

# 赛题二 高性能应用目录差异分析服务

## 1. 总体介绍

### 1.1 背景介绍

在数字化时代，随着信息技术的飞速发展，数据已成为企业、组织乃至个人最为宝贵的资产之一。然而，数据的存储和管理也面临着前所未有的挑战。其中，文件系统备份是确保数据安全性和可恢复性的关键措施。文件系统备份，简而言之，就是创建并保存文件系统的完整副本或部分内容，以便在原始数据丢失、损坏或被篡改时能够迅速恢复。这一过程的实施，不仅有助于预防潜在的数据风险，还能在意外情况发生时为企业和个人提供紧急恢复方案。

为了实现文件系统备份，一项核心工作就是分析当前目录与备份目录之间的差异，从而实施精确备份，提高备份效率。因此本赛题希望参赛同学开发一个系统服务，该系统服务用于分析特定目录的差异并输出差异内容。该服务需处理全量备份和差分备份。

#### 1.1.1 Openharmony 文件管理子系统介绍

文件管理子系统为OpenHarmony提供一套完整的文件数据管理解决方案，提供安全、易用的文件访问能力和完善的文件存储管理能力，包括：

1. 为应用提供安全的沙箱隔离技术，保证应用数据安全基础上权限最小化；
2. 统一的公共文件管理能力，统一公共数据访问入库，保证用户数据安全、纯净；
3. 分布式文件系统和云接入文件系统访问框架，应用可以像使用本地文件一样使用分布式和云端文件；
4. 支持公共数据、跨应用、跨设备的系统级文件分享能力；
5. 提供系统的存储管理能力和基础文件系统能力。



文件管理子系统对应用提供文件访问框架、文件分享框架、存储管理框架能力。

模块	详细描述
文件访问接口	1. 提供完整文件JS 接口，支持基础文件访问能力； 2. 提供本地文件、分布式文件扩展接口。
存储管理	1. 提供数据备份恢复框架能力，支持系统和应用数据备份克隆等场景； 2. 提供应用空间清理和统计、配额管控等空间管理能力； 3. 提供挂载管理、外卡管理、设备管理及多用户管理等存储管理能力。
公共文件	1. 公共数据沙箱隔离，保证用户数据安全、纯净； 2. 统一公共数据访问入口，仅medialibrary； 3. 提供统一的FMF的文件管理框架。
应用文件	1. 为应用提供安全的沙箱隔离技术，保证应用数据安全基础上权限最小化； 2. 支持应用间文件分享和文件跨设备分享，支持群组分享。
分布式能力	1. 提供基础分布式跨端访问能力，支持同账号分布式访问和异账号临时访问； 2. 支持文件跨端迁移能力，支撑应用迁移、分布式剪切板等分布式场景。
基础文件系统	1. 支持ext4、f2fs、exfat、ntfs等本地文件系统； 2. 支持分布式文件系统、nfs等网络文件系统； 3. 文件系统相关工具。

**备份恢复**相关主要在**应用文件管理仓库**里，接下来我们介绍**应用文件管理仓库**。

### 1.1.2 Openharmony 应用文件管理仓库

应用文件服务是为应用提供文件分享和管理能力的服务，包含应用间文件分享、跨设备同应用文件分享以及跨设备跨应用文件分享的能力。当前已具备基于分布式文件系统的跨设备同应用文件分享能力。

#### 目录

```
/foundation/filemanagement/app_file_service
| — interfaces                // 接口声明
|   | — innerkits             // 对内接口声明
|   | — kits                  // 对外接口声明
```

#### 备份恢复

##### 简介

备份恢复是为Openharmony设备上三方应用数据、系统应用数据、公共数据提供一套完整的数据备份和数据恢复解决方案。

备份恢复功能主要由三大部分组成：

- 集成在克隆等系统应用中的 **JS API**：负责触发备份/恢复数据。支持获取能力文件，触发备份应用数据，触发恢复应用数据，设置恢复应用数据时安装应用。
- 集成在待备份恢复应用中的备份 **服务扩展**：负责备份恢复具体应用的数据。应用开发者可通过配置备份恢复策略规则，配置备份恢复场景及过滤隐私等目录。
- 具有独立进程的备份服务：主要负责调度备份恢复任务。具体而言，其具体职责包括获取及检查备份恢复能力、管理备份服务扩展的生命周期与并发程度、协调零拷贝传输文件、在恢复时可选择安装应用。

