Gem #118: Аспекты переносимости имён в файловых системах и GNATCOLL.VFS

Автор: Emmanuel Briot, AdaCore

Краткое содержание: В Gem #118 обсуждаются некоторые аспекты переносимости, связанные с файловыми системами, в частности, вопросы имён файлов, работа с чувствительными и нечувствительными к регистру файловыми системами, наборами символов и символическими ссылками. Кроме того, представляется рабочий пакет GNATCOLL.VFS.

Давайте начнём…

Один из важных вопросов, чтобы адресоваться при портировании кода от одной системы до другого является одним файловых систем.

Есть несколько аспектов обработки имен файлов, которые варьируются через файловые системы. Для одного файловая система могла бы быть чувствительной к регистру или нечувствительной к регистру. Это относится к тому, релевантно ли преобразование регистра имени при доступе к файлу на диске. Кроме того, файловая система может быть сохранением случая или нет. В некоторых системах имена файлов преобразовываются в верхний регистр систематически, когда они выведены на экран (MS-DOS, и VMS находятся в той категории). Системы, которые не сохраняют преобразование регистра, всегда нечувствительны к регистру.

В результате есть три категории файловых систем: "чувствительное к регистру сохранение" (большинство файловых систем Unix), (NTFS) "нечувствительное к регистру сохранение" и "нечувствительный к регистру разрушительный" (FAT и VMS).

При работе в файловой системе, не учитывающей регистр, приложения должны отображать имена файлов с тем же корпусом, который пользователь использовал при их создании. Тем не менее, многие приложения, перенесенные из Unix, просто преобразуют все имена файлов в нижний регистр (чтобы обеспечить уникальность имен файлов внутри) и, таким образом, отображать, что беспокоит пользователя. На практике наш опыт показывает, что имена файлов должны быть преобразованы в нижний регистр при сравнении имен (например, чтобы выяснить, относятся ли два имени к одному файлу или при вычислении хэша). В остальное время корпус всегда должен быть сохранен.

По правде говоря, приведенное выше введение недостаточно точно: атрибуты для корпуса действительно зависят от файловой системы, а не от операционной системы (Windows, Linux, ...). Например, возможно, что ваша машина смонтировала удаленную файловую систему с разными свойствами (например, раздел Windows, установленный на машине Linux). Apple OS-X является особым случаем здесь, поскольку его значение по умолчанию должно быть чувствительным к регистру, но пользователи могут сделать файловую систему нечувствительной к регистру. Все это показывает, что тестирование системы, на которой вы работаете, на практике недостаточно. К сожалению, мы не нашли хороший способ протестировать файловую систему динамически (не говоря уже о том, что это было бы очень дорого, так как для каждого файла нужно было бы проверить, для какой файловой системы он включен).

Еще одна проблема с именами файлов - это набор символов, в котором они закодированы. Когда имя файла содержит только символы ASCII, обычно нет проблем с манипулированием именем. Однако в большинстве систем для использования акцентированных символов в именах файлов действительно. Но некоторые файловые системы не форсируют кодировку и просто видят имя файла в виде серии байтов, интерпретация которых остается для приложений (проводник Windows, терминалы и т. д.), которые будут отображать имя. В общем, эти приложения будут учитывать языковой стандарт пользователя для отображения. Другие файловые системы всегда интерпретируют имена файлов как UTF-8. Опять же, для приложения, чтобы получить это точно,

потребуется тестирование файловой системы для каждого файла, а не просто тестирование самой системы и принятие некоторых значений по умолчанию.

Другая проблема - символические ссылки. Во многих файловых системах доступ к файлам осуществляется через разные пути при использовании символических ссылок. Хотя эти ссылки не влияют при открытии и чтении файла, они усложняют проверку того, относятся ли два пути к одному и тому же физическому файлу. Это можно сделать, проверив каждый компонент пути, чтобы узнать, является ли он ссылкой, и если это так, преобразуйте его в нормализованную форму. Это вычисление может быть дорогостоящим (особенно в медленных или удаленных файловых системах), поэтому его результат должен быть кэширован, когда это возможно.

