基于 Ollama 和 Shimmy 的本地大模型部署方案

目录

项目概述

项目目标

在本地环境中基于 Ollama 和 Shimmy 部署大语言模型，为 Trae 平台提供 AI 代码生成能力，支持软件开发团队的日常工作。

技术栈

- **模型运行时**: Ollama

- **模型适配层**: Shimmy
- **API 网关**: FastAPI
- **容器化**: Docker (可选)
- **监控**: Python 监控脚本

预期收益

- 提供本地化的 AI 代码生成服务
- 支持多种开源大模型
- 优化 CPU 环境下的模型性能
- 与 Trae 平台无缝集成

环境分析

硬件配置

- **CPU**: Intel Core i7-3770 (4 核 8 线程，3.4GHz)
- **内存**: 32GB RAM
- **GPU**: Radeon R7 200 (1GB, 无 CUDA 支持)
- **存储**: 建议 50GB 以上可用空间
- **网络**: 稳定的互联网连接 (用于模型下载)

软件环境

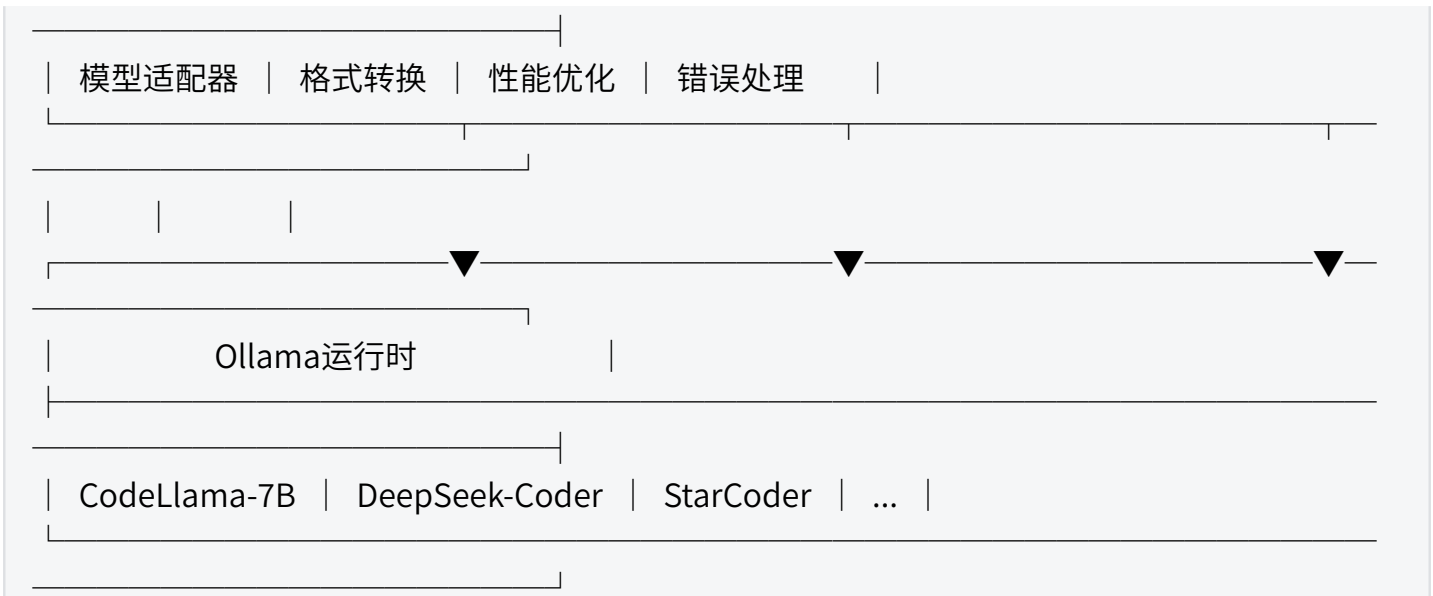
- **操作系统**: Ubuntu 22.04 LTS
- **Python**: 3.9+
- **Docker**: 20.10+ (可选)
- **网络**: 开放必要的端口 (8000, 8080, 11434)

限制与挑战

- **无 GPU 加速**: 只能使用 CPU 进行模型推理
- **老旧 CPU**: 需要优化以获得可接受的性能
- **内存限制**: 32GB 内存限制了可运行模型的大小
- **散热考虑**: 长时间运行可能需要散热措施

系统架构设计

整体架构图



组件说明

1. 客户端应用层

- **Trae IDE**: 主要的开发环境集成
- **Web 界面**: 提供可视化操作界面
- **命令行工具**: 提供快速的命令行访问
- **API 客户端**: 供其他应用程序集成

2. 服务网关层

- **认证授权**: API 密钥验证和权限控制
- **请求路由**: 将请求分发到合适的模型
- **负载均衡**: 在多个模型实例间分配负载
- **缓存机制**: 缓存频繁请求的结果
- **监控统计**: 收集和展示服务指标

3. Shimmy 适配层

- **模型适配**: 统一不同模型的接口
- **格式转换**: 处理输入输出格式转换

- **性能优化**: 应用推理优化技术
- **错误处理**: 统一的错误处理机制

4. Ollama 运行时

- **模型管理**: 模型的下载、加载和卸载
- **推理服务**: 提供模型推理能力
- **资源管理**: 管理 CPU 和内存资源
- **并发控制**: 控制并发请求数量

详细部署步骤

步骤 1: 系统环境准备

1.1 更新系统

```
# 更新系统包列表
sudo apt update
# 升级系统包
sudo apt upgrade -y
# 安装必要的系统工具
sudo apt install -y \
 curl wget git \
 python3 python3-pip python3-venv \
 build-essential \
 net-tools \
 htop \
 screen \
 tmux
```

1.2 配置 Python 环境

```
# 创建Python虚拟环境
python3 -m venv ai-model-env
# 激活虚拟环境
source ai-model-env/bin/activate
# 更新pip
pip install --upgrade pip
# 验证Python版本
python --version
pip --version
```

步骤 2: 安装 Ollama

2.1 下载并安装 Ollama

```
# 下载Ollama安装脚本
curl -fsSL https://ollama.com/install.sh -o install-ollama.sh
# 查看安装脚本内容(可选)
cat install-ollama.sh
# 执行安装脚本
sudo sh install-ollama.sh
# 验证安装
ollama --version
# 启动Ollama服务
sudo systemctl start ollama
# 设置Ollama服务开机自启
sudo systemctl enable ollama
# 检查服务状态
sudo systemctl status ollama
```

2.2 配置 Ollama

```
# 创建Ollama配置目录
sudo mkdir -p /etc/ollama
# 创建Ollama配置文件
sudo tee /etc/ollama/config.yaml <<EOF
# Ollama配置文件
```

```
environment:
# CPU线程数配置
 OLLAMA_NUM_THREADS: "8"
# 内存使用限制
 OLLAMA_MEMORY: "24GB"
# 模型缓存目录
 OLLAMA_MODELS: "/var/lib/ollama/models"
# 日志级别
 OLLAMA_LOG_LEVEL: "info"
# 服务配置
service:
 host: "0.0.0.0"
 port: 11434
EOF
# 重启Ollama服务以应用配置
sudo systemctl restart ollama
# 检查服务状态
sudo systemctl status ollama
```

