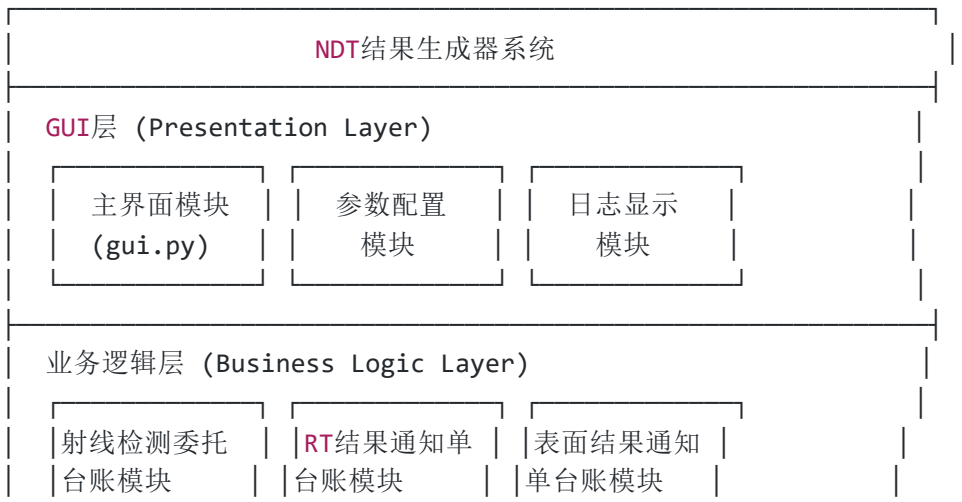
NDT结果生成器系统详细设计文档

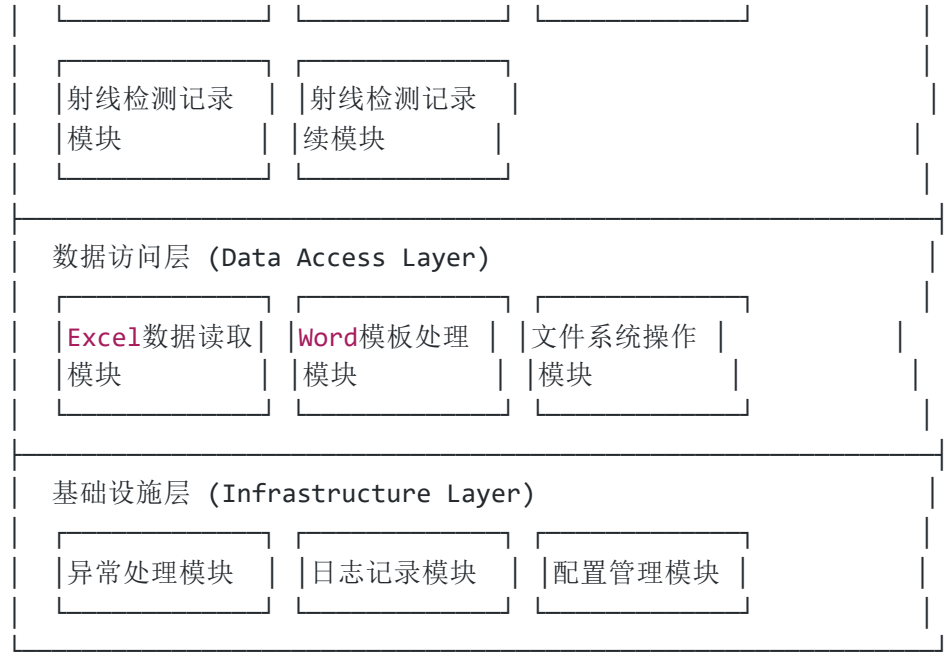
文档信息

- **项目名称**: NDT结果生成器 (NDT Result Generator)
- **版本**: v1.0
- **文档版本**: v1.0
- **编写日期**: 2024-06-24
- **文档状态**: 正式版

