Разработка приложения для архивирования файлов в виде Windows Service

Лабораторная работа

Ревизия: 0.1

Последняя модификация: Vlad Kovtun

Дата последней модификации: 24.09.2010 19:36:00

История изменений

19.09.2010 – Версия 0.1. Первичный документ. Влад Ковтун

Содержание

Лабораторная работа 10. Разработка приложения для архивирования файлов в виде Windows Service

Вопросы

- Постановка задачи.
- Методические рекомендации.

Постановка задачи

Цель

1. Разработать консольное серверное приложение архивирования файлов в виде Windows-сервиса.

2. Разработать консольное клиентское приложение, которое взаимодействует с пользователем и передает необходимые команды Windows-сервису для архивирования и разархивирования файлов.

Задачи

1. Ознакомиться с особенностями построения Windows-сервиса, его установки в систему, запуска, остановки и удаления из системы [2, 3].

2. Ознакомиться с классом GZipStream для архивирования и разархивирования файлов алгоритмом ZIP входящей в состав Microsoft.Net 2.0 [1].

3. Ознакомиться с этапами разработки сервисов Windows [3].

4. Ознакомиться с особенностями межпроцессного взаимодействия (между пользовательским приложением и Windows-сервисом), например посредством технологии .NET Remoting [12, 13].

5. Оценить производительность, при различном количестве и размере файлов (для процессов архивирования и разархивирования).

6. Оформить отчет к лабораторной работе. В отчете обязательно указать, каким образом производится тестирование на корректность.

Задание

Опишем основные ключи командной строки, которые следует реализовать для клиентского приложения.

Для управления клиентским приложением предлагается использовать ключи командной строки:

- /id:<filedir> – полный сетевой путь к директории, в которой хранятся файлы для архивирования или разархивирования. Данный параметр является обязательным, если он не указан, то происходит вывод на экран соответствующего сообщения и подсказки по использованию данного приложения.
- /if:<filenames> - перечисление имен файлов через пробел, которые следует архивировать или разархивировать. Файлы обязаны находиться в директории указанной с помощью ключа /id. Данный ключ позволяет осуществлять выборочное архивирование или разархивирование файлов.
- /od:<filedir> - полный сетевой путь к директории, в которую следует заносить разархивированные файлы. Данный параметр не является обязательным, если он не указан, то разархивированный файлы располагаются в директории, где храниться и сам архив.
- /o:<filename> – полный путь к файлу, либо имя файла, который будет хранить архив.
- /<mode> - указывается режим использования приложения – архивирование (параметр /c (compress)) либо разархивирование (параметр /d (decompress)).
- /? - вывод информации о допустимых ключах командной строки.

Пример

Архивирование 3 файлов, результат архивирования выводится в файл `commonarch.zip`. Отметим, что файловое хранилище расположено на публичном хранилище

\\192.168.1.12\logs. В данном случае, файл архива располагается в директории, что и исходные файлы для архивирования.

```
C:/>winzipper.exe /id:\\192.168.1.12\logs /if access-www.1.log access-www.2.log access-www.3.log /c /o:commonarch.zip
```

Разархивирование файла commonarch.zip, расположенного в публичном хранилище \\192.168.1.13\dat. Результирующие файлы, полученные в результате процесса разархивирования, будут расположены на публичном хранилище \\192.168.1.12\logs.

```
C:/>winzipper.exe /id:\\192.168.1.12\dat /if commonarch.zip /d /od:\\192.168.1.12\logs
```

Вывод

Во время работы приложения клиентского приложения, рекомендуется выводить информацию о статусе приложения, а также о корректности его работы на консоль.

