```go
package main

import (
    "flag"
    "fmt"
    "log"
    "os"
    "path/filepath"
    "strings"

    "github.com/jung-kurt/gofpdf"
)

// 代码块结构
type Chunk struct {
    StartLine int    // 起始行号
    EndLine   int    // 结束行号
    Content   string // 代码内容
}

// 代码转PDF配置
type Config struct {
    InputPath    string  // 输入文件/目录路径
    OutputPath   string  // 输出PDF路径
    ProjectName  string  // 项目名称（用于页眉）
    LinesPerPage int     // 每页行数
    TotalPages   int     // 总页数
    CodeChunks   []Chunk // 代码块（前30页和后30页）
}

func main() {
    // 解析命令行参数
    config := parseFlags()

    // 处理代码文件
    err := processCodeFiles(&config)
    if err != nil {
        log.Fatalf("处理代码文件失败: %v", err)
    }

    // 生成PDF
    err = generatePDF(&config)
    if err != nil {
        log.Fatalf("生成PDF失败: %v", err)
    }

    fmt.Printf("PDF生成成功: %s\n", config.OutputPath)
}

// 解析命令行参数
```

```
func parseFlags() Config {
    inputPath := flag.String("input", "", "输入文件或目录路径")
    outputPath := flag.String("output", "code_document.pdf", "输出PDF路
")
    projectName := flag.String("project", "项目名称", "项目名称（用于页
）")
    linesPerPage := flag.Int("lines-per-page", 50, "每页行数")
    totalPages := flag.Int("total-pages", 60, "总页数")

    flag.Parse()

    if *inputPath == "" {
        log.Fatal("请指定输入文件或目录路径 (-input)")
    }

    return Config{
        InputPath:    *inputPath,
        OutputPath:   *outputPath,
        ProjectName:  *projectName,
        LinesPerPage: *linesPerPage,
        TotalPages:   *totalPages,
    }
}

// 处理代码文件
func processCodeFiles(config *Config) error {
    var allCodeLines []string

    // 判断是文件还是目录
    fileInfo, err := os.Stat(config.InputPath)
    if err != nil {
        return fmt.Errorf("获取文件信息失败: %v", err)
    }

    if fileInfo.IsDir() {
        // 处理目录
        err := filepath.Walk(config.InputPath, func(path string, info
os.FileInfo, err error) error {
            if err != nil {
                return err
            }

            // 跳过目录
            if info.IsDir() {
                return nil
            }

            // 只处理代码文件（可根据需要扩展）
            if isCodeFile(info.Name()) {
                lines, err := readFileLines(path)
                if err != nil {
                    return fmt.Errorf("读取文件 %s 失败: %v", path, err)
```

```
50                  }
```

```
                allCodeLines = append(allCodeLines, lines...)
            }

            return nil
        })

        if err != nil {
            return fmt.Errorf("遍历目录失败: %v", err)
        }
    } else {
        // 处理单个文件
        lines, err := readFileLines(config.InputPath)
        if err != nil {
            return fmt.Errorf("读取文件失败: %v", err)
        }

        allCodeLines = lines
    }

    // 计算需要提取的代码块
    totalLines := len(allCodeLines)

    // 如果总行数超过配置的页数限制，取前一半和后一半
    if totalLines > config.LinesPerPage*config.TotalPages {
        halfPages := config.LinesPerPage * config.TotalPages / 2
        firstChunk := Chunk{
            StartLine: 1,
            EndLine:   halfPages,
            Content:   strings.Join(allCodeLines[:halfPages], "\n"),
        }

        lastChunkStart := totalLines - halfPages
        lastChunk := Chunk{
            StartLine: lastChunkStart + 1,
            EndLine:   totalLines,
            Content:   strings.Join(allCodeLines[lastChunkStart:], "\n"),
        }

        config.CodeChunks = []Chunk{firstChunk, lastChunk}
    } else {
        // 如果总行数不足配置的页数限制，取全部代码
        config.CodeChunks = []Chunk{
            {
                StartLine: 1,
                EndLine:   totalLines,
                Content:   strings.Join(allCodeLines, "\n"),
            },
        }
    }
```

```
1
2       return nil
3   }
4
5   // 判断是否为代码文件
6   func isCodeFile(filename string) bool {
7       codeExtensions := []string{
8           ".go", ".java", ".py", ".js", ".ts", ".html", ".css", ".cpp", ".c", ".h", ".cs",
    ".php",
9       }
10
11      ext := filepath.Ext(filename)
12      for _, e := range codeExtensions {
13          if e == ext {
14              return true
15          }
16      }
17
18      return false
19  }
20
21  // 读取文件行
22  func readFileLines(path string) ([]string, error) {
23      data, err := os.ReadFile(path)
24      if err != nil {
25          return nil, fmt.Errorf("读取文件失败 %s: %w", path, err)
26      }
27
28      // 处理不同的换行符
29      text := string(data)
