安同 OS Python 2 清除计划 - ShellWen Chen

项目申请书

项目名称: 安同 OS Python 2 清除计划
项目主导师: wxiwnd <wxiwnd@aosc.io>
申请人: ShellWen Chen / 陈畅
日期: 2025.06.03
邮箱: me@shellwen.com

1. 项目背景

1. 项目基本需求：

1. 消除安同 OS 中的 Python 2 及其相关依赖，提升系统安全性。
2. 简化系统架构，降低维护成本。
3. 优化包构建仓库的资源利用，同时跟进社区主流趋势。

2. 项目相关仓库：

https://github.com/AOSC-Dev/aosc-os-abbs

2. 技术方法及可行性

1. 依赖包分析

对安同 OS 中的 Python 包进行详细分析，识别纯 Python 2 包、错误声明依赖的 Python 3 包以及双版本支持包，并制定相应的处理方案。

2. 现有软件包评估

检查非 Python 软件包，分析脚本调用、Shebang、构建系统是否依赖 Python 2，并确定强制 Python 2 依赖的处理方式。

3. 实施策略

采用分阶段、渐进式的方法移除 Python 2：

1. **第一阶段 - 隔离 Python 2 软件包**：清理依赖关系，处理 Python 2 依赖，并进行用户通知。

2. **第二阶段 - 将 Python 3 作为默认选项**：更改默认解释器，进行兼容性测试，并进行用户通知。
3. **第三阶段 - 彻底移除 Python 2 及其软件包**：最终清理，系统验证，用户支持，并更新文档。

4. 可行性分析

- Python 2 已终止生命周期，主流发行版已移除或正在移除，有丰富的经验可供参考。
- 开源社区对 Python 2 的依赖已大大降低，迁移阻碍较小。
- 通过分阶段实施，可以降低风险，最小化对用户的影响。

3. 项目实现细节梳理:

1. 依赖包分析细节：

- **Python 包打包现状**：分析混合打包模式、依赖声明不准确和依赖链不清晰的问题，并提出改进方案。
- **Python 2 依赖分析**：通过工具分析当前系统中依赖 `python-2` 的软件包列表，并进行评估。
- **错误依赖分析**：深入调查发现错误依赖声明的软件包，并确定修正方法。

2. 现有软件包评估细节：

对系统软件包进行分类评估，以确定 Python 2 依赖的处理方式。

2.2.1 Python 软件包分类处理

1. **纯 Python 2 包** (仅含 Python 2 代码)：
   - **策略**：识别后评估必要性，计划移除或寻找 Python 3 替代品。
2. **错误声明依赖的 Python 3 包** (仅含 Python 3 代码，但 spec 文件误报 Python 2 依赖)：
   - **策略**：修正其包描述文件（如 spec），移除不必要的 Python 2 依赖后重新打包。
3. **双版本支持包** (同时含 Python 2 和 Python 3 代码)：
   - **策略**：修改打包脚本，剥离 Python 2 相关部分，仅保留并打包 Python 3 支持。

2.2.2 非 Python 软件包评估处理

检查非 Python 软件包中对 Python 2 的间接依赖。

1. **Shell 脚本和 Shebang 检查与修改**：
   - **策略**：扫描脚本中的 `#!/usr/bin/python` 或 `python2` 调用，统一修改为明确指向 `python3` 或评估脚本兼容性。
2. **构建系统检查与修改**：

- **策略**：审查 `Makefile`、`configure` 脚本等，将构建过程中对 Python 2 的调用替换为 Python 3。

3. **处理强制 Python 2 依赖的方案**：
   - **策略**：对于无法轻易迁移的依赖，优先寻求上游更新或替代品；次选禁用相关功能；最后考虑移除该软件包。

