pixcode.py

python · 972 lines · 37.7 KB

Path: pixcode.py
Language: python
Lines: 972
Size: 37.7 KB

```python
#!/usr/bin/env python3
"""
pixcode - 将代码仓库转为分层次、结构化的PDF集合
"""

import os
import sys
import argparse
import fnmatch
import re
from pathlib import Path
from dataclasses import dataclass, field
from datetime import datetime

from reportlab.lib.pagesizes import A4
from reportlab.lib.units import mm
from reportlab.lib.colors import HexColor, white, Color
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer, Table, TableStyle,
    PageBreak,
)
from reportlab.platypus.flowables import Flowable
from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont

# ═══════════════════════════════════════════════════════════
# 字体注册 — 修正版
# ═══════════════════════════════════════════════════════════
# 所有 canvas 绘制统一使用这些变量
FONT_NORMAL = 'Helvetica'
FONT_BOLD = 'Helvetica-Bold'
FONT_MONO = 'Courier'
FONT_MONO_BOLD = 'Courier-Bold'


def _register_fonts():
    """
    注册 CJK 字体。关键点：
    - 比例字体: 用于标题、段落、表格等
    - 等宽字体: 用于代码块。但系统一般没有中文等宽字体，
      所以我们用同一个 CJK 字体同时作为 mono 使用。
      代码块中字符宽度靠 drawString + 手动 x 偏移控制，
      不依赖字体本身等宽。
    """
    global FONT_NORMAL, FONT_BOLD, FONT_MONO, FONT_MONO_BOLD

    candidates = [
    # Windows
        (r'C:\Windows\Fonts\msyh.ttc', 'CJK_Normal'),
        (r'C:\Windows\Fonts\msyhbd.ttc', 'CJK_Bold'),
        (r'C:\Windows\Fonts\simhei.ttf', 'CJK_Normal'),
        (r'C:\Windows\Fonts\simsun.ttc', 'CJK_Normal'),
    # macOS
        ('/System/Library/Fonts/PingFang.ttc', 'CJK_Normal'),
        ('/System/Library/Fonts/STHeiti Medium.ttc', 'CJK_Normal'),
        ('/System/Library/Fonts/STHeiti Light.ttc', 'CJK_Normal'),
        ('/Library/Fonts/Arial Unicode.ttf', 'CJK_Normal'),
    # Linux
        ('/usr/share/fonts/truetype/wqy/wqy-microhei.ttc', 'CJK_Normal'),
```

```
            ('/usr/share/fonts/truetype/wqy/wqy-zenhei.ttc',  'CJK_Normal'),
            ('/usr/share/fonts/opentype/noto/NotoSansCJK-Regular.ttc',  'CJK_Normal'),
            ('/usr/share/fonts/noto-cjk/NotoSansCJK-Regular.ttc',  'CJK_Normal'),
            ('/usr/share/fonts/google-noto-cjk/NotoSansCJK-Regular.ttc',  'CJK_Normal'),
        ]

        registered_normal = False
        registered_bold = False

        for font_path, font_name in candidates:
            if not os.path.exists(font_path):
                continue
            try:
                if font_name == 'CJK_Bold' and not registered_bold:
                    pdfmetrics.registerFont(TTFont('CJK_Bold', font_path))
                    registered_bold = True
                    print(f"  CJK Bold : {font_path}")
                elif font_name == 'CJK_Normal' and not registered_normal:
                    pdfmetrics.registerFont(TTFont('CJK_Normal', font_path))
                    registered_normal = True
                    print(f"  CJK Normal: {font_path}")
            # 没有独立 bold 就用 normal 兜底
                    if not registered_bold:
                        try:
                            pdfmetrics.registerFont(TTFont('CJK_Bold', font_path))
                            registered_bold = True
                        except Exception:
                            pass
            except Exception as e:
                print(f"  Font registration failed for {font_path}: {e}")
                continue

            if registered_normal and registered_bold:
                break

        if registered_normal:
            FONT_NORMAL = 'CJK_Normal'
            FONT_BOLD = 'CJK_Bold' if registered_bold else 'CJK_Normal'
        # 代码块也用 CJK 字体（确保中文能显示）
            FONT_MONO = 'CJK_Normal'
            FONT_MONO_BOLD = 'CJK_Bold' if registered_bold else 'CJK_Normal'
            print(f"  All rendering will use CJK font")
        else:
            print("  No CJK font found! Chinese characters will show as □")
            print("  On Windows: msyh.ttc should exist in C:\\Windows\\Fonts\\")
            print("  On Linux : apt install fonts-wqy-microhei")
            print("  On macOS: PingFang should be built-in")