1. 系统架构设计

1.1 总体架构





1.2 模块依赖关系

```
gui.py (主界面)
├── Ray_Detection.py (射线检测委托台账-Mode2)
├── Ray_Detection_mode1.py (射线检测委托台账-Mode1)
├── NDT_result.py (RT结果通知单台账-Mode2)
├── NDT_result_mode1.py (RT结果通知单台账-Mode1)
├── Surface_Defect.py (表面结果通知单台账-Mode2)
├── Surface_Defect_mode1.py (表面结果通知单台账-Mode1)
├── Radio_test.py (射线检测记录)
└── Radio_test_renewal.py (射线检测记录续)

共同依赖:
├── pandas (数据处理)
├── openpyxl (Excel操作)
```

```
├── python-docx (Word操作)
└── tkinter (GUI框架)
```

2. 核心模块设计

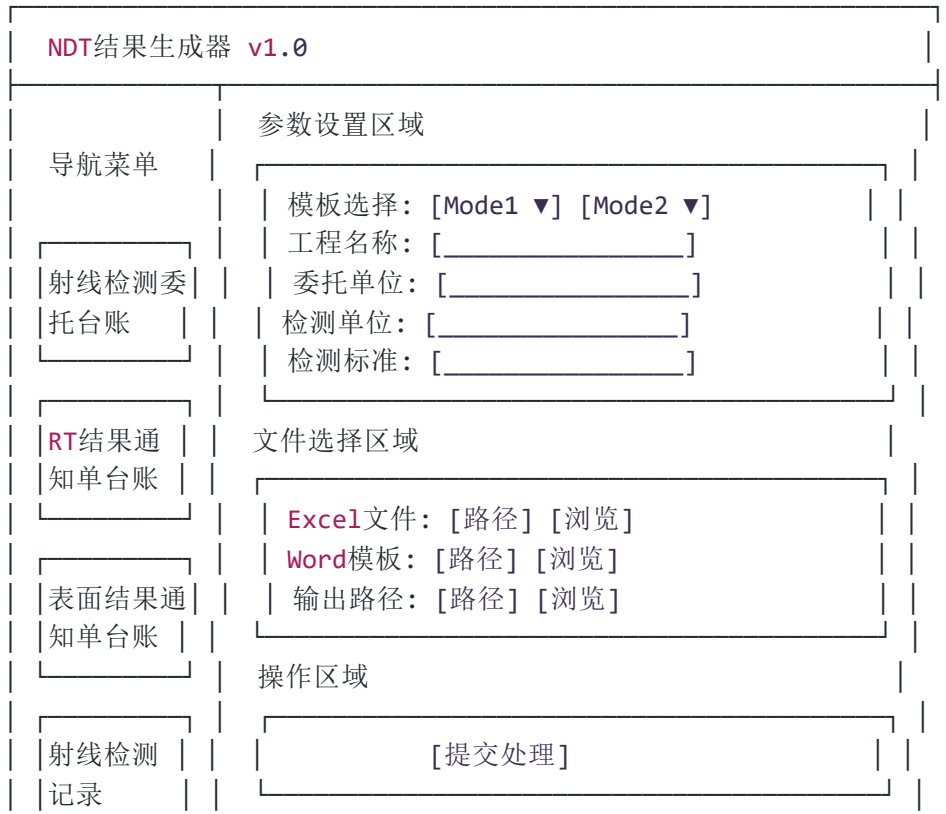
2.1 GUI主界面模块 (gui.py)