步骤 3: 下载模型

3.1 推荐模型列表

```
# 查看可用模型
ollama list
# 下载推荐的代码生成模型
echo "正在下载CodeLlama 7B代码模型..."
ollama pull codellama:7b-code
echo "正在下载DeepSeek Coder 6.7B模型..."
ollama pull deepseek-coder:6.7b
echo "正在下载StarCoder 7B模型..."
ollama pull starcoder:7b
echo "正在下载WizardCoder 7B模型..."
ollama pull wizardcoder:7b
# 查看已安装模型
echo "已安装的模型:"
ollama list
```

3.2 创建模型量化版本

```
# 创建CodeLlama 7B 4-bit量化版本
echo "创建CodeLlama 7B 4-bit量化版本..."
ollama create codellama:7b-code-q4 -f - <<EOF
FROM codellama:7b-code
PARAMETER quantize q4_0
PARAMETER num_ctx 2048
PARAMETER num_thread 8
EOF
# 创建DeepSeek Coder 4-bit量化版本
echo "创建DeepSeek Coder 4-bit量化版本..."
ollama create deepseek-coder:6.7b-q4 -f - <<EOF
FROM deepseek-coder:6.7b
PARAMETER quantize q4_0
PARAMETER num_ctx 2048
PARAMETER num_thread 8
EOF
# 查看量化后的模型
ollama list
```

步骤 4: 安装和配置 Shimmy

4.1 安装 Shimmy

```
# 确保虚拟环境已激活
source ai-model-env/bin/activate
# 安装Shimmy和相关依赖
pip install \
 shimmy \
 openai \
 fastapi \
 uvicorn \
 requests \
 python-dotenv \
 pydantic \
 loguru \
```

```
 prometheus-client \
 python-multipart
```

4.2 创建 Shimmy 配置

```
# 创建项目目录结构
mkdir -p ai-model-server/{shimmy,gateway,scripts,config,logs,models}
cd ai-model-server
# 创建Shimmy配置文件
cat > shimmy/config.yaml <<EOF
# Shimmy配置文件
model:
 default: codellama:7b-code-q4
 available:
- name: codellama:7b-code
 type: code
 description: Meta CodeLlama 7B for code generation
 parameters:
 temperature: 0.7
 max_tokens: 2048
 top_p: 0.95

- name: codellama:7b-code-q4
 type: code
 description: Meta CodeLlama 7B (4-bit quantized) for code generation
 parameters:
 temperature: 0.7
 max_tokens: 2048
 top_p: 0.95

- name: deepseek-coder:6.7b
 type: code
 description: DeepSeek Coder 6.7B for code generation
 parameters:
 temperature: 0.7
 max_tokens: 2048
 top_p: 0.95

- name: deepseek-coder:6.7b-q4
```

```
  type: code
  description: DeepSeek Coder 6.7B (4-bit quantized) for code generation
  parameters:
  temperature: 0.7
  max_tokens: 2048
  top_p: 0.95

- name: starcoder:7b
  type: code
  description: StarCoder 7B for code generation
  parameters:
  temperature: 0.7
  max_tokens: 2048
  top_p: 0.95
server:
  host: 0.0.0.0
  port: 8000
  workers: 2
  timeout: 300
ollama:
  base_url: http://localhost:11434/api
  timeout: 300
logging:
  level: INFO
  file: ../logs/shimmy.log
  rotation: 10 MB
  retention: 7 days
cache:
  enabled: true
  ttl: 3600
  max_size: 1000
EOF
```

4.3 创建 Shimmy 服务脚本

```
# 创建Shimmy启动脚本
cat > shimmy/start_shimmy.sh <<EOF
#!/bin/bash
# Shimmy启动脚本
```

```
# 加载环境变量
source ../../ai-model-env/bin/activate
# 设置工作目录
cd \$(dirname \$0)
# 创建日志目录
mkdir -p ../logs
# 启动Shimmy服务
echo "Starting Shimmy service..."
uvicorn shimmy.server:app \
--host 0.0.0.0 \
--port 8000 \
--workers 2 \
--log-config logging.conf \
--timeout-keep-alive 300 \
--reload
EOF
# 设置脚本执行权限
chmod +x shimmy/start_shimmy.sh
# 创建Shimmy日志配置
cat > shimmy/logging.conf <<EOF
[loggers]
keys=root
[handlers]
keys=console,file
[formatters]
keys=default
[logger_root]
level=INFO
handlers=console,file
[handler_console]
class=StreamHandler
formatter=default
args=(sys.stdout,)
[handler_file]
class=FileHandler
formatter=default
args=('../logs/shimmy.log', 'a', 'utf-8')
[formatter_default]
format=%(asctime)s [%(levelname)s] %(name)s: %(message)s
datefmt=%Y-%m-%d %H:%M:%S
EOF
```

步骤 5: 创建 API 网关

5.1 生成安全的 API 密钥

```
# 生成强API密钥
API_KEY=$(openssl rand -hex 32)
echo "生成的API密钥: ${API_KEY}"
echo "请妥善保存此密钥，后续配置需要使用"
```

5.2 创建 FastAPI 网关配置

```
# 创建API网关配置
cat > gateway/config.yaml <<EOF
# API网关配置
server:
 host: 0.0.0.0
 port: 8080
 workers: 4
 timeout: 300
models:
 default: codellama:7b-code-q4
 endpoints:
 codellama:7b-code:
 url: http://localhost:8000/v1/completions
 type: code
 description: Meta CodeLlama 7B for code generation

 codellama:7b-code-q4:
 url: http://localhost:8000/v1/completions
 type: code
 description: Meta CodeLlama 7B (4-bit quantized)

 deepseek-coder:6.7b:
 url: http://localhost:8000/v1/completions
 type: code
 description: DeepSeek Coder 6.7B
```

```
  deepseek-coder:6.7b-q4:
  url: http://localhost:8000/v1/completions
  type: code
  description: DeepSeek Coder 6.7B (4-bit quantized)
security:
  api_key: ${API_KEY}
  rate_limit:
  requests: 100
  period: 60
logging:
  level: INFO
  file: ../logs/gateway.log
  rotation: 10 MB
  retention: 7 days
cache:
  enabled: true
  ttl: 3600
  max_size: 1000
EOF
```