Допускается перенаправление вывода консоли в текстовый файл, например:

```
C:/>winzipper.exe /id:\\192.168.1.12\dat /if commonarch.zip /d /od:\\192.168.1.12\logs >log.txt
```

Требования

- Архитектура приложения строится по модульному принципу.
- За основу принимается стандартная библиотека C++ (в случае разработки на C++).
- Рекомендуется использовать защищенные ресурсы и указатели.
- Обязательным является обработка исключений.
- Исходный код обязан быть комментирован. Для C++ следует использовать нотацию doxygen [9]. Для C# следует использовать нотацию Microsoft Visual Studio [8].
- Допускается запуск единственного процесса архивирования и разархивирования, другими словами следующий процесс запускается лишь после завершения предыдущего, такой подход позволит существенно упростить Windows-сервис, т.е. позволит уйти от написания многопоточной обработки.

Методические указания для самостоятельной работы

Создание Windows-сервиса

Процесс создания Windows-сервиса детально описан в MSDN [3], однако существует не менее удачное описание [10, 11].

Добавление возможности инсталляции и деинсталляции сервиса в ОС

В уже созданном проекте Windows-сервиса, следует:

1. Перейти в Solution Explorer и открыть дерево проекта Windows-сервиса.

2. Перейти в режим Design отображения файлов. Для этого следует выбрать файл <имя сервиса>.cs и в контекстном меню, доступном для этого файла, следует выбрать View Designer.

3. В режиме Design, следует в контекстом меню, доступном для этого окна, следует выбрать Add Installer. После чего, автоматически будут добавлены два компонента: `ServiceProcessInstaller` и `ServiceInstaller`.

4. Для свойства `StartType` компонента `ServiceInstaller` следует выбрать режим запуска сервиса, например `Manual`.

Инсталляция и деинсталляция сервиса в ОС

Для ручной инсталляции Windows-сервиса, следует воспользоваться утилитой InstallUtil.exe, которая доступна через Visual Studio Command Prompt. В поставку Microsoft Visual Studio обязательно входит утилита Visual Studio Command Prompt, которая формирует корректный набор переменных окружения, для удобства работы с различными инструментами Visual Studio.

В связи с этим, следует перейти: Start->Programs->Microsoft Visual Studio 2005->Visual Studio Tools и запустить Visual Studio 2005 Command Prompt.

Ручная инсталляция сервиса

1. Перейти в папку, где расположен скомпилированный Windows-сервис, например в папку `WinZipService/Bin/Debug/`

2. Запустить утилиту InstallUtil.exe из командной строки, например

```
InstallUtil.exe WinZipService.exe
```

Ручная деинсталляция сервиса

1. Перейти в папку, где расположен скомпилированный Windows-сервис, например в папку `WinZipService/Bin/Debug/`

2. Запустить утилиту InstallUtil.exe из командной строки, например

```
InstallUtil.exe /u WinZipService.exe
```

Создание простейшего приложения для работы с ZIP архивами

Основу приложения по работе с ZIP архивами составляет класс `GZipStream`, который входит в область видимости `System.IO.Compression`.

Реализация межпроцессного взаимодействия

Основу межпроцессного взаимодействия составляет технология Microsoft .NET Remoting [12, 13].

.NET Remoting позволяет клиентскому приложению создать объект (именуемый remotable object) доступный в рамках remoting boundaries и расположенный в домене приложения внутри windows-сервиса, исполняющемся на этом компьютере, или даже на другом компьютере, соединённом сетью. Процесс .NET Remoting содержит приёмник запросов к объекту в домене windows-сервиса. На стороне клиента, любые запросы к удалённому объекту направляются средой выполнения .NET Remoting через объекты Channel, являющиеся обёрткой для средств транспортного уровня, таких как потоки TCP, потоки HTTP и именованные каналы. В результате, запросы к удаленным объектам для клиентского кода ничем не отличаются от локальных вызовов, а созданием экземпляра нужного Channel-объекта, приложение .NET Remoting можно без перекомпиляции перевести на другой коммуникационный протокол. Среда выполнения сама по себе выполняет этапы сериализации и маршалинга объектов в среде между клиентским и серверным доменами приложения.