_register_fonts()

# ============================================================
# 颜色主题 (One Dark)
# ============================================================
COLORS = {
    'bg':       HexColor('#282c34'),
    'bg_light':  HexColor('#2c313a'),
    'fg':       HexColor('#abb2bf'),
    'comment':   HexColor('#5c6370'),
    'keyword':   HexColor('#c678dd'),
    'string':    HexColor('#98c379'),
    'number':    HexColor('#d19a66'),
    'function':  HexColor('#61afef'),
    'type':      HexColor('#e5c07b'),
    'operator':  HexColor('#56b6c2'),
    'accent':    HexColor('#61afef'),
    'accent2':   HexColor('#c678dd'),
    'border':    HexColor('#3e4451'),
    'line_no':   HexColor('#4b5263'),
    'white':     HexColor('#ffffff'),
```

```
    'red':        HexColor('#e06c75'),
    'green':      HexColor('#98c379'),
    'header_bg':  HexColor('#21252b'),
    'tag':        HexColor('#e06c75'),
}

# ============================================================
# 语言检测 & 语法高亮数据
# ============================================================
LANG_MAP= {
    '.py': 'python', '.pyw': 'python',
    '.js': 'javascript', '.mjs': 'javascript', '.cjs': 'javascript',
    '.ts': 'typescript', '.tsx': 'typescript', '.jsx': 'javascript',
    '.java': 'java',
    '.c': 'c', '.h': 'c',
    '.cpp': 'cpp', '.cc': 'cpp', '.cxx': 'cpp', '.hpp': 'cpp',
    '.cs': 'csharp', '.go': 'go', '.rs': 'rust', '.rb': 'ruby',
    '.php': 'php', '.swift': 'swift',
    '.kt': 'kotlin', '.kts': 'kotlin', '.scala': 'scala',
    '.r': 'r', '.R': 'r',
    '.sh': 'bash', '.bash': 'bash', '.zsh': 'bash',
    '.sql': 'sql',
    '.html': 'html', '.htm': 'html',
    '.css': 'css', '.scss': 'css', '.sass': 'css', '.less': 'css',
    '.xml': 'xml', '.xsl': 'xml',
    '.json': 'json', '.yaml': 'yaml', '.yml': 'yaml',
    '.toml': 'toml', '.md': 'markdown', '.txt': 'text',
    '.ini': 'ini', '.cfg': 'ini',
    '.dockerfile': 'docker', '.lua': 'lua',
    '.pl': 'perl', '.pm': 'perl',
    '.ex': 'elixir', '.exs': 'elixir',
    '.erl': 'erlang', '.hrl': 'erlang',
    '.hs': 'haskell', '.ml': 'ocaml', '.mli': 'ocaml',
    '.vim': 'vim', '.el': 'elisp',
    '.clj': 'clojure', '.cljs': 'clojure',
    '.dart': 'dart', '.v': 'v', '.zig': 'zig', '.nim': 'nim',
    '.tf': 'terraform', '.proto': 'protobuf',
    '.graphql': 'graphql', '.gql': 'graphql',
    '.vue': 'vue', '.svelte': 'svelte',
}

KEYWORDS= {
    'python': {
        'False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
        'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
        'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is',
        'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
        'while', 'with', 'yield',
        },
    'javascript': {
        'async', 'await', 'break', 'case', 'catch', 'class', 'const',
        'continue', 'debugger', 'default', 'delete', 'do', 'else', 'export',
        'extends', 'false', 'finally', 'for', 'function', 'if', 'import',
        'in', 'instanceof', 'let', 'new', 'null', 'of', 'return', 'static',
        'super', 'switch', 'this', 'throw', 'true', 'try', 'typeof',
        'undefined', 'var', 'void', 'while', 'with', 'yield',
        },
    'go': {
        'break', 'case', 'chan', 'const', 'continue', 'default', 'defer',
        'else', 'fallthrough', 'for', 'func', 'go', 'goto', 'if', 'import',
        'interface', 'map', 'package', 'range', 'return', 'select', 'struct',
        'switch', 'type', 'var', 'true', 'false', 'nil',
        },
    'rust': {
        'as', 'async', 'await', 'break', 'const', 'continue', 'crate', 'dyn',
        'else', 'enum', 'extern', 'false', 'fn', 'for', 'if', 'impl', 'in',
        'let', 'loop', 'match', 'mod', 'move', 'mut', 'pub', 'ref', 'return',
        'self', 'Self', 'static', 'struct', 'super', 'trait', 'true', 'type',
        'unsafe', 'use', 'where', 'while',
        },
```

```
    'java': {
        'abstract', 'assert', 'boolean', 'break', 'byte', 'case', 'catch',
        'char', 'class', 'const', 'continue', 'default', 'do', 'double',
