. **Использование fmt.Errorf для создания ошибки**: Используйте функцию fmt.Errorf для создания новой ошибки с форматированным сообщением. Верните эту ошибку из вашей функции и обработайте её.
5. **Обработка ошибки из стандартной библиотеки**: Вызовите функцию из стандартной библиотеки Go, которая может вернуть ошибку (например, os.Open). Обработайте возвращаемую ошибку.
6. **Создание кастомного типа ошибки (⭐ сложное)**: Создайте свой собственный тип ошибки, реализовав интерфейс error. Создайте функцию, которая возвращает вашу кастомную ошибку, и обработайте её.