5.3 创建 API 网关主文件

```
# 创建API网关主文件
cat > gateway/main.py <<EOF
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
AI Model API Gateway
基于FastAPI的模型服务网关
"""
from fastapi import FastAPI, HTTPException, Depends, Request
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from pydantic import BaseModel, Field
import requests
import time
import hashlib
import yaml
```

```
import os
from functools import lru_cache
from datetime import datetime, timedelta
from typing import Dict, List, Optional, Any
import logging
from logging.handlers import RotatingFileHandler
# 配置日志
logger = logging.getLogger("api-gateway")
logger.setLevel(logging.INFO)
# 创建日志处理器
file_handler = RotatingFileHandler(
"../logs/gateway.log",
 maxBytes=10*1024*1024,  # 10MB
 backupCount=7,
 encoding="utf-8"
)
console_handler = logging.StreamHandler()
# 设置日志格式
formatter = logging.Formatter(
"%(asctime)s [%(levelname)s] %(name)s: %(message)s",
 datefmt="%Y-%m-%d %H:%M:%S"
)
file_handler.setFormatter(formatter)
console_handler.setFormatter(formatter)
# 添加日志处理器
logger.addHandler(file_handler)
logger.addHandler(console_handler)
# 加载配置
def load_config():
"""加载配置文件"""
 config_path = "config.yaml"
 try:
 with open(config_path, "r", encoding="utf-8") as f:
 config = yaml.safe_load(f)
 logger.info("配置文件加载成功")
 return config
 except Exception as e:
 logger.error(f"配置文件加载失败: {e}")
 raise
config = load_config()
# 创建FastAPI应用
```

```
app = FastAPI(
 title="AI Model API Gateway",
 description="本地大模型服务网关",
 version="1.0.0"
)
# CORS配置
app.add_middleware(
 CORSMiddleware,
 allow_origins=["*"],
 allow_credentials=True,
 allow_methods=["*"],
 allow_headers=["*"],
)
# 全局变量
REQUEST_COUNTER = {}
CACHE = {}
# 数据模型
class ModelRequest(BaseModel):
"""模型请求数据模型"""
 model: str = Field(default=config["models"]["default"], description="模型名称")
 prompt: str = Field(description="输入提示词")
 temperature: float = Field(default=0.7, ge=0.0, le=1.0, description="温度参数")
 max_tokens: int = Field(default=1024, ge=1, le=4096, description="最大生成token数")
 top_p: float = Field(default=0.95, ge=0.0, le=1.0, description="Top P参数")
 stream: bool = Field(default=False, description="是否流式输出")
class ModelResponse(BaseModel):
"""模型响应数据模型"""
 success: bool = Field(description="请求是否成功")
 result: Optional[Dict[str, Any]] = Field(default=None, description="模型返回结果")
 metadata: Dict[str, Any] = Field(description="元数据信息")
# 工具函数
def verify_api_key(request: Request):
"""验证API密钥"""
 api_key = request.headers.get("X-API-Key")
 if not api_key or api_key != config["security"]["api_key"]:
 logger.warning(f"API密钥验证失败: {api_key}")
 raise HTTPException(status_code=401, detail="无效的API密钥")
 return api_key
def rate_limit(request: Request):
"""速率限制"""
 client_ip = request.client.host
```

```
 current_time = time.time()

# 初始化计数器
 if client_ip not in REQUEST_COUNTER:
 REQUEST_COUNTER[client_ip] = {"count": 0, "reset_time": current_time + config["security
"]["rate_limit"]["period"]}

 counter = REQUEST_COUNTER[client_ip]

# 检查是否需要重置计数器
 if current_time > counter["reset_time"]:
 counter["count"] = 0
 counter["reset_time"] = current_time + config["security"]["rate_limit"]["period"]

# 检查速率限制
 if counter["count"] >= config["security"]["rate_limit"]["requests"]:
 reset_time = datetime.fromtimestamp(counter["reset_time"]).strftime("%Y-%m-%d %H:
%M:%S")
 logger.warning(f"速率限制触发: {client_ip}, 当前计数: {counter['count']}")
 raise HTTPException(
 status_code=429,
 detail=f"请求过于频繁，请在 {reset_time} 后重试"
)

 counter["count"] += 1
def generate_cache_key(request: ModelRequest) -> str:
"""生成缓存键"""
 key_data = f"{request.model}:{request.temperature}:{request.max_tokens}:{hashlib.md5(
request.prompt.encode()).hexdigest()}"
 return key_data
def get_cache(key: str) -> Optional[Any]:
"""获取缓存"""
 if not config["cache"]["enabled"]:
 return None

 if key in CACHE:
 cache_entry = CACHE[key]
# 检查缓存是否过期
 if time.time() < cache_entry["expire_time"]:
 logger.debug(f"缓存命中: {key}")
 return cache_entry["data"]
```

```
 else:
 del CACHE[key]
 logger.debug(f"缓存过期: {key}")
 return None
def set_cache(key: str, data: Any):
"""设置缓存"""
 if config["cache"]["enabled"]:
 expire_time = time.time() + config["cache"]["ttl"]
 CACHE[key] = {
"data": data,
"expire_time": expire_time
}
 logger.debug(f"缓存设置: {key}")

# 检查缓存大小
 if len(CACHE) > config["cache"]["max_size"]:
# 删除最旧的缓存
 oldest_key = min(CACHE.keys(), key=lambda k: CACHE[k]["expire_time"])
 del CACHE[oldest_key]
 logger.debug(f"缓存溢出，删除最旧缓存: {oldest_key}")
def get_model_endpoint(model_name: str) -> Dict[str, Any]:
"""获取模型端点配置"""
 if model_name not in config["models"]["endpoints"]:
 logger.error(f"模型不存在: {model_name}")
 raise HTTPException(status_code=404, detail=f"模型 {model_name} 不存在")
 return config["models"]["endpoints"][model_name]
# API路由
@app.get("/health", summary="健康检查")
async def health_check():
"""服务健康检查"""
 try:
# 检查Shimmy服务
 shimmy_health = requests.get("http://localhost:8000/health", timeout=5).json()

# 检查Ollama服务
 ollama_health = requests.get("http://localhost:11434/api/tags", timeout=5).json()

 return {
"status": "healthy",
"timestamp": datetime.now().isoformat(),
"services": {
```

```
"gateway": "running",
"shimmy": shimmy_health.get("status", "unknown"),
"ollama": "running" if "models" in ollama_health else "error"
}
}
 except Exception as e:
 logger.error(f"健康检查失败: {e}")
 return {
"status": "unhealthy",
"timestamp": datetime.now().isoformat(),
"error": str(e)
}
@app.get("/models", summary="列出可用模型")
async def list_models(api_key: str = Depends(verify_api_key)):
"""列出所有可用的模型"""
 models = []
 for model_name, model_config in config["models"]["endpoints"].items():
 models.append({
"name": model_name,
"type": model_config["type"],
"description": model_config["description"]
})

 return {
"models": models,
"default_model": config["models"]["default"],
"total_models": len(models)
}
@app.post("/generate", summary="生成文本", response_model=ModelResponse)
async def generate_text(
 request: ModelRequest,
 api_key: str = Depends(verify_api_key),
 rate_limit_check: None = Depends(rate_limit)
):
"""生成文本内容"""
 start_time = time.time()
 request_id = hashlib.uuid4().hex

 logger.info(f"收到请求: {request_id}, 模型: {request.model}, prompt长度: {len(request.prompt
)}")
```

```
 try:
# 检查缓存
 cache_key = generate_cache_key(request)
 cached_result = get_cache(cache_key)

 if cached_result:
 latency = time.time() - start_time
 logger.info(f"请求完成: {request_id}, 来源: 缓存, 耗时: {latency:.2f}s")
 return {
"success": True,
"result": cached_result,
"metadata": {
"request_id": request_id,
"model": request.model,
"latency": round(latency, 2),
"source": "cache",
"timestamp": datetime.now().isoformat()
}
}

# 获取模型端点
 model_endpoint = get_model_endpoint(request.model)

# 准备请求数据
 payload = {
"model": request.model,
"prompt": request.prompt,
"temperature": request.temperature,
"max_tokens": request.max_tokens,
"top_p": request.top_p,
"stream": request.stream
}

# 发送请求到Shimmy
 response = requests.post(
 model_endpoint["url"],
 json=payload,
 timeout=config["server"]["timeout"]
)

 response.raise_for_status()
```

```
 result = response.json()

# 设置缓存
 set_cache(cache_key, result)

 latency = time.time() - start_time
 logger.info(f"请求完成: {request_id}, 耗时: {latency:.2f}s")

 return {
"success": True,
"result": result,
"metadata": {
"request_id": request_id,
"model": request.model,
"latency": round(latency, 2),
"source": "model",
"timestamp": datetime.now().isoformat()
}
}

 except Exception as e:
 latency = time.time() - start_time
 logger.error(f"请求失败: {request_id}, 错误: {e}, 耗时: {latency:.2f}s")
 return {
"success": False,
"metadata": {
"request_id": request_id,
"model": request.model,
"latency": round(latency, 2),
"error": str(e),
"timestamp": datetime.now().isoformat()
}
}
@app.get("/stats", summary="获取统计信息")
async def get_stats(api_key: str = Depends(verify_api_key)):
"""获取服务统计信息"""
# 获取Ollama模型信息
 try:
 ollama_models = requests.get("http://localhost:11434/api/tags", timeout=5).json()
 except:
 ollama_models = {"models": []}
```

```
 return {
"timestamp": datetime.now().isoformat(),
"active_models": len(ollama_models.get("models", [])),
"cache_size": len(CACHE),
"request_counter": len(REQUEST_COUNTER),
"config": {
"cache_enabled": config["cache"]["enabled"],
"rate_limit": config["security"]["rate_limit"]
}
}
if __name__ == "__main__":
 import uvicorn

 logger.info("启动API网关服务")
 uvicorn.run(
"main:app",
 host=config["server"]["host"],
 port=config["server"]["port"],
 workers=config["server"]["workers"],
 timeout_keep_alive=config["server"]["timeout"],
 log_config=None
)
EOF
# 创建API网关启动脚本
cat > gateway/start_gateway.sh <<EOF
#!/bin/bash
# API网关启动脚本
# 加载环境变量
source ../../ai-model-env/bin/activate
# 设置工作目录
cd \$(dirname \$0)
# 创建日志目录
mkdir -p ../logs
# 启动API网关
echo "Starting API Gateway..."
python main.py
EOF
# 设置脚本执行权限
chmod +x gateway/start_gateway.sh
```