        'else', 'enum', 'extends', 'false', 'final', 'finally', 'float',
        'for', 'if', 'implements', 'import', 'instanceof', 'int', 'interface',
        'long', 'native', 'new', 'null', 'package', 'private', 'protected',
        'public', 'return', 'short', 'static', 'super', 'switch', 'this',
        'throw', 'throws', 'true', 'try', 'void', 'volatile', 'while',
        },
}
KEYWORDS['typescript']       = KEYWORDS['javascript']
KEYWORDS['cpp']   = KEYWORDS['java']    | {
    'auto', 'bool', 'delete', 'explicit', 'friend', 'inline', 'mutable',
    'namespace', 'noexcept', 'nullptr', 'operator', 'override', 'private',
    'protected', 'public', 'register', 'sizeof', 'struct', 'template',
    'thread_local', 'typedef', 'typeid', 'typename', 'union', 'using',
    'virtual', 'wchar_t',
}
KEYWORDS['c']   = KEYWORDS['cpp']
KEYWORDS['csharp']    = KEYWORDS['java']

BUILTIN_FUNCTIONS  = {
    'python': {
        'print', 'len', 'range', 'int', 'str', 'float', 'list', 'dict',
        'set', 'tuple', 'bool', 'type', 'isinstance', 'super', 'property',
        'classmethod', 'staticmethod', 'enumerate', 'zip', 'map', 'filter',
        'sorted', 'reversed', 'any', 'all', 'min', 'max', 'sum', 'abs',
        'round', 'input', 'open', 'hasattr', 'getattr', 'setattr',
        'callable', 'iter', 'next', 'repr', 'hash', 'id', 'dir',
        'vars', 'globals', 'locals', 'format', 'ord', 'chr', 'hex', 'oct',
        },
}

COMMENT_STYLES= {
    'python': '#', 'bash': '#', 'ruby': '#', 'yaml': '#', 'toml': '#',
    'ini': ';',
    'javascript': '//', 'typescript': '//', 'java': '//', 'c': '//',
    'cpp': '//', 'csharp': '//', 'go': '//', 'rust': '//', 'swift': '//',
    'kotlin': '//', 'scala': '//', 'dart': '//',
    'sql': '--', 'lua': '--', 'haskell': '--',
}

# ============================================================
# 默认忽略规则
# ============================================================
DEFAULT_IGNORE_DIRS  = {
    '.git', '.svn', '.hg', '__pycache__', '.pytest_cache', '.mypy_cache',
    '.ruff_cache', 'node_modules', 'bower_components', '.venv', 'venv',
    'env', '.env', '.tox', '.nox', 'dist', 'build', '_build', '.idea',
    '.vscode', '.vs', 'target', 'vendor', '.next', '.nuxt', 'coverage',
    '.coverage', '.terraform', 'egg-info',
}

DEFAULT_IGNORE_PATTERNS = [
    '*.pyc', '*.pyo', '*.pyd', '*.so', '*.dylib', '*.dll', '*.o', '*.a',
    '*.exe', '*.bin', '*.class', '*.jar', '*.war',
    '*.min.js', '*.min.css', '*.map',
    '*.lock', 'package-lock.json', 'yarn.lock', 'pnpm-lock.yaml',
    '*.log', '*.png', '*.jpg', '*.jpeg', '*.gif', '*.bmp', '*.ico',
    '*.svg', '*.webp', '*.mp3', '*.mp4', '*.avi', '*.mov', '*.wav',
    '*.zip', '*.tar', '*.gz', '*.bz2', '*.xz', '*.rar', '*.7z',
    '*.pdf', '*.doc', '*.docx', '*.xls', '*.xlsx', '*.ppt', '*.pptx',
    '*.woff', '*.woff2', '*.ttf', '*.eot', '*.otf',
    '*.db', '*.sqlite', '*.sqlite3',
    '.DS_Store', 'Thumbs.db', '.gitignore', '.gitattributes',
]


# ============================================================
# 数据模型
```

```python
# ============================================================
@dataclass
class FileInfo:
    path: Path
    abs_path: Path
    language: str
    size: int
    line_count: int = 0
    content: str = ""
    index: int = 0


@dataclass
class RepoInfo:
    root: Path
    name: str
    files: list[FileInfo] = field(default_factory=list)
    total_lines: int = 0
    total_size: int = 0
    language_stats: dict = field(default_factory=dict)
    tree_str: str = ""


# ============================================================
# 仓库扫描器
# ============================================================
class RepoScanner:
    def __init__(self, root: str, max_file_size: int = 512 * 1024,
                 extra_ignore: list[str] = None):
        self.root = Path(root).resolve()
        self.max_file_size = max_file_size
        self.extra_ignore = extra_ignore or []

    def _should_ignore_dir(self, dirname: str) -> bool:
        return dirname in DEFAULT_IGNORE_DIRS or dirname.startswith('.')

