BEST SELLER

MESTRE BIBLIOTECAS DE LINGUAGEM GO

FERRAMENTAS ESSENCIAIS PARA APRENDIZAGEM RÁPIDA

Para programadores Go, explore 100 bibliotecas essenciais em apenas 1 hora com nosso melhor guia abrangente

KANRO TOMOYA

Index

Chapter 1 Introduction

1. Purpose

Bem-vindo ao mundo da programação em Go, uma linguagem poderosa e eficiente que se destaca pela sua simplicidade e capacidade de escalabilidade.

Este guia foi criado para ajudá-lo a dominar rapidamente as funcionalidades mais essenciais dos pacotes Go. Se você é um iniciante ou já tem experiência com outras linguagens, encontrará valiosos recursos para acelerar seu aprendizado e aprimorar suas habilidades.

Através de exemplos práticos e diretos, este livro oferece uma abordagem passo a passo para entender como cada pacote pode ser utilizado em diferentes cenários de desenvolvimento de software. Nossa esperança é que este recurso sirva como um trampolim para o seu sucesso na programação Go.

Desfrute da jornada e dos desafios que você irá superar ao se tornar um desenvolvedor Go competente.

Chapter 2 standard library

1. net/http

A biblioteca net/http permite que você crie servidores HTTP em Go.

Ex:net/http

```
package main
import (
	"fmt"
	"net/http"
)
func handler(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "Olá, Mundo!")
}
func main() {
	http.HandleFunc("/", handler)
	http.ListenAndServe(":8080", nil)
}
```

Se você executar esse programa e acessar 'http://localhost:8080' no seu navegador, verá a mensagem "Olá, Mundo!".

Esse exemplo de código cria um servidor HTTP básico. Na função main, http.HandleFunc é usada para registrar uma função de manipulação para a rota raiz "/", que é handler. Esta fun ção handler recebe dois parâmetros: http.ResponseWriter e *http.Request. O http.Response Writer é usado para escrever a resposta que será enviada ao cliente. A função fmt.Fprintf é chamada para enviar a string "Olá, Mundo!" como resposta HTTP. http.ListenAndServe inici a o servidor na porta 8080 e escuta as requisições que chegam. Se você acessar 'http://loca lhost:8080' no seu navegador após executar esse código, o servidor responderá com a men sagem.

2. os

A biblioteca os fornece uma forma portátil de interagir com o sistema operacional.

Ex:os

```
package main
import (
	"fmt"
	"os"
)
func main() {
	path := "/tmp/testfile.txt"
	file, err := os.Create(path)
	if err != nil {
		fmt.Println("Erro ao criar o arquivo:", err)
		return
	}
	defer file.Close()
	fmt.Println("Arquivo criado com sucesso:", path)
}
```

```
Arquivo criado com sucesso: /tmp/testfile.txt
```

Neste exemplo, usamos a função os.Create para criar um novo arquivo no sistema operacio nal. A função retorna um objeto *os.File e um valor de erro. Se err não for nil, significa que h ouve um erro ao criar o arquivo, e a mensagem de erro será impressa. Se o arquivo for cria do com sucesso, a mensagem "Arquivo criado com sucesso: /tmp/testfile.txt" será impressa. O defer file.Close() é usado para garantir que o arquivo seja fechado quando a função main terminar de executar, o que é uma prática importante para evitar vazamentos de recursos n o seu programa.

3. log

A biblioteca log em Go oferece funcionalidades para registrar mensagens de log, permitindo uma maneira fácil de reportar erros, avisos e outras informações importantes durante a exec ução de um programa.

Ex:log

```
package main
import (
    "log"
    "os"
)
func main() {
    arquivoLog, err := os.OpenFile("meu_log.log", os.O_APPEND|os.O_CREATE|os.O_WRO
NLY, 0666)
    if err != nil {
        log.Fatal(err)
    }
    defer arquivoLog.Close()
    log.SetOutput(arquivoLog)
    log.Println("Esta é uma mensagem de log!")
}
```

O arquivo meu_log.log é criado ou modificado, contendo a mensagem "Esta é uma mensag em de log!"

