

# 源代码文档

<b>软件名称</b>	左右道飞MCP服务系统
<b>英文名称</b>	Daofy MCP Server
<b>版本号</b>	V2026.06.08.1
<b>著作权人</b>	吉林省左右软件开发有限公司
<b>开发语言</b>	Python 3.10+
<b>代码规模</b>	约 3300 行 (60页)

文件: server.py (原始行号 1-55)

```

1  """
2  Daofy 主程序
3
4  版权所有 (C) 吉林省左右软件开发有限公司
5  Copyright (C) Equilibrium Software Development Co., Ltd, Jilin
6  Update & Mod By Crystalxp (黑夜杀手 QQ:281309196)
7
8  提供 MCP 协议服务,注册所有工具并启动服务器
9  """
10
11 import asyncio
12 import sys
13 import os
14 import time
15 import winreg
16 from pathlib import Path
17 from typing import Any, Optional
18 import io
19
20 os.environ['PYTHONIOENCODING'] = 'utf-8'
21 os.environ['PYTHONUTF8'] = '1'
22
23 # 保护: 子进程(multiprocessing spawn) 的 stdout 已经 pipe,
24 # TextIOWrapper 可能失败。失败时跳过不影响子进程通信。
25 if __name__ != '__mp_main__':
26     try:
27         sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8', errors='replace', line_buffering=False)
28         sys.stderr = io.TextIOWrapper(sys.stderr.buffer, encoding='utf-8', errors='replace', line_buffering=False)
29     except Exception:
30         import logging as _logging
31         _logger = _logging.getLogger(__name__)
32         _logger.warning("stdout/stderr 编码设置失败, 部分输出可能乱码", exc_info=True)
33
34 # 添加项目根目录到 Python 路径
35 project_root = Path(__file__).parent.parent
36 if str(project_root) not in sys.path:
37     sys.path.insert(0, str(project_root))
38
39 # =====
40 # multiprocessing 子进程保护
41 # Windows spawn模式下,子进程会重新导入 __main__ 模块(即本文件),
42 # 导致所有服务模块被重新导入(885个模块),启动极慢。
43 # 检测到是子进程时,跳过所有服务导入,只保留必要的模块。
44 # =====
45 _is_multiprocessing_child = __name__ == '__mp_main__'
46
47 if _is_multiprocessing_child:
48     # 子进程不需要任何MCP服务,直接跳过
49     # ProcessPoolExecutor的worker只需要能pickle/unpickle函数即可
50     pass
51 else:
52
53     from mcp.server import Server
54     from mcp.server.stdio import stdio_server
55     from mcp.types import CallToolResult, TextContent, Tool, Resource, ReadResourceResult, TextResourceContents

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 56-110)

```

1
2     from src.services.config_manager import ConfigManager
3     from src.services.compiler_service import CompilerService
4     from src.services.knowledge_base import DelphiKnowledgeBaseService
5     from src.services.knowledge_base.thirdparty_knowledge_base import ThirdPartyKnowledgeBase
6     from src.tools.project import handle_project as _handle_project
7     # project 模块统一管理编译器服务, 保留别名供初始化用
8     from src.tools.compile_project import set_compiler_service as sp1
9     from src.tools.compile_file import set_compiler_service as sp2
10    from src.tools.get_args import set_compiler_service as sp3
11    from src.tools.config import set_config_manager, search_compilers
12    from src.tools.environment import check_environment, set_config_manager as scm, set_thirdparty_kb_service as stks
13    from src.tools.knowledge_base import (
14        set_delphi_kb_service,
15        set_thirdparty_kb_service,
16        _resolve_project_path,
17    )
18    from src.tools.read_source_file import set_knowledge_base_services, read_source_file
19    from src.tools import knowledge_base as kb_tools
20    from src.tools import thirdparty_knowledge_base as thirdparty_kb_tools
21    from src.tools import async_tasks as async_tools
22    from src.tools import pasfmt
23    from src.tools.install_package import handle_package, set_compiler_service as sip
24    from src.tools import document_kb_tools as doc_tools
25    from src.tools.code_hosting import code_hosting
26    from src.tools import file_tool
27    from src.tools import dfm_utils as dfm_utils_mod
28    from src.tools import manage_component as manage_component_mod
29    from src.tools import create_component_dfm as create_component_dfm_mod
30    from src.tools.coding_rules import get_coding_rules as _get_coding_rules
31    from src.tools.tool_help import get_tool_help
32    from src.tools.experience import experience as _experience
33    from src.config.tool_docs import TOOL_NAMES, TOOL_SHORT_DESC
34    from src.utils.logger import init_default_logger, log_api_call
35    from src.__version__ import __version__, __copyright__
36    from src.utils import updater
37
38    # 后台版本检查结果缓存 (由 startup 异步任务填充)
39    _update_check_result: Optional[dict] = None
40    _update_check_done: bool = False
41
42    # 文件变更监听器 (由 startup 异步任务启动)
43    _project_file_watcher: Optional[object] = None
44
45    # 服务器启动时间 (用于 /health 资源)
46    _server_start_time: float = 0.0
47    # 最近一次 KB 构建时间
48    _last_kb_build_time: Optional[float] = None
49
50    # 初始化日志
51    logger = init_default_logger()
52
53
54    def _auto_detect_delphi_help_dir() -> Optional[str]:
55        """自动检测最新安装的 Delphi 帮助文档目录"""

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 111-165)

```

1   try:
2       key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"SOFTWARE\Embarcadero\BDS")
3       versions = []
4       i = 0
5       while True:
6           try:
7               versions.append(winreg.EnumKey(key, i))
8               i += 1
9           except OSError:
10              break
11          winreg.CloseKey(key)
12
13          versions.sort(key=lambda x: float(x) if x.replace('.', '').isdigit() else 0, reverse=True)
14          for ver in versions:
15              try:
16                  vk = winreg.OpenKey(winreg.HKEY_CURRENT_USER, rf"SOFTWARE\Embarcadero\BDS\{ver}")
17                  root_dir = winreg.QueryValueEx(vk, "RootDir")[0]
18                  winreg.CloseKey(vk)
19                  help_dir = Path(root_dir) / "Help" / "Doc"
20                  if help_dir.exists():
21                      logger.info(f"自动检测到 Delphi 帮助目录 (版本 {ver}): {help_dir}")
22                      return str(help_dir)
23              except Exception:
24                  logger.debug("读取注册表版本键失败", exc_info=True)
25                  continue
26          except Exception:
27              logger.debug("打开注册表 BDS 键失败", exc_info=True)
28
29          # 注册表失败, 尝试默认路径 (版本号与注册表一致: 37.0=Delphi13, 23.0=Delphi12, 22.0=Delphi11...)
30          # 只有 17.0 (XE) 及以上版本使用此目录结构
31          for ver in ["37.0", "23.0", "22.0", "21.0", "20.0", "19.0", "18.0", "17.0"]:
32              path = rf"C:\Program Files (x86)\Embarcadero\Studio\{ver}\Help\Doc"
33              if Path(path).exists():
34                  logger.info(f"使用默认帮助目录: {path}")
35                  return path
36          return None
37
38
39 def _get_smart_hint(name: str, result: Any, arguments: dict) -> Optional[str]:
40     """
41     智能提示: 根据工具名和返回结果, 生成下一步建议。
42
43     Args:
44         name: 工具名
45         result: 工具返回结果 (dict 或 CallToolResult)
46         arguments: 调用参数
47
48     Returns:
49         建议文本, 无建议时返回 None
50     """
51     if name == "delphi_kb":
52         action = arguments.get("action", "search")
53         if action == "search":
54             if isinstance(result, dict):
55                 results = result.get('results') or result.get('data') or []

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 166-220)

```

1         if isinstance(results, list) and len(results) > 0:
2             return ("hint: use "
3                 'delphi_file(action="read", file_path="...") to read full source')
4         elif action == "stats":
5             return ("hint: if KB data is stale, "
6                 "use delphi_kb(action='build', kb_type='project') to rebuild")
7
8         elif name == "get_coding_rules":
9             # 仅在 section=None (默认模式) 时提示
10            section = arguments.get("section")
11            if section is None or section == "":
12                return ("hint: use section param for specific chapters:\n"
13                    '  get_coding_rules(section="writing") - before writing\n'
14                    '  get_coding_rules(section="review") - after compile, before review\n'
15                    '  get_coding_rules(section="safety") - security-sensitive ops')
16
17            elif name == "check_environment":
18                action = arguments.get("action", "check")
19                if action == "detect" or action == "check":
20                    if isinstance(result, dict):
21                        compilers = result.get('compilers') or result.get('data')
22                        if compilers and len(compilers) > 0:
23                            return ("hint: environment ready, "
24                                "use project(action='compile') to verify")
25                        else:
26                            return ("hint: no compiler detected, "
27                                "check Delphi installation, "
28                                "or use check_environment(action='detect', search_path=...)")
29
30            elif name == "package":
31                action = arguments.get("action", "")
32                if action == "install":
33                    if isinstance(result, CallToolResult):
34                        is_error = result.isError
35                    elif isinstance(result, dict):
36                        is_error = (
37                            result.get('status') == 'failed'
38                            or result.get('success') is False
39                            or (result.get('error') is not None and result.get('error') != ''))
40                    )
41                else:
42                    is_error = False
43                if not is_error:
44                    return ("hint: install done, "
45                        "use package(action='list') to verify IDE registration")
46
47            # P4: 版本更新提示 (检查完成且有新版本时通知 AI)
48            if _update_check_done and _update_check_result and _update_check_result.get("update_available"):
49                return (
50                    f"📦 发现新版本 Daofy: v{ _update_check_result['current'] } → "
51                    f"v{ _update_check_result['latest'] }!\n"
52                    f"请使用 `daofy_update(action=\"check\")` 查看详情, "
53                    f"或 `daofy_update(action=\"update\")` 通过 git pull 更新.\n"
54                    f"发布说明: { _update_check_result['release_url']}"
55                )

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 221-275)

```
1
2     return None
3
4
5 async def run_server():
6     """运行 MCP Server"""
7     global _server_start_time
8     _server_start_time = time.time()
9     logger.info(f"启动 Daofy v{__version__}")
10    logger.info(f"__copyright__")
11
12    # 初始化配置管理器
13    config_manager = ConfigManager()
14    logger.info("配置管理器初始化完成")
15
16    # 初始化编译服务
17    compiler_service = CompilerService(config_manager)
18    logger.info("编译服务初始化完成")
19
20    # 初始化知识库服务
21    kb_service = DelphiKnowledgeBaseService()
22    logger.info("知识库服务初始化完成")
23
24    # 初始化第三方库知识库服务
25    thirdparty_kb_service = ThirdPartyKnowledgeBase()
26    thirdparty_kb_tools.set_thirdparty_knowledge_base_service(thirdparty_kb_service)
27    logger.info("第三方库知识库服务初始化完成")
28
29    # 设置工具的服务实例
30    sp1(compiler_service)
31    sp2(compiler_service)
32    sp3(compiler_service)
33    sip(compiler_service)
34    scm(config_manager)
35    set_config_manager(config_manager)
36    stks(thirdparty_kb_service)
37    set_knowledge_base_services(kb_service, thirdparty_kb_service)
38    set_delphi_kb_service(kb_service)
39    # 项目 KB 服务由 project_path 参数动态创建,不在启动时初始化
40    # set_project_kb_service(kb_service) # kb_service 是 Delphi RTL KB,不适合作为项目 KB
41    set_thirdparty_kb_service(thirdparty_kb_service)
42
43    # 设置 DFM 工具编译器路径
44    newest = config_manager.get_newest_compiler()
45    if newest and newest.path:
46        if os.path.isfile(newest.path):
47            dfm_utils_mod.set_compiler_path(newest.path)
48            create_component_dfm_mod.set_compiler_path(newest.path)
49            logger.info(f"DFM 工具编译器路径已设置: {newest.path}")
50        else:
51            logger.warning("编译器文件不存在: %s, DFM 转换功能将不可用。请重新检测编译器。", newest.path)
52    else:
53        logger.warning("未找到可用编译器, DFM 转换功能将不可用")
54
55    # 设置事件签名解析器的 KB 服务引用
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 276-330)

```

1   from src.tools.dfm_parser import set_kb_services as _set_dfm_kb
2   _set_dfm_kb(delphi_kb=kb_service, thirdparty_kb=thirdparty_kb_service)
3
4   logger.info("工具服务实例设置完成")
5
6   # 创建 MCP Server 实例
7   server = Server("daofy-for-delphi")
8   logger.info("MCP Server 实例创建完成")
9
10  # =====
11  # MCP 工具注册
12  # 所有工具必须同时在 list_tools() 和 call_tool() 中注册
13  # =====
14  @server.list_tools()
15  async def list_tools():
16      """列出所有可用工具"""
17      return [
18          # ===== 项目全生命周期管理 ☆☆☆ =====
19          Tool(
20              name="project",
21              description=TOOL_SHORT_DESC["project"],
22              inputSchema={
23                  "type": "object",
24                  "properties": {
25                      "action": {
26                          "type": "string",
27                          "enum": ["compile", "compile_file", "dry_run", "info", "create",
28                                  "set", "add_config", "remove_config", "add_source",
29                                  "remove_source", "audit", "ast", "runtime"],
30                          "description": "操作类型。先 tool_help('project') 查看各 action 的参数说明。"
31                      },
32                      "project_path": {"type": "string", "description": "项目文件路径(.dproj/.dpr/.dpk/.pas)"},
33                      "dry_run": {"type": "boolean", "default": False, "description": "仅预览编译参数不实际执行"},
34                  },
35                  "additionalProperties": True,
36                  "required": ["action"]
37              }
38          ),
39
40          # ===== 知识库搜索/管理 ☆☆☆ =====
41          Tool(
42              name="delphi_kb",
43              description=TOOL_SHORT_DESC["delphi_kb"],
44              inputSchema={
45                  "type": "object",
46                  "properties": {
47                      "action": {"type": "string", "enum": ["search", "stats", "build", "scan", "web", "read", "bu
48                                  "query": {"type": "string", "description": "搜索关键词 (action=search时需要)"},
49                      "kb_type": {"type": "string", "enum": ["all", "delphi", "project", "thirdparty", "document"],
50                      "search_type": {"type": "string", "enum": ["function", "procedure", "class", "record", "inter
51                      "top_k": {"type": "integer", "default": 200, "description": "最大返回结果数 (默认200, 最大500)"}
52                      "project_path": {"type": "string", "description": "项目路径 (搜索project/thirdparty知识库时需要)"}
53                      "version": {"type": "string", "description": "Delphi版本 (构建知识库时使用)"},
54                      "async_mode": {"type": "boolean", "default": True, "description": "是否异步执行 (build操作时生效)"}
55                      "rebuild": {"type": "boolean", "default": False, "description": "是否强制重建 (build操作时生效)"}