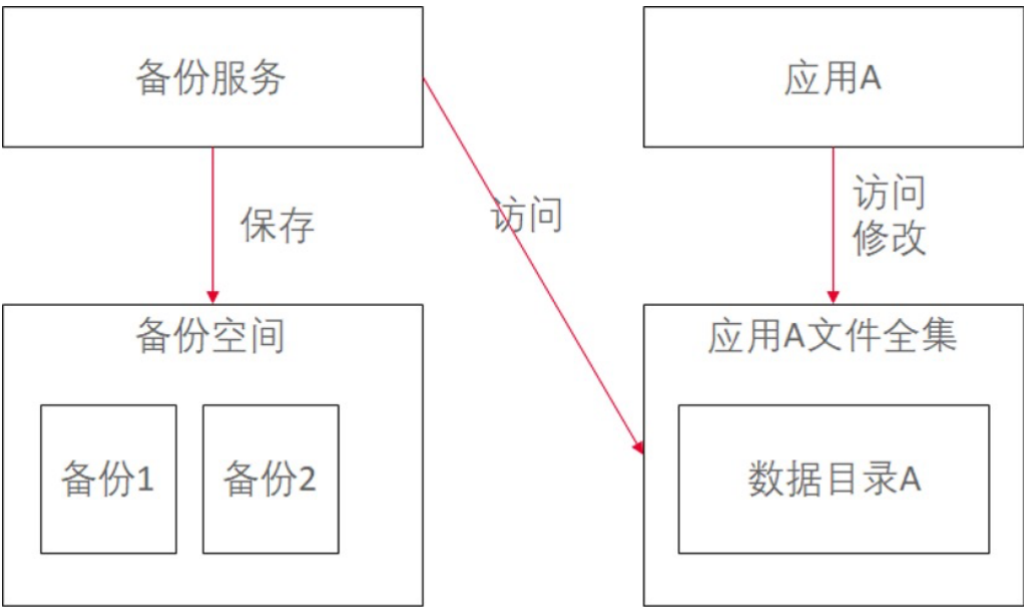
目录

```
/foundation/filemanagement/app_file_service
|—— frameworks                               // 框架层
|   |—— native
|       |—— backup_ext                       // 备份服务扩展
|—— interfaces                             // 接口存放目录
|   |—— api
|       |—— inner_api                       // 内部接口声明
|           |—— kits
|               |—— js
|                   |—— backup               // js外部接口
|—— services
|   |—— backup_sa                           // 备份恢复服务
|—— tests                                 // 测试用例
|—— tools                                 // 备份恢复工具
|—— utils                                 // 工具套
```

其中services目录里实现的备份恢复服务以及tools里实现的备份恢复工具是我们重点关注的，之后会在demo解析部分详细介绍。

1.2 赛题描述

本赛题的问题场景如下图。数据目录A是应用A涉及的所有文件的集合，我们希望对该目录备份，并在必要的时候予以恢复。所以本赛题希望同学开发如下图所示的备份服务，该备份服务可以对目标应用A的数据目录进行备份，将备份保存在备份空间中，并在后续依据备份对数据进行恢复。



## 1.3 参考技术路线

实现并测试此备份服务有多种方式：

- 基本：在鸿蒙里单独实现一个目录差异分析备份工具，通过命令行调用能对指定目录进行读取并生成全量备份文件
- 进阶：不仅支持全量备份，也支持差分备份，同时能将备份工具结合到openharmony原本的备份服务里，将高性能目录差异分析功能融入现有备份服务，对实际的运行中应用能做到定时文件备份以及触发恢复等。

## 2. 初赛内容

---

初赛要求大家提交一个技术报告，该技术报告需要包含两部分内容：

- 技术预研报告
- 技术路线报告

### 2.1 技术预研报告

技术预研报告希望大家能对相关技术进行调研，并形成一份技术调研报告，我们推荐的相关技术调研包括：

- OpenHarmony文件管理子系统的分析
- Openharmony现有备份技术调研，包括如何使用、总体代码结构解析、备份恢复流程、性能分析、优缺点分析等各个方面
- 其他任何你认为需要调研的内容.....