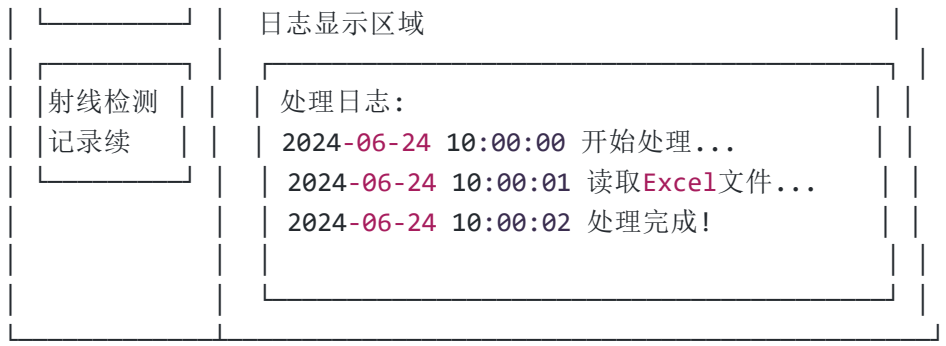
2.1.1 类结构设计

```
class NDTReportGeneratorGUI:
    """NDT结果生成器主界面类"""

    def __init__(self):
        """初始化GUI界面"""

    # 界面创建方法
    def create_main_interface(self)
    def create_left_navigation(self)
    def create_right_content_area(self)

    # 模块界面创建方法
    def create_ray_detection_interface(self)
    def create_ndt_result_interface(self)
    def create_surface_defect_interface(self)
    def create_radio_test_interface(self)
    def create_radio_test_renewal_interface(self)

    # 事件处理方法
    def on_template_change(self, module_name)
    def on_submit_click(self, module_name)
    def on_file_browse(self, entry_widget, file_type)

    # 数据处理方法
```

```
    def process_ray_data(self)
    def process_ndt_data(self)
    def process_surface_data(self)
    def process_radio_data(self)
    def process_radio_renewal_data(self)

    # 工具方法
    def show_log(self, module_name, message)
    def validate_parameters(self, module_name)
    def update_template_path(self, module_name, template_mode)
```

2.1.2 界面布局设计





2.2 数据处理模块设计

2.2.1 通用数据处理流程

```
def process_excel_to_word(excel_path, word_template_path, output_path,
                          project_name, client_name, *args):
    """通用的Excel到Word处理流程"""

    # 1. 数据读取阶段
    df = read_excel_data(excel_path)
    validate_data_format(df)

    # 2. 数据预处理阶段
    df_cleaned = clean_data(df)
    column_mapping = map_columns(df_cleaned)

    # 3. 数据分组阶段
    grouped_data = group_by_order_number(df_cleaned)

    # 4. 文档生成阶段
    for order_number, order_data in grouped_data:
        doc = load_word_template(word_template_path)
        fill_document_data(doc, order_data, column_mapping)
        replace_parameters(doc, project_name, client_name, *args)
```

```
            save_document(doc, output_path, order_number)

        return True
```

2.2.2 检测级别值映射模块

```
    def get_detection_level_by_method(detection_method):
        """检测方法与检测级别值映射"""

        method_mapping = {
            '硬度检测': '力学    级',
            'YD': '力学    级',
            '光谱检测': '光谱分析   级',
            'PMIN': '光谱分析   级',
            'UT': 'UT   级',
            'PT': 'PT   级',
            'MT': 'MT   级',
            'RT': 'RT   级',
            'TOFD': 'TOFD   级',
            'PA': 'PA   级'
        }

        method = str(detection_method).strip().upper()

        # 精确匹配
        if method in method_mapping:
            return method_mapping[method]

        # 模糊匹配
        for key, value in method_mapping.items():
            if key in method:
                return value

        return ""
```

3. 数据库设计

3.1 Excel数据结构

3.1.1 射线检测委托台账数据结构

列号	列名	数据类型	说明	示例
A	委托日期	Date	委托日期	2024-06-20
B	完成日期	Date	完成日期	2024-06-21
C	委托单编号	String	唯一标识	WTS-2024-001
D	检件编号	String	检件编号	JJ-001
E	焊口编号	String	焊口编号	HK-001
F	焊工号	String	焊工编号	HG-001
G	规格	String	管道规格	DN100
H	材质	String	材质信息	20#
I	合格级别	String	合格级别	II级
J	检测比例	String	检测比例	100%