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 331-385)

```

1         "incremental": {"type": "boolean", "default": False, "description": "是否增量更新 (build操作时
2         "build_thirdparty": {"type": "boolean", "default": True, "description": "构建项目KB时是否同时构
3         "build_project": {"type": "boolean", "default": True, "description": "是否构建项目KB"},
4         "directory": {"type": "string", "description": "扫描目录 (action=scan时使用, 或build document时
5         "extensions": {"type": "array", "items": {"type": "string"}, "description": "文件扩展名过滤 (a
6         "content_type": {"type": "string", "description": "文档类型过滤 (action=search kb_type=documen
7         "url": {"type": "string", "description": "网页URL (action=web时使用) 或文档URL (action=read时使用
8         "doc_id": {"type": "string", "description": "文档ID (action=read时使用, 与url二选一)"},
9         "file_path": {"type": "string", "description": "文件路径 (action=read时使用)"},
10        "offset": {"type": "integer", "default": 0, "description": "读取偏移量 (action=read时使用)"},
11        "limit": {"type": "integer", "default": 5000, "description": "读取字节数限制 (action=read时使用)
12        "max_pages": {"type": "integer", "default": 100, "description": "最大抓取页数 (build document
13        "max_depth": {"type": "integer", "default": 3, "description": "最大抓取深度 (build document KB
14        "domain_filter": {"type": "string", "description": "域名过滤 (build document KB时使用)"},
15        "url_pattern": {"type": "string", "description": "URL模式过滤 (build document KB时使用)"},
16        "exclude": {"type": "array", "items": {"type": "string"}, "description": "排除目录列表 (build
17        "max_workers": {"type": "integer", "description": "最大工作进程数 (action=scan时使用)"},
18        "show_progress": {"type": "boolean", "default": True, "description": "是否显示进度"},
19    }
20    }
21),
22
23    # ===== Delphi 文件专用操作 - 读/写/格式化/备份管理 ★★★★★ =====
24    Tool(
25        name="delphi_file",
26        description=TOOL_SHORT_DESC["delphi_file"],
27        inputSchema={
28            "type": "object",
29            "required": ["action"],
30            "properties": {
31                # ---- 全局参数 (所有 action 都可用) ----
32                "action": {"type": "string", "enum": ["read", "write", "batch_write", "format", "backup", "us
33                "file_path": {"type": "string", "description": "目标文件路径, 支持 .pas/.dfm/.dproj/.dpk/.fmx/
34
35                # ---- [仅 action=read] 参数 ----
36                "search_type": {"type": "string", "enum": ["path", "class", "function", "record"], "descripti
37                "type_name": {"type": "string", "description": "[仅 action=read, search_type=class] 类名/接口
38                "class_name": {"type": "string", "description": "[仅 action=read, search_type=class] 类名 (与
39                "record_name": {"type": "string", "description": "[仅 action=read, search_type=record] Recorc
40                "function_name": {"type": "string", "description": "[仅 action=read, search_type=function] 函
41                "start_line": {"type": "integer", "default": 0, "description": "起始行号 (从0开始, 左闭右开区间
42                "limit": {"type": "integer", "default": 500, "description": "[仅 action=read] 最大返回行数 (默
43                "show_line_numbers": {"type": "boolean", "default": False, "description": "[仅 action=read] }
44                "end_line": {"type": "integer", "description": "结束行号 (不包含该行, 左闭右开区间), 不传则到文件
45                "search_in": {"type": "string", "enum": ["all", "delphi", "thirdparty"], "default": "all", "c
46                "project_path": {"type": "string", "description": "[仅 action=read, search_type=class/function
47
48                # ---- [仅 action=write/batch_write] 参数 ----
49                "content": {"type": "string", "description": "[action=write 必需] 写入的内容。不传 start_line/
50                "encoding": {"type": "string", "default": "auto", "description": "[write/batch_write/uses] 写
51                "auto_format": {"type": "boolean", "default": False, "description": "[write/batch_write/uses]
52                "backup": {"type": "boolean", "default": True, "description": "[write/batch_write/uses] 写入前
53                "preview": {"type": "boolean", "default": False, "description": "[write/batch_write] 预览模式
54
55                # ---- [仅 action=batch_write] 参数 ----

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 386-440)

```

1      # 推荐使用 batch_write 进行所有部分写入（替代多次 write 调用）。
2      # edits 以原始文件为参照系，内部自动处理行号偏移，无需 AI 手动计算。
3      "edits": {
4          "type": "array",
5          "description": "【action=batch_write 必需】编辑列表，传入顺序不限。以备份文件为参照系，内部自
6          "items": {
7              "type": "object",
8              "required": ["start_line", "content"],
9              "properties": {
10                 "start_line": {"type": "integer", "description": "起始行号（0-indexed inclusive）"},
11                 "end_line": {"type": "integer", "description": "结束行号（0-indexed exclusive）"},
12                 "content": {"type": "string", "description": "替换内容（完整替代 [start_line, end_
13                 "description": {"type": "string", "description": "可选的文字描述，仅用于返回消息标记
14             }
15         }
16     },
17     "force": {"type": "boolean", "default": False, "description": "[仅 action=batch_write] 强制写
18
19     # ---- [仅 action=format] 参数 ----
20     "mode": {"type": "string", "enum": ["file", "code", "check"], "default": "file", "descriptio
21     "code": {"type": "string", "description": "[仅 action=format, mode=code] 待格式化的代码文本"},
22     "config_path": {"type": "string", "description": "[仅 action=format] pasfmt 配置文件路径（可选
23     "uses_style": {"type": "string", "enum": ["compact", "pasfmt_default"], "description": "[仅 a
24     "dry_run": {"type": "boolean", "default": False, "description": "[仅 action=format] true=仅检
25
26     # ---- [仅 action=backup] 参数 ----
27     "backup_action": {"type": "string", "enum": ["create", "list", "restore"], "default": "create
28     "version": {"type": "integer", "description": "[仅 action=backup, backup_action=restore] 要恢
29
30     # ---- [仅 action=uses] 参数 ----
31     "uses_action": {"type": "string", "enum": ["add", "remove"], "description": "[仅 action=uses]
32     "unit_name": {"type": "string", "description": "[仅 action=uses] 单元名，如 Vcl.Dialogs、Syste
33     "uses_section": {"type": "string", "enum": ["interface", "implementation"], "default": "inter
34     }
35     }
36     ),
37
38     # ===== 组件管理 ★★ =====
39     Tool(
40         name="manage_component",
41         description=TOOL_SHORT_DESC["manage_component"],
42         inputSchema={
43             "type": "object",
44             "properties": {
45                 "action": {"type": "string", "enum": ["create", "add", "remove", "modify"], "default": "creat
46                 "description": "操作类型：create=生成DFM, add=添加组件, remove=删除组件, modify=修改
47                 "target_dfm": {"type": "string", "description": "目标 DFM 文件路径（add/remove/modify 时必需）"},
48                 "target_pas": {"type": "string", "description": "目标 PAS 文件路径（add/remove/modify 时可选，）
49                 "component_name": {"type": "string", "description": "组件名称（remove/modify 时必需，指定操作的
50                 "parent_name": {"type": "string", "description": "父组件名称（add 时可选，默认添加到根组件下）"},
51                 "new_component_class": {"type": "string", "description": "新组件类名（add 时必需，如 TButton）"},
52                 "new_component_name": {"type": "string", "description": "新组件实例名（add 时可选，默认自动生成！
53                 "properties": {"type": "object", "additionalProperties": {"type": "string"},
54                 "description": "组件属性字典（add/modify 时使用，如 {"Caption": "OK", "OnClick
55                 "dfm_text": {"type": "string", "description": "待添加的 DFM 文本片段（add 时可选，替代 new_comp

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 441-495)

```

1         "code": {"type": "string", "description": "[create 必需] Pascal 实现代码, 必须包含 function Cre
2         "uses": {"type": "array", "items": {"type": "string"}, "description": "[create] 需引用的单元列
3         "type_decl": {"type": "string", "description": "[create] 类型声明段 (可选), 用于声明 Form 类、
4         "init_code": {"type": "string", "description": "[create] 初始化代码 (可选), 在 CreateComponent
5         "compile_timeout": {"type": "integer", "default": 60, "description": "编译超时秒数"},
6         "exec_timeout": {"type": "integer", "default": 15, "description": "执行超时秒数 (组件创建代码可
7         },
8         "required": ["action"]
9     }
10 ),
11
12 # ===== 环境检查 ★★★ =====
13 Tool(
14     name="check_environment",
15     description=TOOL_SHORT_DESC["check_environment"],
16     inputSchema={
17         "type": "object",
18         "properties": {
19             "action": {"type": "string", "enum": ["check", "detect", "install", "format_install"], "defau
20             "search_path": {"type": "string", "description": "额外搜索路径 (action=detect时使用)"},
21             "install_dir": {"type": "string", "description": "安装目录 (action=install/format_install时使
22             "delphi_version": {"type": "string", "default": "11", "description": "Delphi版本 (action=form
23         }
24     }
25 ),
26
27 # ===== 异步任务管理 ★ =====
28 Tool(
29     name="async_task",
30     description=TOOL_SHORT_DESC["async_task"],
31     inputSchema={
32         "type": "object",
33         "properties": {
34             "action": {"type": "string", "enum": ["start", "status", "result", "list", "cancel"], "descri
35             "task_id": {"type": "string", "description": "任务ID (action=status/result/cancel时使用)"},
36             "long_poll_seconds": {"type": "integer", "default": 0, "minimum": 0, "maximum": 30, "descript
37             "task_type": {"type": "string", "description": "任务类型 (action=start时使用), 如: build_know.
38             "task_params": {"type": "object", "description": "任务参数 (action=start时使用, 根据task_type不
39             "show_progress": {"type": "boolean", "default": True, "description": "是否显示进度"},
40         }
41     }
42 ),
43
44 # ===== 组件包管理 ★★ =====
45 Tool(
46     name="package",
47     description=TOOL_SHORT_DESC["package"],
48     inputSchema={
49         "type": "object",
50         "properties": {
51             "action": {"type": "string", "enum": ["install", "list"], "default": "install", "description"
52             "package_path": {"type": "string", "description": "[install] 包文件路径(.dproj/.dpk/.grouppro
53             "target_platform": {"type": "string", "enum": ["win32", "win64"], "default": "win32", "descri
54             "build_configuration": {"type": "string", "default": "Debug", "description": "[install] 构建
55             "timeout": {"type": "integer", "default": 300, "description": "[install] 超时时间(秒)"},

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 496-550)

```

1         "install": {"type": "boolean", "default": True, "description": "[install] 是否自动安装到 IDE"}
2     },
3     "required": ["action"]
4 }
5 ),
6
7 # ===== 编码规则 (AI 必读) ★★★★★ =====
8 Tool(
9     name="get_coding_rules",
10    description=TOOL_SHORT_DESC["get_coding_rules"],
11    inputSchema={
12        "type": "object",
13        "properties": {
14            "project_path": {"type": "string", "description": "项目路径 (可选), 用于查找项目自定义的编码规则"},
15            "section": {"type": "string", "description": "章节名称 (可选), 如 workflow/writing/review/saf"},
16        }
17    }
18 ),
19
20 # ===== 代码托管平台统一工具 =====
21 Tool(
22     name="code_hosting",
23     description=TOOL_SHORT_DESC["code_hosting"],
24     inputSchema={
25         "type": "object",
26         "properties": {
27             "platform": {"type": "string", "enum": ["gitea", "github", "gitlab", "gitee", "gitcode"], "description": "平台名称"},
28             "action": {"type": "string", "enum": ["create_token", "init_labels", "create_issue", "close_issue", "add_comment"], "description": "操作类型"},
29             "base_url": {"type": "string", "description": "平台实例地址, 如 https://code.qdac.cc:3000 (API 操作需要)"},
30             "token": {"type": "string", "description": "API 访问令牌 (API 操作需要)"},
31             "repo": {"type": "string", "description": "仓库名, 格式 owner/repo (API 操作需要)"},
32             "issue_number": {"type": "integer", "description": "工单编号 (close_issue/add_comment 需要)"},
33             "title": {"type": "string", "description": "工单标题 (create_issue 需要)"},
34             "body": {"type": "string", "description": "正文内容, 支持 Markdown (create_issue/add_comment 需要)"},
35             "labels": {"type": "array", "items": {"type": "string"}, "description": "标签名称列表 (create_issue 需要)"},
36             "comment": {"type": "string", "description": "关闭工单时的说明 (close_issue 可选)"},
37             "state": {"type": "string", "enum": ["open", "closed", "all"], "description": "工单过滤状态 (close_issue 可选)"},
38             "username": {"type": "string", "description": "用户名 (create_token 需要)"},
39             "password": {"type": "string", "description": "密码 (create_token 需要)"},
40             "token_name": {"type": "string", "description": "Token 名称 (create_token 可选, 默认 delphi-mc"},
41             # Git 操作参数
42             "dir": {"type": "string", "description": "Git 仓库本地路径 (git_* 操作需要)"},
43             "repo_url": {"type": "string", "description": "远程仓库 URL (git_clone 需要)"},
44             "mirror": {"type": "string", "description": "GitHub 镜像源地址, 如 https://hub.fastgit.xyz (git_clone 需要)"},
45             "branch": {"type": "string", "description": "分支名 (git_clone/git_push 可选)"},
46             "message": {"type": "string", "description": "提交信息 (git_commit 需要)"},
47             "files": {"type": "array", "items": {"type": "string"}, "description": "要 add 的文件列表 (git_add 需要)"},
48             "remote": {"type": "string", "description": "远程名称 (git_push/git_push_retry 可选, 默认 origin)"},
49             "retry_interval": {"type": "integer", "description": "重试间隔秒数 (git_push_retry 可选, 默认 10)"},
50             "task_id": {"type": "string", "description": "异步任务ID (配合 async_task 工具查询)"},
51         }
52     },
53     "required": ["action"]
54 }
55 ),

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 551-605)

```

1      # ===== 工具帮助（按需获取详细文档）=====
2      Tool(
3          name="tool_help",
4          description=TOOL_SHORT_DESC["tool_help"],
5          inputSchema={
6              "type": "object",
7              "properties": {
8                  "tool_name": {
9                      "type": "string",
10                     "enum": TOOL_NAMES,
11                     "description": "工具名",
12                 },
13             },
14             "required": ["tool_name"],
15         }
16     ),
17
18     # ===== Daofy 自身更新管理 =====
19     Tool(
20         name="daofy_update",
21         description="检查 Daofy 版本更新、执行 git pull 更新。发现新版本时智能提示中会自动通知。",
22         inputSchema={
23             "type": "object",
24             "properties": {
25                 "action": {
26                     "type": "string",
27                     "enum": ["check", "update", "version"],
28                     "default": "check",
29                     "description": "check=检查新版, update=执行 git pull 更新, version=显示当前版本",
30                 },
31             },
32             "required": ["action"],
33         }
34     ),
35
36     # ===== 经验记忆管理 =====
37     Tool(
38         name="experience",
39         description=TOOL_SHORT_DESC["experience"],
40         inputSchema={
41             "type": "object",
42             "properties": {
43                 "action": {
44                     "type": "string",
45                     "enum": ["save", "search", "get", "list", "update", "merge", "prune", "delete", "rebuild_
46                     "description": "操作类型: save=保存经验(自动去重), search=语义搜索, get=查看详情, list=浏览列
47                 },
48                 "problem": {"type": "string", "description": "[save] 问题描述"},
49                 "solution": {"type": "string", "description": "[save] 解决步骤"},
50                 "tools_used": {"type": "array", "items": {"type": "string"}, "description": "[save] 用到的工具"},
51                 "tags": {"type": "array", "items": {"type": "string"}, "description": "[save/search/list] 标
52                 "context": {"type": "object", "description": "[save] 上下文信息"},
53                 "query": {"type": "string", "description": "[search] 搜索关键词"},
54                 "top_k": {"type": "integer", "default": 5, "description": "[search] 返回条数"},
55                 "id": {"type": "string", "description": "[get/update/delete] 经验ID"},

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 606-660)

```

1         "ids": {"type": "array", "items": {"type": "string"}, "description": "[merge] 待合并的经验ID列表"},
2         "keep": {"type": "string", "description": "[merge] 保留的目标ID (可选, 不传则创建新记录)"},
3         "sort_by": {"type": "string", "default": "updated_at", "enum": ["updated_at", "created_at", "id"], "description": "[merge] 排序方式"},
4         "limit": {"type": "integer", "default": 20, "description": "[list/prune] 返回条数"},
5         "force": {"type": "boolean", "default": False, "description": "[save] 发现高相似度经验时仍强制更新"},
6     },
7     "required": ["action"],
8 }
9 ),
10
11 # ===== 软著文档生成 =====
12 Tool(
13     name="generate_copyright",
14     description=TOOL_SHORT_DESC.get("generate_copyright", "生成软著文档"),
15     inputSchema={
16         "type": "object",
17         "properties": {
18             "action": {
19                 "type": "string",
20                 "enum": ["generate", "validate", "update_config", "status", "list", "generate_content", "prune"],
21                 "default": "generate",
22                 "description": "操作类型: generate=生成文档; validate=检查配置; update_config=更新配置; status=获取状态; list=获取列表; generate_content=生成内容; prune=删除经验",
23             },
24             "config": {
25                 "type": "object",
26                 "description": "配置更新 (仅 action=update_config 时必需)",
27             },
28             "doc_type": {
29                 "type": "string",
30                 "enum": ["all", "source", "manual", "summary"],
31                 "default": "all",
32                 "description": "文档类型 (仅 action=generate 时生效)",
33             },
34             "output_dir": {
35                 "type": "string",
36                 "description": "输出目录 (可选, 默认 docs/copyright)",
37             },
38         },
39         "required": ["action"],
40     }
41 ),
42
43 # ===== Delphi 自动化截图 =====
44 Tool(
45     name="automate_delphi",
46     description=TOOL_SHORT_DESC.get("automate_delphi", "Delphi 自动化测试"),
47     inputSchema={
48         "type": "object",
49         "properties": {
50             "app_path": {
51                 "type": "string",
52                 "description": "Delphi exe 文件路径",
53             },
54             "script": {
55                 "type": "string",