GNAT Components Collection (GNATCOLL) представляет собой полезный пакет для абстрагирования таких аспектов, а именно GNATCOLL.VFS. Этот пакет предоставляет несколько типов, которые используются для управления файлами и их именами:

```
type Filesystem_String is new String;
type Virtual_File is tagged private;
```

Первый тип, указанный выше, предназначен для первоначальной замены строк, которые обычно используются для представления имени файла. Нет конвертации в Юникод или из него. Цель состоит в том, чтобы напомнить пользователям, что точная интерпретация не должна быть строкой, которая может отображаться как есть, а рядом байтов, которые необходимо интерпретировать в контексте определенного набора символов (чаще всего UTF-8, но также ISO -8859-1 и варианты).

Использование второго типа предполагает большее изменение для большинства приложений: идея заключается в том, что он инкапсулирует и кэширует различную информацию о файле и его имени и тем самым абстрагирует понятия, такие как чувствительность к регистру.

Рассмотрим некоторые примеры. Сначала нам нужно получить представление для файла с диска. Для этого мы можем использовать одну из функций Create, доступных в GNATCOLL. Например,

```
declare
   F : Virtual_File;
begin
   F := Create ("/tmp/Foo.txt");
end;
```

Если мы передадим F в какую-либо подпрограмму, которая должна отображать ее в графическом интерфейсе, например, мы ожидаем, что имя будет отображаться точно так же, как «Foo.txt», а не как версия с полным кодом «foo.txt», даже на файловую систему без регистра. Чтобы получить имя, мы могли бы, например, написать:

```
declare
   Name : constant Filesystem_String := F.Base_Name;
begin
   Put_Line (+Name);
end;
```

Как отмечалось ранее, имя следует рассматривать как ряд байтов, интерпретация которых зависит от вашей системы. В большинстве случаев относительно безопасно предположить, что это UTF-8. В этом случае GNATCOLL.VFS предоставляет оператор «+» для преобразования файловой системы в String.

Если мы создадим еще один экземпляр Virtual_File, мы можем проверить, соответствуют ли две ссылки одному и тому же файлу. Результат будет справедливым для Windows, например, но не для Unix.

```
declare
   F2 : constant Virtual_File := Create ("/tmp/foo.txt");
begin
   if F = F2 then
      null;
   end if;
end;
```

В Unix мы могли бы создать символическую ссылку из "/temp" к "/tmp". Если мы хотим, чтобы наше приложение правильно поддерживало символические ссылки и распознавало что "/temp/foo.txt" и "/tmp/foo.txt" действительно тот же файл, нам нужно сказать GNATCOLL, что мы готовы заплатить потерей производительности, вызывая:

```
GNATCOLL.VFS.Symbolic_Links_Support (True);
```

По умолчанию эта поддержка отключена. При загрузке большого проекта в GPS, например (с несколькими тысячами файлов), в медленной файловой системе (ClearCase), не проверять явно для символических ссылок, по крайней мере, на порядок быстрее. Вот почему это остается явным выбором для приложения.

Однако GNATCOLL достаточно умен, чтобы кэшировать символическое разрешение, а также нормализовать имя файла. Поэтому, если вы повторно используете Virtual_File несколько раз, ему больше не нужно будет выполнять системные вызовы.

API в GNATCOLL.VFS гораздо более обширен, чем то, что мы видели выше, и предоставляет способы проверить, есть ли у нас каталог, является ли файл доступным для записи, эффективно считывает содержимое файла в память, получает список файлов в каталог и даже изменять файлы. В каждом из этих случаев GNATCOLL будет обеспечивать правильную форму имени файла при общении с системой.

На самом деле GNATCOLL.VFS также обеспечивает поддержку удаленных файловых систем (это основа удаленного режима в GPS), где сетевые операции выполняются прозрачно при доступе к файлу, но это будет предметом другого Gem.

Преобразование приложения в GNATCOLL.VFS - это небольшая работа. Но это обеспечивает ряд преимуществ с точки зрения мобильности и производительности.

Связанный со статьёй текст программы

Attached Files отсутствуют

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Emmanuel Briot
AdaCore

Emmanuel Briot был с AdaCore с 1998 года. Он участвовал в различных проектах, в частности, ориентированных на графические пользовательские интерфейсы, включая GtkAda, GPS, XML / Ada,

GnatTracker и наш внутренний CRM. Он имеет инженерную степень в Ecole Nationale des Telecommunications (Брест, Франция).

Last Updated: 10/13/2017
Posted on: 2/13/2012

Обсуждение…