### 2.2 技术路线报告

在技术路线报告中，希望各位同学可以完整、详细地描述自己针对该赛题的方案设计，以便我们了解各位同学的决赛计划，我们希望同学在报告中体现如下内容：

- 备份系统的总体设计
- 备份系统各个部分的实现计划使用的技术
- 备份系统架构图以及流程图
- 其他任何你想补充的内容.....

### 2.3 提交方式

参赛同学请将完整的报告以pdf格式保存在git仓库中，如果有代码也可以一并上传到git仓库，并将git仓库的地址通过邮件发送至助教邮箱。

## 3. 决赛内容

---

决赛需要大家提交完整的项目代码，以及完整的技术报告两部分内容。

### 3.1完整的项目代码

决赛需要参赛同学提交完整的项目代码，我们将会采用示例数据目录来验证项目的正确性和性能表现。

#### 3.1.1 示例数据目录定义

本赛题提供了示例数据目录来模拟应用在不同时间点的数据状态，该示例数据目录的架构描述如下：

- 应用A有一个独有的数据目录 ( $A_1, A_2, A_3$ )，分别代表从早到晚三个不同时间点的数据状态 ( $T_1 < T_2 < T_3$ )。
- 数据目录包含多层文件/文件夹，每层的文件数量和大小因层数不同而异。

#### 3.1.2 备份数据格式

对于备份数据格式，一个可能的示例如下（此处仅为示例，同学们可以自行设计）：

- 每个备份是一个目录，该目录包括：
  - 一个文本文件：该文本文件保存了该备份的文件或目录的状态信息。每个状态信息一行，示例如下：

```
(added) && (path) || (startpos1 endpos1)
(delete) && (path)
(modified) && (path) || ((startpos1 endpos1) || (startpos2 endpos2)|| ..... )
```
  - 一个目录：该目录保存了具体文件的变化。具体形式自行确定。

#### 3.1.3 正确性要求

针对上述示例数据目录，本赛题有如下正确性要求：

1. 备份：全量备份和差分备份可以选择性支持一种或多种
  1. 全量备份：以数据目录 $A_1$ 为目标，对数据目录进行全量备份，注意保留相关的备份数据分析信息。
  2. 差分备份：差分备份**需要完成下述两个要求**：
    1. 普通差分备份：以数据目录 $A_1$ 和 $A_2$ 为目标，需要分析数据目录 $A_2$ 和 $A_1$ 的差异，并进行差分备份，输出 $A_2$ 和 $A_1$ 间的差异数据全集。建议将大文件进行适当粒度的分块（4KB起步）。
    2. 多次差分备份：以数据目录 $A_1$ 、 $A_2$ 和 $A_3$ 为目标，需要分析数据目录 $A_2$ 和 $A_1$ 的差异，输出备份数据1。然后分析 $A_3$ 和 $A_2$ 的差异，并在备份数据1的基础上叠加差异，输出备份数据2。不允许直接获取 $A_3$ 和 $A_1$ 的差异。
2. 备份恢复：在数据目录发生更改以后，可以根据上述生成的备份恢复到某一时刻的数据。
  - 实现方式基础要求：可以单独在Openharmony上实现一个程序对指定的文件目录进行备份和恢复
  - 实现方式进阶要求：能将备份工具结合到openharmony原本的备份服务里，将高性能目录差异分析功能融入现有备份服务，对实际的运行中应用能做到定时文件备份以及触发恢复等。

我们会检查源代码，并通过先调用备份服务，再调用备份恢复服务来检查正确性。

#### 3.1.4 性能要求

在确保通过正确性测试的要求后，我们会对大家备份和恢复两个过程的性能进行评估，并结合多项数据指标进行打分，指标包括：

- 该过程的耗时
- 该过程中CPU和内存等资源的占用

## 3.2 完整的技术报告

决赛技术报告在初赛报告的基础上，需要完善与项目相关的部分，具体需要完善的部分包括：

- 项目的完整设计方案
- 项目正确性展示
- 项目备份性能测试（备份耗时、资源使用情况）
- 项目总结与不足
- 其他任何你想补充的内容.....