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 661-715)

```

1         "description": "JSON 脚本（文件路径 或 JSON 字符串）。"
2             " 格式: [{\"cmd\": \"goto\", \"target\": \"TMainForm\", \"capture\": \"main_001
3             " 协议: JSON请求/响应, cmd字段支持: goto/click/rclick/dblclick/hover/move/dr
4         },
5         "snapshots_dir": {
6             "type": "string",
7             "description": "截图输出目录（可选, 默认 docs/copyright/snapshots）",
8         },
9         "wait_timeout": {
10            "type": "number",
11            "default": 10,
12            "description": "等待 Delphi 管道就绪的超时秒数（默认 10s）",
13        },
14        "keep_alive": {
15            "type": "boolean",
16            "default": False,
17            "description": "执行完后是否保持进程运行。True=常驻供后续复用, False=执行完退出（默认）",
18        },
19        },
20        "required": ["app_path", "script"],
21    }
22    ),
23    ]
24
25    # =====
26    # 参数类型校验 – MCP 客户端可能传错类型（如 string 代替 bool）
27    # =====
28
29    def _coerce_bool(val, default: bool = False) -> bool:
30        """将任意输入安全转换为 bool。"""
31        if isinstance(val, bool):
32            return val
33        if isinstance(val, str):
34            return val.lower() in ('1', 'true', 'yes', 'on')
35        if isinstance(val, (int, float)):
36            return val != 0
37        return default
38
39    def _coerce_int(val, default: int = 0, minv=None, maxv=None) -> int:
40        """将任意输入安全转换为 int, 支持范围裁剪。"""
41        if isinstance(val, int):
42            return val
43        if isinstance(val, str):
44            try:
45                v = int(val)
46            except (ValueError, TypeError):
47                return default
48            if minv is not None:
49                v = max(v, minv)
50            if maxv is not None:
51                v = min(v, maxv)
52            return v
53        if isinstance(val, float):
54            v = int(val)
55            if minv is not None:

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 716-770)

```

1         v = max(v, minv)
2         if maxv is not None:
3             v = min(v, maxv)
4         return v
5     return default
6
7     def _coerce_list(val, default=None):
8         """将任意输入安全转换为 list。"""
9         if isinstance(val, list):
10            return val
11        if isinstance(val, (str, bytes)):
12            return [val]
13        if val is None:
14            return default or []
15        return default or []
16
17        # =====
18        # 工具分发 - 将工具名映射到对应的 handler 函数
19        async def _handle_project_tool(arguments: dict) -> Any:
20            return await _handle_project(**arguments)
21
22        async def _handle_delphi_kb(arguments: dict) -> Any:
23            action = arguments.get("action", "search")
24            kb_type = arguments.get("kb_type", "all")
25            if action == "search":
26                return await doc_tools.search_documents(arguments) if kb_type == "document" else await kb_tools.search_kr
27            elif action == "stats":
28                return await doc_tools.get_document_statistics(arguments) if kb_type == "document" else await kb_tools.ge
29            elif action == "build":
30                async_mode = _coerce_bool(arguments.get("async_mode"), True)
31                if not async_mode:
32                    return await kb_tools.build_unified_knowledge_base(arguments)
33                version = arguments.get("version")
34                rebuild = _coerce_bool(arguments.get("rebuild"), False)
35                kb_type_map = {"all": "build_knowledge_base", "delphi": "build_knowledge_base",
36                             "thirdparty": "build_thirdparty_knowledge_base", "project": "init_project_knowledge_base",
37                             "document": "build_document_knowledge_base"}
38                task_type = kb_type_map.get(kb_type, "build_knowledge_base")
39                incremental = arguments.get("incremental", False)
40                if task_type == "build_document_knowledge_base":
41                    directory = arguments.get("directory")
42                    if not directory:
43                        detected = _auto_detect_delphi_help_dir()
44                        if detected:
45                            directory = detected
46                            logger.info(f"自动检测到 Delphi 帮助目录: {directory}")
47                        else:
48                            logger.warning("未提供 directory 且未检测到 Delphi 帮助目录")
49                task_params = {"urls": arguments.get("urls", []), "directory": directory,
50                             "extensions": arguments.get("extensions", [".chm"]),
51                             "start_url": arguments.get("start_url"), "max_pages": arguments.get("max_pages", 100),
52                             "max_depth": arguments.get("max_depth", 3), "domain_filter": arguments.get("domain_fil
53                             "url_pattern": arguments.get("url_pattern"), "exclude": arguments.get("exclude"),
54                             "rebuild": arguments.get("rebuild", False)}
55            elif task_type == "init_project_knowledge_base":

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 771-825)

```

1         resolved_path = _resolve_project_path(arguments.get("project_path"))
2         task_params = {"project_path": resolved_path, "version": version,
3                       "rebuild": rebuild, "build_thirdparty": arguments.get("build_thirdparty", True),
4                       "build_project": arguments.get("build_project", True)}
5     else:
6         task_params = {"version": version, "rebuild": rebuild, "incremental": incremental}
7     return await async_tools.start_async_task({"task_type": task_type, "task_params": task_params,
8                                               "show_progress": arguments.get("show_progress", True),
9                                               "_on_complete": arguments.get("_on_complete")})
10    elif action == "build_embedding":
11        pp = _resolve_project_path(arguments.get("project_path"))
12        if not pp:
13            return {"error": "未检测到项目路径"}
14        return await async_tools.start_async_task({"task_type": "build_embedding", "task_params": {"project_path":
15                                                  "show_progress": True,
16                                                  "_on_complete": arguments.get("_on_complete")})
17    elif action == "scan":
18        return await doc_tools.scan_documents(arguments) if kb_type == "document" else {"error": "action=scan 仅"}
19    elif action == "web":
20        return await doc_tools.add_web_document(arguments) if kb_type == "document" else {"error": "action=web 仅"}
21    elif action == "read":
22        if arguments.get("url") or arguments.get("doc_id"):
23            return await doc_tools.read_document(arguments)
24        elif arguments.get("file_path"):
25            return await read_source_file(arguments)
26        return {"error": "action=read 需要 url/doc_id 或 file_path 参数"}
27    return {"error": f"未知action: {action}"}
28
29    async def _handle_file_tool(arguments: dict) -> Any:
30        return await file_tool.handle_file_tool(arguments)
31
32    async def _handle_manage_component(arguments: dict) -> Any:
33        # manage_component_mod 是函数(被 __init__.py re-export 了), 直接调用
34        return await manage_component_mod(
35            action=arguments.get("action", "create"),
36            target_dfm=arguments.get("target_dfm"),
37            target_pas=arguments.get("target_pas"),
38            component_name=arguments.get("component_name"),
39            parent_name=arguments.get("parent_name"),
40            new_component_class=arguments.get("new_component_class"),
41            new_component_name=arguments.get("new_component_name"),
42            properties=arguments.get("properties"),
43            dfm_text=arguments.get("dfm_text"),
44            code=arguments.get("code", ""),
45            uses=arguments.get("uses"),
46            type_decl=arguments.get("type_decl", ""),
47            init_code=arguments.get("init_code", ""),
48            compile_timeout=arguments.get("compile_timeout", 60),
49            exec_timeout=arguments.get("exec_timeout", 15),
50        )
51
52    async def _handle_check_environment(arguments: dict) -> Any:
53        action = arguments.get("action", "check")
54        if action == "detect":
55            return await search_compilers(search_path=arguments.get("search_path"))

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 826-880)

```

1     elif action == "check":
2         return await check_environment()
3     elif action == "install":
4         return await pasfmt.download_and_install_pasfmt(install_dir=arguments.get("install_dir"))
5     elif action == "format_install":
6         return await pasfmt.download_and_install_pasfmt_rad(delphi_version=arguments.get("delphi_version", "11"),
7         return {"error": f"未知action: {action}"}
8
9     async def _handle_async_task(arguments: dict) -> Any:
10        action = arguments.get("action", "list")
11        handlers = {"start": async_tools.start_async_task, "status": async_tools.get_task_status,
12                  "result": async_tools.get_task_result, "list": async_tools.list_tasks,
13                  "cancel": async_tools.cancel_task}
14        handler = handlers.get(action)
15        if handler:
16            return await handler(arguments)
17        return {"error": f"未知action: {action}"}
18
19    async def _handle_package(arguments: dict) -> Any:
20        return await handle_package(**arguments)
21
22    async def _handle_get_coding_rules(arguments: dict) -> Any:
23        return await _get_coding_rules(project_path=arguments.get("project_path"), section=arguments.get("section"))
24
25    async def _handle_code_hosting(arguments: dict) -> Any:
26        try:
27            if "action" not in arguments:
28                return {"status": "failed", "message": "missing required parameter: action"}
29            # 使用 asyncio.to_thread 避免同步 HTTP 阻塞事件循环
30            return await asyncio.to_thread(code_hosting, **arguments)
31        except Exception as e:
32            logger.error(f"code_hosting 执行失败: {e}", exc_info=True)
33            return {"status": "failed", "message": f"code_hosting failed: {e}"}
34
35    async def _handle_tool_help(arguments: dict) -> Any:
36        return get_tool_help(tool_name=arguments.get("tool_name", ""))
37
38    async def _handle_experience(arguments: dict) -> dict:
39        """处理 experience 工具调用, 带 asyncio 超时保护 (30s)。"""
40        import asyncio
41        try:
42            result = await asyncio.wait_for(
43                asyncio.to_thread(_experience, **arguments),
44                timeout=30,
45            )
46            return result
47        except asyncio.TimeoutError:
48            return {
49                "status": "failed",
50                "message": "experience 操作超时 (30s), 可能是 embedding 模型加载/下载耗时过长。"
51                " 建议: 先调用 delphi_kb(action=build_embedding) 预加载模型, "
52                " 再使用 experience 的语义搜索功能。",
53            }
54        except Exception as e:
55            return {"status": "failed", "message": f"experience failed: {e}"}

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 881-935)

```
1
2  async def _handle_daofy_update(arguments: dict) -> dict:
3      """处理 daofy_update 工具调用。"""
4      action = arguments.get("action", "check")
5
6      if action == "version":
7          install_type = "git" if updater.is_git_installation() else "pip"
8          return {
9              "version": updater.get_current_version(),
10             "install_type": install_type,
11             "python": sys.version,
12         }
13
14     if action == "check":
15         result = await asyncio.get_running_loop().run_in_executor(
16             None, updater.check_for_update
17         )
18         if result is None:
19             return {
20                 "error": "无法检查更新（网络不可达或 GitHub API 异常）",
21                 "hint": "请检查网络连接后重试",
22             }
23         install_type = "git" if updater.is_git_installation() else "pip"
24         result["install_type"] = install_type
25         if result["update_available"]:
26             if install_type == "git":
27                 result["message"] = (
28                     f"发现新版本 v{result['latest']}! "
29                     f"当前版本 v{result['current']}。"
30                     f"使用 daofy_update(action='update') 执行 git pull 更新。"
31                 )
32             else:
33                 result["message"] = (
34                     f"发现新版本 v{result['latest']}! "
35                     f"当前版本 v{result['current']}。"
36                     f"请运行: pip install --upgrade daofy-for-delphi"
37                 )
38         else:
39             result["message"] = f"当前已是最新版本: v{result['current']}"
40         return result
41
42     if action == "update":
43         if not updater.is_git_installation():
44             info = updater.check_for_update()
45             latest = info["latest"] if info else "unknown"
46             return {
47                 "success": False,
48                 "message": (
49                     "当前为 pip 安装模式, 不支持 git pull 更新。 \n"
50                     f"请手动运行: pip install --upgrade daofy-for-delphi"
51                     f"{f' (最新版: v{latest})' if latest != 'unknown' else ''}"
52                 ),
53             }
54         result = await updater.git_pull_update()
55         if result["success"] and result["updated"]:
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 936-990)

```
1         # 更新全局缓存
2         global _update_check_result
3         _update_check_result = None
4         return result
5
6         return {"error": f"未知 action: {action}"}
7
8     async def _handle_generate_copyright(arguments: dict) -> dict:
9         """处理 generate_copyright 工具调用。"""
10        try:
11            result = await asyncio.wait_for(
12                asyncio.to_thread(_generate_copyright, **arguments),
13                timeout=300,
14            )
15            return result
16        except asyncio.TimeoutError:
17            return {"status": "failed", "message": "generate_copyright 执行超时 (300s)"}
18        except Exception as e:
19            return {"status": "failed", "message": f"generate_copyright failed: {e}"}
20
21    async def _handle_automate_delphi(arguments: dict) -> dict:
22        """处理 automate_delphi 工具调用。"""
23        import asyncio
24        app_path = arguments.get("app_path", "")
25        script = arguments.get("script", "")
26        snapshots_dir = arguments.get("snapshots_dir", "")
27        wait_timeout = arguments.get("wait_timeout", 10)
28        keep_alive = arguments.get("keep_alive", False)
29
30        if not app_path or not script:
31            return {"status": "error", "message": "缺少必需参数: app_path, script"}
32
33        try:
34            result = await asyncio.wait_for(
35                asyncio.to_thread(_execute_script,
36                                app_path=app_path,
37                                script=script,
38                                snapshots_dir=snapshots_dir,
39                                wait_for_pipe=wait_timeout,
40                                keep_alive=keep_alive),
41                timeout=300,
42            )
43            return result
44        except asyncio.TimeoutError:
45            return {
46                "status": "failed",
47                "message": "automate_delphi 执行超时 (300s)",
48            }
49        except Exception as e:
50            return {"status": "failed", "message": f"automate_delphi failed: {e}"}
51
52    _TOOL_HANDLERS = {
53        "project": _handle_project_tool,
54        "delphi_kb": _handle_delphi_kb,
55        "delphi_file": _handle_file_tool,
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 991-1045)

```

1      "file_tool": _handle_file_tool, # 旧名兼容别名
2      "manage_component": _handle_manage_component,
3      "check_environment": _handle_check_environment,
4      "async_task": _handle_async_task,
5      "package": _handle_package,
6      "get_coding_rules": _handle_get_coding_rules,
7      "code_hosting": _handle_code_hosting,
8      "tool_help": _handle_tool_help,
9      "experience": _handle_experience,
10     "daofy_update": _handle_daofy_update,
11     "generate_copyright": _handle_generate_copyright,
12     "automate_delphi": _handle_automate_delphi,
13 }
14
15 @server.call_tool()
16 async def call_tool(name: str, arguments: dict):
17     """调用工具（由 _TOOL_HANDLERS dispatch）"""
18     import time as _time
19     from datetime import datetime as _datetime
20     import asyncio as _asyncio
21
22     _call_start = _time.monotonic()
23     _call_start_dt = _datetime.now()
24
25     logger.info(f"调用工具: {name}")
26     result = None
27
28     try:
29         handler = _TOOL_HANDLERS.get(name)
30         if handler:
31             # — MCP 推送通知注入 —
32             # 对于 code_hosting 等支持异步任务的工具，注入 _on_complete 回调
33             # 任务完成时自动推送 TaskStatusNotification 到 MCP 客户端，无需轮询
34             try:
35                 from mcp.types import (
36                     TaskStatusNotification, TaskStatusNotificationParams,
37                 )
38                 _session = server.request_context.session
39                 _loop = _asyncio.get_running_loop()
40
41                 def _make_on_complete(session, loop):
42                     def _on_complete(task_info):
43                         """后台任务完成回调 - 推送 TaskStatusNotification"""
44                         # 映射 local TaskStatus → MCP Literal 状态值
45                         status_map = {
46                             'COMPLETED': 'completed',
47                             'FAILED': 'failed',
48                             'CANCELLED': 'cancelled',
49                         }
50                         ts = task_info.status.name # e.g. 'COMPLETED'
51                         mcp_status = status_map.get(ts, 'completed')
52                         # 确保 datetime 类型
53                         created = task_info.created_at
54                         updated = task_info.completed_at or _datetime.now()
55

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1046-1100)

