# 总结文档

# 一、BOSS Agent

## 1. 从Redis获取异常数据

使用异步方式从Redis获取所有异常数据，并在获取后删除，避免重复处理。

async def get\_all\_anomalies\_async():  
 anomalies = []  
 async for anomaly\_json in fetch\_and\_delete\_anomaly\_data(redis\_url):  
 # 处理异常数据  
 if isinstance(anomaly\_json, dict) and 'anomalies' in anomaly\_json:  
 anomalies.extend(anomaly\_json['anomalies'])  
 elif isinstance(anomaly\_json, list):  
 anomalies.extend(anomaly\_json)  
 else:  
 anomalies.append(anomaly\_json)  
 return anomalies  
  
all\_anomalies = asyncio.run(get\_all\_anomalies\_async())

## 2. 从数据库获取相关日志

为每个异常收集相关日志（通过异常的时间点去 三个log日志表 找应的时间的 相关日志），并将日志格式化为适合大模型处理的格式。（注：三个日志表：application\_logs、security\_logs、system\_logs）

logs\_by\_anomaly\_raw = collect\_and\_analyze\_logs\_for\_anomalies(all\_anomalies, db\_config)  
logs\_by\_anomaly = {}  
for anomaly\_id, logs\_result in logs\_by\_anomaly\_raw.items():  
 logs\_by\_anomaly[anomaly\_id] = prepare\_logs\_for\_llm(logs\_result)

记录日志收集开始信息，并调用函数批量收集和分析所有异常的相关日志。

将收集到的日志转换为适合大语言模型处理的格式。

## 3. 构建提示词并调用大语言模型进行分析

构建详细的提示词，包含异常数据汇总、每个异常的详细信息和相关日志：开始构建提示词，记录日志，并从异常数据中提取异常列表和统计摘要。

prompt = f"""  
你是麒麟操作系统多智能体智能运维管家，你的任务是分析系统中的异常指标和相关日志，进行根因分析，并提出解决方案。  
  
### 异常指标数据汇总  
总异常数量: {summary.get('total', 0)}  
按类型统计: {json.dumps(summary.get('by\_type', {}), ensure\_ascii=False, indent=2)}  
按严重程度统计: {json.dumps(summary.get('by\_severity', {}), ensure\_ascii=False, indent=2)}  
按设备统计: {json.dumps(summary.get('by\_device', {}), ensure\_ascii=False, indent=2)}  
  
### 详细异常指标数据:  
"""

构建提示词的开头部分，包含角色设定和异常数据汇总信息。

# 添加每个异常的详细信息和相关日志  
 for anomaly in anomalies:  
 anomaly\_id = anomaly.get('id', 'unknown')  
 anomaly\_type = anomaly.get('type', '未知')  
 measurement = anomaly.get('measurement', '未知')  
 severity = anomaly.get('severity', '未知')  
 device = anomaly.get('device', '未知')  
 start\_time = anomaly.get('start\_time', '未知')  
 duration = anomaly.get('duration', 0)  
 complexity = anomaly.get('complexity\_score', 0)  
 impact = anomaly.get('business\_impact', '未知')  
 affected\_services = ', '.join(anomaly.get('affected\_services', []))  
 metrics = anomaly.get('metrics', {})

遍历所有异常，提取每个异常的详细信息。

prompt += f"""  
异常ID: {anomaly\_id}  
类型: {anomaly\_type}  
测量项: {measurement}  
严重程度: {severity}  
设备: {device}  
开始时间: {start\_time}  
持续时间: {duration}秒  
复杂度: {complexity}  
业务影响: {impact}  
受影响服务: {affected\_services}  
  
指标数据:  
{json.dumps(metrics, ensure\_ascii=False, indent=2)}  
  
相关日志:  
"""

将每个异常的详细信息添加到提示词中。

# 添加相关日志  
 logs = logs\_by\_anomaly.get(anomaly\_id, {})  
   
 # 添加ERROR日志 (使用新的日志结构)  
 error\_logs = logs.get('error\_logs', [])  
 if error\_logs:  
 prompt += f"ERROR级别日志 ({len(error\_logs)}条):\n"  
 for log in error\_logs:  
 source = log.get('source', '未知')  
 time = log.get('time', '未知时间')  
 message = log.get('message', '未知消息')  
 prompt += f"- [{source}] {time}: {message}\n"

# 添加WARNING日志  
 warning\_logs = logs.get('warning\_logs', [])  
 if warning\_logs:  
 prompt += f"WARNING级别日志 ({len(warning\_logs)}条):\n"  
 for log in warning\_logs[:5]: # 只添加前5条  
 source = log.get('source', '未知')  
 time = log.get('time', '未知时间')  
 message = log.get('message', '未知消息')  
 prompt += f"- [{source}] {time}: {message}\n"

# 添加INFO日志（如果ERROR和WARNING日志较少）  
 info\_logs = logs.get('info\_logs', [])  
 if info\_logs and not error\_logs and len(warning\_logs) < 2:  
 prompt += f"INFO级别日志 ({len(info\_logs)}条):\n"  
 for log in info\_logs[:5]: # 只添加前5条  
 source = log.get('source', '未知')  
 time = log.get('time', '未知时间')  
 message = log.get('message', '未知消息')  
 prompt += f"- [{source}] {time}: {message}\n"

结构化输出：

# 提示LLM进行分析并给出结果  
 prompt += """  
 根据上述异常指标数据和相关日志，请对系统进行全面分析：  
 1. 识别主要问题和根本原因  
 2. 对每个异常进行详细分析，确定问题类型、根本原因、相关因素和证据  
 3. 提出解决问题的建议，包括立即行动和长期措施  
 4. 分析受影响的服务及其影响程度  
 5. 描述分析过程和推理链  
   