3.1.2 RT结果通知单台账数据结构

列号	列名	数据类型	说明	示例
B	完成日期	Date	完成日期	2024-06-21

列号	列名	数据类型	说明	示例
C	委托单编号	String	唯一标识	WTS-2024-001
D	检件编号	String	检件编号	JJ-001
E	焊口编号	String	焊口编号	HK-001
F	焊工号	String	焊工编号	HG-001
K	返修补片	String	检测结果	合格
N	检测方法	String	检测方法	RT
O	备注	String	备注信息	无
Q	单元名称	String	单元名称	单元A
W	实际不合格	Integer	不合格数量	0

3.2 Word模板结构

3.2.1 占位符设计

参数占位符:
- 工程名称参数值 / 工程名称值
- 委托单位参数值 / 委托单位值
- 检测单位参数值 / 检测单位值
- 检测方法参数 / 检测方法值
- 检测标准参数值 / 检测标准值
- 检测级别值 (新增智能映射)

数据占位符:
- 委托单编号值
- 单元名称值

- 合格级别值
- 检测比例值
- 日期相关占位符

表格数据:
- 动态表格行数据填充
- 保持表格格式和样式

4. 接口设计

4.1 模块间接口

4.1.1 GUI到业务模块接口

```
# 射线检测委托台账接口
def process_ray_detection_mode1(excel_path, word_path, output_path,
                                project_name, client_name, detection_standard,
                                acceptance_standard, detection_method,
                                detection_level, appearance_check, groove_form):
    """Mode1模式处理接口"""

def process_ray_detection_mode2(excel_path, word_path, output_path,
                                project_name, detection_category,
                                detection_standard, detection_method, groove_form):
    """Mode2模式处理接口"""

# RT结果通知单台账接口
def process_ndt_result_mode1(excel_path, word_path, output_path,
                             project_name, client_name, detection_unit,
                             detection_method, detection_standard):
    """Mode1模式处理接口"""

def process_ndt_result_mode2(excel_path, word_path, output_path,
```

```
                              project_name, client_name, detection_method):
    """Mode2模式处理接口"""
```

4.1.2 数据处理接口

```
# Excel数据读取接口
def read_excel_data(file_path, sheet_name=None):
    """读取Excel数据"""

def validate_excel_format(df, required_columns):
    """验证Excel格式"""

# Word文档处理接口
def load_word_template(template_path):
    """加载Word模板"""

def fill_word_document(doc, data_dict):
    """填充Word文档"""

def save_word_document(doc, output_path, filename):
    """保存Word文档"""
```

4.2 错误处理接口

```
class NDTProcessingError(Exception):
    """NDT处理异常基类"""

class ExcelReadError(NDTProcessingError):
    """Excel读取异常"""

class WordTemplateError(NDTProcessingError):
    """Word模板异常"""
```

```
class DataValidationError(NDTProcessingError):
    """数据验证异常"""

def handle_processing_error(error, context):
    """统一错误处理"""
```

5. 安全设计

5.1 数据安全

- **文件访问控制**: 验证文件路径合法性，防止路径遍历攻击
- **数据验证**: 严格验证输入数据格式和内容
- **临时文件清理**: 及时清理处理过程中的临时文件

5.2 异常处理

- **异常捕获**: 全面的异常捕获和处理机制
- **错误日志**: 详细记录错误信息，便于问题排查
- **用户提示**: 友好的错误提示，不暴露系统内部信息

6. 性能设计

6.1 内存优化

- **分批处理**: 大数据量时采用分批处理策略
- **及时释放**: 及时释放不再使用的对象和资源
- **内存监控**: 监控内存使用情况，防止内存泄漏

6.2 处理效率

- **多线程**: GUI与数据处理分离，避免界面冻结
- **缓存机制**: 缓存常用的模板和配置信息
- **优化算法**: 优化数据处理算法，提高处理速度