```

1         notif = TaskStatusNotification(
2             params=TaskStatusNotificationParams(
3                 taskId=task_info.task_id,
4                 status=mcp_status,
5                 statusMessage=task_info.message[:500] if task_info.message else None,
6                 createdAt=created,
7                 lastUpdatedAt=updated,
8                 ttl=3600000, # 1 hour retention
9             )
10        )
11        # 从后台线程调度到 asyncio 事件循环
12        asyncio.run_coroutine_threadsafe(
13            session.send_notification(notif),
14            loop
15        )
16        return _on_complete
17
18        arguments['_on_complete'] = _make_on_complete(_session, _loop)
19    except (LookupError, AttributeError, ImportError) as _ctx_err:
20        logger.debug(f"无法注入 MCP 推送回调: {_ctx_err}")
21        # 非 MCP 环境 (如测试) 或无 request_context 时静默跳过
22
23        result = await handler(arguments)
24    else:
25        raise ValueError(f"未知工具: {name}")
26
27    # 计算调用用时
28    _call_end = _time.monotonic()
29    _call_end_dt = _datetime.now()
30    _duration = _call_end - _call_start
31
32    # P2: 智能提示
33    hint = _get_smart_hint(name, result, arguments)
34    if hint:
35        if isinstance(result, CallToolResult):
36            if result.content and hasattr(result.content[0], 'text'):
37                result.content[0].text = result.content[0].text + "\n\n" + hint
38        elif isinstance(result, dict):
39            msg = result.get('message', '')
40            if isinstance(msg, str):
41                result['message'] = msg + "\n\n" + hint
42
43    # P3: API 调用日志 (排除注入的 _on_complete 回调, 防止 json.dumps 序列化函数报错)
44    _log_args = {k: v for k, v in arguments.items() if k != '_on_complete'}
45    log_api_call(logger, name, _log_args, result)
46
47    import json as _json
48    _show_timing = config_manager.get_show_timing()
49    # 统一提取 data: dict→直接使用, CallToolResult→提取 TextContent 文本
50    if isinstance(result, dict):
51        data = result
52        is_error = (result.get('status') == 'failed'
53                   or result.get('success') is False
54                   or (result.get('error') is not None and result.get('error') != ''))
55    elif isinstance(result, CallToolResult):

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1101-1155)

```
1         extracted = None
2         if result.content and len(result.content) > 0:
3             ct = result.content[0]
4             if hasattr(ct, 'text'):
5                 extracted = ct.text
6         if extracted is not None:
7             try:
8                 parsed = _json.loads(extracted)
9                 if isinstance(parsed, dict):
10                    data = {k: v for k, v in parsed.items() if v is not None}
11                else:
12                    data = extracted
13            except (_json.JSONDecodeError, TypeError):
14                data = extracted
15        else:
16            data = str(result)
17            is_error = getattr(result, 'isError', False)
18        elif isinstance(result, (str, bytes)):
19            data = result
20            is_error = False
21        else:
22            data = str(result)
23            is_error = False
24
25        response = {'success': not is_error, 'data': data}
26        if isinstance(result, CallToolResult):
27            response['isError'] = is_error
28        if _show_timing and isinstance(data, dict):
29            response['timing'] = {
30                'duration': round(_duration * 1000, 1),
31                'startTime': _call_start_dt.strftime('%Y-%m-%dT%H:%M:%S.%f')[:-3],
32                'endTime': _call_end_dt.strftime('%Y-%m-%dT%H:%M:%S.%f')[:-3],
33            }
34        if isinstance(response, (dict, list)):
35            try:
36                text = _json.dumps(response, ensure_ascii=False, indent=2, default=str)
37            except (TypeError, ValueError):
38                text = str(response)
39        else:
40            text = str(response)
41        result = CallToolResult(content=[TextContent(type="text", text=text)], isError=is_error)
42        return result
43
44    except Exception as e:
45        _call_end = _time.monotonic()
46        _call_end_dt = _datetime.now()
47        _duration = _call_end - _call_start
48        error_result = {
49            "error": str(e),
50            "timing": {
51                'duration': round(_duration * 1000, 1),
52                'startTime': _call_start_dt.strftime('%Y-%m-%dT%H:%M:%S.%f')[:-3],
53                'endTime': _call_end_dt.strftime('%Y-%m-%dT%H:%M:%S.%f')[:-3],
54            }
55        }
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1156-1210)

```
1         log_api_call(logger, name, arguments, {"error": str(e)})
2         logger.error(f"工具调用失败: {str(e)}", exc_info=True)
3         import json as _json
4         return CallToolResult(
5             content=[TextContent(type="text", text=_json.dumps(error_result, ensure_ascii=False, indent=2))],
6             isError=True
7         )
8
9     # 注册 MCP 资源
10    _resources_dir = project_root / "config"
11
12    @server.list_resources()
13    async def list_resources():
14        """列出可用资源"""
15        resources = []
16        coding_rules_path = _resources_dir / "CODING_RULES.mdc"
17        if coding_rules_path.exists():
18            resources.append(Resource(
19                uri="delphi://coding-rules",
20                name="CODING_RULES",
21                title="Delphi 编码规范",
22                description="Delphi 源码编码规则, 包含命名规范、格式化、类型声明顺序、修改/审核代码规则等",
23                mimeType="text/markdown"
24            ))
25        resources.append(Resource(
26            uri="delphi://health",
27            name="health",
28            title="Daofy 服务器状态",
29            description="服务器运行状态、版本号、文件监听器状态等健康检查信息",
30            mimeType="application/json"
31        ))
32        return resources
33
34    @server.read_resource()
35    async def read_resource(uri: str):
36        """读取资源内容"""
37        from pydantic import AnyUrl # pydantic import 延迟以避免不必要的依赖加载
38
39        if uri == "delphi://coding-rules":
40            rules_path = _resources_dir / "CODING_RULES.mdc"
41            if rules_path.exists():
42                with open(rules_path, 'r', encoding='utf-8') as f:
43                    content = f.read()
44                return ReadResourceResult(
45                    contents=[TextResourceContents(
46                        uri=AnyUrl(uri),
47                        mimeType="text/markdown",
48                        text=content
49                    )]
50                )
51            return ReadResourceResult(
52                contents=[TextResourceContents(
53                    uri=AnyUrl(uri),
54                    text="编码规则文件不存在"
55                )]
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1211-1265)

```

1         )
2
3         if uri == "delphi://health":
4             import json as _json
5             uptime = time.monotonic() - _server_start_time
6             watcher_running = (
7                 _project_file_watcher is not None
8             ) if '_project_file_watcher' in dir() else False
9             health = {
10                "version": __version__,
11                "uptime_seconds": round(uptime, 1),
12                "uptime": f"{int(uptime // 3600)}h{int((uptime % 3600) // 60)}m{int(uptime % 60)}s",
13                "file_watcher_active": watcher_running,
14            }
15            return ReadResourceResult(
16                contents=[TextResourceContents(
17                    uri=AnyUrl(uri),
18                    mimeType="application/json",
19                    text=_json.dumps(health, ensure_ascii=False, indent=2)
20                )]
21            )
22
23            raise ValueError(f"未知资源: {uri}")
24
25            # =====
26            # 后台版本检查 - 启动时异步检测 GitHub 有无新版本
27            # =====
28
29            async def _background_version_check():
30                """后台检查 Daofy 版本更新, 结果存入全局变量。"""
31                global _update_check_result, _update_check_done
32                try:
33                    logger.info("正在后台检查 Daofy 版本更新...")
34                    loop = asyncio.get_running_loop()
35                    result = await loop.run_in_executor(None, updater.check_for_update)
36                    if result:
37                        _update_check_result = result
38                        if result.get("update_available"):
39                            logger.warning(
40                                "发现新版本! 当前: %s, 最新: %s → %s",
41                                result["current"], result["latest"], result["release_url"],
42                            )
43                        else:
44                            logger.info("当前已是最新版本: %s", result["current"])
45                    else:
46                        logger.debug("版本检查未返回结果 (可能网络不可达)")
47                except Exception as e:
48                    logger.debug(f"版本检查失败 (不影响正常运行): {e}")
49                finally:
50                    _update_check_done = True
51
52            # 启动后台版本检查 (不阻塞启动)
53            asyncio.create_task(_background_version_check())
54
55            # =====

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1266-1320)

```

1   # 自动构建项目知识库 - 启动时检测项目目录并后台构建
2   # =====
3
4   async def _auto_build_project_kb():
5       """自动检测项目目录并后台构建项目知识库（不阻塞启动）"""
6       try:
7           logger.info("正在自动检测项目目录...")
8           loop = asyncio.get_running_loop()
9
10          # 在 executor 中执行 CWD 扫描（可能涉及文件系统 I/O）
11          project_path = await loop.run_in_executor(
12              None, _resolve_project_path, None
13          )
14
15          if not project_path:
16              logger.info(
17                  "未检测到项目文件（.dproj），跳过自动构建项目知识库"
18              )
19              return
20
21          logger.info(
22              "检测到项目： %s，正在后台自动构建项目知识库...",
23              project_path,
24          )
25
26          # 使用现有异步任务机制提交后台构建
27          # rebuild=False → 增量更新（只索引变更文件）
28          # 首次运行时 KB 不存在， build_project_knowledge_base 会自动全量构建
29          # Step 1 的热切换机制保证：已有 KB 重建时不阻塞搜索
30          task_params = {
31              "project_path": project_path,
32              "rebuild": False,
33              "build_thirdparty": True,
34              "build_project": True,
35          }
36          result = await async_tools.start_async_task({
37              "task_type": "init_project_knowledge_base",
38              "task_params": task_params,
39              "show_progress": False,
40          })
41
42          # start_async_task 返回 CallToolResult, isError=False 表示任务已成功提交到后台
43          if result.isError:
44              logger.warning(
45                  "自动构建项目知识库提交失败： %s", project_path
46              )
47          else:
48              logger.info(
49                  "自动构建项目知识库任务已提交到后台： %s", project_path
50              )
51
52          # — Step 3: 启动文件变更监听（如果 watchdog 可用） —
53          await _start_project_file_watcher(project_path)
54
55      except Exception as e:

```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1321-1375)

```
1         logger.debug(  
2             "自动构建项目知识库失败（不影响正常运行）： %s", e  
3         )  
4  
5     async def _start_project_file_watcher(project_path: str) -> None:  
6         """启动项目文件变更监听器，自动触发增量 KB 更新。  
7  
8         在 executor 中启动，不阻塞事件循环。watchdog 不可用时静默降级。  
9         """  
10        global _project_file_watcher  
11        try:  
12            from src.services.knowledge_base.file_watcher import (  
13                ProjectFileWatcher,  
14            )  
15  
16            project_dir = str(Path(project_path).parent)  
17            loop = asyncio.get_running_loop()  
18  
19            def _start_watcher() -> Optional[object]:  
20                w = ProjectFileWatcher(project_path, project_dir)  
21                w.start()  
22                return w  
23  
24            watcher = await loop.run_in_executor(None, _start_watcher)  
25            if watcher:  
26                _project_file_watcher = watcher  
27                logger.info(  
28                    "文件变更监听已启动 (watchdog 可用): %s", project_dir  
29                )  
30            else:  
31                logger.info(  
32                    "文件变更监听未启动 (watchdog 不可用): %s", project_dir  
33                )  
34  
35        except Exception as e:  
36            logger.debug(  
37                "启动文件变更监听失败（不影响正常运行）： %s", e  
38            )  
39  
40        # 启动后台项目知识库自动构建（不阻塞启动）  
41        asyncio.create_task(_auto_build_project_kb())  
42  
43        # 启动服务器  
44        logger.info("MCP Server 启动完成,准备接收请求...")  
45        async with stdio_server() as (read_stream, write_stream):  
46            await server.run(  
47                read_stream,  
48                write_stream,  
49                server.create_initialization_options()  
50            )  
51  
52  
53    def _cleanup_resources():  
54        """清理资源：关闭后台任务、DB连接、临时文件等"""  
55        logger.info("清理资源中...")
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1376-1430)

```
1   try:
2       from src.tools.knowledge_base import _cleanup_pkb_cache # 延迟导入避免循环import
3       _cleanup_pkb_cache()
4   except Exception:
5       logger.warning("清理 pkb_cache 时发生异常", exc_info=True)
6   try:
7       from src.tools.dfm_utils import _cleanup_dfm_temp_dirs
8       _cleanup_dfm_temp_dirs()
9   except Exception:
10      logger.warning("清理 DFM 临时文件时发生异常", exc_info=True)
11  try:
12      from src.services.experience_service import cleanup as _cleanup_exp
13      _cleanup_exp()
14  except Exception:
15      logger.warning("清理经验库时发生异常", exc_info=True)
16  global _project_file_watcher
17  if _project_file_watcher is not None:
18      try:
19          _project_file_watcher.stop()
20      except Exception:
21          logger.warning("停止文件监听时发生异常", exc_info=True)
22      _project_file_watcher = None
23  logger.info("资源清理完成")
24
25
26  def _build_arg_parser() -> "argparse.ArgumentParser":
27      """构建 CLI 参数解析器 (不依赖任何服务, --help/--version 立即退出)"""
28      import argparse
29      parser = argparse.ArgumentParser(
30          prog="daofy",
31          description="Daofy for Delphi – MCP Server (Delphi 编译 + 知识库)",
32      )
33      parser.add_argument(
34          "--version",
35          action="store_true",
36          help="显示版本信息并退出",
37      )
38      parser.add_argument(
39          "--config",
40          metavar="PATH",
41          help="自定义 compilers.json 路径 (默认: config/compilers.json)",
42      )
43      return parser
44
45
46  def main():
47      """主函数"""
48      # — 早退: --help/--version 不触发任何服务初始化 —
49      # 避免在没装 Delphi / 默认路径失效的环境下 --help 报错
50      parser = _build_arg_parser()
51      args = parser.parse_args()
52      if args.version:
53          print(f"Daofy v{__version__}")
54          print(f"Python {sys.version.split()[0]}")
55          print(f"{{__copyright__}}")
```

吉林省左右软件开发有限公司 版权所有

文件: server.py (原始行号 1431-1445)

```
1         return
2
3     try:
4         asyncio.run(run_server())
5     except KeyboardInterrupt:
6         logger.info("服务器已停止")
7     except Exception as e:
8         logger.error(f"服务器运行失败: {str(e)}", exc_info=True)
9         sys.exit(1)
10    finally:
11        _cleanup_resources()
12
13
14    if __name__ == "__main__":
15        main()
```

吉林省左右软件开发有限公司 版权所有

文件: tools/file\_tool.py (原始行号 1-55)

```

1  """
2  delphi_file - Delphi 文件专用操作工具 (MCP 注册名 delphi_file, 原 file_tool)
3
4  整合读取/写入/格式化/备份管理, 覆盖 Delphi 文件操作完整生命周期。
5  MCP 客户端以 delphi_file 名注册, 旧名 file_tool 仍作为别名兼容。
6
7  Action 模式:
8  read      读取文件内容 (继承 read_source_file, 支持按路径/类名/函数名搜索)
9  write     写入文件内容 (自动备份到 __history, 支持 DFM 透明转换)
10 batch_write 批量写入 (edits 数组, 内部自动按 start_line 排序后依次替换, 以备份文件为参照系)
11 format    格式化 Delphi 源码 (继承 format_delphi, pasfmt 驱动)
12 backup   备份管理 (创建/恢复/列表/对比)
13 uses     增删 uses 子句中的单元 (命名空间冲突检测 + 自动排序)
14
15 返回值统一为 dict, 遵循项目规范:
16  success: {"status": "success", "message": "...", ...}
17  error:   {"status": "failed", "message": "..."}
18  """
19
20  import os
21  import locale
22  import shutil
23  import tempfile
24  import re
25  import threading
26  from typing import Any, Optional, Dict, List
27  from mcp.types import CallToolResult
28  from ..utils.logger import get_logger
29  from ..utils.file_backup import create_backup, list_backups, restore_backup, detect_encoding
30  from . import pasfmt
31  from .read_source_file import read_source_file as _read_file, search_and_read_file as _search_read_file
32  from . import dfm_utils
33
34  logger = get_logger(__name__)
35
36  _DELPHI_EXTENSIONS = {'.pas', '.dpr', '.dpk', '.dfm', '.fmx', '.inc', '.dproj'}
37
38  _SYSTEM_SENSITIVE_DIRS: List[str] = []
39  if os.name == 'nt':
40      _windir = os.environ.get('WINDIR', r'C:\Windows')
41      _SYSTEM_SENSITIVE_DIRS = [
42          os.path.join(_windir, 'System32', 'config'),
43          os.path.join(_windir, 'System32', 'drivers', 'etc'),
44      ]
45  else:
46      _SYSTEM_SENSITIVE_DIRS = ['/etc/shadow', '/etc/ssh']
47
48
49  def _validate_path(file_path: str) -> Optional[str]:
50      """校验文件路径安全性, 返回 None 表示安全, 否则返回错误信息"""
51      if '\0' in file_path:
52          return "路径包含 null 字节"
53      try:
54          resolved = os.path.abspath(os.path.realpath(file_path))
55      except (OSError, ValueError) as e:

```

吉林省左右软件开发有限公司 版权所有

文件: tools/file\_tool.py (原始行号 56-110)

```

1         return "路径解析失败: %s" % str(e)
2     for sensitive_dir in _SYSTEM_SENSITIVE_DIRS:
3         try:
4             resolved_relative = os.path.relpath(resolved, sensitive_dir)
5             if not resolved_relative.startswith('.'):
6                 return "路径位于系统敏感目录中: %s" % sensitive_dir
7         except ValueError:
8             pass
9     return None
10
11
12 def _is_delphi_file(file_path: str) -> bool:
13     """判断是否是 Delphi 源文件"""
14     ext = os.path.splitext(file_path)[1].lower()
15     return ext in _DELPHI_EXTENSIONS
16
17
18 def _is_dfm_file(file_path: str) -> bool:
19     """判断是否是 DFM/FMX 表单文件 (Delphi VCL/FireMonkey 表单, 可能为二进制或文本格式)"""
20     ext = os.path.splitext(file_path)[1].lower()
21     return ext in {'.dfm', '.fmx'}
22
23
24 def _wrap_error(msg: str) -> Dict[str, Any]:
25     """构造错误 dict"""
26     return {"status": "failed", "message": msg}
27
28
29 def _normalize_encoding_name(enc: str) -> str:
30     """
31     归一化编码名: 去除 BOM 后缀、-/_ 变体, 统一小写。
32
33     utf-8-sig 在归一化层面视为 utf-8 (BOM 在打开时自动剥离/添加),
34     用于编码兼容性比较。
35     """
36     if not enc:
37         return ""
38     e = enc.lower().replace("_", "-")
39     if e == "utf-8-sig":
40         e = "utf-8"
41     return e
42
43
44 def _is_encoding_compatible(user_enc: str, detected_enc: str) -> bool:
45     """
46     检查用户指定的 encoding 与文件实际编码是否兼容。
47
48     兼容规则:
49     - 完全相等 (归一化后)
50     - utf-8 ↔ utf-8-sig 互相兼容
51     - utf-16 系列 (utf-16/utf-16-le/utf-16-be) 只在同子系列内兼容
52
53     Args:
54     user_enc: 用户显式指定的 encoding 参数
55     detected_enc: detect_encoding 实际检测结果

```

吉林省左右软件开发有限公司 版权所有

文件: tools/file\_tool.py (原始行号 111-165)

```

1
2     Returns:
3         True 表示兼容可继续; False 表示编码不匹配需拒绝
4         """
5     u = _normalize_encoding_name(user_enc)
6     d = _normalize_encoding_name(detected_enc)
7
8     if u == d:
9         return True
10
11     # utf-8 家族内部互兼
12     utf8_family = {"utf-8"}
13     if u in utf8_family and d in utf8_family:
14         return True
15
16     # utf-16 家族内部互兼 (仅在两边都是 utf-16 系列时)
17     utf16_family = {"utf-16", "utf-16-le", "utf-16-be"}
18     if u in utf16_family and d in utf16_family:
19         return True
20
21     return False
22
23
24 async def _read_content(
25     file_path: str,
26     start_line: int = 0,
27     limit: int = 500,
28     search_in: str = "all",
29     project_path: Optional[str] = None,
30     end_line: Optional[int] = None,
31     show_line_numbers: bool = False,
32 ) -> Dict[str, Any]:
33     """
34     读取文件内容的内部实现。
35
36     如果文件在本地直接存在, 检测编码后直接读取 (支持 UTF-8/GBK/UTF-16 等)。
37     否则委托给 read_source_file 走知识库搜索路径。
38
39     start_line/end_line 为 0-indexed 左闭右开区间:
40     - start_line=0 表示从文件第 1 行开始
41     - end_line 不传时等价于「读到 limit 行为止」
42
43     show_line_numbers: 为 True 时每行前面添加行号前缀 (如 " 0: unit Unit1;"),
44     行号为 0-indexed 绝对行号 (与 batch_write / write 的 start_line 直接对齐)。
45     """
46     # 直接文件读取 (支持编码检测 + 降级链)
47     if os.path.isfile(file_path):
48         # 构建编码降级链: 检测编码 → UTF-8 → CP_ACP (系统 ANSI 代码页)
49         detected = detect_encoding(file_path)
50         fallback_encodings = [detected]
51
52         # UTF-8 无 BOM (与检测编码不同时补充)
53         if detected not in ('utf-8', 'utf-8-sig'):
54             fallback_encodings.append('utf-8')
55

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1-55)

```
1  """
2  编译服务
3
4  版权所有 (C) 吉林省左右软件开发有限公司
5  Copyright (C) Equilibrium Software Development Co., Ltd, Jilin
6  Update & Mod By Crystalxp (黑夜杀手 QQ:281309196)
7
8  核心业务逻辑, 协调参数生成、进程执行、结果解析等组件
9  """
10
11 import time
12 import os
13 import winreg
14 from typing import Optional, List
15 from datetime import datetime
16 from pathlib import Path
17 from ..models.compile_request import ProjectCompileRequest, FileCompileRequest, TargetPlatform, CompileOptions, OutputParser
18 from ..models.compile_result import CompileResult, CompileStatus
19 from ..models.command_args import CommandArgs
20 from ..models.compile_history import CompileHistoryEntry
21 from .args_generator import ArgsGenerator
22 from .process_manager import ProcessManager
23 from .config_manager import ConfigManager
24 from ..utils import get_console_encoding
25 from ..utils.parser import OutputParser
26 from ..utils.validator import Validator
27 from ..utils.dproj_parser import DprojParser
28 from ..utils.unit_dependency_analyzer import SmartLibraryPathResolver
29 from ..utils.logger import get_logger
30
31 logger = get_logger(__name__)
32
33
34 class CompilerService:
35     """编译服务"""
36
37     def __init__(self, config_manager: ConfigManager):
38         """初始化编译服务
39
40         Args:
41             config_manager: 配置管理器
42         """
43         self.config_manager = config_manager
44         self.args_generator = ArgsGenerator()
45         self.process_manager = ProcessManager()
46         self.output_parser = OutputParser()
47         self.validator = Validator()
48         self.history: list = []
49
50         # MSBuild 路径
51         self.msbuild_path = self._find_msbuild()
52
53         logger.info("编译服务初始化完成")
54
55     def _find_msbuild(self) -> Optional[str]:
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 56-110)

```

1      """
2      查找 MSBuild 可执行文件。
3
4      搜索优先级:
5      1. vswhere.exe (VS 2017+ 官方工具, 最准确)
6      2. %ProgramFiles(x86)%\Microsoft Visual Studio\Installer\vswhere.exe
7      3. 常见 VS 安装路径列表 (回退方案)
8
9      Returns:
10     MSBuild 路径, 如果未找到则返回 None
11     """
12     import subprocess
13
14     # 方法1: 使用 vswhere.exe 查询最新 VS 的 MSBuild 路径
15     vswhere_candidates = [
16         Path(os.environ.get("ProgramFiles(x86)", "C:\\Program Files (x86)"))
17         / "Microsoft Visual Studio" / "Installer" / "vswhere.exe",
18         Path(os.environ.get("ProgramFiles", "C:\\Program Files"))
19         / "Microsoft Visual Studio" / "Installer" / "vswhere.exe",
20     ]
21     for vswhere in vswhere_candidates:
22         if vswhere.exists():
23             try:
24                 result = subprocess.run(
25                     [
26                         str(vswhere),
27                         "-latest", "-products", "*",
28                         "-requires", "Microsoft.Component.MSBuild",
29                         "-find", "MSBuild\\**\\Bin\\MSBuild.exe",
30                     ],
31                     capture_output=True, text=True, timeout=15,
32                     creationflags=getattr(subprocess, 'CREATE_NO_WINDOW', 0),
33                 )
34                 if result.returncode == 0 and result.stdout.strip():
35                     for line in result.stdout.strip().splitlines():
36                         msbuild_path = line.strip()
37                         if msbuild_path and Path(msbuild_path).exists():
38                             logger.info(f"通过 vswhere 找到 MSBuild: {msbuild_path}")
39                             return msbuild_path
40             except (subprocess.TimeoutExpired, OSError) as e:
41                 logger.debug(f"vswhere 查询失败: {e}")
42
43     # 方法2: 常见 VS 安装路径列表 (回退方案)
44     pf86 = os.environ.get("ProgramFiles(x86)", "C:\\Program Files (x86)")
45     pf = os.environ.get("ProgramFiles", "C:\\Program Files")
46     editions = ["BuildTools", "Community", "Professional", "Enterprise"]
47     years = ["2019", "2022"]
48     possible_paths = []
49     for year in years:
50         for edition in editions:
51             base = Path(pf86) if year == "2019" else Path(pf)
52             possible_paths.append(
53                 str(base / f"Microsoft Visual Studio\\{year}\\{edition}\\MSBuild\\Current\\Bin\\MSBuild.exe")
54             )
55 
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 111-165)

```
1     for path in possible_paths:
2         if Path(path).exists():
3             logger.info(f"找到 MSBuild: {path}")
4             return path
5
6     logger.warning("未找到 MSBuild, 将回退到直接编译")
7     return None
8
9     def _get_delphi_root_from_registry(self, version: Optional[str] = None) -> Optional[str]:
10        """
11        从注册表获取 Delphi 安装根目录
12
13        Args:
14            version: Delphi 版本号(如 "22.0"), 如果为 None 则使用最新版本
15
16        Returns:
17            Delphi 安装根目录, 如果未找到则返回 None
18        """
19        try:
20            # 打开 Delphi 注册表项
21            key = winreg.OpenKey(
22                winreg.HKEY_CURRENT_USER,
23                r"SOFTWARE\Embarcadero\BDS",
24                0,
25                winreg.KEY_READ | winreg.KEY_WOW64_32KEY
26            )
27
28            versions = []
29            index = 0
30            while True:
31                try:
32                    version_key = winreg.EnumKey(key, index)
33                    index += 1
34
35                    # 打开本子项
36                    version_path_key = winreg.OpenKey(key, version_key)
37                    try:
38                        root_dir, _ = winreg.QueryValueEx(version_path_key, "RootDir")
39                        if root_dir and Path(root_dir).exists():
40                            versions.append((version_key, root_dir))
41                    except OSError:
42                        pass
43                    finally:
44                        winreg.CloseKey(version_path_key)
45
46                except OSError:
47                    break
48
49            winreg.CloseKey(key)
50
51            if not versions:
52                return None
53
54            # 如果指定了版本, 查找对应版本
55            if version:
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 166-220)

```
1         for v, root in versions:
2             if v == version:
3                 return root
4
5         # 返回最新版本（版本号最大的）
6         versions.sort(key=lambda x: x[0], reverse=True)
7         return versions[0][1]
8
9     except Exception as e:
10        logger.warning(f"从注册表获取 Delphi 路径失败: {e}")
11        return None
12
13    def _get_rsvars_path(self, version: Optional[str] = None) -> Optional[str]:
14        """
15        获取 rsvars.bat 路径
16
17        Args:
18            version: Delphi 版本号, 如果为 None 则使用最新版本
19
20        Returns:
21            rsvars.bat 完整路径, 如果未找到则返回 None
22        """
23        root_dir = self._get_delphi_root_from_registry(version)
24        if not root_dir:
25            logger.error("无法从注册表获取 Delphi 安装路径")
26            return None
27
28        rsvars_path = Path(root_dir) / "bin" / "rsvars.bat"
29        if rsvars_path.exists():
30            logger.info(f"找到 rsvars.bat: {rsvars_path}")
31            return str(rsvars_path)
32        else:
33            logger.error(f"rsvars.bat 不存在: {rsvars_path}")
34            return None
35
36    def _check_process_running(self, process_name: str) -> Optional[dict]:
37        """
38        检查指定进程是否正在运行
39
40        Args:
41            process_name: 进程名称（不含.exe扩展名）
42
43        Returns:
44            如果进程正在运行, 返回包含进程信息的字典; 否则返回 None
45        """
46        import subprocess
47        import json
48        import re
49
50        if not re.match(r'^[A-Za-z0-9_-]+\$', process_name):
51            logger.warning("进程名包含非法字符, 已拒绝: %s", process_name)
52            return None
53
54        try:
55            result = subprocess.run(
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 221-275)

```
1         ['powershell', '-Command',
2         "Get-Process -Name '%s' -ErrorAction SilentlyContinue | "
3         "Select-Object Id, ProcessName, Path | ConvertTo-Json" % process_name],
4         capture_output=True,
5         text=True,
6         timeout=10,
7         creationflags=getattr(subprocess, 'CREATE_NO_WINDOW', 0)
8     )
9
10    if result.returncode == 0 and result.stdout.strip():
11        output = result.stdout.strip()
12
13        if output.startswith('['):
14            processes = json.loads(output)
15            if processes:
16                proc = processes[0]
17            else:
18                return None
19        elif output.startswith('{'):
20            proc = json.loads(output)
21        else:
22            return None
23
24        logger.info("检测到进程正在运行: %s (PID: %s)", proc.get('ProcessName'), proc.get('Id'))
25        return {
26            'pid': proc.get('Id'),
27            'name': proc.get('ProcessName'),
28            'path': proc.get('Path')
29        }
30
31    return None
32
33    except Exception as e:
34        logger.warning("检查进程时发生错误: %s", str(e))
35    return None
36
37    def _cleanup_dcu_files(self, file_path: str):
38        """
39        清理源文件所在目录的 .dcu 文件
40
41        Args:
42            file_path: 源文件路径
43        """
44        import glob
45
46        file_path_obj = Path(file_path)
47        file_dir = file_path_obj.parent
48
49        if not file_dir.exists():
50            return
51
52        dcu_pattern = str(file_dir / "*.dcu")
53        dcu_files = glob.glob(dcu_pattern)
54
55        if dcu_files:
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 276-330)

```

1         logger.info(f"在 {file_dir} 中找到 {len(dcu_files)} 个 .dcu 文件")
2
3         total_deleted = 0
4         for dcu_file in dcu_files:
5             try:
6                 os.unlink(dcu_file)
7                 logger.debug(f"已删除: {dcu_file}")
8                 total_deleted += 1
9             except Exception as e:
10                logger.warning(f"删除失败 {dcu_file}: {str(e)}")
11
12        if total_deleted > 0:
13            logger.info(f"共删除 {total_deleted} 个 .dcu 文件")
14
15    def _execute_build_event(self, event_name: str, event_cmd: str, project_dir: str,
16                            ignore_exit_code: bool = False, timeout: int = 60,
17                            context: dict = None) -> tuple:
18        """
19        执行编译事件
20
21        Args:
22            event_name: 事件名称(用于日志)
23            event_cmd: 事件命令
24            project_dir: 项目目录
25            ignore_exit_code: 是否忽略退出码
26            timeout: 超时时间(秒)
27            context: 上下文变量字典,包含项目信息等
28
29        Returns:
30            (success, error_message, output) 元组
31        """
32        import subprocess
33        import tempfile
34        import re
35
36        logger.info("执行 %s: %s", event_name, event_cmd[:200])
37
38        _DANGEROUS_PATTERNS = re.compile(
39            r'(?:\^|\b)(?:rm\s+|del\s+|[sq]|format\s+[a-z]:|net\s+user|'
40            r'reg(?:edit(?:32)?|\.exe)?\s+add|powershell\s+-enc|'
41            r'cmd(?:\.exe)?\s+/c\s*(?:curl|wget|bitsadmin))',
42            re.IGNORECASE
43        )
44        if _DANGEROUS_PATTERNS.search(event_cmd):
45            error_msg = "%s 命令包含危险模式, 已拒绝执行: %s" % (event_name, event_cmd[:200])
46            logger.error(error_msg)
47            return (False, error_msg, None)
48
49        # 替换 Delphi 支持的变量
50        if context:
51            # 项目相关变量
52            event_cmd = event_cmd.replace('${PROJECTDIR}', context.get('project_dir', project_dir))
53            event_cmd = event_cmd.replace('${PROJECTPATH}', context.get('project_path', ''))
54            event_cmd = event_cmd.replace('${PROJECTFILENAME}', context.get('project_filename', ''))
55            event_cmd = event_cmd.replace('${PROJECTNAME}', context.get('project_name', ''))

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 331-385)