## 3.3 提交方式

和初赛方式相同，请各位参赛同学请将完整的项目代码和技术报告保存在git仓库中，并将git仓库的地址通过邮件发送至助教邮箱。

# 4. demo解析

---

## 4.1 单独实现程序处理备份恢复

### 1. 项目结构

创建一个新的项目目录，组织代码结构如下：

```
backup_tool/  
|--- src/  
|   |--- main.cpp  
|   |--- backup.cpp  
|   |--- backup.h  
|   |--- restore.cpp  
|   |--- restore.h  
|--- CMakeLists.txt
```

### 2. CMakeLists.txt

创建一个 `CMakeLists.txt` 文件，用于定义构建过程：

```
cmake_minimum_required(VERSION 3.4.1)  
  
project(backup_tool)  
  
set(CMAKE_CXX_STANDARD 17)  
  
add_executable(backup_tool src/main.cpp src/backup.cpp src/restore.cpp)  
  
target_include_directories(backup_tool PRIVATE src)
```

### 3. 定义备份和恢复的头文件

创建 `backup.h` 和 `restore.h` 文件，定义备份和恢复的函数接口。

```

#ifndef BACKUP_H
#define BACKUP_H

#include <string>

void FullBackup(const std::string &srcDir, const std::string &backupDir);
void DiffBackup(const std::string &prevDir, const std::string &newDir, const
std::string &backupDir);

#endif // BACKUP_H

```

```

// restore.h
#ifndef RESTORE_H
#define RESTORE_H

#include <string>

void RestoreBackup(const std::string &backupDir, const std::string &restoreDir);

#endif // RESTORE_H

```

#### 4. 实现备份和恢复功能

创建 `backup.cpp` 和 `restore.cpp` 文件，分别实现全量备份、差分备份和恢复功能。

```

#include "backup.h"
#include <filesystem>
#include <fstream>
#include <iostream>

namespace fs = std::filesystem;

void FullBackup(const std::string &srcDir, const std::string &backupDir) {
    try {
        fs::path srcPath(srcDir);
        fs::path backupPath(backupDir);

        // Create backup directory if it does not exist
        fs::create_directories(backupPath);

        // Copy all files and directories from srcDir to backupDir
        for (const auto &entry : fs::recursive_directory_iterator(srcPath)) {
            const auto &path = entry.path();
            auto relativePath = path.lexically_relative(srcPath);
            fs::copy(path, backupPath / relativePath,
fs::copy_options::overwrite_existing | fs::copy_options::recursive);
        }

        // Write metadata
        std::ofstream metaFile(backupPath / "metadata.txt");
        for (const auto &entry : fs::recursive_directory_iterator(srcPath)) {
            const auto &path = entry.path();
            metaFile << path.string() << "\n";
        }
        metaFile.close();
    }
}

```

```

        std::cout << "Full backup completed successfully.\n";
    } catch (const fs::filesystem_error &err) {
        std::cerr << "Filesystem error: " << err.what() << '\n';
    }
}

void DiffBackup(const std::string &prevDir, const std::string &newDir, const
std::string &backupDir) {
    try {
        fs::path prevPath(prevDir);
        fs::path newPath(newDir);
        fs::path backupPath(backupDir);

        // Create backup directory if it does not exist
        fs::create_directories(backupPath);

        // Compare directories and identify changes
        for (const auto &entry : fs::recursive_directory_iterator(newPath)) {
            const auto &path = entry.path();
            auto relativePath = path.lexically_relative(newPath);
            auto prevFilePath = prevPath / relativePath;
            if (!fs::exists(prevFilePath) || fs::file_size(prevFilePath) !=
fs::file_size(path) || fs::last_write_time(prevFilePath) != fs::last_write_time(path))
            {
                fs::copy(path, backupPath / relativePath,
fs::copy_options::overwrite_existing | fs::copy_options::recursive);
            }
        }

        // Write metadata
        std::ofstream metaFile(backupPath / "metadata.txt");
        for (const auto &entry : fs::recursive_directory_iterator(newPath)) {
            const auto &path = entry.path();
            auto relativePath = path.lexically_relative(newPath);
            auto prevFilePath = prevPath / relativePath;
            if (!fs::exists(prevFilePath) || fs::file_size(prevFilePath) !=
fs::file_size(path) || fs::last_write_time(prevFilePath) != fs::last_write_time(path))
            {
                metaFile << "modified && " << relativePath.string() << "\n";
            }
        }
        metaFile.close();

        std::cout << "Differential backup completed successfully.\n";
    } catch (const fs::filesystem_error &err) {
        std::cerr << "Filesystem error: " << err.what() << '\n';
    }
}