7. 部署架构设计

7.1 打包架构

```
PyInstaller打包结构:
├── NDT结果生成器.exe (主程序)
│   ├── Python解释器
│   ├── 核心依赖库
│   │   ├── pandas
│   │   ├── openpyxl
│   │   ├── python-docx
│   │   └── tkinter
│   ├── 业务模块
│   │   ├── gui.py
│   │   ├── Ray_Detection.py
│   │   ├── NDT_result.py
│   │   ├── Surface_Defect.py
│   │   └── 其他模块
│   └── 资源文件
└── 生成器/ (外部模板文件夹)
    ├── Excel/
    ├── word/
    └── wod/
```

7.2 运行时架构

运行时环境：

```
┌─────────────────────────────────────┐
│  Windows操作系统 (Win7+)            │
├─────────────────────────────────────┤
│  .NET Framework / Visual C++        │
├─────────────────────────────────────┤
│  NDT结果生成器.exe                  │
│  ├── 内嵌Python运行时               │
│  ├── GUI界面 (tkinter)              │
│  ├── 数据处理引擎                   │
│  └── 文档生成引擎                   │
├─────────────────────────────────────┤
│  文件系统                           │
│  ├── 输入文件 (.xlsx)               │
│  ├── 模板文件 (.docx)               │
│  └── 输出文件 (.docx)               │
└─────────────────────────────────────┘
```

8. 配置管理设计

8.1 配置文件结构

```
# 默认配置
DEFAULT_CONFIG = {
    "paths": {
        "excel_templates": "生成器/Excel/",
        "word_templates_word": "生成器/word/",
        "word_templates_wod": "生成器/wod/",
        "output_default": "生成器/输出报告/"
    },
    "ui": {
        "window_size": "1200x800",
```

```
            "font_family": "Microsoft YaHei",
            "font_size": 10,
            "theme_color": "#f0f0f0"
        },
        "processing": {
            "max_records_per_batch": 1000,
            "enable_multithreading": True,
            "log_level": "INFO"
        }
    }
```

8.2 模板配置

```
    # 模板配置映射
    TEMPLATE_CONFIG = {
        "ray_detection": {
            "mode1": {
                "template_path": "生成器/word/1_射线检测委托台账_Mode1.docx",
                "parameters": ["工程名称", "委托单位", "检测标准", "验收规范",
                               "检测方法", "检测技术等级", "外观检查", "坡口形式"]
            },
            "mode2": {
                "template_path": "生成器/word/1_射线检测委托台账_Mode2.docx",
                "parameters": ["工程名称", "检测类别号", "检测标准", "检测方法", "坡口形式"]
            }
        },
        "ndt_result": {
            "mode1": {
                "template_path": "生成器/wod/2_RT结果通知台账_Mode1.docx",
                "parameters": ["工程名称", "委托单位", "检测单位", "检测方法", "检测标准"]
            },
            "mode2": {
                "template_path": "生成器/wod/2_RT结果通知台账_Mode2.docx",
                "parameters": ["工程名称", "委托单位", "检测方法"]
```

```
            }
        }
    }
```

9. 日志设计

9.1 日志级别

```
import logging

# 日志级别定义
LOG_LEVELS = {
    "DEBUG": logging.DEBUG,      # 调试信息
    "INFO": logging.INFO,        # 一般信息
    "WARNING": logging.WARNING,  # 警告信息
    "ERROR": logging.ERROR,      # 错误信息
    "CRITICAL": logging.CRITICAL # 严重错误
}

# 日志格式
LOG_FORMAT = "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
```