```
1
2     # 输入文件相关变量
3     event_cmd = event_cmd.replace('${INPUTPATH}', context.get('input_path', ''))
4     event_cmd = event_cmd.replace('${INPUTFILENAME}', context.get('input_filename', ''))
5     event_cmd = event_cmd.replace('${INPUTTEXT}', context.get('input_ext', ''))
6
7     # 输出文件相关变量
8     event_cmd = event_cmd.replace('${OUTPUTDIR}', context.get('output_dir', ''))
9     event_cmd = event_cmd.replace('${OUTPUTPATH}', context.get('output_path', ''))
10    event_cmd = event_cmd.replace('${OUTPUTFILENAME}', context.get('output_filename', ''))
11    event_cmd = event_cmd.replace('${OUTPUTTEXT}', context.get('output_ext', ''))
12
13    # 配置相关变量
14    event_cmd = event_cmd.replace('${Config}', context.get('config', 'Debug'))
15    event_cmd = event_cmd.replace('${Platform}', context.get('platform', 'Win32'))
16    event_cmd = event_cmd.replace('${DEFINES}', context.get('defines', ''))
17
18    # 路径相关变量
19    event_cmd = event_cmd.replace('${DIR}', context.get('dir', project_dir))
20    event_cmd = event_cmd.replace('${INCLUDEPATH}', context.get('include_path', ''))
21    event_cmd = event_cmd.replace('${PATH}', context.get('path', ''))
22
23    # Delphi 环境变量
24    event_cmd = event_cmd.replace('${BDS}', context.get('bds', ''))
25    event_cmd = event_cmd.replace('${LOCALCOMMAND}', context.get('local_command', ''))
26
27    # 系统变量
28    event_cmd = event_cmd.replace('${SystemRoot}', context.get('system_root', ''))
29    event_cmd = event_cmd.replace('${WINDIR}', context.get('windir', ''))
30
31    # 兼容旧版本的变量替换
32    event_cmd = event_cmd.replace('${PROJECTDIR}', project_dir)
33
34    try:
35        logger.warning("编译事件来自 .dproj 文件配置, 请确保项目文件可信")
36        # 使用控制台编码 (可能为 UTF-8 或 GBK), 确保 cmd.exe 正确读取中文路径
37        batch_encoding = get_console_encoding()
38        with tempfile.NamedTemporaryFile(
39            mode='w', suffix='.bat', delete=False, encoding=batch_encoding
40        ) as bat_f:
41            bat_f.write('@echo off\n')
42            bat_f.write(event_cmd + '\n')
43            bat_path = bat_f.name
44
45        try:
46            result = subprocess.run(
47                ['cmd.exe', '/c', bat_path],
48                cwd=project_dir,
49                capture_output=True,
50                encoding=get_console_encoding(),
51                timeout=timeout,
52                creationflags=getattr(subprocess, 'CREATE_NO_WINDOW', 0)
53            )
54        finally:
55            try:
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 386-440)

```
1         os.unlink(bat_path)
2     except OSError:
3         pass
4
5     if result.returncode != 0 and not ignore_exit_code:
6         error_msg = "%s 失败(退出码 %d): %s" % (event_name, result.returncode, result.stderr)
7         logger.error(error_msg)
8         return (False, error_msg, result.stdout + result.stderr)
9
10    logger.info("%s 执行成功", event_name)
11    if result.stdout:
12        logger.debug("%s 输出: %s", event_name, result.stdout)
13
14    return (True, None, result.stdout + result.stderr)
15
16    except subprocess.TimeoutExpired:
17        error_msg = "%s 执行超时(%d秒)" % (event_name, timeout)
18        logger.error(error_msg)
19        return (False, error_msg, None)
20    except Exception as e:
21        error_msg = "%s 执行失败: %s" % (event_name, str(e))
22        logger.error(error_msg, exc_info=True)
23        return (False, error_msg, None)
24
25    def _extract_config_from_dproj(self, project_path: str, options: 'CompileOptions') -> 'CompileOptions':
26        """
27        从 .dproj 文件中提取配置
28
29        Args:
30            project_path: 项目文件路径(.dproj 或 .dpr)
31            options: 原始编译选项
32
33        Returns:
34            合并后的编译选项
35        """
36        # 如果传入的是 .dpr/.dpk 文件,查找对应的 .dproj 文件
37        if project_path.endswith('.dpr') or project_path.endswith('.dpk'):
38            ext = '.dpr' if project_path.endswith('.dpr') else '.dpk'
39            dproj_path = project_path[:-len(ext)] + '.dproj'
40            if not Path(dproj_path).exists():
41                logger.info(f"未找到对应的 .dproj 文件: {dproj_path}")
42                return options
43            elif project_path.endswith('.dproj'):
44                dproj_path = project_path
45            else:
46                return options
47
48        # 解析 .dproj 文件
49        parser = DprojParser(dproj_path)
50        if not parser.parse():
51            logger.warning(f"解析 .dproj 文件失败: {dproj_path}")
52            return options
53
54        # 获取配置和平台
55        config = options.build_configuration
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 441-495)

```
1 platform = "Win64" if options.target_platform == TargetPlatform.WIN64 else "Win32"
2
3 # 提取单元搜索路径
4 # 1. 先获取基础路径(不限制配置和平台)
5 base_unit_paths = parser.get_unit_search_paths()
6 # 2. 再获取平台特定路径
7 platform_unit_paths = parser.get_unit_search_paths(config, platform)
8 # 3. 合并所有路径
9 dproj_unit_paths = base_unit_paths + platform_unit_paths
10
11 if dproj_unit_paths:
12     # 合并用户指定的路径和 .dproj 中的路径
13     all_paths = list(options.unit_search_paths) + dproj_unit_paths
14     # 去重
15     options.unit_search_paths = list(dict.fromkeys(all_paths))
16     logger.info(f"从 .dproj 文件中提取了 {len(dproj_unit_paths)} 个单元搜索路径")
17
18 # 提取条件编译符号
19 dproj_defines = parser.get_conditional_defines(config, platform)
20 if dproj_defines:
21     all_defines = list(options.conditional_defines) + dproj_defines
22     options.conditional_defines = list(dict.fromkeys(all_defines))
23     logger.info(f"从 .dproj 文件中提取了 {len(dproj_defines)} 个条件编译符号")
24
25 # 提取输出路径
26 if not options.output_path:
27     dproj_output = parser.get_output_path(config, platform)
28     if dproj_output:
29         options.output_path = dproj_output
30         logger.info(f"从 .dproj 文件中提取了输出路径: {dproj_output}")
31
32 # 智能解析第三方库路径
33 # 注意: 如果用户显式传入了 unit_search_paths, resolver 会直接返回, 不会进行智能分析
34 try:
35     logger.info("开始解析第三方库路径...")
36     resolver = SmartLibraryPathResolver()
37
38     # 传入用户显式指定的路径, resolver 会判断是否跳过智能分析
39     user_provided_paths = list(options.unit_search_paths) if options.unit_search_paths else None
40     resolved_paths, info = resolver.resolve_library_paths(
41         project_path,
42         platform,
43         user_search_paths=user_provided_paths
44     )
45
46     if info.get("mode") == "user_provided":
47         logger.info(f"使用用户显式传入的 {len(resolved_paths)} 个路径")
48     elif resolved_paths:
49         options.unit_search_paths = resolved_paths
50         logger.info(f"智能解析选择了 {len(resolved_paths)} 个第三方库路径"
51             f" (从 {info.get('total_paths_count', 0)} 个全局路径中筛选, "
52             f"解决了 {info.get('resolved_units', 0)} 个单元依赖)")
53         logger.debug(f"解析详情: {info}")
54
55     # 记录未找到的单元
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 496-550)

```
1         if info.get('still_missing_units'):
2             logger.warning(f"仍有 {len(info['still_missing_units'])} 个单元未找到: "
3                             f"{info['still_missing_units'][:5]}")
4
5         except Exception as e:
6             logger.warning(f"智能解析第三方库路径失败: {e}, 将使用默认方式")
7
8         return options
9
10    async def compile_dpr_direct(self, request: ProjectCompileRequest) -> CompileResult:
11        """
12        直接使用 dcc32/dcc64 编译 .dpr 文件 (不依赖 .dproj)
13
14        Args:
15            request: 工程编译请求
16
17        Returns:
18            编译结果
19        """
20        logger.info(f"直接编译 .dpr 文件: {request.project_path}")
21        start_time = time.time()
22
23        try:
24            # 1. 验证项目路径
25            is_valid, error_msg = self.validator.validate_project_path(request.project_path)
26            if not is_valid:
27                logger.error(f"项目路径验证失败: {error_msg}")
28                return CompileResult(
29                    status=CompileStatus.FAILED,
30                    error_code="INVALID_PROJECT_PATH",
31                    error_message=error_msg,
32                    duration=int((time.time() - start_time) * 1000)
33                )
34
35            # 2. 确定输出路径 - 默认输出到 Win32 子目录
36            project_dir = str(Path(request.project_path).parent)
37            project_name = Path(request.project_path).stem
38
39            if request.options.output_path:
40                output_base = request.options.output_path
41            else:
42                # 平台→输出目录映射, 从 ArgsGenerator 复用
43                from ..services.args_generator import ArgsGenerator
44                lib_dir = ArgsGenerator._PLATFORM_LIB_DIR.get(request.options.target_platform, 'Win32')
45                output_base = str(Path(project_dir) / lib_dir)
46
47            # 确保输出目录存在
48            Path(output_base).mkdir(parents=True, exist_ok=True)
49
50            # 3. 获取编译器配置
51            compiler_config = self.config_manager.get_compiler(request.options.compiler_version)
52            if not compiler_config:
53                # 尝试自动查找 dcc32
54                dcc32_path = self._find_dcc32_from_registry()
55                if dcc32_path:
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 551-605)

```

1         logger.info(f"自动找到 dcc32: {dcc32_path}")
2         compiler_path = dcc32_path
3     else:
4         error_msg = "未配置默认编译器且无法自动查找"
5         logger.error(error_msg)
6         return CompileResult(
7             status=CompileStatus.FAILED,
8             error_code="COMPILER_NOT_FOUND",
9             error_message=error_msg,
10            duration=int((time.time() - start_time) * 1000)
11        )
12    else:
13        compiler_path = compiler_config.path
14
15    # 4. 根据目标平台选择编译器
16    target_platform = request.options.target_platform
17    if target_platform != TargetPlatform.WIN32:
18        compiler_name = self._get_platform_compiler_name(target_platform)
19        bin_dir = Path(compiler_path).parent
20        target_compiler = str(bin_dir / compiler_name)
21        if Path(target_compiler).exists():
22            compiler_path = target_compiler
23            logger.info(f"切换到 {target_platform.value} 编译器: {compiler_path}")
24        else:
25            # 跨平台编译器不存在, 回退查找
26            found = self._find_compiler_from_registry(target_platform)
27            if found:
28                compiler_path = found
29                logger.info(f"从注册表找到 {target_platform.value} 编译器: {compiler_path}")
30            else:
31                logger.warning(f"{target_platform.value} 编译器 ({compiler_name}) 未找到, 尝试使用 dcc32")
32
33    # 5. 验证编译器路径
34    is_valid, error_msg = self.validator.validate_compiler_path(compiler_path)
35    if not is_valid:
36        logger.error(f"编译器路径验证失败: {error_msg}")
37        return CompileResult(
38            status=CompileStatus.FAILED,
39            error_code="INVALID_COMPILER_PATH",
40            error_message=error_msg,
41            duration=int((time.time() - start_time) * 1000)
42        )
43
44    # 6. 检查目标程序是否正在运行
45    running_process = self._check_process_running(project_name)
46    if running_process:
47        error_msg = f"目标程序 '{project_name}.exe' 正在运行 (PID: {running_process['pid']}), 无法编译。请先关闭
48        logger.warning(error_msg)
49        return CompileResult(
50            status=CompileStatus.FAILED,
51            error_code="PROCESS_RUNNING",
52            error_message=error_msg,
53            duration=int((time.time() - start_time) * 1000),
54            log=f"进程信息: PID={running_process['pid']}, 路径={running_process.get('path', '未知')}"
55        )

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 606-660)

```
1
2     # 7. 生成命令行参数
3     args = self._generate_dpr_args(request.project_path, request.options, output_base)
4
5     # 8. 验证参数
6     if not self.args_generator.validate_args(args):
7         logger.error("参数验证失败")
8         return CompileResult(
9             status=CompileStatus.FAILED,
10            error_code="INVALID_ARGS",
11            error_message="编译参数包含非法字符",
12            duration=int((time.time() - start_time) * 1000)
13        )
14
15    logger.info(f"编译命令: {compiler_path} {' '.join(args)}")
16
17    # 9. 执行编译
18    try:
19        return_code, stdout, stderr = await self.process_manager.execute(
20            compiler_path,
21            args,
22            request.options.timeout
23        )
24
25    # 10. 解析输出
26    errors = self.output_parser.parse_errors(stdout + stderr)
27    warnings = self.output_parser.parse_warnings(stdout + stderr)
28
29    # 11. 构建结果
30    duration = int((time.time() - start_time) * 1000)
31    output_file = str(Path(output_base) / f"{project_name}.exe")
32
33    if return_code == 0:
34        output_files = self._collect_output_files(
35            request.project_path,
36            request.options.target_platform.value,
37            request.options.build_configuration or "Debug",
38            output_base,
39        )
40        result = CompileResult(
41            status=CompileStatus.SUCCESS,
42            output_file=output_file,
43            output_files=output_files,
44            warnings=warnings,
45            errors=errors,
46            duration=duration,
47            log=stdout + stderr
48        )
49        logger.info(f"编译成功,输出文件: {output_file},耗时 {duration}ms")
50    else:
51        result = CompileResult(
52            status=CompileStatus.FAILED,
53            error_code="COMPILATION_FAILED",
54            error_message=self.output_parser.extract_error_summary(stdout + stderr),
55            warnings=warnings,
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 661-715)

```
1         errors=errors,
2         duration=duration,
3         log=stdout + stderr
4     )
5     logger.error(f"编译失败,耗时 {duration}ms")
6
7     # 12. 保存编译历史
8     self._save_history(request.project_path, result.status.value, duration, result.error_message)
9
10    return result
11
12    except TimeoutError as e:
13        duration = int((time.time() - start_time) * 1000)
14        logger.error(f"编译超时: {str(e)}")
15        result = CompileResult(
16            status=CompileStatus.TIMEOUT,
17            error_code="COMPILATION_TIMEOUT",
18            error_message=str(e),
19            duration=duration
20        )
21        self._save_history(request.project_path, result.status.value, duration, str(e))
22        return result
23
24    except Exception as e:
25        duration = int((time.time() - start_time) * 1000)
26        error_msg = f"直接编译过程发生异常: {str(e)}"
27        logger.error(error_msg, exc_info=True)
28        result = CompileResult(
29            status=CompileStatus.FAILED,
30            error_code="INTERNAL_ERROR",
31            error_message=error_msg,
32            duration=duration
33        )
34        self._save_history(request.project_path, result.status.value, duration, error_msg)
35        return result
36
37    def _generate_dpr_args(self, project_path: str, options: 'CompileOptions', output_base: str) -> List[str]:
38        """生成 .dpr 文件的直接编译参数"""
39        from ..utils.delphi_env import get_delphi_library_paths, expand_delphi_path_macros
40
41        args = []
42
43        # 项目文件
44        args.append(project_path)
45
46        # 输出目录 (存放 .exe 文件)
47        args.append(f'-E{output_base}')
48
49        # 中间文件目录 (存放 .dcu 文件)
50        dcu_dir = str(Path(output_base) / "dcu")
51        args.append(f'-N{dcu_dir}')
52
53        # 条件编译符号
54        if options.conditional_defines:
55            defines = ";".join(options.conditional_defines)
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 716-770)

```

1         args.append(f'-$D+{defines}')
2
3         # 命名空间搜索路径 - 默认添加 System 命名空间
4         default_namespaces = ["System", "Winapi", "System.Win", "Vcl", "Vcl.Imaging",
5                               "Vcl.Touch", "Vcl.Samples", "Vcl.Shell", "Data", "Dataspn",
6                               "Web", "Soap", "Xml",
7                               "Common", "Protocol", "Engine", "Security", "Transport",
8                               "Integration", "Platforms"]
9         args.append('-NS' + ";" + ".join(default_namespaces))
10
11        # 单元搜索路径 - 如果未提供, 则自动获取 Delphi 默认库搜索路径
12        unit_paths = options.unit_search_paths if options.unit_search_paths else []
13        if not unit_paths:
14            # 自动获取 Delphi 库搜索路径
15            # 使用目标平台名 (首字母大写, 用于注册表查询)
16            from ..services.args_generator import ArgsGenerator
17            platform = ArgsGenerator._PLATFORM_LIB_DIR.get(options.target_platform, 'Win32')
18            delphi_lib_paths = get_delphi_library_paths(platform=platform)
19            # 展开路径中的宏变量
20            for p in delphi_lib_paths:
21                expanded = expand_delphi_path_macros(p, version=None, platform=platform)
22                if expanded and expanded not in unit_paths:
23                    unit_paths.append(expanded)
24
25        if unit_paths:
26            paths = ";" + ".join(unit_paths)
27            args.append(f'-U{paths}')
28
29        # 资源搜索路径
30        if options.resource_search_paths:
31            paths = ";" + ".join(options.resource_search_paths)
32            args.append(f'-R{paths}')
33
34        # 优化选项
35        if options.optimize:
36            args.append('-S0+')
37        else:
38            args.append('-S0-')
39
40        # 调试信息
41        if options.debug:
42            args.append('-SD+')
43        else:
44            args.append('-SD-')
45
46        # 警告级别
47        args.append(f'-W{options.warning_level}')
48
49        # 禁用警告
50        for warning in options.disabled_warnings:
51            args.append(f'-W-{warning}')
52
53        # 输出类型
54        output_type_map = {
55            OutputType.CONSOLE: "-CC",

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 771-825)