```

restore.cpp :

```

#include "restore.h"
#include <filesystem>
#include <fstream>
#include <iostream>
#include <sstream>

```



```

namespace fs = std::filesystem;

void RestoreBackup(const std::string &backupDir, const std::string &restoreDir) {
    try {
        fs::path backupPath(backupDir);
        fs::path restorePath(restoreDir);

        // Create restore directory if it does not exist
        fs::create_directories(restorePath);

        // Read metadata
        std::ifstream metaFile(backupPath / "metadata.txt");
        std::string line;
        while (std::getline(metaFile, line)) {
            std::string action, path;
            std::istringstream iss(line);
            if (iss >> action >> path) {
                fs::path fullPath = backupPath / path;
                if (action == "modified") {
                    fs::copy(fullPath, restorePath / path,
fs::copy_options::overwrite_existing | fs::copy_options::recursive);
                }
            }
        }
        metaFile.close();

        std::cout << "Restore completed successfully.\n";
    } catch (const fs::filesystem_error &err) {
        std::cerr << "Filesystem error: " << err.what() << '\n';
    }
}

```

## 5. 主程序入口

创建 `main.cpp` 文件，实现命令行接口，用于备份和恢复操作。

```

#include "backup.h"
#include "restore.h"
#include <iostream>
#include <map>
#include <vector>
#include <string>

void PrintUsage() {
    std::cout << "Usage:\n";
    std::cout << "  backup_tool backup --srcDir=<source_directory> --backupDir=
<backup_directory>\n";
    std::cout << "  backup_tool diff_backup --prevDir=<previous_directory> --newDir=
<new_directory> --backupDir=<backup_directory>\n";
    std::cout << "  backup_tool restore --backupDir=<backup_directory> --restoreDir=
<restore_directory>\n";
}

std::map<std::string, std::string> ParseArguments(int argc, char *argv[]) {

```

```

std::map<std::string, std::string> args;
for (int i = 1; i < argc; ++i) {
    std::string arg = argv[i];
    if (arg.substr(0, 2) == "--") {
        auto pos = arg.find('=');
        if (pos != std::string::npos) {
            std::string key = arg.substr(0, pos);
            std::string value = arg.substr(pos + 1);
            args[key] = value;
        } else if (i + 1 < argc) {
            args[arg] = argv[++i];
        }
    }
}
return args;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        PrintUsage();
        return 1;
    }

    std::string command = argv[1];
    auto args = ParseArguments(argc, argv);

    if (command == "backup") {
        if (args.find("--srcDir") != args.end() && args.find("--backupDir") !=
args.end()) {
            std::string srcDir = args["--srcDir"];
            std::string backupDir = args["--backupDir"];
            FullBackup(srcDir, backupDir);
        } else {
            PrintUsage();
            return 1;
        }
    } else if (command == "diff_backup") {
        if (args.find("--prevDir") != args.end() && args.find("--newDir") != args.end()
&& args.find("--backupDir") != args.end()) {
            std::string prevDir = args["--prevDir"];
            std::string newDir = args["--newDir"];
            std::string backupDir = args["--backupDir"];
            DiffBackup(prevDir, newDir, backupDir);
        } else {
            PrintUsage();
            return 1;
        }
    } else if (command == "restore") {
        if (args.find("--backupDir") != args.end() && args.find("--restoreDir") !=
args.end()) {
            std::string backupDir = args["--backupDir"];
            std::string restoreDir = args["--restoreDir"];
            RestoreBackup(backupDir, restoreDir);
        } else {
            PrintUsage();
            return 1;
        }
    } else {

```

```
        PrintUsage();  
        return 1;  
    }  
  
    return 0;  
}
```