30      text = strings.ReplaceAll(text, "\r\n", "\n") // Windows换行符转换
    Unix换行符
31      text = strings.ReplaceAll(text, "\r", "\n")   // 旧Mac换行符转换为Unix换
    符
32
33      // 处理特殊空格字符，防止在PDF中显示为方框
34      text = strings.ReplaceAll(text, "\t", "    ") // 将制表符替换为4个空格
35
36      // 处理其他可能导致方框显示的特殊空格字符
37      text = strings.Map(func(r rune) rune {
38          // 如果是不可见字符但不是普通空格、换行符或回车符，则替换为普通
    格
39          if (r < 32 || r == 127 || (r >= 128 && r <= 159)) && r != '\n' && r
    != '\r' && r != ' ' {
40              return ' '
41          }
42          return r
43      }, text)
44
45      return strings.Split(text, "\n"), nil
46  }
47
```

```
48  // 生成PDF
49  func generatePDF(config *Config) error {
50      // 创建PDF对象 - 使用中文支持
```

```
    pdf := gofpdf.New("P", "mm", "A4", "")
    pdf.SetAutoPageBreak(true, 10) // 减小页脚边距，使内容更紧凑

    // 设置总页数占位符
    pdf.AliasNbPages("11")

    // 添加中文字体支持
    pdf.AddUTF8Font("msyh", "", "fonts/微软雅黑.ttf")

    // 使用Courier字体处理代码（等宽字体，更适合代码显示）
    pdf.SetFont("Courier", "", 8)

    // 设置字符间距
    pdf.SetCellMargin(0)

    // 添加页眉和页脚
    pdf.SetHeaderFunc(func() {
        pdf.SetY(10) // 增加页眉顶部间距
        pdf.SetFont("msyh", "", 10)
        pdf.CellFormat(0, 8, fmt.Sprintf("%s ", config.ProjectName), "", 0, "L", false, 0, "")
        pdf.Ln(10) // 增加页眉底部间距
        // 添加分割线
        pdf.SetLineWidth(0.5)
        pdf.Line(10, pdf.GetY(), 200, pdf.GetY())
        pdf.Ln(5) // 分割线后间距
    })

    pdf.SetFooterFunc(func() {
        pdf.SetY(-12)                                                    // 小页脚底部边距
        pdf.SetFont("msyh", "", 8)                                       // 使用微软雅黑显示中文
        pdf.CellFormat(0, 8, fmt.Sprintf("第 %d 页，共 11 页", pdf.PageNo()), "", 0, "C", false, 0, "") // 使用占位符表示总页数
        pdf.Ln(4)                                                        // 减页脚内部间距
    })

    // 处理每个代码块
    for i, chunk := range config.CodeChunks {
        // 为每个代码块添加标题页
        pdf.AddPage()
        pdf.SetFont("msyh", "", 14) // 使用微软雅黑显示中文

        if len(config.CodeChunks) > 1 {
            if i == 0 {
                pdf.CellFormat(0, 10, fmt.Sprintf("前 %d 行代码", chunk.EndLine), "", 1, "C", false, 0, "")
            } else {
                pdf.CellFormat(0, 10, fmt.Sprintf("后 %d 行代码 (总行数: %d)", chunk.EndLine-chunk.StartLine+1,
```

```
        config.CodeChunks[0].EndLine+chunk.EndLine-chunk.StartLine+1), "", 1,
        "C", false, 0, "")
46              }
47          } else {
48              // pdf.CellFormat(0, 10, fmt.Sprintf("代码文件: %s",
        filepath.Base(config.InputPath)), "", 1, "C", false, 0, "")
49          }
50
```

```
        pdf.Ln(5) // 减小标题页与内容之间的间距

        // 分割代码内容为行
        lines := strings.Split(chunk.Content, "\n")

        // 按每页行数分页显示代码
        linesPerPage := config.LinesPerPage

        for i := 0; i < len(lines); i += linesPerPage {
            // 如果不是第一页，添加新页
            if i > 0 {
                pdf.AddPage()
            }

            // 每页显示指定行数
            end := i + linesPerPage
            if end > len(lines) {
                end = len(lines)
            }

            pageLines := lines[i:end]

            // 每页重置行号计数器
            pageLineNumber := 1

            // 创建代码表格
            for _, line := range pageLines {
                // 行号宽度
                lineNumWidth := float64(15) // 增加行号宽度，为行号和代码之创造更多空间

                // 行号单元格
                pdf.SetTextColor(128, 128, 128)
             // 灰色
                pdf.CellFormat(lineNumWidth, 5, fmt.Sprintf("%4d", pageLineNumber), "", 0, "R", false, 0, "") // 右对齐行号，每页从1开始

                // 添加间距单元格
                spacerWidth := float64(5)                                  // 额外的间距度
                pdf.CellFormat(spacerWidth, 5, "", "", 0, "L", false, 0, "") // 空白距单元格

                // 代码单元格
                pdf.SetTextColor(0, 0, 0) // 黑色
                availableWidth := float64(190) - lineNumWidth - spacerWidth
                pdf.MultiCell(availableWidth, 5, line, "", "L", false) // 左对齐代内容

                pageLineNumber++ // 页内行号递增
            }

```

```
47          // 表格底部横线 - 减小间距
48          pdf.Ln(0.5)
49          pdf.SetLineWidth(0.2) // 减小线宽
50          pdf.Line(20, pdf.GetY(), 190, pdf.GetY())
```

```
            pdf.Ln(2) // 减小底部间距
        }
    }

    // 输出PDF
    return pdf.OutputFileAndClose(config.OutputPath)
}

```