```
1         OutputType.GUI: "-CG",
2         OutputType.DLL: "-LD"
3     }
4     args.append(output_type_map[options.output_type])
5
6     # 运行时库
7     if options.runtime_library == RuntimeLibrary.DYNAMIC:
8         args.append('-Y+')
9     else:
10        args.append('-Y-')
11
12    logger.debug(f"生成的 .dpr 编译参数: {' '.join(args)}")
13    return args
14
15    @staticmethod
16    def _get_platform_compiler_name(platform: TargetPlatform) -> str:
17        """根据目标平台返回编译器可执行文件名"""
18        _PLATFORM_COMPILER_MAP = {
19            TargetPlatform.WIN32: 'dcc32.exe',
20            TargetPlatform.WIN64: 'dcc64.exe',
21            TargetPlatform.OSX64: 'dccosx64.exe',
22            TargetPlatform.OSXARM64: 'dccosxarm64.exe',
23            TargetPlatform.IOSDEVICE64: 'dcciosarm64.exe',
24            TargetPlatform.IOSDEVICE: 'dcciosarm64.exe',
25            TargetPlatform.IOSSIMULATOR: 'dcciosarm64.exe',
26            TargetPlatform.ANDROID: 'dccaarm.exe',
27            TargetPlatform.ANDROID64: 'dccaac64.exe',
28            TargetPlatform.LINUX64: 'dcclinux64.exe',
29        }
30        return _PLATFORM_COMPILER_MAP.get(platform, 'dcc32.exe')
31
32    def _find_compiler_from_registry(self, platform: TargetPlatform) -> Optional[str]:
33        """从注册表查找指定平台的编译器路径"""
34        root_dir = self._get_delphi_root_from_registry()
35        if not root_dir:
36            return None
37
38        compiler_name = self._get_platform_compiler_name(platform)
39        compiler_path = Path(root_dir) / "bin" / compiler_name
40        if compiler_path.exists():
41            return str(compiler_path)
42
43        return None
44
45    def _find_dcc32_from_registry(self) -> Optional[str]:
46        """从注册表查找 dcc32.exe 路径（向后兼容）"""
47        return self._find_compiler_from_registry(TargetPlatform.WIN32)
48
49    async def compile_project_with_msbuild(self, request: ProjectCompileRequest) -> CompileResult:
50        """
51        使用 MSBuild 编译 Delphi 工程
52
53        Args:
54            request: 工程编译请求
55        """
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 826-880)

```
1 Returns:
2     编译结果
3     """
4     logger.info(f"使用 MSBuild 编译工程: {request.project_path}")
5     start_time = time.time()
6
7     try:
8         # 1. 检查 MSBuild 是否可用
9         if not self.msbuild_path:
10            error_msg = "未找到 MSBuild,无法使用 MSBuild 编译"
11            logger.error(error_msg)
12            return CompileResult(
13                status=CompileStatus.FAILED,
14                error_code="MSBUILD_NOT_FOUND",
15                error_message=error_msg,
16                duration=int((time.time() - start_time) * 1000)
17            )
18
19        # 2. 验证项目路径
20        is_valid, error_msg = self.validator.validate_project_path(request.project_path)
21        if not is_valid:
22            logger.error(f"项目路径验证失败: {error_msg}")
23            return CompileResult(
24                status=CompileStatus.FAILED,
25                error_code="INVALID_PROJECT_PATH",
26                error_message=error_msg,
27                duration=int((time.time() - start_time) * 1000)
28            )
29
30        # 3. 确定项目文件(.dproj)
31        if request.project_path.endswith(('.dpr', '.dpk')):
32            ext = '.dpr' if request.project_path.endswith('.dpr') else '.dpk'
33            dproj_path = request.project_path[:-len(ext)] + '.dproj'
34            if not Path(dproj_path).exists():
35                error_msg = f"未找到对应的 .dproj 文件: {dproj_path}"
36                logger.error(error_msg)
37                return CompileResult(
38                    status=CompileStatus.FAILED,
39                    error_code="DPROJ_NOT_FOUND",
40                    error_message=error_msg,
41                    duration=int((time.time() - start_time) * 1000)
42                )
43            elif request.project_path.endswith('.dproj'):
44                dproj_path = request.project_path
45            else:
46                error_msg = "项目文件必须是 .dpr、.dpk 或 .dproj 文件"
47                logger.error(error_msg)
48                return CompileResult(
49                    status=CompileStatus.FAILED,
50                    error_code="INVALID_PROJECT_FILE",
51                    error_message=error_msg,
52                    duration=int((time.time() - start_time) * 1000)
53                )
54
55        # 4. 提取并执行编译事件
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 881-935)

```

1         platform = "Win64" if request.options.target_platform == TargetPlatform.WIN64 else "Win32"
2         config = request.options.build_configuration or "Debug"
3
4         project_dir = str(Path(dproj_path).parent)
5         project_path = str(Path(dproj_path).parent)
6         project_filename = Path(dproj_path).name
7         project_name = Path(dproj_path).stem
8
9         # 4.5 检查目标程序是否正在运行
10        running_process = self._check_process_running(project_name)
11        if running_process:
12            error_msg = f"目标程序 '{project_name}.exe' 正在运行 (PID: {running_process['pid']}), 无法编译。请先关闭
13            logger.warning(error_msg)
14            return CompileResult(
15                status=CompileStatus.FAILED,
16                error_code="PROCESS_RUNNING",
17                error_message=error_msg,
18                duration=int((time.time() - start_time) * 1000),
19                log=f"进程信息: PID={running_process['pid']}, 路径={running_process.get('path', '未知')}"
20            )
21
22        # 构建上下文变量 (os 已在模块顶部导入)
23        context = {
24            # 项目相关
25            'project_dir': project_dir,
26            'project_path': project_path,
27            'project_filename': project_filename,
28            'project_name': project_name,
29
30            # 输入文件相关(对于项目编译,输入文件就是项目文件)
31            'input_path': project_path,
32            'input_filename': project_filename,
33            'input_ext': Path(dproj_path).suffix,
34
35            # 输出文件相关
36            'output_dir': request.options.output_path or project_dir,
37            'output_path': request.options.output_path or project_dir,
38            'output_filename': f"{project_name}.exe",
39            'output_ext': '.exe',
40
41            # 配置相关
42            'config': config,
43            'platform': platform,
44            'defines': ';'.join(request.options.conditional_defines) if request.options.conditional_defines else
45
46            # 路径相关
47            'dir': project_dir,
48            'include_path': ';'.join(request.options.unit_search_paths) if request.options.unit_search_paths else
49            'path': os.environ.get('PATH', ''),
50
51            # Delphi 环境变量
52            'bds': os.environ.get('BDS', r'C:\Program Files (x86)\Embarcadero\Studio\22.0'),
53            'local_command': '',
54
55            # 系统变量

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 936-990)

```
1         'system_root': os.environ.get('SystemRoot', r'C:\Windows'),
2         'windir': os.environ.get('WINDIR', r'C:\Windows'),
3     }
4
5     # 解析 .dproj 文件获取编译事件
6     parser = DprojParser(dproj_path)
7     build_events = {}
8     if parser.parse():
9         build_events = parser.get_build_events(config, platform)
10
11     # 执行 PreBuildEvent
12     if build_events.get('pre_build'):
13         success, error_msg, output = self._execute_build_event(
14             "PreBuildEvent",
15             build_events['pre_build'],
16             project_dir,
17             build_events.get('pre_build_ignore_exit_code', False),
18             60,
19             context
20         )
21
22     if not success:
23         return CompileResult(
24             status=CompileStatus.FAILED,
25             error_code="PRE_BUILD_EVENT_FAILED",
26             error_message=error_msg,
27             duration=int((time.time() - start_time) * 1000),
28             log=output or ""
29         )
30
31     # 5. 构建 MSBuild 参数
32     args = []
33
34     # 项目文件
35     args.append(dproj_path)
36
37     # 目标平台
38     args.append(f"/p:Platform={platform}")
39
40     # 配置(Debug/Release)
41     args.append(f"/p:Config={config}")
42
43     # 输出路径
44     if request.options.output_path:
45         args.append(f"/p:DCC_ExeOutput={request.options.output_path}")
46
47     # 条件编译符号
48     if request.options.conditional_defines:
49         defines = ";".join(request.options.conditional_defines)
50         args.append(f"/p:DCC_Define={defines}")
51
52     # 单元搜索路径
53     if request.options.unit_search_paths:
54         paths = ";".join(request.options.unit_search_paths)
55         args.append(f"/p:DCC_UnitSearchPath={paths}")
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 991-1045)

```
1
2     # 其他参数
3     args.append("/v:minimal") # 最小输出级别
4
5     # 记录完整编译参数到日志
6     msbuild_cmd = f'msbuild {" ".join(args)}'
7     logger.info(f"MSBuild 参数: {msbuild_cmd}")
8
9     # 5. 获取 rsvars.bat 路径
10    rsvars_path = self._get_rsvars_path()
11    if not rsvars_path:
12        error_msg = "无法找到 rsvars.bat, 请检查 Delphi 安装"
13        logger.error(error_msg)
14        return CompileResult(
15            status=CompileStatus.FAILED,
16            error_code="RSVARS_NOT_FOUND",
17            error_message=error_msg,
18            duration=int((time.time() - start_time) * 1000)
19        )
20
21    # 6. 创建临时批处理文件来设置环境并执行 MSBuild
22    import tempfile
23    # 将参数中的路径用引号包裹 (处理空格)
24    quoted_args = []
25    for arg in args:
26        if ' ' in arg:
27            quoted_args.append('"%s"' % arg)
28        else:
29            quoted_args.append(arg)
30    # 使用控制台编码 (可能为 UTF-8 或 GBK), 确保 cmd.exe 正确读取中文字符
31    batch_encoding = get_console_encoding()
32    with tempfile.NamedTemporaryFile(mode='w', suffix='.bat', delete=False,
33                                    encoding=batch_encoding) as f:
34        f.write('@echo off\n')
35        f.write('call "%s"\n' % rsvars_path)
36        f.write('msbuild %s\n' % ' '.join(quoted_args))
37        batch_file = f.name
38
39    logger.info("创建批处理文件: %s", batch_file)
40    logger.debug("=== 完整编译命令 ===\n"
41                "rsp: %s\n%s" % (rsvars_path, msbuild_cmd))
42
43    # 6. 执行批处理文件
44    try:
45        return_code, stdout, stderr = await self.process_manager.execute(
46            'cmd.exe',
47            ['/c', batch_file],
48            request.options.timeout
49        )
50
51    # 删除临时文件
52    try:
53        os.unlink(batch_file)
54    except OSError:
55        pass
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1046-1100)

```
1
2     # 记录 MSBuild 输出中的编译器版本/行数信息
3     for line in (stdout or '').split('\n'):
4         line = line.strip()
5         if line and ('Embarcadero Delphi' in line or 'dcc' in line.lower() or 'lines' in line.lower()):
6             logger.debug(f"编译器输出: {line}")
7
8     # 6. 解析输出
9     errors = self.output_parser.parse_errors(stdout + stderr)
10    warnings = self.output_parser.parse_warnings(stdout + stderr)
11
12    # 7. 构建结果
13    duration = int((time.time() - start_time) * 1000)
14
15    if return_code == 0:
16        # 编译成功
17        # 执行 PostBuildEvent
18        if build_events.get('post_build'):
19            # 检查执行条件
20            execute_when = build_events.get('post_build_execute_when', 'Always')
21            should_execute = False
22
23            if execute_when == 'Always':
24                should_execute = True
25            elif execute_when == 'TargetOutOfDate':
26                # 目标过期时执行(编译成功意味着目标已更新)
27                should_execute = True
28
29            if should_execute:
30                success, error_msg, output = self._execute_build_event(
31                    "PostBuildEvent",
32                    build_events['post_build'],
33                    project_dir,
34                    build_events.get('post_build_ignore_exit_code', False),
35                    60,
36                    context
37                )
38
39                if not success:
40                    logger.warning(f"PostBuildEvent 失败,但编译已成功: {error_msg}")
41
42                output_file = self._extract_output_file(stdout, request.options.output_path)
43                output_files = self._collect_output_files(
44                    dproj_path,
45                    request.options.target_platform.value,
46                    config,
47                    output_file if (output_file and Path(output_file).parent.exists()) else None,
48                )
49                # 若 _extract_output_file 没找到 (MSBuild 输出无 Output: 行), 从 output_files 回填
50                if not output_file and output_files:
51                    output_file = output_files[0]
52                result = CompileResult(
53                    status=CompileStatus.SUCCESS,
54                    output_file=output_file,
55                    output_files=output_files,
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1101-1155)

```
1         warnings=warnings,
2         errors=errors,
3         duration=duration,
4         log=stdout + stderr
5     )
6     logger.info(f"MSBuild 编译成功,耗时 {duration}ms")
7     else:
8         # 编译失败
9         result = CompileResult(
10            status=CompileStatus.FAILED,
11            error_code="COMPILATION_FAILED",
12            error_message=self.output_parser.extract_error_summary(stdout + stderr),
13            warnings=warnings,
14            errors=errors,
15            duration=duration,
16            log=stdout + stderr
17        )
18        logger.error(f"MSBuild 编译失败,耗时 {duration}ms")
19
20        # 8. 保存编译历史
21        self._save_history(request.project_path, result.status.value, duration, result.error_message)
22
23        return result
24
25    except TimeoutError as e:
26        duration = int((time.time() - start_time) * 1000)
27        logger.error(f"MSBuild 编译超时: {str(e)}")
28        result = CompileResult(
29            status=CompileStatus.TIMEOUT,
30            error_code="COMPILATION_TIMEOUT",
31            error_message=str(e),
32            duration=duration
33        )
34        self._save_history(request.project_path, result.status.value, duration, str(e))
35        return result
36
37    except Exception as e:
38        duration = int((time.time() - start_time) * 1000)
39        error_msg = f"MSBuild 编译过程发生异常: {str(e)}"
40        logger.error(error_msg, exc_info=True)
41        result = CompileResult(
42            status=CompileStatus.FAILED,
43            error_code="INTERNAL_ERROR",
44            error_message=error_msg,
45            duration=duration
46        )
47        self._save_history(request.project_path, result.status.value, duration, error_msg)
48        return result
49
50    async def compile_project(self, request: ProjectCompileRequest) -> CompileResult:
51        """
52        编译 Delphi 工程
53
54        优先使用 MSBuild 编译,如果 MSBuild 不可用或没有 .dproj 文件则回退到直接调用编译器
55        """
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1156-1210)

```
1      Args:
2          request: 工程编译请求
3
4      Returns:
5          编译结果
6      """
7      logger.info(f"开始编译工程: {request.project_path}")
8
9      # 检查项目类型 (.dpr/.dpc 可以直编, .dproj 只能 MSBuild)
10     ext = Path(request.project_path).suffix.lower()
11     can_compile_direct = ext in ('.dpr', '.dpc')
12     is_source_file = ext in ('.dpr', '.dpc')
13
14     # 检查是否存在 .dproj 文件
15     dproj_exists = False
16     if is_source_file:
17         dproj_path = request.project_path[:-len(ext)] + '.dproj'
18         dproj_exists = Path(dproj_path).exists()
19         if not dproj_exists:
20             logger.info(f"未找到 .dproj 文件, 将使用 dcc32 直接编译{ext} 文件")
21
22     # 如果是源码文件且没有 .dproj 文件, 或者 MSBuild 不可用, 使用直接编译
23     if (is_source_file and not dproj_exists) or not self.msbuild_path:
24         if is_source_file and not dproj_exists:
25             logger.info(f"使用 dcc32 直接编译{ext} 文件")
26         else:
27             logger.info("MSBuild 不可用, 使用直接编译器调用")
28         return await self.compile_dpr_direct(request)
29
30     # 优先使用 MSBuild 编译
31     logger.info("使用 MSBuild 编译")
32     return await self.compile_project_with_msbuild(request)
33
34     async def compile_file(self, request: FileCompileRequest) -> CompileResult:
35         """
36         编译单个 Delphi 单元文件(仅语法检查)
37
38         Args:
39             request: 单文件编译请求
40
41         Returns:
42             编译结果
43         """
44         logger.info(f"开始编译文件: {request.file_path}")
45         start_time = time.time()
46
47         try:
48             # 1. 验证文件路径
49             is_valid, error_msg = self.validator.validate_file_path(request.file_path)
50             if not is_valid:
51                 logger.error(f"文件路径验证失败: {error_msg}")
52                 return CompileResult(
53                     status=CompileStatus.FAILED,
54                     error_code="INVALID_FILE_PATH",
55                     error_message=error_msg,
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1211-1265)