## 6. 编译运行

1. 在项目根目录下创建构建目录并进入构建目录：

```
mkdir build  
cd build
```

2. 生成构建文件：

```
cmake ..
```

3. 编译项目：

```
make
```

4. 运行备份工具：

**全量备份：**

```
./backup_tool backup --srcDir=./test/ --backupDir=./test2/
```

**差分备份：**

```
./backup_tool diff_backup --prevDir=./test/ --newDir=./test2/ --  
backupDir=./diff_backup/
```

**恢复备份：**

```
./backup_tool restore --backupDir=./backup/ --restoreDir=./restore/
```

运行全量备份：



image-20240523163549840

## 4.2 OpenHarmony 全量备份工具backup\_tool

`backup_tool` 是OpenHarmony备份恢复服务下提供的命令行工具，能对指定应用的文件目录进行全量备份以及恢复，位于 `OpenHarmony/foundation/filemanagement/app_file_service/tools/backup_tool` 下

### 功能特性

1. **全量备份：**
  - 备份指定目录下的所有文件和子目录，确保数据的完整性。
2. **错误处理和日志记录：**
  - 提供详细的错误处理和日志记录机制，确保用户可以实时了解备份过程的状态和遇到的问题。
3. **事件驱动的回调机制：**

- 通过回调函数机制，在备份过程中处理各种事件，如文件准备好、应用备份开始或结束、所有应用备份结束以及备份服务意外死亡等。

#### 4. 多线程支持：

- 采用多线程设计，提高备份过程的并发性和效率。

## 使用方法

`backup_tool` 通过命令行参数来指定备份操作的相关配置。以下是基本的使用方法：

```
backup_tool backup --isLocal=true --bundle com.sample.app2 --pathCapFile /data/backup/tmp
```

- `--isLocal`：指定是否为本地备份。该参数为布尔值，`true` 表示本地备份，`false` 表示远程备份。
- `--bundle`：指定需要备份的应用，该应用的文件目录为备份源目录
- `--pathCapFile`：指定备份能力文件，此文件描述了备份的能力集合。
- 目标目录为指定的/data/backup下

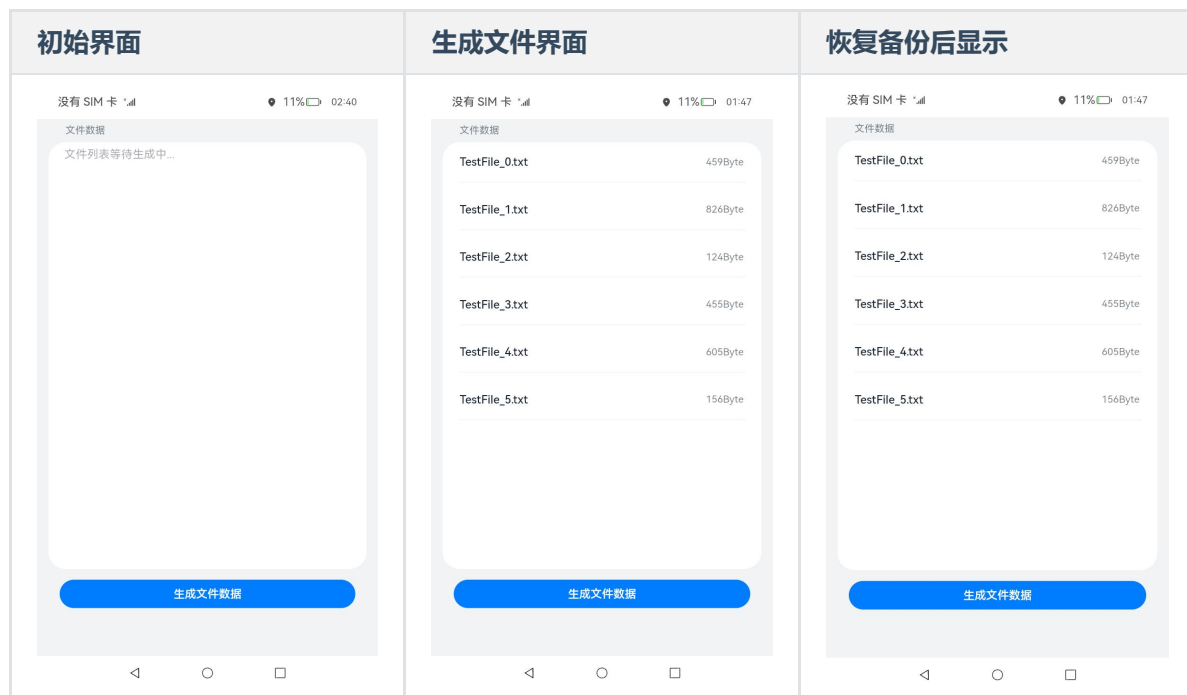
以上例子即为将app2的应用目录下的文件都备份到/data/backup/下

## 例子

这里以官方提供的一个应用（[https://gitee.com/openharmony/applications\\_app\\_samples/tree/master/code/BasicFeature/FileManagement/FileBackupExtension](https://gitee.com/openharmony/applications_app_samples/tree/master/code/BasicFeature/FileManagement/FileBackupExtension)）为例：