步骤 6: 创建监控和管理脚本

6.1 创建系统监控脚本

```
# 创建监控脚本目录
mkdir -p scripts/monitoring
# 创建系统监控脚本
cat > scripts/monitoring/system_monitor.sh <<EOF
#!/bin/bash
# 系统监控脚本
# 配置
LOG_FILE="../../logs/system_monitor.log"
CHECK_INTERVAL=60  # 检查间隔(秒)
ALERT_THRESHOLDS=(
"cpu:80"      # CPU使用率阈值
"memory:80"   # 内存使用率阈值
"disk:80"     # 磁盘使用率阈值
)
# 创建日志目录
mkdir -p \$(dirname \$LOG_FILE)
# 日志函数
log_message() {
 echo "\$(date '+%Y-%m-%d %H:%M:%S') [\$1] \$2" | tee -a \$LOG_FILE
}
# 检查CPU使用率
check_cpu() {
 local cpu_usage=\$(top -bn1 | grep "Cpu(s)" | awk '{print \$2}' | cut -d'%' -f1 | awk '{printf "%
.1f", \$1}')
 local threshold=\$(echo \${ALERT_THRESHOLDS[0]} | cut -d':' -f2)

 echo "CPU使用率: \${cpu_usage}%"

 if (( \$(echo "\${cpu_usage} > \${threshold}" | bc -l) )); then
 log_message "WARNING" "CPU使用率过高: \${cpu_usage}% (阈值: \${threshold}%)"
 fi
}
# 检查内存使用率
check_memory() {
```

```
 local mem_usage=\$(free | grep Mem | awk '{printf("%.1f", \$3/\$2 * 100.0)}')
 local threshold=\$(echo \${ALERT_THRESHOLDS[1]} | cut -d':' -f2)

 echo "内存使用率: \${mem_usage}%"

 if (( \$(echo "\${mem_usage} > \${threshold}" | bc -l) )); then
 log_message "WARNING" "内存使用率过高: \${mem_usage}% (阈值: \${threshold}%)"
 fi
}
# 检查磁盘使用率
check_disk() {
 local disk_usage=\$(df -h / | tail -1 | awk '{print \$5}' | cut -d'%' -f1)
 local threshold=\$(echo \${ALERT_THRESHOLDS[2]} | cut -d':' -f2)

 echo "磁盘使用率: \${disk_usage}%"

 if [ \${disk_usage} -gt \${threshold} ]; then
 log_message "WARNING" "磁盘使用率过高: \${disk_usage}% (阈值: \${threshold}%)"
 fi
}
# 检查服务状态
check_services() {
 echo "服务状态检查:"

# 检查Ollama服务
 if systemctl is-active --quiet ollama; then
 echo "✓ Ollama服务正在运行"
 else
 echo " Ollama服务未运行"
 log_message "ERROR" "Ollama服务未运行"
 fi

# 检查Shimmy服务
 if netstat -tlnp | grep -q ":8000"; then
 echo "✓ Shimmy服务正在运行"
 else
 echo " Shimmy服务未运行"
 log_message "ERROR" "Shimmy服务未运行"
 fi

# 检查API网关
```

```
 if netstat -tlnp | grep -q ":8080"; then
 echo "✓ API网关正在运行"
 else
 echo " API网关未运行"
 log_message "ERROR" "API网关未运行"
 fi
}
# 检查模型状态
check_models() {
 echo "模型状态检查:"

 if command -v ollama &> /dev/null; then
 local model_count=\$(ollama list | wc -l)
 echo "已安装模型数量: \$((model_count - 1))"  # 减去表头

# 检查默认模型是否存在
 local default_model=\$(grep "default:" ../../shimmy/config.yaml | awk '{print \$2}')
 if ollama list | grep -q "\${default_model}"; then
 echo "✓ 默认模型 \${default_model} 已安装"
 else
 echo " 默认模型 \${default_model} 未安装"
 log_message "ERROR" "默认模型 \${default_model} 未安装"
 fi
 else
 echo " Ollama命令不可用"
 log_message "ERROR" "Ollama命令不可用"
 fi
}
# 健康检查
health_check() {
 echo "健康检查:"

# 检查API网关健康状态
 if command -v curl &> /dev/null; then
 local health_response=\$(curl -s http://localhost:8080/health)
 if echo "\${health_response}" | grep -q "\"status\":\"healthy\""; then
 echo "✓ API网关健康检查通过"
 else
 echo " API网关健康检查失败"
 log_message "ERROR" "API网关健康检查失败: \${health_response}"
 fi
```

```
 else
 echo " curl命令不可用，无法进行健康检查"
 fi
}
# 主函数
main() {
 log_message "INFO" "开始系统监控检查"

 echo "==================== 系统监控报告 ===================="
 echo "检查时间: \$(date '+%Y-%m-%d %H:%M:%S')"
 echo

 check_cpu
 check_memory
 check_disk
 echo

 check_services
 echo

 check_models
 echo

 health_check
 echo

 echo "============================================================"
 echo

 log_message "INFO" "系统监控检查完成"
}
# 运行主函数
main
EOF
# 设置脚本执行权限
chmod +x scripts/monitoring/system_monitor.sh
# 创建定期监控脚本
cat > scripts/monitoring/start_monitoring.sh <<EOF
#!/bin/bash
# 启动监控服务
# 配置
```

```
CHECK_INTERVAL=60  # 检查间隔(秒)
LOG_FILE="../../logs/monitoring_service.log"
# 创建日志目录
mkdir -p \$(dirname \$LOG_FILE)
# 日志函数
log_message() {
 echo "\$(date '+%Y-%m-%d %H:%M:%S') [\$1] \$2" | tee -a \$LOG_FILE
}
# 主函数
main() {
 log_message "INFO" "启动监控服务，检查间隔: \${CHECK_INTERVAL}秒"

 while true; do
# 执行系统监控
./system_monitor.sh

# 等待下一次检查
 sleep \${CHECK_INTERVAL}
 done
}
# 运行主函数
main
EOF
# 设置脚本执行权限
chmod +x scripts/monitoring/start_monitoring.sh
```