9.2 日志记录策略

```
class NDTLogger:
    """NDT系统日志记录器"""

    def __init__(self, module_name):
        self.logger = logging.getLogger(module_name)
        self.setup_logger()
```

```
    def setup_logger(self):
        """设置日志记录器"""
        handler = logging.StreamHandler()
        formatter = logging.Formatter(LOG_FORMAT)
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)
        self.logger.setLevel(logging.INFO)

    def log_processing_start(self, file_path):
        """记录处理开始"""
        self.logger.info(f"开始处理文件: {file_path}")

    def log_processing_progress(self, current, total):
        """记录处理进度"""
        self.logger.info(f"处理进度: {current}/{total}")

    def log_processing_complete(self, output_count):
        """记录处理完成"""
        self.logger.info(f"处理完成，生成 {output_count} 个文件")

    def log_error(self, error_msg, exception=None):
        """记录错误信息"""
        if exception:
            self.logger.error(f"{error_msg}: {str(exception)}")
        else:
            self.logger.error(error_msg)
```

10. 测试设计

10.1 单元测试设计

```
import unittest
from unittest.mock import Mock, patch
```

```python
class TestNDTProcessing(unittest.TestCase):
    """NDT处理模块单元测试"""

    def setUp(self):
        """测试初始化"""
        self.test_excel_path = "test_data/test.xlsx"
        self.test_word_path = "test_templates/test.docx"
        self.test_output_path = "test_output/"

    def test_excel_reading(self):
        """测试Excel读取功能"""
        # 测试正常文件读取
        # 测试文件不存在情况
        # 测试文件格式错误情况

    def test_data_validation(self):
        """测试数据验证功能"""
        # 测试必要列存在性验证
        # 测试数据格式验证
        # 测试数据完整性验证

    def test_detection_level_mapping(self):
        """测试检测级别值映射"""
        # 测试8种检测方法的映射
        # 测试大小写不敏感
        # 测试未知方法处理

    def test_word_generation(self):
        """测试Word文档生成"""
        # 测试模板加载
        # 测试数据填充
        # 测试文档保存
```

10.2 集成测试设计

```
class TestNDTIntegration(unittest.TestCase):
    """NDT系统集成测试"""

    def test_end_to_end_processing(self):
        """端到端处理测试"""
        # 测试完整的处理流程
        # 验证输出文件正确性
        # 验证处理日志完整性

    def test_gui_integration(self):
        """GUI集成测试"""
        # 测试界面与业务逻辑集成
        # 测试参数传递正确性
        # 测试错误处理机制

    def test_template_switching(self):
        """模板切换测试"""
        # 测试Mode1/Mode2切换
        # 测试参数动态显示
        # 测试路径自动更新
```

11. 维护设计

11.1 版本管理

```
版本号规则：v主版本.次版本.修订版本
- 主版本：重大功能变更或架构调整
- 次版本：新功能添加或重要改进
- 修订版本：Bug修复或小幅优化

示例：
v1.0.0 - 初始版本
```

v1.1.0 - 添加新的检测类型支持
v1.0.1 - 修复数据处理Bug

11.2 代码维护

```
# 代码注释规范
def process_excel_to_word(excel_path, word_template_path, output_path,
                          project_name, client_name):
    """
    处理Excel数据并生成Word报告

    Args:
        excel_path (str): Excel文件路径
        word_template_path (str): Word模板文件路径
        output_path (str): 输出目录路径
        project_name (str): 工程名称
        client_name (str): 委托单位名称

    Returns:
        bool: 处理成功返回True，失败返回False

    Raises:
        ExcelReadError: Excel文件读取失败
        WordTemplateError: Word模板处理失败
        DataValidationError: 数据验证失败

    Example:
        >>> success = process_excel_to_word(
        ...     "data.xlsx", "template.docx", "output/",
        ...     "测试工程", "测试单位"
        ... )
        >>> print(success)
```

```
        True
    """
```

11.3 文档维护

文档维护策略：
1. 代码变更时同步更新相关文档
2. 定期审查文档的准确性和完整性
3. 维护版本变更日志
4. 保持用户文档的时效性

文档类型：
- 技术设计文档（本文档）
- 用户操作手册
- 系统部署指南
- API接口文档
- 测试报告

文档版本: v1.0 **最后更新**: 2024-06-24 **审核状态**: 已审核 **文档状态**: 正式版