```

1         duration=int((time.time() - start_time) * 1000)
2     )
3
4     # 2. 获取编译器配置 (优先使用传入版本, 默认用最新安装的)
5     compiler_config = (
6         self.config_manager.get_compiler(request.compiler_version)
7         if request.compiler_version
8         else self.config_manager.get_newest_compiler()
9     )
10    if not compiler_config:
11        error_msg = "未找到可用的编译器"
12        logger.error(error_msg)
13        return CompileResult(
14            status=CompileStatus.FAILED,
15            error_code="COMPILER_NOT_FOUND",
16            error_message=error_msg,
17            duration=int((time.time() - start_time) * 1000)
18        )
19
20    # 3. 查找项目文件并提取配置
21    file_dir = str(Path(request.file_path).parent)
22    file_name = Path(request.file_path).stem
23
24    # 尝试查找项目文件
25    dproj_path = None
26    project_unit_paths = []
27    project_include_paths = []
28    project_namespaces = []
29
30    # 方法1: 查找同目录下的.dproj文件
31    dproj_files = list(Path(file_dir).glob('*.*dproj'))
32    if dproj_files:
33        # 优先选择与文件名匹配的项目, 否则选择第一个
34        for dproj in dproj_files:
35            if dproj.stem == file_name:
36                dproj_path = str(dproj)
37                break
38        if not dproj_path:
39            dproj_path = str(dproj_files[0])
40
41    # 方法2: 检查文件是否在项目的DCCReference中
42    if dproj_path:
43        try:
44            parser = DprojParser(dproj_path)
45            if parser.parse():
46                # 检查文件是否属于项目
47                if parser.is_file_in_project(Path(request.file_path).name):
48                    logger.info(f"文件 {Path(request.file_path).name} 属于项目 {Path(dproj_path).name}")
49
50                # 提取单元搜索路径
51                project_unit_paths = parser.get_unit_search_paths()
52                logger.info(f"从项目文件中提取了 {len(project_unit_paths)} 个单元搜索路径")
53
54                # 提取include路径 (从单元搜索路径中推断)
55                for path in project_unit_paths:

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1266-1320)

```

1         if 'include' in path.lower():
2             project_include_paths.append(path)
3
4             # 提取命名空间
5             project_namespaces = parser.get_namespace()
6             if project_namespaces:
7                 logger.info(f"从项目文件中提取了 {len(project_namespaces)} 个命名空间")
8             else:
9                 logger.info(f"文件 {Path(request.file_path).name} 不属于项目 {Path(dproj_path).name}")
10                dproj_path = None
11            except Exception as e:
12                logger.warning(f"解析项目文件失败: {e}")
13                dproj_path = None
14
15            # 4. 删除源文件所在目录的旧版 .dcu 文件
16            self._cleanup_dcu_files(request.file_path)
17
18            # 5. 准备命名空间和include路径
19            # 合并项目命名空间和默认命名空间
20            default_namespaces = [
21                "System", "Winapi", "System.Win", "Vcl", "Vcl.Imaging",
22                "Vcl.Touch", "Vcl.Samples", "Vcl.Shell", "Data", "Datanap",
23                "Web", "Soap", "Xml"
24            ]
25            if project_namespaces:
26                # 合并并去重
27                namespaces = list(dict.fromkeys(project_namespaces + default_namespaces))
28            else:
29                namespaces = default_namespaces
30
31            # 合并include路径
32            default_include_paths = [
33                str(Path(file_dir) / "Thirdpart" / "Jedi" / "Jcl" / "source" / "include"),
34                str(Path(file_dir) / "Thirdpart" / "Jedi" / "Jcl" / "source" / "include" / "jedi")
35            ]
36            if project_include_paths:
37                include_paths = list(dict.fromkeys(project_include_paths + default_include_paths))
38            else:
39                include_paths = default_include_paths
40
41            # 合并单元搜索路径
42            if project_unit_paths and request.unit_search_paths:
43                # 用户提供的路径优先
44                all_unit_paths = list(dict.fromkeys(request.unit_search_paths + project_unit_paths))
45                logger.info(f"合并单元搜索路径: 用户 {len(request.unit_search_paths)} 个 + 项目 {len(project_unit_paths)} 个")
46            elif project_unit_paths:
47                all_unit_paths = project_unit_paths
48            else:
49                all_unit_paths = request.unit_search_paths or []
50
51            # 输出目录 - 使用固定的绝对路径
52            output_dir = str(Path(file_dir) / "Win32" / "Debug")
53
54            # 确保输出目录存在
55            Path(output_dir).mkdir(parents=True, exist_ok=True)

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1321-1375)

```
1
2     # 5. 从编译器配置获取 Delphi 数值版本号, 用于标准库路径
3     delphi_version = compiler_config.registry_version
4     if not delphi_version:
5         # 注册表不可用时 (如手动配置的编译器), 从编译器 --version 输出解析
6         from ..utils.delphi_versions import detect_registry_version_from_compiler
7         detected = detect_registry_version_from_compiler(compiler_config.path)
8         if detected:
9             delphi_version = detected
10            logger.info(f"从编译器输出检测到版本: {delphi_version}")
11        else:
12            delphi_version = "22.0" # 最终回退
13
14        args = self.args_generator.generate_for_file(
15            request.file_path,
16            all_unit_paths,
17            request.warning_level,
18            request.disabled_warnings,
19            namespaces=namespaces,
20            include_paths=include_paths if include_paths else None,
21            output_dir=output_dir,
22            delphi_version=delphi_version,
23            conditional_defines=request.conditional_defines,
24        )
25
26        # 5. 执行编译
27        return_code, stdout, stderr = await self.process_manager.execute(
28            compiler_config.path,
29            args,
30            timeout=60 # 单文件编译超时时间较短
31        )
32
33        # 5. 解析输出
34        errors = self.output_parser.parse_errors(stdout + stderr)
35        warnings = self.output_parser.parse_warnings(stdout + stderr)
36
37        # 6. 构建结果
38        duration = int((time.time() - start_time) * 1000)
39
40        if return_code == 0:
41            result = CompileResult(
42                status=CompileStatus.SUCCESS,
43                warnings=warnings,
44                errors=errors,
45                duration=duration,
46                log=stdout + stderr
47            )
48            logger.info(f"文件编译成功,耗时 {duration}ms")
49        else:
50            result = CompileResult(
51                status=CompileStatus.FAILED,
52                error_code="SYNTAX_ERROR",
53                error_message=self.output_parser.extract_error_summary(stdout + stderr),
54                warnings=warnings,
55                errors=errors,
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1376-1430)

```
1         duration=duration,
2         log=stdout + stderr
3     )
4     logger.error(f"文件编译失败,耗时 {duration}ms")
5
6     return result
7
8     except Exception as e:
9         duration = int((time.time() - start_time) * 1000)
10        error_msg = f"文件编译过程发生异常: {str(e)}"
11        logger.error(error_msg, exc_info=True)
12        return CompileResult(
13            status=CompileStatus.FAILED,
14            error_code="INTERNAL_ERROR",
15            error_message=error_msg,
16            duration=duration
17        )
18
19    def get_args(self, request: ProjectCompileRequest) -> CommandArgs:
20        """
21        获取命令行参数(不执行编译)
22
23        Args:
24            request: 工程编译请求
25
26        Returns:
27            命令行参数对象
28        """
29        logger.info(f"生成命令行参数: {request.project_path}")
30
31        # 获取编译器配置
32        compiler_config = self.config_manager.get_compiler(request.options.compiler_version)
33        if not compiler_config:
34            raise ValueError(f"编译器配置不存在: {request.options.compiler_version or '默认编译器'}")
35
36        # 根据目标平台选择编译器
37        compiler_path = compiler_config.path
38        if request.options.target_platform == TargetPlatform.WIN64:
39            if 'dcc32.exe' in compiler_path:
40                compiler_path = compiler_path.replace('dcc32.exe', 'dcc64.exe')
41
42        # 生成参数
43        args = self.args_generator.generate(request.project_path, request.options)
44
45        # 生成完整命令
46        full_command = self.args_generator.format_command(compiler_path, args)
47
48        # 收集警告
49        warnings = []
50        if request.options.output_path:
51            is_valid, error_msg = self.validator.validate_output_path(request.options.output_path)
52            if not is_valid:
53                warnings.append(f"输出路径警告: {error_msg}")
54
55        return CommandArgs(
```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1431-1485)

```

1         compiler_executable=compiler_path,
2         project_file=request.project_path,
3         arguments=args,
4         full_command=full_command,
5         warnings=warnings
6     )
7
8     @staticmethod
9     def _collect_output_files(
10         project_path: str,
11         target_platform: str = "win32",
12         build_configuration: str = "Debug",
13         output_path_override: Optional[str] = None,
14     ) -> List[str]:
15         """扫描编译产物目录，收集所有生成的文件路径（exe/dll/bpl/bpi/dcp 等）
16
17         Args:
18             project_path: .dproj/.dpr/.dpc 文件路径
19             target_platform: 目标平台（win32/win64 等）
20             build_configuration: 编译配置（Debug/Release）
21             output_path_override: 显式指定的输出路径（若提供则优先使用）
22
23         Returns:
24             所有存在的输出文件路径列表
25         """
26         proj = Path(project_path)
27         project_name = proj.stem
28         project_dir = proj.parent
29
30         # 确定候选输出目录
31         plat_map = {"win32": "Win32", "win64": "Win64"}
32         plat_dir = plat_map.get(target_platform.lower(), "Win32")
33         cfg = build_configuration or "Debug"
34
35         dproj_path = proj.with_suffix('.dproj') if proj.suffix.lower() != '.dproj' else proj
36         dproj_output = None
37         if dproj_path.exists():
38             try:
39                 from ..utils.dproj_parser import DprojParser
40                 parser = DprojParser(str(dproj_path))
41                 if parser.parse():
42                     dproj_output = parser.get_output_path(config=cfg, platform=plat_dir)
43             except Exception as e:
44                 logger.debug("DProj 解析失败（继续用候选目录列表）: %s", e)
45
46         # 收集所有可能输出目录（去重、按可信度排序）
47         candidate_dirs: List[Path] = []
48         if output_path_override:
49             candidate_dirs.append(Path(output_path_override))
50         if dproj_output:
51             candidate_dirs.append(Path(dproj_output))
52         candidate_dirs.append(project_dir / plat_dir / cfg) # Win32/Debug
53         candidate_dirs.append(project_dir) # 项目根目录（MSBuild 缺省输出）
54
55         # 输出产物列表（daudit 需要 .map 做堆栈解析，.drc 不需要）

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1486-1540)

```

1     extensions = ['.exe', '.dll', '.bpl', '.bpi', '.dcp', '.map']
2     found: List[str] = []
3
4     for output_dir in candidate_dirs:
5         if not output_dir.exists():
6             continue
7         for ext in extensions:
8             candidate = output_dir / f"{project_name}{ext}"
9             if candidate.exists() and str(candidate.resolve()) not in found:
10                found.append(str(candidate.resolve()))
11            # 也通配查找以项目名开头的 .bpl/.bpi/.dcp (平台后缀变体)
12            if not any(f.endswith(('.bpl', '.bpi', '.dcp')) for f in found):
13                for ext in ('.bpl', '.bpi', '.dcp'):
14                    for p in output_dir.glob(f"{project_name}*{ext}"):
15                        if p.exists() and str(p.resolve()) not in found:
16                            found.append(str(p.resolve()))
17
18            # 对于 .dpk 包编译, bpl 可能输出到全局 Bpl 目录
19            if proj.suffix.lower() == '.dpk' or dproj_path.suffix.lower() == '.dpk':
20                bpl_search_dirs = [
21                    Path.home() / "Documents" / "Embarcadero" / "Studio",
22                    Path(r"C:\Users\Public\Documents\Embarcadero\Studio"),
23                ]
24                for bpl_base in bpl_search_dirs:
25                    if bpl_base.exists():
26                        for ver_dir in sorted(bpl_base.iterdir(), reverse=True):
27                            bpl_dir = ver_dir / "Bpl"
28                            if bpl_dir.exists():
29                                for ext in ('.bpl', '.bpi', '.dcp'):
30                                    for p in bpl_dir.glob(f"{project_name}*{ext}"):
31                                        if p.exists() and str(p.resolve()) not in found:
32                                            found.append(str(p.resolve()))
33                                    break # 只检查最新版本
34
35            return found
36
37    def _extract_output_file(self, output: str, output_path: Optional[str]) -> Optional[str]:
38        """从输出中提取输出文件路径"""
39        if output_path:
40            return output_path
41        # 尝试从 dcc32 输出行解析: "Output: C:\path\to\output.exe"
42        for line in output.splitlines():
43            if 'Output:' in line and '.exe' in line:
44                idx = line.index('Output:') + 7
45                path = line[idx:].strip().rstrip('.')
46                if os.path.exists(path):
47                    return path
48        return None
49
50    def _save_history(self, project_path: str, status: str, duration: int, error_message: Optional[str]):
51        """保存编译历史"""
52        entry = CompileHistoryEntry(
53            timestamp=datetime.now(),
54            project_path=project_path,
55            status=status,

```

吉林省左右软件开发有限公司 版权所有

文件: services/compiler\_service.py (原始行号 1541-1544)

```
1         duration=duration,  
2         error_message=error_message  
3     )  
4     self.config_manager.add_history_entry(entry)
```

吉林省左右软件开发有限公司 版权所有

文件: services/config\_manager.py (原始行号 1-55)

```

1  """
2  配置管理器
3
4  版权所有 (C) 吉林省左右软件开发有限公司
5  Copyright (C) Equilibrium Software Development Co., Ltd, Jilin
6  Update & Mod By Crystalxp (黑夜杀手 QQ:281309196)
7
8  负责编译器配置和编译历史的读写
9  """
10
11 import json
12 import os
13 import winreg
14 from pathlib import Path
15 from typing import Optional, List
16 from datetime import datetime
17 from ..models.compiler_config import CompilerConfig, ConfigFile
18 from ..models.compile_history import CompileHistoryEntry, HistoryFile
19 from ..utils.delphi_versions import PROJECT_VERSION_PREFIX_MAP
20 from ..utils.logger import get_logger
21
22 logger = get_logger(__name__)
23
24
25 class ConfigManager:
26     """配置管理器"""
27
28     def __init__(self, config_path: Optional[str] = None, history_path: Optional[str] = None):
29         """
30         初始化配置管理器
31
32         Args:
33             config_path: 编译器配置文件路径, 默认从 src/config/ 或项目根 config/ 自动查找
34                         (两个候选位置都支持, 按存在性自动选择并输出提示)
35             history_path: 编译历史文件路径, 默认与 config_path 同目录下的 history.json
36         """
37         if config_path is None:
38             # 自愈路径: AGENTS.md 文档与历史部署位置不一致
39             # 候选: (1) src/config/compiler.json (2) <项目根>/config/compiler.json
40             # 选择第一个存在的, 都不存在时回退到 (1) 以便后续 _auto_detect_compilers 写入
41             _default_root = Path(__file__).parent.parent # src/
42             _candidates = [
43                 _default_root / "config" / "compilers.json", # src/config/ (历史内置)
44                 _default_root.parent / "config" / "compilers.json", # 项目根 config/ (AGENTS.md 描述)
45             ]
46             config_path = str(_candidates[0]) # 默认: src/config/
47             for _candidate in _candidates:
48                 if _candidate.exists():
49                     config_path = str(_candidate)
50                 if _candidate != _candidates[0]:
51                     # 仅在切换到非默认位置时输出提示 (首次启动/迁移后)
52                     logger.info(
53                         "compilers.json 默认路径 (%s) 不存在, 已自愈切换到: %s",
54                         _candidates[0], config_path,
55                     )

```

吉林省左右软件开发有限公司 版权所有