No código acima, a função OpenFile da biblioteca os é usada para abrir ou criar um arquivo chamado "meu_log.log". O modo os.O_APPEND é usado para adicionar mensagens ao fina l do arquivo se ele já existir, os.O_CREATE cria o arquivo se não existir, e os.O_WRONLY abre o arquivo somente para escrita. A função log.SetOutput define o destino dos logs para o arquivo que acabamos de abrir, em vez de usar a saída padrão (como o console). A funçã o log.Println é usada para escrever uma mensagem de log no arquivo. O uso de defer arqui voLog.Close() garante que o arquivo será fechado quando a função main terminar, o que é uma prática importante para liberar recursos.

4. image

A biblioteca image oferece ferramentas para manipular imagens em diferentes formatos. Ela permite carregar, modificar e salvar imagens, trabalhando com pixels e outras operações básicas.

Ex:image

```
package main
import (
    "image"
    "image/color"
    "image/png"
    "os"
)
func main() {
    img := image.NewRGBA(image.Rect(0, 0, 100, 100))
    yellow := color.RGBA{255, 255, 0, 255}
    for x := 0; x < 100; x++ {
        for y := 0; y < 100; y++ {
            img.Set(x, y, yellow)
        }
    }
    f, err := os.Create("yellow_image.png")
    if err != nil {
        log.Fatal(err)
    }
    defer f.Close()
    png.Encode(f, img)
}
```

O arquivo yellow_image.png é criado, contendo uma imagem de 100x100 pixels completamente amarela.

O código utiliza a função image.NewRGBA para criar uma nova imagem com 100x100 pixels, onde cada pixel pode ser modificado individualmente. A cor amarela é definida usando color.RGBA, onde cada valor representa a intensidade de vermelho, verde, azul e o canal alfa (transparência), respectivamente. A imagem é preenchida com a cor amarela usando um laço duplo que itera sobre cada pixel da imagem. A função os.Create é usada para criar um novo arquivo chamado "yellow_image.png". Após isso, a imagem é codificada em formato PNG usando png.Encode e salva no arquivo. A função defer f.Close() é crucial para assegurar que o arquivo aberto seja fechado após a conclusão das operações, o que ajuda a evitar vazamentos de recursos.

5. database/sql

O pacote database/sql em Go proporciona uma maneira generalizada para interação com b ancos de dados SQL. Este pacote oferece funcionalidades para executar consultas, atualiza ções e obter os resultados em formas controláveis e seguras.

Ex:database/sql

```
package main
import (
    "database/sql"
    _ "github.com/mattn/go-sqlite3"
    "fmt"
)
func main() {
    db, err := sql.Open("sqlite3", "file:test.db?cache=shared&mode=memory")
    if err != nil {
        fmt.Println("Erro ao abrir banco de dados:", err)
        return
    }
    defer db.Close()
    _, err = db.Exec("CREATE TABLE IF NOT EXISTS users (name TEXT)")
    if err != nil {
        fmt.Println("Erro ao criar tabela:", err)
        return
    }
    _, err = db.Exec("INSERT INTO users (name) VALUES (?)", "Alice")
    if err != nil {
        fmt.Println("Erro ao inserir usuário:", err)
        return
    }
    rows, err := db.Query("SELECT name FROM users")
    if err != nil {
        fmt.Println("Erro ao consultar usuários:", err)
        return
    }
    defer rows.Close()
    for rows.Next() {
        var name string
        if err := rows.Scan(&name); err != nil {
            fmt.Println("Erro ao ler dados:", err)
            return
        }
        fmt.Println("Usuário:", name)
    }
}
```

```
```

```
Usuário: Alice
```

Este código inicia com a importação do pacote database/sql e do driver SQLite. Primeiro, el e tenta abrir uma conexão com um banco de dados SQLite em memória. Após isso, tenta cri ar uma tabela de usuários se ela não existir e insere um novo usuário com nome "Alice". Po r fim, faz uma consulta para buscar todos os usuários e imprime seus nomes. defer é utiliza do para garantir que os recursos do banco de dados sejam liberados corretamente após seu uso. Cada operação tem tratamento de erro para garantir que qualquer problema durante a execução seja devidamente reportado e tratado.