    def _should_ignore_file(self, filename: str) -> bool:
        for pattern in DEFAULT_IGNORE_PATTERNS + self.extra_ignore:
            if fnmatch.fnmatch(filename, pattern) or \
               fnmatch.fnmatch(filename.lower(), pattern.lower()):
                return True
        return False

    def _detect_language(self, filepath: Path) -> str:
        special = {
          'dockerfile': 'docker', 'makefile': 'makefile',
          'cmakelists.txt': 'cmake', 'rakefile': 'ruby',
          'gemfile': 'ruby', 'requirements.txt': 'text',
          'pipfile': 'toml', 'cargo.toml': 'toml',
          'go.mod': 'go', 'go.sum': 'text',
        }
        name = filepath.name.lower()
        if name in special:
            return special[name]
        return LANG_MAP.get(filepath.suffix.lower(), 'text')

    def _is_text_file(self, filepath: Path) -> bool:
        try:
            with open(filepath, 'rb') as f:
                chunk = f.read(8192)
            return b'\x00' not in chunk
        except (IOError, OSError):
            return False

    def scan(self) -> RepoInfo:
        repo = RepoInfo(root=self.root, name=self.root.name)
        files = []

        for dirpath, dirnames, filenames in os.walk(self.root):
            dirnames[:] = sorted(d for d in dirnames if not self._should_ignore_dir(d))
```

```
                for filename in sorted(filenames):
                    if self._should_ignore_file(filename):
                        continue
                    filepath = Path(dirpath) / filename
                    rel_path = filepath.relative_to(self.root)
                    try:
                        size = filepath.stat().st_size
                    except OSError:
                        continue
                    if size > self.max_file_size or size == 0:
                        continue
                    if not self._is_text_file(filepath):
                        continue
                    try:
                        content = filepath.read_text(encoding='utf-8', errors='replace')
                    except (IOError, OSError):
                        continue
                    line_count = content.count('\n') + (
                        1 if content and not content.endswith('\n') else 0)
                    files.append(FileInfo(
                        path=rel_path, abs_path=filepath,
                        language=self._detect_language(filepath),
                        size=size, line_count=line_count, content=content,
                    ))

            files.sort(key=lambda f: str(f.path))
            for i, f in enumerate(files, 1):
                f.index = i

            repo.files = files
            repo.total_lines = sum(f.line_count for f in files)
            repo.total_size = sum(f.size for f in files)

            lang_stats = {}
            for f in files:
                lang_stats.setdefault(f.language, {'files': 0, 'lines': 0})
                lang_stats[f.language]['files'] += 1
                lang_stats[f.language]['lines'] += f.line_count
            repo.language_stats = dict(sorted(
                lang_stats.items(), key=lambda x: x[1]['lines'], reverse=True))
            repo.tree_str = self._build_tree(files)
            return repo

        def _build_tree(self, files: list[FileInfo]) -> str:
            tree = {}
            for f in files:
                parts = f.path.parts
                node = tree
                for part in parts[:-1]:
                    node = node.setdefault(part + '/', {})
                node[parts[-1]] = None
            lines = [f"{self.root.name}/"]
            self._tree_lines(tree, lines, "")
            return '\n'.join(lines)

        def _tree_lines(self, node: dict, lines: list, prefix: str):
            items = list(node.items())
            for i, (name, subtree) in enumerate(items):
                is_last = (i == len(items) - 1)
                connector = '└── ' if is_last else '├── '
                lines.append(f"{prefix}{connector}{name}")
                if subtree is not None:
                    extension = '    ' if is_last else '│   '
                    self._tree_lines(subtree, lines, prefix + extension)


# ============================================================
# 工具
# ============================================================
def xml_escape(text: str) -> str:
```

```
        return (text
                .replace('&', '&amp;')
                .replace('<', '&lt;')
                .replace('>', '&gt;')
                .replace('"', '&quot;')
                .replace("'", '&#39;'))


def _char_width(char: str, font_size: float) -> float:
    """
    估算单个字符的渲染宽度。
    CJK字符约为 font_size 宽，ASCII约为 font_size * 0.6。
    """
    cp = ord(char)
    # CJK 统一表意文字 + 全角标点等
    if (cp >= 0x2E80 and cp <= 0x9FFF) or \
       (cp >= 0xF900 and cp <= 0xFAFF) or \
       (cp >= 0xFE30 and cp <= 0xFE4F) or \
       (cp >= 0xFF00 and cp <= 0xFFEF) or \
       (cp >= 0x20000 and cp <= 0x2FA1F) or \
       (cp >= 0x3000 and cp <= 0x303F):
        return font_size * 1.0
    else:
        return font_size * 0.6


def _str_width(text: str, font_size: float) -> float:
    """估算字符串的渲染宽度"""
    return sum(_char_width(c, font_size) for c in text)


def _truncate_to_width(text: str, font_size: float, max_width: float) -> str:
    """将字符串截断到不超过 max_width 像素宽度"""
    w = 0.0
    for i, c in enumerate(text):
        w += _char_width(c, font_size)
        if w > max_width:
            return text[:i] + '...'