本sample主要给备份流程提供应用，用于生成数据和显示数据。

## 效果预览



使用说明：

- 点击按钮“生成文件数据”，应用会随机生成2个到10个txt文件，大小在1Byte到1024Byte之间，用于校验备份和恢复的数据为同一数据。
- 本应用仅适用于生成和显示数据，具体的备份和恢复的操作目前需要使用hdc shell命令操作。
- 具体操作方面，进入应用点击按钮生成文件数据后，进入hdc shell，使用`backup_tool backup --isLocal=true --bundle com.samples.backupextension --pathCapFile /data/backup/tmp`命令进行备份。然后可以把应用删掉，重新安装应用后进入hdc shell，使用`backup_tool restore --bundle`

com.samples.backupextension --pathCapFile /data/backup/tmp, 提示成功后, 重新打开应用可以看到备份的文件显示在列表里。

#### 4. 备份与恢复操作:

应用备份

```
backup_tool backup --isLocal=true --bundle com.sample.backupextension --pathCapFile /data/backup/tmp
```

备份成功提示:

```
BundleStarted errCode = 0, BundleName = com.example.backupextension
FileReady owner = com.example.backupextension, fileName = manage.json, sn = 0, fd = 9
FileReady owner = com.example.backupextension, fileName = part.0.tar, sn = 0, fd = 9
BundleFinished errCode = 0, BundleName = com.example.backupextension
backup successful
```

应用恢复

```
backup_tool restore --bundle com.sample.backupextension --pathCapFile /data/backup/tmp
```

恢复成功提示:

```
BundleStarted errCode = 0, BundleName = com.example.backupextension
FileReady owner = com.example.backupextension, fileName = part.0.tar, sn = 0, fd = 8
FileReady owner = com.example.backupextension, fileName = manage.json, sn = 0, fd = 8
BundleFinished errCode = 0, BundleName = com.example.backupextension
Restore successful
```

backup详细说明可以使用backup -help查看。

## 内部工作原理

代码目录:

备份逻辑主要在tools\_op\_backup.cpp中实现:

#### 1. 初始化会话:

- 通过 `BSessionBackup::Init` 方法初始化备份会话, 并注册相关回调函数。

#### 2. 配置备份参数:

- 解析命令行参数, 设置源目录和备份目录。

#### 3. 启动备份流程:

- 调用 `BSessionBackup::Start` 方法, 正式启动备份过程。

#### 4. 文件传输和状态更新:

- 在备份过程中, 回调函数 `OnFileReady` 被调用, 用于处理文件传输和状态更新。

#### 5. 完成备份:

- 当所有文件备份完成后, 调用 `OnAllBundlesFinished` 回调函数, 通知用户备份操作成功完成。

## 回调函数

在备份过程中, `backup_tool` 提供了一系列回调函数, 以处理不同的事件:

- `OnFileReady`: 当文件准备好进行备份时调用, 负责文件的传输和状态更新。
- `OnBundleStarted`: 当某个应用的备份开始时调用。
- `OnBundleFinished`: 当某个应用的备份结束时调用。
- `OnAllBundlesFinished`: 当所有应用的备份完成时调用。
- `OnBackupServiceDied`: 当备份服务意外死亡时调用。

目前还不支持差分备份, 这一块可以选择基于此工具继续去完善。

## 参考链接

---

- 鸿蒙文件管理子系统介绍: <https://gitee.com/openharmony/docs/blob/master/zh-cn/readme/%E6%96%87%E4%BB%B6%E7%AE%A1%E7%90%86%E5%AD%90%E7%B3%BB%E7%B%9F.md>
- 应用文件管理服务: [https://gitee.com/openharmony/filemanagement\\_app\\_file\\_service](https://gitee.com/openharmony/filemanagement_app_file_service)