PFS过滤器和系统选项设计方案说明书

|  |  |  |  |
| --- | --- | --- | --- |
| 更新记录 | | | |
| 2010-8-5 | 文档初稿 | V0.8 | 刘东学 |
|  |  |  |  |

1. 方案起源

本方案设计源自”梦幻诛仙”海外版翻译工作流程的一个需求。

海外版翻译工作的整体流程不属于本文档的描述范围，这里仅说明与此相关的环节。

海外版翻译工作由一个专门的本地化工具负责实现。本地化工具负责加载一个待翻译的文件包(.zip格式)，翻译完成后，本地化工具导出翻译后的资源包client.pfs(仍旧是.zip格式)。为了方便测试，这个翻译之后的资源包，最好能够直接放到客户端程序所在的目录下直接能被加载并测试。

用例：

假定，本地安装版本存在cfg.pfs, ui.pfs,…资源包，本地化工具导出的资源包为client.pfs，里面含有cfg目录和ui目录对应的翻译文件，那么，将client.pfs直接置于客户端目录下，与cfg.pfs并列放置，通过尽量简单的配置，可以使得客户端能够优先从client.pfs的cfg目录中读取翻译之后的cfg资源，从ui目录中读取翻译之后的ui资源，而client.pfs中并不存在的那些资源，依旧从原始的cfg.pfs,ui.pfs中读取。

另外，尽量能够做到，在不修改上述配置的情况下，删除client.pfs之后依旧保证客户端程序无障碍运行。

1. 配置方案

修改mount.xml，新增指定mount点的过滤器属性，同时以可选方式mount client.pfs资源包。

配置方案和用法如下：

<data>

<option group="0" name="base-filter" value="1"/>

<mount dir="/run" device="%current-dir%" mt="2"/>

<mount dir="/a" device="/run/client" mt="18"/>

<mount dir="/cfg" device="/run/cfg" filter="/a/cfg" mt="2"/>

<mount dir="/wtf" device="/run/wtf" mt="2"/>

<mount dir="/show" device="/run/show" mt="2"/>

<mount dir="/ui" device="/run/ui" filter="/a/ui" mt="2"/>

<mount dir="/sound" device="/run/sound" mt="2"/>

<mount dir="/pak" device="/run/pak" mt="2"/>

<mount dir="/role/character" device="/run/character" mt="2"/>

<mount dir="/role/monster" device="/run/monster" mt="2"/>

<mount dir="/role/npc" device="/run/npc" mt="2"/>

<mount dir="/map" device="/run/map" mt="2"/>

<mount dir="/effect" device="/run/effect" mt="2"/>

</data>

为了简化说明，下面以/cfg的过滤说明该配置的工作方式。

<option group="0" name="base-filter" value="1"/>

指出，整个pfs是否启用这种过滤器。如果没有配置，缺省不启用，这可以保证正常模式下过滤器不生效，不至于影响PFS原本的加载效率。

<mount dir="/a" device="/run/client" mt="18"/>

配置一个单纯的过滤包（可以理解为补丁包，mt类型为16+2,16表示该mountdevice是否可选，对于可选项，mount失败，不予理睬）。

<mount dir="/cfg" device="/run/cfg" filter="/a/cfg" mt="2"/>

这里的 filter属性指明/a/cfg 这个PFS子系统的子目录为当前/cfg子系统的过滤包。

1. 测试方案

测试环境：

1. 在release目录下准备如下文件

cfg/a.txt 内容 a.txt in cfg/a.txt \n

cfg/ab.txt 内容 ab.txt in cfg/ab.txt \n

cfg/f/c.txt 内容 c.txt in cfg/f/c.txt \n

a/cfg/b.txt 内容 --- b.txt in a/cfg/b.txt

a/cfg/ab.txt 内容 --- ab.txt in a/cfg/ab.txt

a/cfg/f/c.txt 内容 --- c.txt in a/cfg/f/c.txt

a/cfg/f/fc.txt 内容 --- fc.txt in a/cfg/f/fc.txt

将目录a中的打成a.pfs包。注意,a.pfs中不含有目录a本身。

2. wtf/mount.xml进行如下配置:

<option group="0" name="base-filter" value="1"/>

<mount dir="/run" device="%current-dir%" mt="2"/>

<mount dir="/a" device="/run/a" mt="18"/>

<mount dir="/cfg" device="/run/cfg" filter="/a/cfg" mt="2"/>

测试结论：

1. filter开启时：

open如下文件应该打开的是a目录下对应的文件

/cfg/ab.txt --> /a/cfg/ab.txt

/cfg/f/c.txt --> /a/cfg/f/c.txt

/cfg/f/fc.txt --> /a/cfg/f/fc.txt

open如下文件应该打开的是cfg目录下对应的文件

/cfg/a.txt --> /cfg/a.txt

/cfg/b.txt --> /a/cfg/b.txt

open如下文件失败目录下对应的文件

/cfg/c.txt --> <fail to open>

2. filter关闭时:

open如下文件应该打开的是a目录下对应的文件

/cfg/a.txt --> /cfg/a.txt

/cfg/ab.txt --> /cfg/ab.txt

/cfg/f/c.txt --> /cfg/f/c.txt

open如下文件失败

/cfg/b.txt --> <fail to open>

/cfg/c.txt --> <fail to open>

/cfg/f/fc.txt --> <fail to open>

3. 附加保证

同样的filename机制下，启用filter和关闭filter表现应该一致。

如果在a/cfg/能读到的文件，所有path应该都能读到。

如果在a/cfg/不能读到的文件，所有path应该不能从a中读到。

如果在a/cfg/不能读到的文件，但能在/cfg目录下能读到的文件，所有path行为一致。

具体测试代码，详见http://172.16.10.57/repos/whf/XiaoPang/pfs/projects/mountmanager\_test.cpp。