    return text


# ============================================================
# 自定义 Flowable: 代码块
# ============================================================
class CodeBlockChunk(Flowable):
    """
    渲染一段代码行，保证高度不超页面。
    所有文字用 FONT_MONO（已注册的CJK 字体）绘制。
    """

    def __init__(self, lines: list[str], language: str,
                 start_line: int = 1, width: float = None,
                 font_size: float = 6.5):
        super().__init__()
        self.code_lines = lines
        self.language = language
        self.start_line = start_line
        self.block_width = width or (A4[0] - 30 * mm)
        self.font_size = font_size
        self.line_height = font_size * 1.6
        self.padding = 6

        self.kw_set = KEYWORDS.get(language, set())
        self.builtin_set = BUILTIN_FUNCTIONS.get(language, set())
        self.line_comment = COMMENT_STYLES.get(language)

    def wrap(self, availWidth, availHeight):
        self.block_width = min(self.block_width, availWidth)
        h = len(self.code_lines) * self.line_height + self.padding * 2
        return (self.block_width, h)
```

```
481
482         def draw(self):
483             canv = self.canv
484             num_lines = len(self.code_lines)
485             total_h = num_lines * self.line_height + self.padding * 2
486
487         # 背景
488             canv.setFillColor(COLORS['bg'])
489             canv.roundRect(0, 0, self.block_width, total_h, 4, fill=1, stroke=0)
490
491         # 行号区域
492             max_no = self.start_line + num_lines
493             line_no_width = max(35, len(str(max_no)) * 7 + 14)
494
495             canv.setFillColor(COLORS['header_bg'])
496             canv.roundRect(0, 0, line_no_width, total_h, 4, fill=1, stroke=0)
497             canv.rect(line_no_width - 4, 0, 4, total_h, fill=1, stroke=0)
498
499             canv.setStrokeColor(COLORS['border'])
500             canv.setLineWidth(0.5)
501             canv.line(line_no_width, 0, line_no_width, total_h)
502
503             y = total_h - self.padding - self.line_height + 3
504             code_x = line_no_width + 8
505             code_area_width = self.block_width - code_x - 6
506
507             for i, line in enumerate(self.code_lines):
508                 line_no = self.start_line + i
509
510           # 行号 — 用 Courier (纯数字没问题)
511                 canv.setFont('Courier', self.font_size)
512                 canv.setFillColor(COLORS['line_no'])
513                 canv.drawRightString(line_no_width - 6, y, str(line_no))
514
515           # 代码 — 截断到可用宽度
516                 display = _truncate_to_width(line, self.font_size, code_area_width)
517
518                 self._draw_line(canv, code_x, y, display)
519                 y -= self.line_height
520
521         def _draw_line(self, canv, x, y, line):
522             fs = self.font_size
523             stripped = line.lstrip()
524
525         # 整行注释
526             if self.line_comment and stripped.startswith(self.line_comment):
527                 canv.setFont(FONT_MONO, fs)
528                 canv.setFillColor(COLORS['comment'])
529                 canv.drawString(x, y, line)
530                 return
531
532         # Python 装饰器
533             if self.language == 'python' and stripped.startswith('@'):
534                 canv.setFont(FONT_MONO, fs)
535                 canv.setFillColor(COLORS['function'])
536                 canv.drawString(x, y, line)
537                 return
538
539         # 字符串行
540             if stripped and stripped[0] in ('"', "'", '`'):
541                 canv.setFont(FONT_MONO, fs)
542                 canv.setFillColor(COLORS['string'])
543                 canv.drawString(x, y, line)
544                 return
545
546         # 分词着色
547             tokens = re.split(r'(\s+)', line)
548             cur_x = x
549
550             for token in tokens:
```

```
                if not token:
                    continue
                if token.isspace():
                    cur_x += _str_width(token, fs)
                    continue

                color = COLORS['fg']
                bold = False

                word = re.sub(r'^[^\w]*|[^\w]*$', '', token)

                if word in self.kw_set:
                    color = COLORS['keyword']
                    bold = True
                elif word in self.builtin_set:
                    color = COLORS['type']
                elif self.language == 'python' and word in ('self', 'cls'):
                    color = COLORS['red']
                elif re.match(r'^\d+\.?\d*$', word):
                    color = COLORS['number']
                elif any(token.startswith(c) for c in
                         ('"', "'", '`', 'f"', "f'", 'r"', "r'", 'b"', "b'")):
                    color = COLORS['string']

                canv.setFillColor(color)
                font = FONT_MONO_BOLD if bold else FONT_MONO
                canv.setFont(font, fs)
                canv.drawString(cur_x, y, token)
                cur_x += _str_width(token, fs)


class HeaderBar(Flowable):
    def __init__(self, text: str, subtext: str = "", width: float = None):
        super().__init__()
        self.text = text
        self.subtext = subtext