6.2 创建系统管理脚本

```
# 创建系统管理脚本
cat > scripts/system_manager.sh <<EOF
#!/bin/bash
# AI模型服务管理脚本
# 配置
PROJECT_DIR=\$(cd \$(dirname \$0)/.. && pwd)
LOG_DIR=\${PROJECT_DIR}/logs
ENV_DIR=\${PROJECT_DIR}/../ai-model-env
# 颜色定义
RED='\033[0;31m'
GREEN='\033[0;32m'
```

```
YELLOW='\033[1;33m'
NC='\033[0m' # No Color
# 日志函数
log_message() {
 echo -e "\$(date '+%Y-%m-%d %H:%M:%S') [\$1] \$2"
}
# 检查环境
check_environment() {
 log_message "INFO" "检查运行环境..."

# 检查Python虚拟环境
 if [ ! -d "\${ENV_DIR}" ]; then
 log_message "ERROR" "Python虚拟环境不存在: \${ENV_DIR}"
 exit 1
 fi

# 检查项目目录
 if [ ! -d "\${PROJECT_DIR}/shimmy" ] || [ ! -d "\${PROJECT_DIR}/gateway" ]; then
 log_message "ERROR" "项目目录结构不完整"
 exit 1
 fi

# 检查Ollama
 if ! command -v ollama &> /dev/null; then
 log_message "ERROR" "Ollama未安装"
 exit 1
 fi

 log_message "INFO" "环境检查完成"
}
# 启动服务
start_services() {
 log_message "INFO" "启动AI模型服务..."

# 检查环境
 check_environment

# 启动Ollama服务
 if ! systemctl is-active --quiet ollama; then
 log_message "INFO" "启动Ollama服务..."
 sudo systemctl start ollama
```

```
# 等待服务启动
 sleep 10

 if systemctl is-active --quiet ollama; then
 log_message "SUCCESS" "Ollama服务启动成功"
 else
 log_message "ERROR" "Ollama服务启动失败"
 exit 1
 fi
 else
 log_message "INFO" "Ollama服务已在运行"
 fi

# 启动Shimmy服务
 if ! netstat -tlnp | grep -q ":8000"; then
 log_message "INFO" "启动Shimmy服务..."

# 创建日志目录
 mkdir -p \${LOG_DIR}

# 在新的screen会话中启动Shimmy
 screen -dmS shimmy_service bash -c "
 cd \${PROJECT_DIR}/shimmy &&
 source \${ENV_DIR}/bin/activate &&
./start_shimmy.sh > \${LOG_DIR}/shimmy_startup.log 2>&1
"

# 等待服务启动
 sleep 15

 if netstat -tlnp | grep -q ":8000"; then
 log_message "SUCCESS" "Shimmy服务启动成功"
 else
 log_message "ERROR" "Shimmy服务启动失败"
 log_message "ERROR" "查看日志: tail -f \${LOG_DIR}/shimmy.log"
 fi
 else
 log_message "INFO" "Shimmy服务已在运行"
 fi
```

```
# 启动API网关
 if ! netstat -tlnp | grep -q ":8080"; then
 log_message "INFO" "启动API网关..."

# 在新的screen会话中启动API网关
 screen -dmS gateway_service bash -c "
 cd \${PROJECT_DIR}/gateway &&
 source \${ENV_DIR}/bin/activate &&
./start_gateway.sh > \${LOG_DIR}/gateway_startup.log 2>&1
"

# 等待服务启动
 sleep 10

 if netstat -tlnp | grep -q ":8080"; then
 log_message "SUCCESS" "API网关启动成功"
 else
 log_message "ERROR" "API网关启动失败"
 log_message "ERROR" "查看日志: tail -f \${LOG_DIR}/gateway.log"
 fi
 else
 log_message "INFO" "API网关已在运行"
 fi

# 启动监控服务
 if ! screen -list | grep -q "monitoring_service"; then
 log_message "INFO" "启动监控服务..."

 screen -dmS monitoring_service bash -c "
 cd \${PROJECT_DIR}/scripts/monitoring &&
./start_monitoring.sh > \${LOG_DIR}/monitoring_startup.log 2>&1
"

 log_message "SUCCESS" "监控服务启动成功"
 else
 log_message "INFO" "监控服务已在运行"
 fi

 log_message "INFO" "服务启动完成"
 show_status
}
```

```
# 停止服务
stop_services() {
 log_message "INFO" "停止AI模型服务..."

# 停止API网关
 if screen -list | grep -q "gateway_service"; then
 log_message "INFO" "停止API网关..."
 screen -S gateway_service -X quit
 log_message "SUCCESS" "API网关停止成功"
 else
 log_message "INFO" "API网关未在运行"
 fi

# 停止Shimmy服务
 if screen -list | grep -q "shimmy_service"; then
 log_message "INFO" "停止Shimmy服务..."
 screen -S shimmy_service -X quit
 log_message "SUCCESS" "Shimmy服务停止成功"
 else
 log_message "INFO" "Shimmy服务未在运行"
 fi

# 停止监控服务
 if screen -list | grep -q "monitoring_service"; then
 log_message "INFO" "停止监控服务..."
 screen -S monitoring_service -X quit
 log_message "SUCCESS" "监控服务停止成功"
 else
 log_message "INFO" "监控服务未在运行"
 fi

# 停止Ollama服务 (可选)
 read -p "是否停止Ollama服务? (y/N): " choice
 if [[ \${choice} =~ ^[Yy]$ ]]; then
 if systemctl is-active --quiet ollama; then
 log_message "INFO" "停止Ollama服务..."
 sudo systemctl stop ollama
 log_message "SUCCESS" "Ollama服务停止成功"
 else
 log_message "INFO" "Ollama服务未在运行"
 fi
```

```
 fi

 log_message "INFO" "服务停止完成"
}
# 重启服务
restart_services() {
 log_message "INFO" "重启AI模型服务..."

 stop_services
 sleep 5
 start_services

 log_message "INFO" "服务重启完成"
}
# 显示状态
show_status() {
 log_message "INFO" "服务状态报告:"

 echo -e "\n${YELLOW}=== 服务状态 ===${NC}"

# Ollama服务状态
 if systemctl is-active --quiet ollama; then
 echo -e "Ollama服务: ${GREEN}运行中${NC}"
 else
 echo -e "Ollama服务: ${RED}已停止${NC}"
 fi

# Shimmy服务状态
 if netstat -tlnp | grep -q ":8000"; then
 echo -e "Shimmy服务: ${GREEN}运行中${NC} (端口: 8000)"
 else
 echo -e "Shimmy服务: ${RED}已停止${NC}"
 fi

# API网关状态
 if netstat -tlnp | grep -q ":8080"; then
 echo -e "API网关: ${GREEN}运行中${NC} (端口: 8080)"
 else
 echo -e "API网关: ${RED}已停止${NC}"
 fi
```

```
# 监控服务状态
 if screen -list | grep -q "monitoring_service"; then
 echo -e "监控服务: ${GREEN}运行中${NC}"
 else
 echo -e "监控服务: ${RED}已停止${NC}"
 fi

 echo -e "\n${YELLOW}=== 网络端口 ===${NC}"
 netstat -tlnp | grep -E ":8000|:8080|:11434"

 echo -e "\n${YELLOW}=== Screen会话 ===${NC}"
 screen -list

 echo -e "\n${YELLOW}=== 系统资源 ===${NC}"
 echo "CPU使用率: \$(top -bn1 | grep "Cpu(s)" | awk '{print \$2}' | cut -d'%' -f1)%"
 echo "内存使用率: \$(free | grep Mem | awk '{printf("%.1f%%", \$3/\$2 * 100.0)}')"
 echo "磁盘使用率: \$(df -h / | tail -1 | awk '{print \$5}')"

 echo -e "\n${YELLOW}=== 已安装模型 ===${NC}"
 ollama list
}
# 查看日志
view_logs() {
 local service_name=\$1

 case \${service_name} in
"shimmy")
 log_file=\${LOG_DIR}/shimmy.log
;;
"gateway")
 log_file=\${LOG_DIR}/gateway.log
;;
"monitoring")
 log_file=\${LOG_DIR}/system_monitor.log
;;
"all")
 echo "显示所有日志文件:"
 ls -la \${LOG_DIR}/*.log
 return 0
;;
*)
```

```
 echo "用法: \$0 logs [shimmy|gateway|monitoring|all]"
 return 1
;;
 esac

 if [ -f "\${log_file}" ]; then
 echo "正在显示日志: \${log_file}"
 echo "按 Ctrl+C 退出"
 sleep 2
 tail -f \${log_file}
 else
 log_message "ERROR" "日志文件不存在: \${log_file}"
 fi
}
# 清理资源
clean_resources() {
 log_message "INFO" "清理系统资源..."

 read -p "确定要清理日志文件和缓存吗? (y/N): " choice
 if [[ ! \${choice} =~ ^[Yy]$ ]]; then
 log_message "INFO" "取消清理操作"
 return 0
 fi

# 清理日志文件
 log_message "INFO" "清理日志文件..."
 rm -f \${LOG_DIR}/*.log

# 清理缓存
 log_message "INFO" "清理缓存..."
 if [ -d "\${PROJECT_DIR}/gateway/__pycache__" ]; then
 rm -rf \${PROJECT_DIR}/gateway/__pycache__
 fi

 if [ -d "\${PROJECT_DIR}/shimmy/__pycache__" ]; then
 rm -rf \${PROJECT_DIR}/shimmy/__pycache__
 fi

# 清理Ollama缓存 (可选)
 read -p "是否清理Ollama模型缓存? (y/N): " choice
 if [[ \${choice} =~ ^[Yy]$ ]]; then
```

```
 log_message "INFO" "清理Ollama模型缓存..."
 sudo systemctl stop ollama
 sudo rm -rf /var/lib/ollama/models/*
 sudo systemctl start ollama
 fi

 log_message "SUCCESS" "资源清理完成"
}
# 健康检查
health_check() {
 log_message "INFO" "执行健康检查..."

 local all_healthy=true

# 检查服务状态
 if ! systemctl is-active --quiet ollama; then
 log_message "ERROR" "Ollama服务未运行"
 all_healthy=false
 fi

 if ! netstat -tlnp | grep -q ":8000"; then
 log_message "ERROR" "Shimmy服务未运行"
 all_healthy=false
 fi

 if ! netstat -tlnp | grep -q ":8080"; then
 log_message "ERROR" "API网关未运行"
 all_healthy=false
 fi

# 检查API响应
 if command -v curl &> /dev/null; then
 local response=\$(curl -s http://localhost:8080/health)
 if echo "\${response}" | grep -q "\"status\":\"healthy\""; then
 log_message "SUCCESS" "API网关健康检查通过"
 else
 log_message "ERROR" "API网关健康检查失败: \${response}"
 all_healthy=false
 fi
 fi
```

```
# 检查模型可用性
 if command -v curl &> /dev/null; then
 local api_key=\$(grep api_key \${PROJECT_DIR}/gateway/config.yaml | awk '{print \$2}')
 local models_response=\$(curl -s http://localhost:8080/models -H "X-API-Key: \${api_key}
")
 if echo "\${models_response}" | grep -q "\"models\":"; then
 log_message "SUCCESS" "模型列表获取成功"
 else
 log_message "ERROR" "模型列表获取失败: \${models_response}"
 all_healthy=false
 fi
 fi

 if \${all_healthy}; then
 log_message "SUCCESS" "所有健康检查通过"
 return 0
 else
 log_message "ERROR" "健康检查失败，请查看日志了解详情"
 return 1
 fi
}
# 显示帮助信息
show_help() {
 echo "AI模型服务管理脚本"
 echo "用法: \$0 [命令] [参数]"
 echo
 echo "命令:"
 echo "  start             启动所有服务"
 echo "  stop              停止所有服务"
 echo "  restart           重启所有服务"
 echo "  status            显示服务状态"
 echo "  logs [服务名]       查看日志 (shimmy|gateway|monitoring|all)"
 echo "  health            执行健康检查"
 echo "  clean             清理资源"
 echo "  help              显示帮助信息"
 echo
 echo "示例:"
 echo "  \$0 start           # 启动所有服务"
 echo "  \$0 logs shimmy       # 查看Shimmy日志"
 echo "  \$0 health          # 执行健康检查"
}
```

```
# 主函数
main() {
 if [ \$# -eq 0 ]; then
 show_help
 exit 1
 fi

 case \$1 in
"start")
 start_services
;;
"stop")
 stop_services
;;
"restart")
 restart_services
;;
"status")
 show_status
;;
"logs")
 view_logs \$2
;;
"health")
 health_check
;;
"clean")
 clean_resources
;;
"help")
 show_help
;;
*)
 echo "未知命令: \$1"
 show_help
 exit 1
;;
 esac
}
# 运行主函数
main "\$@"
```

```
EOF
# 设置脚本执行权限
chmod +x scripts/system_manager.sh
```

