配置管理

本文档提供了《原神》和《崩坏：星穹铁道》相关工具（包括圣遗物扫描器、圣遗物导出器、仓库扫描器）的配置说明。通过命令行参数，用户可以灵活配置各工具的行为。

1. 圣遗物扫描器配置

配置项

- **最小星级**（`--min-star`）：只有星级不低于此值的圣遗物会被考虑。
- **最小等级**（`--min-level`）：只有等级不低于此值的圣遗物会被考虑。
- **忽略重复物品**（`--ignore-dup`）：如果设置为`true`，会忽略重复的圣遗物。
- **显示详细信息**（`--verbose`）：如果设置为`true`，会输出更多的日志信息。
- **指定圣遗物数量**（`--number`）：允许指定要扫描的圣遗物的具体数量。

使用示例

```
artifact_scanner --min-star 5 --min-level 10 --ignore-dup --verbose --number 100
```

2. 圣遗物导出器配置

配置项

- **输出格式**（`--format`，`-f`）：设置导出圣遗物的格式。
- **输出目录**（`--output-dir`，`-o`）：指定导出圣遗物文件的目录。

使用示例

```
artifact_exporter --format Mona --output-dir "./exports"
```

3. 仓库扫描器逻辑配置

配置项

- **最大扫描行数**（`--max-row`）：设置在停止前可以扫描的最大行数。
- **翻页时滚轮停顿时间**（`--scroll-delay`）：在翻页操作中，滚轮停顿的时间（毫秒）。
- **切换物品最大等待时间**（`--max-wait-switch-item`）：在切换到下一个物品时，最大的等待时间（毫秒）。
- **云游戏切换物品等待时间**（`--cloud-wait-switch-item`）：在云游戏中切换到下一个物品时，等待的时间（毫秒）。

使用示例

```
genshin_repository_scanner --max-row 100 --scroll-delay 100 --max-wait-switch-item 1000 --cloud-wait-switch-item 500
```

通过上述配置项和示例，用户可以根据自己的需求灵活配置《原神》相关工具的行为，以达到最佳的使用效果。

版本控制

1. 版本控制系统

本项目使用Git作为版本控制系统。

Git是一个分布式版本控制系统，它可以记录文件的变更历史，并允许多人协同开发。通过Git，我们可以轻松地管理项目的版本，回溯到任意历史版本，以及合并不同分支的代码。

Git的基本概念包括仓库（Repository）、分支（Branch）、提交（Commit）和合并（Merge）等。可以使用Git命令行工具或图形化界面工具来操作Git。

2. 版本命名规则

主版本.次版本.修补版本

- **主版本**：做出了不兼容的API修改，
- **次版本**：添加了向下兼容的功能性新增，
- **修补版本**：修正了向下兼容的问题。

3. 分支策略

- **主分支（main）**：用于存放已发布版本的代码。
- **修改分支（modification）**：用于日常开发。

4. 提交信息规范

提交信息包括三个部分：类型、标题和描述。格式如下：

```
<类型>: <标题>

<描述>
```

类型包括：

- feat（新功能）
- fix（修补bug）
- docs（文档改变）
- style（格式化）
- refactor（重构）
- test（增加测试）
- chore（构建过程或辅助工具的变动）

5. 版本发布流程

1. 将`modification`分支合并到`main`分支。
2. 在`main`分支上完成版本准备工作，如版本号更新、文档编写等。
3. 在`main`分支上打上版本标签。
4. 创建新的`modification`分支进行进一步的开发。

持续集成 (CI)

1. 触发条件

- **自动触发**：当代码被推送到`modification`分支时自动触发CI流程。
- **手动触发**：使用GitHub Actions直接启动。

2. 步骤概述

1. **代码检出**：从GitHub仓库检出代码，支持大文件存储（LFS）。
2. **依赖缓存**：缓存Rust项目的依赖，加快后续构建速度。
3. **环境变量设置**：设置环境变量，如`CARGO_TERM_COLOR`，以优化构建输出。
4. **版本信息提取**：从Git历史中提取版本信息，用于版本标记。
5. **工具链设置**：安装并使用Rust的nightly版本进行构建和测试。
6. **版本号设置**：在`Cargo.toml`中设置基于Git版本信息的版本号。
7. **构建**：对项目进行构建，生成可执行文件。
8. **重命名输出**：根据Git版本信息重命名构建输出，以便区分。
9. **上传构建产物**：将构建好的可执行文件上传为GitHub Actions的工件。

持续部署 (CD)

1. 触发条件

- **自动触发**：当代码合并到主分支后自动触发CD流程。
- **手动触发**：使用GitHub Actions直接启动。

2. 步骤概述

1. **环境准备**：准备部署环境，包括但不限于服务器配置、数据库准备等。
2. **获取构建产物**：从CI流程中获取最新的构建产物。
3. **部署准备**：根据目标平台（Windows、macOS、Linux）准备相应的部署脚本或工具。
4. **执行部署**：将构建产物部署到生产环境或测试环境。
5. **健康检查**：执行自动化的健康检查，确保部署的应用运行正常。
6. **回滚机制**：如果部署失败或健康检查未通过，自动执行回滚，恢复到上一个稳定版本。

运维计划

1. 系统监控与日志

- **组件健康检查**：定期检查每个组件（系统控制、交互逻辑、OCR等）的运行状态，确保它们正常工作。
- **日志管理**：为每个组件实现详细的日志记录机制，包括错误日志、操作日志和性能日志，便于问题追踪和性能分析。
- **性能监控**：监控关键组件的性能指标，如响应时间和资源使用情况，及时发现和解决性能瓶颈。

2. 定期维护

- **代码更新与部署**：定期检查和更新项目代码，包括第三方库和依赖，使用CI/CD流程自动化部署更新。
- **系统兼容性测试**：定期测试项目在不同系统（Windows、Linux、macOS）上的兼容性，确保功能正常。
- **数据备份**：对关键数据（如OCR模型数据、用户配置）进行定期备份，防止数据丢失。

3. 文档与培训

- **文档更新**：持续更新项目文档，包括安装指南、用户手册和故障排除指南。
- **团队培训**：定期对团队成员进行技术和安全培训，提高团队的运维能力和安全意识。

4. 自动化

- **自动化测试**：实现自动化测试流程，确保每次代码更新后自动运行测试，快速发现问题。
- **自动化部署**：使用CI/CD工具（如GitHub Actions）自动化部署流程，简化更新和部署操作，减少人为错误。