        self.bar_width = width or (A4[0] - 30 * mm)
        self.bar_height = 28 if subtext else 22

    def wrap(self, availWidth, availHeight):
        self.bar_width = min(self.bar_width, availWidth)
        return (self.bar_width, self.bar_height)

    def draw(self):
        canv = self.canv
        canv.setFillColor(COLORS['accent'])
        canv.roundRect(0, 0, self.bar_width, self.bar_height, 4, fill=1, stroke=0)
        canv.setFont(FONT_BOLD, 10)
        canv.setFillColor(COLORS['white'])
        canv.drawString(10, self.bar_height - 15, self.text)
        if self.subtext:
            canv.setFont(FONT_NORMAL, 7)
            canv.setFillColor(HexColor('#d0e8ff'))
            canv.drawString(10, 5, self.subtext)


class StatBox(Flowable):
    def __init__(self, label: str, value: str, color: Color,
                 width: float = 80, height: float = 50):
        super().__init__()
        self.label = label
        self.value = value
        self.color = color
        self.box_width = width
        self.box_height = height

    def wrap(self, availWidth, availHeight):
        return (self.box_width, self.box_height)

    def draw(self):
```

```
            canv = self.canv
            canv.setFillColor(self.color)
            canv.roundRect(0, 0, self.box_width, self.box_height, 6, fill=1, stroke=0)
            canv.setFillColor(white)
            canv.setFont(FONT_BOLD, 16)
            canv.drawCentredString(self.box_width / 2, self.box_height - 25,
                                   str(self.value))
            canv.setFont(FONT_NORMAL, 8)
            canv.setFillColor(HexColor('#ffffffcc'))
            canv.drawCentredString(self.box_width / 2, 8, self.label)


# ============================================================
# PDF 生成器
# ============================================================
class PDFGenerator:
    def __init__(self, repo: RepoInfo, output_dir: str):
        self.repo = repo
        self.output_dir = Path(output_dir)
        self.output_dir.mkdir(parents=True, exist_ok=True)
        self.page_width, self.page_height = A4
        self.margin = 15 * mm
        self.content_width = self.page_width - 2 * self.margin
        self.avail_height = self.page_height - self.margin - 15 * mm

    def generate_all(self):
        print(f"\n□ Project: {self.repo.name}")
        print(f" Files: {len(self.repo.files)}, Lines: {self.repo.total_lines:,}")
        print(f" Output: {self.output_dir}\n")
        self._generate_index_pdf()
        for f in self.repo.files:
            self._generate_file_pdf(f)
        print(f"\n□ Done! Generated {len(self.repo.files) + 1} PDFs")

    def _page_footer(self, canvas, doc):
        canvas.saveState()
        canvas.setFont(FONT_NORMAL, 7)
        canvas.setFillColor(HexColor('#999999'))
        canvas.drawString(self.margin, 10 * mm,
                          f"pixcode · {self.repo.name}")
        canvas.drawRightString(self.page_width - self.margin, 10 * mm,
                               f"Page {doc.page}")
        canvas.restoreState()

    def _make_doc(self, filename):
        return SimpleDocTemplate(
            str(filename), pagesize=A4,
            leftMargin=self.margin, rightMargin=self.margin,
            topMargin=self.margin, bottomMargin=15 * mm,
        )

    def _cjk_style(self, name, parent_name='Normal', **kwargs):
        styles = getSampleStyleSheet()
        parent = styles[parent_name]
        defaults = {'fontName': FONT_NORMAL,'fontSize': parent.fontSize}
        defaults.update(kwargs)
        return ParagraphStyle(name, parent=parent, **defaults)

    def _max_lines_for_height(self, avail_h, font_size=6.5):
        line_h = font_size * 1.6
        padding = 12
        return max(1, int((avail_h - padding) / line_h))

    # ── INDEX PDF ──────────────────────────────────
    def _generate_index_pdf(self):
        filename = self.output_dir / "00_INDEX.pdf"
        doc = self._make_doc(filename)
        story = []
        cw = self.content_width

```

```
        # 标题
        story.append(Spacer(1, 10 * mm))
        title_style = self._cjk_style(
            'CTitle', 'Title', fontSize=28,
            textColor=COLORS['accent'], fontName=FONT_BOLD,
            spaceAfter=4 * mm,
        )
        story.append(Paragraph(xml_escape(self.repo.name), title_style))

        sub_style = self._cjk_style(
            'CSub', fontSize=10,
            textColor=HexColor('#888888'), spaceAfter=8 * mm,
        )
        story.append(Paragraph(
            f"Code Repository Overview · Generated "
            f"{datetime.now().strftime('%Y-%m-%d %H:%M')}",
            sub_style))

        # 统计卡片
        bw = (cw - 20) / 4
        stat_data = [[
            StatBox("FILES", str(len(self.repo.files)),
                    COLORS['accent'], bw, 50),
            StatBox("LINES", f"{self.repo.total_lines:,}",
                    COLORS['accent2'], bw, 50),
            StatBox("SIZE", self._fmt_size(self.repo.total_size),
                    COLORS['green'], bw, 50),
            StatBox("LANGUAGES", str(len(self.repo.language_stats)),
                    COLORS['type'], bw, 50),
        ]]
        t = Table(stat_data, colWidths=[bw + 5] * 4)