步骤 7: 启动和验证系统

7.1 启动系统

```
# 进入项目目录
cd ai-model-server
# 启动所有服务
./scripts/system_manager.sh start
# 检查服务状态
./scripts/system_manager.sh status
# 执行健康检查
./scripts/system_manager.sh health
```

7.2 测试 API 服务

```
# 获取API密钥
API_KEY=$(grep api_key gateway/config.yaml | awk '{print $2}')
echo "API密钥: ${API_KEY}"
# 测试健康检查API
curl -X GET http://localhost:8080/health
# 测试模型列表API
curl -X GET http://localhost:8080/models \
-H "X-API-Key: ${API_KEY}"
# 测试代码生成API
curl -X POST http://localhost:8080/generate \
-H "X-API-Key: ${API_KEY}" \
-H "Content-Type: application/json" \
-d '{
"model": "codellama:7b-code-q4",
"prompt": "请用Python编写一个快速排序算法",
"temperature": 0.7,
"max_tokens": 512
}'
```

模型选择与配置

推荐模型对比

模型名称	参数规模	专长领域	推理速度	内存占用	推荐指数
CodeLlama-7B-Code	7B	代码生成	中等	14-16GB	
DeepSeek-Coder-6.7B	6.7B	代码生成	较快	12-14GB	
StarCoder-7B	7B	代码生成	中等	14-16GB	
WizardCoder-7B	7B	代码生成	中等	14-16GB	
CodeLlama-7B-Instruct	7B	指令遵循	中等	14-16GB	