1. 实现方案

PFS接口新增如下内容：

挂载类型定义：

enum

{

+ MT\_OPTIONAL = 0x00000010, // 该设备可有可无，没有时忽略此设备。(当前只用于MountManager)

};

typedef unsigned int MOUNT\_TYPE;

+inline bool IsMountOptional( MOUNT\_TYPE mt )

+{

+ return (mt & MT\_OPTIONAL) != 0;

+}

选项组定义：

+//

+// enumeration of pfs option groups

+//

+enum

+{

+ PFS\_OPTION\_GLOBAL = 0, // global properties that can be identified by property name.

+ PFS\_OPTION\_LOG = 1, // log properties(such as log level, log file... reserved)

+ PFS\_OPTION\_INTERNET = 2, // internet properties( such as connect timeout, receive timeout, send timeout. reserved)

+ PFS\_OPTION\_BASE\_FILTER = 10,// 可以通过global属性 base-filter 开启或者关闭

+ // pfs base path filter( name is the base to be replaced, value is the base to replace)

+};

+typedef int PFS\_OPTION\_GROUP;

错误码定义：

enum

{

+ EC\_FILTER\_ERROR = -1100,// 一般过滤器错误

+ EC\_FILTER\_INVALID\_ORI\_DIR, // 无效的过滤器原目录

+ EC\_FILTER\_INVALID\_NEW\_DIR, // 无效的过滤器新目录

+ EC\_FILTER\_ITEM\_NOT\_EXISTS, // 不存在此过滤器配置项目

+

+ EC\_OPTION\_GENERAL\_ERROR = -1200,// 一般属性错误

+ EC\_OPTION\_INVALID\_GROUP, // 无效的属性组

+ EC\_OPTION\_INVALID\_NAME, // 无效的属性名

+ EC\_OPTION\_INVALID\_VALUE, // 无效的属性值

+ EC\_OPTION\_OUT\_OF\_RANGE, // 属性值超出可允许的范围

+ EC\_OPTION\_READ\_ONLY, // 属性只读（不支持修改此属性）

+ EC\_OPTION\_SYSTEM\_ERROR, // 属性修改/读取时发生系统错误

};

+ //

+ // PFS 系统属性接口

+ //

+ // SetOption: 设置PFS系统属性

+ // GetOption: 读取PFS系统属性

+ // 参数:

+ // group - 属性组ID，参见PFS\_OPTION\_GROUP定义

+ // name - 指定组下的属性标志

+ // value - 指定组下对应的属性的属性值

+ //

+ // 返回值:

+ // 对于SetOption,设置成功时返回true，对于只读属性，执行SetOption操作，总是会返回false，PFS LastError会设置EC\_OPTION\_READ\_ONLY.

+ // 对于GetOption,读取成功时返回true. 对于不存在的属性，总是返回false，PFS LastError会设置EC\_OPTION\_INVALID\_GROUP或EC\_OPTION\_INVALID\_NAME.

+ static bool SetOption( PFS\_OPTION\_GROUP group, const std::wstring& name, const std::wstring& value );

+ static bool GetOption( PFS\_OPTION\_GROUP group, const std::wstring& name, std::wstring& value );

+

+ // 开启或关闭BaseFilter。

+ // 为了不影响效率，缺省情况下不开启BaseFilter，除非显式开启.

+ // 也可以通过global属性 base-filter 开启或者关闭

+ // 开启或关闭成功返回true，开启或关闭失败返回false。

+ static bool EnableBaseFilter(bool bEnable);

+

+ // 获取BaseFilter开启与否的状态

+ // BaseFilter开启返回true，没有开启返回false。

+ static bool IsBaseFilterEnabled();

+

+ // 设置/获取base目录的优先替换目录(过滤器方式)

+ // 这2个接口也可以通过SetOption/GetOption(PFS\_OPTION\_BASE\_FILTER,baseDirOriginal,baseDirNew)接口进行操作。

+ // 但是，都必须通过EnableBaseFilter(true)开启或通过EnableBaseFilter(false)关闭。

+ // filter不存在时，GetBaseFilter()返回false，GetLastError()返回 EC\_FILTER\_ITEM\_NOT\_EXISTS。

+ static bool SetBaseFilter( const std::wstring& baseDirOriginal, const std::wstring& baseDirNew );

+ static bool GetBaseFilter( const std::wstring& baseDirOriginal, std::wstring& baseDirNew );