        t.setStyle(TableStyle([
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ]))
        story.append(t)
        story.append(Spacer(1, 8 * mm))

        # 语言统计表
        story.append(HeaderBar("Language Statistics", width=cw))
        story.append(Spacer(1, 3 * mm))

        ns = self._cjk_style('CN', fontSize=8)
        lang_data = [[
            Paragraph('<b>Language</b>', ns),
            Paragraph('<b>Files</b>', ns),
            Paragraph('<b>Lines</b>', ns),
            Paragraph('<b>%</b>', ns),
        ]]
        for lang, stats in self.repo.language_stats.items():
            pct = (stats['lines'] / max(self.repo.total_lines, 1)) * 100
            lang_data.append([
                Paragraph(f'<font color="{COLORS["accent"].hexval()}">'
                          f'{xml_escape(lang)}</font>', ns),
                Paragraph(str(stats['files']), ns),
                Paragraph(f"{stats['lines']:,}", ns),
                Paragraph(f"{pct:.1f}%", ns),
            ])
        lt = Table(lang_data,
                   colWidths=[cw * 0.35, cw * 0.2, cw * 0.25, cw * 0.2])
        lt.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), COLORS['header_bg']),
            ('TEXTCOLOR', (0, 0), (-1, 0), white),
            ('FONTSIZE', (0, 0), (-1, -1), 8),
            ('BOTTOMPADDING', (0, 0), (-1, -1), 4),
            ('TOPPADDING', (0, 0), (-1, -1), 4),
            ('GRID', (0, 0), (-1, -1), 0.5, COLORS['border']),
            ('ROWBACKGROUNDS', (0, 1), (-1, -1),
             [white, HexColor('#f8f9fa')]),
            ('ALIGN', (1, 0), (-1, -1), 'RIGHT'),
```

```
            ]))
            story.append(lt)
            story.append(Spacer(1,        8 *  mm))

        # 目录树
            story.append(HeaderBar("Directory          Structure",      width=cw))
            story.append(Spacer(1,        3 *  mm))

            tree_lines      = self.repo.tree_str.split('\n')
            if   len(tree_lines)        > 120:
                tree_lines      = tree_lines[:120]        + [
                    f'     ...     ({len(tree_lines)}          entries     total)']

            self._add_code_chunks(story,          tree_lines,      'text',      cw,
                                  first_avail=self.avail_height              - 300,
                                  later_avail=self.avail_height              - 10)
            story.append(Spacer(1,        6 *  mm))

        # 文件索引表
            story.append(PageBreak())
            story.append(HeaderBar("File          Index",
                                   f"{len(self.repo.files)}              files",      width=cw))
            story.append(Spacer(1,        3 *  mm))

            fs   = self._cjk_style('FE',             fontSize=7,     fontName=FONT_NORMAL)
            fh   = [
                Paragraph('<b>#</b>',      fs),
                Paragraph('<b>File      Path</b>',    fs),
                Paragraph('<b>Lang</b>',      fs),
                Paragraph('<b>Lines</b>',       fs),
                Paragraph('<b>Size</b>',      fs),
                Paragraph('<b>PDF</b>',     fs),
            ]
            fdata    = [fh]
            for   f   in   self.repo.files:
                pdf_name   = self._file_pdf_name(f)
                fdata.append([
                    Paragraph(str(f.index),           fs),
                    Paragraph(
                        f'<font     color="{COLORS["accent"].hexval()}">'
                        f'{xml_escape(str(f.path))}</font>',                fs),
                    Paragraph(f.language,        fs),
                    Paragraph(f"{f.line_count:,}",              fs),
                    Paragraph(self._fmt_size(f.size),               fs),
                    Paragraph(
                        f'<font     color="{COLORS["accent2"].hexval()}">'
                        f'{xml_escape(pdf_name)}</font>',           fs),
                ])
            fcols    = [cw  *  0.06,    cw  *  0.38,    cw  *  0.12,
                        cw  *  0.12,    cw  *  0.12,    cw  *  0.20]
            ft   = Table(fdata,      colWidths=fcols,      repeatRows=1)
            ft.setStyle(TableStyle([
                ('BACKGROUND',  (0,    0),    (-1,    0),    COLORS['header_bg']),
                ('TEXTCOLOR',     (0,    0),    (-1,    0),    white),
                ('FONTSIZE',      (0,    0),    (-1,    -1),    7),
                ('BOTTOMPADDING',  (0,    0),    (-1,    -1),    3),
                ('TOPPADDING',    (0,    0),    (-1,    -1),    3),
                ('GRID',      (0,    0),    (-1,    -1),    0.3,    COLORS['border']),
                ('ROWBACKGROUNDS',(0,    1),    (-1,    -1),
                  [white,     HexColor('#f8f9fa')]),
                ('ALIGN',      (0,    0),    (0,    -1),    'CENTER'),
                ('ALIGN',      (3,    0),    (4,    -1),    'RIGHT'),
            ]))
            story.append(ft)

            doc.build(story,
                      onFirstPage=self._page_footer,
                      onLaterPages=self._page_footer)
            print(f"       □ 00_INDEX.pdf    ({len(self.repo.files)}              files     indexed)")

```

```
    # ──────────────────────── FILE PDF ────────────────────────