В процессе выполнения запросов любые вызовы методов, направленные объекту, включая идентификатор метода и любые передаваемые параметры, сериализуются в байтовый поток и передаются посредством канала связи, реализованного для конкретного протокола, принимающему прокси-объекту на серверной стороне («маршализируются»). Передача происходит путём записи данных в транспортный ввод канала. На серверной стороне прокси читает поток данных из вывода канала и выполняет вызов удалённого компонента от лица клиента. Результаты сериализуются и передаются через канал клиенту, где прокси читает результат и передаёт его вызывающему приложению. Если удалённому объекту нужно обеспечить обратный вызов (callback) клиентскому объекту, клиентский объект обратного вызова должен быть помечен как remotable, а инфраструктура .NET Remoting должна быть сконфигурирована на создание прослушивателя для него. Сервер может подключиться к нему по другому каналу, или по уже существующему если соединение, на котором он основан, поддерживает двунаправленный обмен данными. Канал может быть составлен из нескольких канальных объектов, возможно даже с разными транспортными механизмами. Таким образом, система, основанная на .NET Remoting, может состоять из нескольких подсистем, связанных подключёнными друг к другу гетерогенными сетями, включая Интернет.

Тестирование

Тестирование приложения осуществляется в несколько этапов:

- Разработка Unit Test в рамках проекта. Данный подход позволит проверить корректность реализации алгоритма.
- Воспользоваться тестовыми пакетными файлами для проверки корректности функционирования.

- Разработать простейшее приложение, которое формирует большие тестовые пакетные файлы, которые затем использовать для тестирования основного приложения.

Теоретические сведения

Архитектура

Данное приложение построено согласно клиент-серверной архитектуре, где Windows-сервис является сервером, а приложение, которое взаимодействует с пользователем – клиентское. Процесс архивирования и разархивирования изображен на рис. 1 и 2, соответственно.

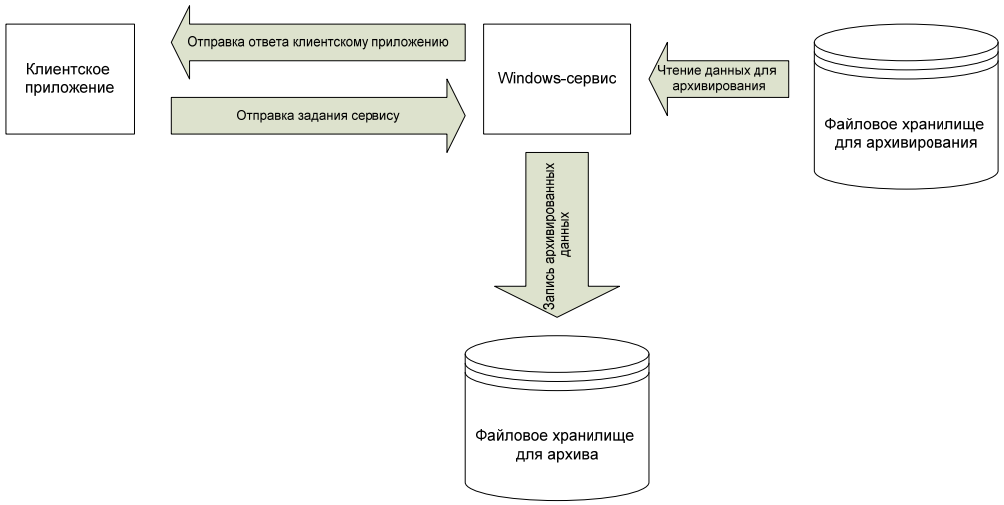

Рис. 1. Процесс архивирования файлов

Рис. 2. Процесс разархивирования файлов

Алгоритм

Исходя из архитектуры приложения, опишем алгоритм Windows-сервиса и алгоритм пользовательского приложения.

Алгоритм Windows-сервиса

1. Главный цикл обработки сообщений, которые могут передаваться как с локального, так и удаленного компьютера. Реализуется посредством Microsoft Service Controller (для

команд по работе сервисом) и технологии .NET Remoting (для выполнения особых действий по архивированию и разархивированию файлов).