3. 移除方案实施细节：

为了系统性地、安全地移除安同 OS 中的 Python 2 及其相关包，我们将严格按照以下三个阶段的具体工作计划执行：

第一阶段具体工作：隔离 Python 2 软件包

目标：明确识别和隔离所有直接或间接依赖 Python 2 的软件包，清理不必要的依赖，为后续移除工作奠定基础。

1. **清理依赖关系**：
   - **识别错误依赖**：通过自动化脚本扫描所有软件包的 `spec` 文件（或安同 OS 使用的包描述文件），结合检查软件包内实际安装的文件路径（例如，仅包含 `python3` 相关路径而无 `python2` 路径的包却声明了 `python-2` 依赖），以及分析构建日志，找出错误声明 `python-2` 依赖的软件包。
   - **修正 spec 文件**：对于仅支持 Python 3 但错误声明了 Python 2 依赖的包，修改其 `spec` 文件，移除 `BuildRequires` 或 `Requires` (或等效字段) 中的 `python-2` 或相关 Python 2 库条目。
   - **重新构建和打包**：使用安同 OS 的标准构建系统（如 `abuild`、`dpkg-buildpackage` 或类似工具），对修正了 `spec` 文件的软件包进行重新构建和打包，生成不含 Python 2 依赖的新版本。
   - **验证新包**：对新构建的包进行基础安装和功能测试，确保移除错误依赖后包功能不受影响。
2. **处理 Python 2 依赖**：
   - **评估必要性**：对于确实依赖 Python 2 的软件包，评估其在安同 OS 中的核心程度、用户使用频率以及是否有社区维护的 Python 3 版本或功能等效的替代品。
   - **寻找替代方案**：
     - 优先在 Python 3 生态中寻找功能兼容或更优的库/应用。
     - 对于无法直接替换的工具，考虑使用其他编程语言实现的类似工具。
     - 如果某些功能非核心且难以迁移，考虑在后续版本中废弃该功能。
   - **重写构建脚本**：如果软件包的构建过程（如 `configure` 脚本、`Makefile` 等）依赖 Python 2，尝试将其修改为使用 `python3`。这可能涉及更新脚本中的解释器路径、调整语法或使用的库。

- **标记无法立即移除的包**：对于少数关键且短期内无法迁移的 Python 2 依赖包，进行特殊标记，并制定长期替换计划。

3. **用户通知**：
   - **通知渠道**：通过安同 OS 官方网站公告、邮件列表、Wiki、主要的社区沟通渠道（如 Telegram 群组、IRC 频道）发布清晰的通知。
   - **通知内容**：
     - 明确移除 Python 2 的原因、目标和大致时间表。
     - 告知用户此变更可能对他们自行安装或开发的依赖 Python 2 的脚本/应用产生影响。
     - 提供指导，帮助用户识别其系统中或项目中对 Python 2 的依赖（例如，检查 `#!/usr/bin/python` 或 `#!/usr/bin/env python` 的脚本，检查项目依赖文件等）。
     - 提供寻求帮助和反馈的渠道。
     - 预告下一阶段将把 Python 3 作为默认 Python 解释器。

第二阶段具体工作：将 Python 3 作为默认选项

目标：在系统层面将 Python 3 设置为默认的 Python 解释器，同时确保系统核心功能和广泛使用的应用在此变更后能够平稳过渡。

1. **更改默认解释器**：
   - **修改符号链接**：执行命令将系统中的 `/usr/bin/python` 符号链接从指向 `python2`（或其具体版本如 `python2.7`）更改为指向 `python3`（或其具体版本如 `python3.x`）。例如：`sudo ln -sf /usr/bin/python3 /usr/bin/python`。
   - **检查shebangs**：在更改符号链接之前或之后，使用脚本大规模扫描系统中（特别是 `/usr/bin`，`/usr/sbin`，`/opt` 等路径下）的脚本文件，识别所有使用 `#!/usr/bin/python` 或 `#!/usr/bin/env python` 作为 shebang 的脚本。
   - **修正脚本**：
     - 对于明确需要 Python 3 运行的脚本，如果其 shebang 是模糊的 `/usr/bin/python`，则无需修改（因为默认已改为 Python 3），或显式改为 `#!/usr/bin/python3`。
     - 对于少数仍需 Python 2 运行的脚本（应在第一阶段被识别并尽可能迁移），其 shebang 必须显式修改为 `#!/usr/bin/python2` (或指向 Python 2 的具体路径)，以避免在默认解释器更改后出错。这些脚本是第三阶段清理的目标。
2. **兼容性测试**：
   - **测试范围**：对安同 OS 的核心组件进行全面的回归测试，包括但不限于：
     - 包管理器及其相关工具。
     - 系统启动和服务管理脚本。
     - 桌面环境（如果提供）的核心组件和常用应用程序。
     - 常用的系统管理工具和命令行工具。