    def _generate_file_pdf(self, file_info: FileInfo):
        pdf_name = self._file_pdf_name(file_info)
        filename = self.output_dir / pdf_name
        doc = self._make_doc(filename)
        story = []
        cw = self.content_width

        # 头部
        story.append(HeaderBar(
            str(file_info.path),
            f"{file_info.language} · {file_info.line_count:,} lines · "
            f"{self._fmt_size(file_info.size)}",
            width=cw,
        ))
        story.append(Spacer(1, 4 * mm))

        # 元信息
        meta = self._cjk_style('Meta', fontSize=8,
                               textColor=HexColor('#666666'))
        for item in [
            f"<b>Path:</b> {xml_escape(str(file_info.path))}",
            f"<b>Language:</b> {file_info.language}",
            f"<b>Lines:</b> {file_info.line_count:,}",
            f"<b>Size:</b> {self._fmt_size(file_info.size)}",
        ]:
            story.append(Paragraph(item, meta))
        story.append(Spacer(1, 4 * mm))

        # 代码
        all_lines = file_info.content.split('\n')
        first_page_used = 28 + 4 * mm + 4 * 14 + 4 * mm + 10
        first_avail = self.avail_height - first_page_used
        later_avail = self.avail_height - 10

        self._add_code_chunks(story, all_lines, file_info.language, cw,
                              first_avail=first_avail,
                              later_avail=later_avail)

        doc.build(story,
                  onFirstPage=self._page_footer,
                  onLaterPages=self._page_footer)
        print(f"  □ {pdf_name} ({file_info.line_count} lines)")

    def _add_code_chunks(self, story, all_lines, language, width,
                         first_avail, later_avail, font_size=6.5):
        """将代码行拆分为安全大小的 chunk 加入 story"""
        offset = 0
        first_chunk = True
        while offset < len(all_lines):
            avail = first_avail if first_chunk else later_avail
            n = self._max_lines_for_height(avail, font_size)
            chunk = all_lines[offset:offset + n]

            story.append(CodeBlockChunk(
                chunk, language,
                start_line=offset + 1,
                width=width, font_size=font_size,
            ))

            offset += n
            first_chunk = False
            if offset < len(all_lines):
                story.append(Spacer(1, 1))

    def _file_pdf_name(self, f: FileInfo) -> str:
        safe_path = str(f.path).replace('/', '_').replace('\\', '_')
        safe_path = re.sub(r'[^\w\-_.]', '_', safe_path)
        return f"{f.index:03d}_{safe_path}.pdf"
```

```python
    @staticmethod
    def _fmt_size(size: int) -> str:
        if size < 1024:
            return f"{size} B"
        elif size < 1024 * 1024:
            return f"{size / 1024:.1f} KB"
        else:
            return f"{size / 1024 / 1024:.1f} MB"


# ============================================================
# CLI
# ============================================================
def main():
    parser = argparse.ArgumentParser(
        prog='pixcode',
        description='Convert code repository to structured PDFs '
          'for LLM collaboration',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  pixcode .                          # Current directory
  pixcode /path/to/repo -o ./pdfs    # Specify output
  pixcode . --max-size 1024          # Max file size 1MB
  pixcode . --ignore "*.test.js"     # Extra ignore patterns
    """,
    )
    parser.add_argument('repo', nargs='?', default='.',
                        help='Path to code repository (default: .)')
    parser.add_argument('-o', '--output', default=None,
                        help='Output directory')
    parser.add_argument('--max-size', type=int, default=512,
                        help='Max file size in KB (default: 512)')
    parser.add_argument('--ignore', nargs='*', default=[],
                        help='Extra file patterns to ignore')
    parser.add_argument('--list-only', action='store_true',
                        help="Only list files, don't generate PDFs")

    args = parser.parse_args()
    repo_path = Path(args.repo).resolve()
    if not repo_path.is_dir():
        print(f"❌ Error: '{args.repo}' is not a directory")
        sys.exit(1)

    print(f"🔍 Scanning {repo_path}...")
    scanner = RepoScanner(str(repo_path),
                          max_file_size=args.max_size * 1024,
                          extra_ignore=args.ignore)
    repo = scanner.scan()

    if not repo.files:
        print("⚠️ No files found!")
        sys.exit(0)

    if args.list_only:
        print(f"\n📦 {repo.name} ({len(repo.files)} files)\n")
        print(repo.tree_str)
        print(f"\n{'Language':<15} {'Files':>6} {'Lines':>8}")
        print('─' * 32)
        for lang, stats in repo.language_stats.items():
            print(f"{lang:<15} {stats['files']:>6} {stats['lines']:>8}")
        print('─' * 32)
        print(f"{'Total':<15} {len(repo.files):>6} {repo.total_lines:>8}")
        return

    output_dir = args.output or f"./pixcode_output/{repo.name}"
    generator = PDFGenerator(repo, output_dir)
    generator.generate_all()


```

```
971 if __name__ == '__main__':
972     main()
```