软件配置与运维文档

1. 配置管理

1.1 配置项识别

- **数据库配置**：包括数据库地址、端口、用户名、密码等信息。
- **应用配置**：应用程序运行所需的配置，如服务器端口、日志级别等。
- **环境配置**：开发、测试、生产环境的特定配置。

1.2 配置管理流程

- **配置文件存储**：使用配置文件（ `.env` ）存储不同环境的配置信息。
- **配置加载顺序**：优先加载环境变量，其次加载配置文件。
- **安全性**：敏感信息加密存储，使用环境变量传递。

1.3 配置管理工具

- **工具选择**：使用Python中的 `python-dotenv` 库管理 `.env` 文件配置。
- **配置中心**：使用Python中的 `configparser` 模块实现集中配置管理。

2. 版本控制

2.1 版本控制策略

- **分支策略**：采用Git Flow的策略，主要分支包括 `master` 、 `develop` 、 `feature/*` 、 `release/*` 、 `hotfix/*` 。

2.2 版本控制工具

- **工具选择**：使用Git进行版本控制。
- **平台选择**：使用GitHub托管代码仓库。

2.3 分支管理

- **主分支**： `master` 分支始终保持可部署状态。
- **开发分支**： `develop` 分支用于日常开发。
- **功能分支**： `feature/*` 分支用于新功能开发，完成后合并至 `develop` 。
- **发布分支**： `release/*` 分支用于准备新版本发布，完成后合并至 `master` 和 `develop` 。
- **热修复分支**： `hotfix/*` 分支用于紧急修复线上问题，完成后合并至 `master` 和 `develop` 。

3. 持续集成

3.1 持续集成流程

- **代码提交**：每次代码提交触发CI流水线。
- **代码构建**：使用Python中的 `setuptools` 进行项目构建。
- **单元测试**：运行单元测试，确保代码功能正确。

3.2 持续集成工具

- **CI服务器**：使用GitHub Actions。
- **构建工具**：使用Python的 `setuptools` 。
- **测试工具**：使用 `unittest` 。

4. 部署和运维计划

4.1 部署流程

- **环境准备**：确保目标环境具备必要的依赖和配置。
- **部署验证**：通过自动化测试验证部署成功。

4.2 运维监控

- **监控工具**：使用Python中的 `prometheus_client` 和 `grafana-api` 监控系统性能和应用状态。
- **日志管理**：使用 `loguru` 库管理日志。
- **告警机制**：配置告警规则，通过邮件或短信通知运维人员。

4.3 故障处理

- **故障排查**：通过日志和监控系统快速定位问题。
- **应急预案**：制定应急预案，确保在发生故障时快速恢复服务。
- **回滚机制**：确保在发布失败时能够快速回滚到上一个稳定版本。