   
 响应格式（严格按照以下JSON格式返回）：  
 {  
 "success": true,  
 "analysis\_result": {  
 "request\_id": "req\_XXXXXXXXXX",  
 "analysis\_timestamp": "ISO格式的当前时间",  
 "overall\_assessment": {  
 "primary\_issue": "主要问题的简短描述",  
 "root\_cause": "根本原因的简短描述",  
 "severity\_level": 0.0-10.0的严重程度分数,  
 "business\_impact": "low|moderate|high|critical",  
 "estimated\_resolution\_time": "预计解决时间"  
 },  
 "anomaly\_analysis": [  
 {  
 "anomaly\_id": "异常的ID",  
 "analysis": {  
 "issue\_type": "问题类型",  
 "root\_cause": "根本原因",  
 "contributing\_factors": ["相关因素1", "相关因素2", ...],  
 "evidence": ["证据1", "证据2", ...],  
 "severity": 0.0-10.0的严重程度分数,  
 }  
 }  
 ],  
 "recommendations": {  
 "immediate\_actions": [  
 {  
 "action": "操作标识",  
 "description": "操作描述",  
 "priority": "high|medium|low",  
 "estimated\_duration": "预计持续时间",  
 "risk\_level": "high|medium|low",  
 "affected\_services": ["受影响服务1", ...]  
 }  
 ],  
 "long\_term\_actions": [  
 {  
 "action": "操作标识",  
 "description": "操作描述",  
 "priority": "high|medium|low",  
 "estimated\_duration": "预计持续时间",  
 "risk\_level": "high|medium|low",  
 "affected\_services": ["受影响服务1", ...]  
 }  
 ]  
 },  
 "affected\_services\_summary": {  
 "primary\_affected": ["主要受影响服务1", ...],  
 "secondary\_affected": ["次要受影响服务1", ...],  
 "impact\_assessment": {  
 "服务名1": "high|medium|low",  
 ...  
 }  
 },  
 "langraph\_workflow": {  
 "nodes\_executed": ["执行的节点1", ...],  
 "reasoning\_chain": ["推理步骤1", ...],  
 "confidence\_factors": {  
 "因素1": 0.0-1.0的分数,  
 ...  
 }  
 }  
 }  
 }  
 仅返回JSON格式的分析结果，不要包含其他文本。  
 """

## 4. 将分析结果发送到后端API

将分析结果通过HTTP POST请求发送到后端API服务。开始异常处理块，定义API端点URL和HTTP请求头。记录发送日志，并通过POST请求将分析结果发送到后端API，设置超时时间为300秒。

url = "http://localhost:8001/api/intelligent/process\_llm\_analysis"  
headers = {  
 "Content-Type": "application/json",  
 "Accept": "application/json"  
}  
  
response = requests.post(url, json=result, headers=headers, timeout=300)

同时获得发送给后端后得到的响应结果;记录响应状态码和响应时间

if response.status\_code == 200:  
 response\_data = response.json()  
 logger.info("✅ 后端处理成功!")  
 logger.info(f"时间戳: {response\_data.get('timestamp', 'N/A')}")  
 logger.info(f"Ansible目录: {response\_data.get('ansible\_directory', 'N/A')}")

如果响应状态码为200（成功），则解析响应数据并记录成功信息。

# 显示处理结果摘要  
 summary = response\_data.get('summary', {})  
 logger.info(f"处理摘要: 总异常={summary.get('total\_anomalies', 0)}, "  
 f"规则引擎处理={summary.get('rule\_engine\_processed', 0)}, "  
 f"LLM处理={summary.get('llm\_processed', 0)}")  
 # 显示详细结果  
 rule\_engine\_results = response\_data.get('rule\_engine\_results', [])  
 llm\_results = response\_data.get('llm\_results', [])  
   
 logger.info(f"规则引擎结果: {len(rule\_engine\_results)} 个")  
 for i, rule\_result in enumerate(rule\_engine\_results, 1):  
 logger.info(f" 结果 {i}: {rule\_result.get('success', False)} - {rule\_result.get('selected\_script', 'N/A')}")  
   
 logger.info(f"LLM分析结果: {len(llm\_results)} 个")  
 for i, llm\_result in enumerate(llm\_results, 1):  
 logger.info(f" 结果 {i}: {llm\_result.get('success', False)} - {llm\_result.get('generated\_script', 'N/A')}")

## 5. 特点

异步处理: 使用 asyncio 异步获取Redis数据，提高效率

模块化设计: 依赖多个工具模块实现不同功能

完善的异常处理: 包含网络、数据库、JSON解析等多层异常处理

详细日志记录: 全流程日志跟踪，便于调试和监控

灵活配置: 通过状态参数传递配置信息

## 6. 总结

BOSS节点模块是智能运维系统的核心分析模块，负责从Redis获取异常数据并从数据库收集相关日志，通过构建结构化提示词调用大语言模型进行根因分析，将分析结果发送到后端API服务，并提供完整的异常处理和日志记录机制。该模块采用异步数据处理、模块化设计和多层异常处理，实现了从数据收集、智能分析到结果处理的完整工作流，是整个系统进行异常诊断和解决方案生成的关键组件。

# 二、MCP

## 1. 置信度模块

## 2. role规则引擎模块

## 3. llm大模型模块

## 4. tools工具模块

# 三、工具到沙盒测试

这个模块实现了一个麒麟操作系统自动化优化测试系统，通过SSH连接到远程服务器，执行性能测试和优化脚本，并返回结果

# 四、工具到麒麟执行