1.1. Получение сообщение на архивирование файла.

1.2. Проверка сообщения на корректность.

1.3. Уведомление удаленного компьютера о корректности принятых данных и запуске задания на выполнение.

1.4. В зависимости от типа сообщения производятся различные действия.

1.4.1. Старт сервиса.

1.4.1.1.Создание серверного канала TCP/IP с указанием порта, для принятия подключений.

1.4.1.2. Регистрация канала в системе.

1.4.1.3. Создание серверного объекта .NET Remoting для приема подключений удаленных клиентов.

1.4.1.4. Регистрация объекта .NET Remoting в системе, что приводит к запуску цикла прослушивания обращений к серверу .NET Remoting.

1.4.2. Стоп сервиса.

1.4.3. Выполнение особого действия (выполнение функции сервера .NET Remoting). Запуск функции архивирования файла.

1.4.3.1. Проверка на существование файла, который следует заархивировать.

1.4.3.2. Открытие потока файла на чтение.

1.4.3.3. Открытие протока файла, для последующей записи туда заархивированных данных.

1.4.3.4. Открытие потока с поддержкой ZIP архивирования, с последующей записью туда данных из открытого потока файла, этап 1.4.3.2.

1.4.3.5. Чтение блоков двоичных данных из потока исходного файла и запись их в ZIP поток.

1.4.3.6. Закрытие потока с поддержкой ZIP архивирования.

1.4.3.7. Закрытие потока файла с результирующими данными.

1.4.3.8. Закрытие потока файла с исходными данными.

1.4.4. Выполнение особого действия (выполнение функции сервера .NET Remoting). Запуск функции разархивирования файла.

1.4.4.1. Проверка на существование ZIP-файла, который следует разархивировать.

1.4.4.2. Открытие потока ZIP-файла на чтение.

1.4.3.3. Открытие протока файла, для последующей записи туда разархивированных данных.

1.4.4.4. Открытие потока с поддержкой ZIP архивирования, для последующего чтения оттуда данных из открытого потока файла, этап 1.4.4.2.

1.4.4.5. Чтение блоков двоичных ZIP-данных из потока исходного файла и запись их в результирующий (разархивированный) поток.

1.4.4.6. Закрытие потока с поддержкой ZIP архивирования.

1.4.3.7. Закрытие потока файла с результирующими данными.

1.4.3.8. Закрытие потока файла с исходными данными.

Алгоритм клиентского приложения

1. Анализ корректности параметров командной строки.

2. Создается экземпляр клиентского TCP/IP подключения.

3. Создание прокси-объекта сервера .NET Remoting.

4. В зависимости от параметров командной строки вызывается соответствующий метод прокси-объекта сервера .NET Remoting.

5. Получение статуса вызова соответствующего вызова.

Литература

1. Microsoft Developer Network. URL: http://www.msdn.com

2. MSDN. Introduction to Windows Service Applications

3. MSDN. How to: Create Windows Services.

4. MSDN. How to: Add Installers to Your Service Application

5. MSDN. How to: Install and Uninstall Services

6. MSDN. How to: Start Services

7. MSDN. How to: Debug Windows Service Applications

8. MSDN. XML Documentation Comments.

9. Формирование документации к исходному коду с помощью средства doxygen. URL: www.nrjetix.com/r-and-d/lectures

10. Simple Windows Service Sample. URL: http://www.codeproject.com/KB/dotnet/simplewindowsservice.aspx

11. Creating a simple Windows Service. URL: http://www.codeproject.com/KB/TipsnTricks/SimpleWindowsService.aspx

12. Absolute beginner's introduction to .NET Remoting. URL: http://www.codeproject.com/KB/IP/absoluteremoting.aspx

13. Simple but potentially useful example of .NET Remoting. URL: http://www.codeproject.com/KB/IP/processactivator.aspx