模型配置优化

量化配置

```
# 4-bit量化配置 (推荐)
PARAMETER quantize q4_0
PARAMETER num_ctx 2048
PARAMETER num_thread 8
# 8-bit量化配置 (平衡)
PARAMETER quantize q8_0
PARAMETER num_ctx 2048
PARAMETER num_thread 8
```

```
# 无量化配置 (性能最佳但内存占用高)
PARAMETER num_ctx 2048
PARAMETER num_thread 8
```

推理参数调优

```
# 代码生成优化参数
temperature: 0.7     # 控制随机性，0.7适合代码生成
max_tokens: 2048     # 最大生成长度
top_p: 0.95          # 控制多样性
num_ctx: 2048        # 上下文窗口大小
num_thread: 8        # 使用的CPU线程数
```

性能优化策略

CPU 优化

线程配置

```
# 设置Ollama使用的CPU线程数
export OLLAMA_NUM_THREADS=8
# 设置Python线程池大小
export PYTHON_THREADPOOL_SIZE=8
# 重启服务应用配置
sudo systemctl restart ollama
```

内存优化

```
# 设置Ollama内存使用限制
export OLLAMA_MEMORY=24GB
# 优化系统内存分配
```

```
echo 'vm.swappiness = 10' | sudo tee -a /etc/sysctl.conf
echo 'vm.dirty_ratio = 15' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

模型优化

量化技术

```
# 创建量化模型
ollama create codellama:7b-code-q4 -f - <<EOF
FROM codellama:7b-code
PARAMETER quantize q4_0
PARAMETER num_ctx 2048
PARAMETER num_thread 8
EOF
```

上下文窗口优化

```
# 调整上下文窗口大小
num_ctx: 2048  # 代码生成推荐大小
# num_ctx: 4096  # 需要更多内存但支持更长上下文
```

缓存策略

多级缓存配置

```
cache:
 enabled: true
 ttl: 3600              # 缓存时间(秒)
 max_size: 1000         # 最大缓存条目数
 memory_cache_size: 500   # 内存缓存大小
 disk_cache_size: 10000   # 磁盘缓存大小(MB)
```

监控与管理

系统监控

关键指标监控

- **CPU 使用率**: 监控 CPU 负载，避免过载
- **内存使用率**: 监控内存使用，防止 OOM
- **磁盘使用率**: 监控存储空间
- **网络流量**: 监控 API 请求流量
- **服务响应时间**: 监控模型推理性能
- **错误率**: 监控 API 错误率

告警机制

```
# 配置告警阈值
CPU_THRESHOLD=80   # CPU使用率超过80%告警
MEM_THRESHOLD=80   # 内存使用率超过80%告警
DISK_THRESHOLD=80  # 磁盘使用率超过80%告警
ERROR_THRESHOLD=5  # 错误率超过5%告警
```

日志管理

日志轮转配置

```
# 配置logrotate
sudo tee /etc/logrotate.d/ai-model-server <<EOF
/home/user/ai-model-server/logs/*.log {
 daily
 rotate 7
```

```
    compress
    delaycompress
    missingok
    notifempty
    create 0644 user user
    postrotate
    # 重启相关服务(如果需要)
    echo "日志轮转完成"
    endscript
}
EOF
```

安全配置

API 安全

API 密钥管理

```
# 生成强API密钥
API_KEY=$(openssl rand -hex 32)
echo "生成的API密钥: ${API_KEY}"
# 更新配置文件
sed -i "s|api_key: .*|api_key: ${API_KEY}|g" gateway/config.yaml
# 重启API网关
./scripts/system_manager.sh restart
```

访问控制

```
security:
  api_key: your_secure_api_key_here
  rate_limit:
    requests: 100     # 每分钟最大请求数
    period: 60        # 时间窗口(秒)
```

```
allowed_ips:      # 允许访问的IP地址
- 127.0.0.1
- 192.168.1.0/24
```

网络安全

防火墙配置

```
# 配置UFW防火墙
sudo ufw allow 22/tcp        # SSH
sudo ufw allow 8080/tcp      # API网关
sudo ufw allow 8000/tcp      # Shimmy服务
sudo ufw allow 11434/tcp     # Ollama服务
sudo ufw enable
sudo ufw status
```

客户端集成

Trae IDE 集成

Trae Agent 配置

```
{
 "name": "Local AI Code Generator",
 "description": "本地AI代码生成服务",
 "model": {
 "provider": "custom",
 "endpoint": "http://localhost:8080/generate",
 "api_key": "your_secure_api_key_here",
 "default_model": "codellama:7b-code-q4"
 },
 "capabilities": [
```

```
 "code_generation",
 "code_completion",
 "code_refactoring",
 "documentation"
 ],
 "settings": {
 "temperature": 0.7,
 "max_tokens": 1024
 }
}
```

命令行工具

CLI 客户端

```
# 创建命令行客户端
cat > scripts/ai-code-cli.py <<EOF
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
AI代码生成命令行工具
"""
import requests
import argparse
import json
import os
from dotenv import load_dotenv
def load_config():
"""加载配置"""
 config = {}

# 从环境变量加载
 load_dotenv()
 config['api_key'] = os.getenv('AI_API_KEY')
 config['endpoint'] = os.getenv('AI_ENDPOINT', 'http://localhost:8080/generate')

# 从配置文件加载
 if os.path.exists('config.json'):
```

```
 with open('config.json', 'r') as f:
 config.update(json.load(f))

 return config
def generate_code(prompt, model=None, max_tokens=1024, temperature=0.7):
"""生成代码"""
 config = load_config()

 if not config.get('api_key'):
 print("错误: API密钥未配置")
 return None

 payload = {
"prompt": prompt,
"max_tokens": max_tokens,
"temperature": temperature
}

 if model:
 payload["model"] = model

 try:
 response = requests.post(
 config['endpoint'],
 json=payload,
 headers={
'X-API-Key': config['api_key'],
'Content-Type': 'application/json'
},
 timeout=300
)

 response.raise_for_status()
 result = response.json()

 if result.get('success') and result.get('result'):
 return result['result']['choices'][0]['text']
 else:
 print(f"错误: {result.get('metadata', {}).get('error', '未知错误')}")
 return None
```

```
except Exception as e:
print(f"请求失败: {e}")
return None
def main():
parser = argparse.ArgumentParser(description='AI代码生成工具')

parser.add_argument('prompt', help='代码生成提示词')
parser.add_argument('--model', help='使用的模型名称')
parser.add_argument('--max-tokens', type=int, default=1024, help='最大生成token数')
parser.add_argument('--temperature', type=float, default=0.7, help='温度参数')
parser.add_argument('--output', help='输出文件路径')
parser.add_argument('--list-models', action='store_true', help='列出可用模型')

args = parser.parse_args()

config = load_config()

if args.list_models:
try:
response = requests.get(
config['endpoint'].replace('/generate', '/models'),
headers={'X-API-Key': config['api_key']},
timeout=10
)
response.raise_for_status()
models = response.json().get('models', [])

print("可用模型:")
for model in models:
print(f"  {model['name']}: {model['description']}")

except Exception as e:
print(f"获取模型列表失败: {e}")
return

print("正在生成代码...")
code = generate_code(
args.prompt,
model=args.model,
max_tokens=args.max_tokens,
temperature=args.temperature
```