    - 构建系统和开发者工具链。
- **测试方法**：
    - 执行预定义的自动化测试套件。
    - 进行手动功能测试，覆盖常见用户场景和边缘案例。
    - 进行压力测试和性能基准测试，确保系统稳定性和性能不受影响。
    - 收集社区测试反馈。
3. **用户通知**：
    - **通知渠道**：与第一阶段类似，通过官方渠道和社区平台进行通知。
    - **通知内容**：
        - 宣布 `/usr/bin/python` 已正式指向 Python 3。
        - 强调这对开发者和用户编写或运行 Python 脚本的意义（例如，现在可以直接使用 `python script.py` 来执行 Python 3 脚本）。
        - 提供详细的迁移指南，包括：
            - 如何检查和更新脚本的 shebang。
            - Python 2 与 Python 3 的主要语法差异和常见迁移问题。
            - 针对常见依赖 Python 2 的第三方应用，提供迁移建议或替代方案。
        - 预告下一阶段将彻底移除 Python 2 及其相关软件包。

第三阶段具体工作：彻底移除 Python 2 及其软件包

目标：从系统中完全移除 Python 2 解释器、标准库及其所有依赖 Python 2 的软件包，确保系统纯净且仅依赖 Python 3。

1. **最终清理**：
    - **移除 Python 2 核心包**：从安同 OS 的构建仓库中移除 `python-3` 及其相关软件包。
    - **验证依赖图**：再次检查系统的依赖关系图，确保没有任何包仍然隐式或显式地指向已移除的 Python 2 组件。
    - **清理残余文件**：检查并清理可能由 Python 2 包留下的配置文件、缓存或数据目录（如果包管理器卸载不彻底）。
2. **系统验证**：
    - **全面系统测试**：重复第二阶段的兼容性测试，但这次是在完全没有 Python 2 的环境下进行。重点关注之前可能被忽略的、对 Python 2 有隐藏依赖的组件。
    - **关键应用功能验证**：确保所有核心系统服务、用户常用应用程序、开发工具链等在 Python 2 完全移除后依然能够正常启动和运行，且功能完整。
3. **用户支持**：
    - **FAQ 和故障排除指南**：根据测试阶段和早期用户反馈，整理并发布常见问题解答 (FAQ) 和故障排除指南。
    - **特定应用迁移支持**：对于用户反馈的、仍然依赖 Python 2 的重要第三方应用，积极与社区合作寻找迁移方案，或提供 Python 3 替代方案（如果可能）。

- **案例收集与分享**：鼓励用户分享他们迁移自定义脚本和应用的经验。

4. **最终文档更新**：
   - **更新系统需求**：在所有安同 OS 的官方文档（如安装指南、开发者文档、用户手册）中，明确声明系统不再包含或支持 Python 2。
   - **Python 3 作为标准**：更新所有涉及 Python 开发或脚本编写的文档，将 Python 3 作为唯一的、标准的 Python 环境进行描述。
   - **最佳实践**：提供使用 Python 3 进行开发和脚本编写的最佳实践指南，包括虚拟环境使用、包管理、编码规范等。
   - **移除 Python 2 的声明**：在发行说明 (Release Notes) 中显著标明 Python 2 已被彻底移除。

4. 规划：

1. 项目研发第一阶段（07 月 01 日 - 08 月 15 日）：

- 完成 Python 2 软件包的隔离工作，清理错误依赖。
- 评估和处理部分非 Python 软件包对 Python 2 的依赖。
- 开始进行用户通知，收集反馈。

2. 项目研发第二阶段（08 月 16 日 - 09 月 30 日）：

- 完成将 Python 3 设置为默认解释器的工作。
- 对系统核心组件进行全面兼容性测试。
- 为用户提供迁移指南和初步支持。
- （如果可能）彻底移除 `python-2` 和其相关软件包

3. 期望：

希望通过这个项目，成功移除安同 OS 中的 Python 2，提升系统安全性和现代化水平。同时，也希望积累系统级迁移和维护的经验，为后续参与更多系统相关的开源贡献打下基础。