```
)

 if code:
 print("\n" + "="*50)
 print("生成的代码:")
 print("="*50)
 print(code)
 print("="*50)

 if args.output:
 with open(args.output, 'w', encoding='utf-8') as f:
 f.write(code)
 print(f"\n代码已保存到: {args.output}")
if __name__ == '__main__':
 main()
EOF
# 设置执行权限
chmod +x scripts/ai-code-cli.py
# 创建配置文件
cat > scripts/config.json <<EOF
{
"api_key": "your_secure_api_key_here",
"endpoint": "http://localhost:8080/generate"
}
EOF
```

维护与更新

定期维护任务

每日维护

```
#!/bin/bash
# 每日维护脚本
echo "开始每日维护..."
```

```
# 检查系统更新
sudo apt update -y
# 清理系统缓存
sudo apt clean
# 清理日志文件
find /home/user/ai-model-server/logs -name "*.log" -mtime +7 -delete
# 检查磁盘空间
df -h
# 检查服务状态
/home/user/ai-model-server/scripts/system_manager.sh health
echo "每日维护完成"
```

每周维护

```
#!/bin/bash
# 每周维护脚本
echo "开始每周维护..."
# 更新系统包
sudo apt upgrade -y
# 更新Python依赖
source /home/user/ai-model-env/bin/activate
pip install --upgrade pip
pip install --upgrade -r requirements.txt
# 清理旧的Docker镜像(如果使用Docker)
if command -v docker &> /dev/null; then
 docker system prune -f
fi
# 检查模型更新
/home/user/ai-model-server/scripts/check_model_updates.sh
# 重启服务
/home/user/ai-model-server/scripts/system_manager.sh restart
echo "每周维护完成"
```

模型更新管理

模型更新检查

```
cat > scripts/check_model_updates.sh <<EOF
#!/bin/bash
# 模型更新检查脚本
LOG_FILE="../logs/model_updates.log"
log_message() {
 echo "\$(date '+%Y-%m-%d %H:%M:%S') [\$1] \$2" | tee -a \${LOG_FILE}
}
# 获取本地模型列表
local_models=\$(ollama list | awk 'NR>1 {print \$1}')
# 检查更新
for model in \${local_models}; do
 log_message "INFO" "检查模型更新: \${model}"

# 拉取最新版本
 if ollama pull \${model}; then
 log_message "INFO" "模型 \${model} 更新成功"
 else
 log_message "ERROR" "模型 \${model} 更新失败"
 fi
done
log_message "INFO" "模型更新检查完成"
EOF
chmod +x scripts/check_model_updates.sh
```

故障排除

常见问题及解决方案

1. Ollama 服务无法启动

```
# 检查Ollama服务状态
sudo systemctl status ollama
# 查看Ollama日志
journalctl -u ollama -f
# 检查端口占用
```

```
netstat -tlnp | grep :11434
# 重启Ollama服务
sudo systemctl restart ollama
```

2. 内存不足问题

```
# 检查内存使用情况
free -h
# 检查进程内存占用
top -o %MEM
# 调整Ollama内存配置
sudo sed -i 's|OLLAMA_MEMORY: .*|OLLAMA_MEMORY: "20GB"|g' /etc/ollama/config.yaml
sudo systemctl restart ollama
```

3. API 请求超时

```
# 检查服务状态
./scripts/system_manager.sh status
# 检查网络连接
curl -v http://localhost:8080/health
# 查看日志文件
./scripts/system_manager.sh logs gateway
# 调整超时配置
sed -i 's|timeout: .*|timeout: 600|g' gateway/config.yaml
```

4. 模型推理速度慢

```
# 检查CPU使用情况
htop
# 检查内存使用情况
free -h
# 使用量化模型
# 修改配置文件使用q4后缀的模型
sed -i 's|default: .*|default: codellama:7b-code-q4|g' shimmy/config.yaml
```

性能调优检查清单

- [x] ~~使用 4-bit 量化模型~~
- [x] ~~调整 CPU 线程数配置~~
- [x] ~~优化内存分配~~
- [x] ~~启用缓存机制~~
- [x] ~~监控系统资源使用~~
- [x] ~~调整推理参数~~

附录：配置文件模板

Ollama 配置模板

```
# /etc/ollama/config.yaml
environment:
  OLLAMA_NUM_THREADS: "8"
  OLLAMA_MEMORY: "24GB"
  OLLAMA_MODELS: "/var/lib/ollama/models"
  OLLAMA_LOG_LEVEL: "info"
service:
  host: "0.0.0.0"
  port: 11434
```

Shimmy 配置模板

```
# shimmy/config.yaml
model:
  default: codellama:7b-code-q4
  available:
    - name: codellama:7b-code
      type: code
      description: Meta CodeLlama 7B for code generation
      parameters:
        temperature: 0.7
        max_tokens: 2048
```

```
        top_p: 0.95

    - name: codellama:7b-code-q4
      type: code
      description: Meta CodeLlama 7B (4-bit quantized)
      parameters:
        temperature: 0.7
        max_tokens: 2048
        top_p: 0.95
server:
  host: 0.0.0.0
  port: 8000
  workers: 2
  timeout: 300
ollama:
  base_url: http://localhost:11434/api
  timeout: 300
logging:
  level: INFO
  file: ../logs/shimmy.log
  rotation: 10 MB
  retention: 7 days
cache:
  enabled: true
  ttl: 3600
  max_size: 1000
```

API 网关配置模板

```
# gateway/config.yaml
server:
  host: 0.0.0.0
  port: 8080
  workers: 4
  timeout: 300
models:
  default: codellama:7b-code-q4
  endpoints:
    codellama:7b-code:
      url: http://localhost:8000/v1/completions
      type: code
      description: Meta CodeLlama 7B for code generation

    codellama:7b-code-q4:
      url: http://localhost:8000/v1/completions
      type: code
      description: Meta CodeLlama 7B (4-bit quantized)
security:
```

```
  api_key: your_secure_api_key_here
  rate_limit:
    requests: 100
    period: 60
logging:
  level: INFO
  file: ../logs/gateway.log
  rotation: 10 MB
  retention: 7 days
cache:
  enabled: true
  ttl: 3600
  max_size: 1000
```

启动脚本模板

```
#!/bin/bash
# 一键启动脚本
echo "正在启动AI模型服务..."
# 加载环境变量
source ~/ai-model-env/bin/activate
# 启动Ollama服务
sudo systemctl start ollama
# 等待Ollama启动
sleep 10
# 启动Shimmy服务
cd ~/ai-model-server/shimmy
screen -dmS shimmy_service bash -c "./start_shimmy.sh"
# 等待Shimmy启动
sleep 15
# 启动API网关
cd ~/ai-model-server/gateway
screen -dmS gateway_service bash -c "./start_gateway.sh"
# 等待网关启动
sleep 10
# 启动监控服务
cd ~/ai-model-server/scripts/monitoring
screen -dmS monitoring_service bash -c "./start_monitoring.sh"
echo "服务启动完成！"
echo "API网关: http://localhost:8080"
echo "使用管理脚本查看状态: ./scripts/system_manager.sh status"
```

文档版本: 1.1

创建日期: 2025 年 11 月 5 日

适用环境: Ubuntu 22.04 LTS, i7-3770, 32GB RAM

维护团队: AI 开发团队

> （注：文档部分内容可能由 AI 生成）