][-РЕЛИЗ:

НАШ DROPBOX НА WINDOWS AZURE

Василий Гай
vasiliy.gai@gmail.com

Артём Гончаров

Разрабатываем облачное файловое хранилище

Облачные сервисы хранения данных сейчас не использует только ленивый. Dropbox, SkyDrive, Bitcasa… перечислить все не представляется возможным. Мы расскажем, как сделать на Windows Azure файловое хранилище с небольшим количеством свистелок и плюшек.

ВВЕДЕНИЕ В WINDOWS AZURE

Windows Azure — одна из платформ для реализации облачных сервисов. При разработке проекта из всего множества сервисов, предоставляемых Azure, мы использовали только два: Applications (среда выполнения приложений) и Data Management (сервис хранения данных).

Приложение, выполняемое в Azure, представляет собой облачный сервис. Существует три типа облачных сервисов (ролей): worker-роль, web-роль и vm-роль. Web-роль обычно используется для создания веб-приложений, worker-роль — для выполнения длительных вычислений в фоновом режиме, vm-роль — образ операционной системы.

В рамках одного проекта в облаке может взаимодействовать несколько экземпляров web- и worker-ролей. При этом для выполнения каждой роли выделяется отдельная виртуальная машина, которая создается при развертывании сервиса в облаке.

Сервис хранения данных позволяет хранить объекты нескольких типов: таблицы (структурированное хранилище), бинарные объекты, очереди (используются для обмена информацией между ролями), диски.

СТРУКТУРА ПРОЕКТА

В разработанный проект облачного файлового хранилища входит клиентское приложение и сервер, выполняющийся в облаке (в worker-роли). Для организации взаимодействия клиента и сервера мы выбрали технологию Windows Communication Foundation (WCF). Разработанный WCF-сервис реализует дуплексный контракт, что позволяет клиенту и сервису обмениваться сообщениями, вызывая операции друг друга. Дуплексный контракт определяется в виде пары интерфейсов:

1. интерфейс от клиента к сервису (IServiceOperations — операции, которые клиент может вызывать на сервисе);
2. интерфейс обратного вызова от сервиса к клиенту (IClientOperations — операции, которые сервис может вызывать на клиенте), см. атрибут ServiceContract.

Атрибут ServiceBehavior описывает поведение сервиса. Значение параметра InstanceContextMode устанавливает время жизни экземпляров сервиса, параметр ConcurrencyMode — степень параллелизма, то есть количество

запросов, которые могут быть одновременно направлены к одному экземпляру сервиса. Установленное сочетание параметров указывает, что создается один экземпляр сервиса, с которым параллельно может работать несколько потоков. Учитывая, что к сервису одновременно обращается несколько потоков, доступ к локальным переменным сервиса должен осуществляться с учетом требований потокобезопасности.

Описание поведения сервиса

```
[ServiceBehavior(
    InstanceContextMode = InstanceContextMode.Single,
    ConcurrencyMode = ConcurrencyMode.Multiple,
    IncludeExceptionDetailInFaults = false,
    AddressFilterMode = AddressFilterMode.Any)]
```

Описание контракта обратного вызова

```
[ServiceContract(
    CallbackContract = typeof(IClientOperations),
    SessionMode = SessionMode.Required)]
```

Значение параметра SessionMode указывает, что обмен сообщениями между клиентом и сервером — часть одного диалога (сеанса).

Взаимодействие между клиентом и сервисом инициирует клиент, который вызывает на сервисе операцию Register. Эта операция отмечена атрибутом IsInitiating = true, это указывает на то, что только данная операция может начинать сеанс обмена сообщениями.

Операция Register, в зависимости от переданных ей параметров, выполняет создание аккаунта пользователя на сервисе или регистрацию пользователя в списке клиентов.

В сервисе для описания информации о подключенных клиентах (о сессиях) используется класс ClientInformation, который включает следующие сведения о клиенте:

1. Идентификатор сессии.
2. Имя пользователя.
3. Контракт обратного вызова.
4. Уникальный хеш клиента.
5. Имя файла, с которым клиент выполняет действие.

Поскольку несколько клиентов могут выполнять чтение и обновление информации о сессиях одновременно, для организации доступа к списку сессий используется класс ReaderWriterLockSlim. Данный класс предоставляет блокировку, которая позволяет организовать доступ к ресурсу таким образом, что выполнять чтение могут одновременно несколько потоков, а монопольно записывать — только один. Методы EnterWriteLock/ExitWriteLock выполняют вход в блокировку (выход из нее) в режиме записи, а методы EnterReadLock/ExitReadLock — в блокировке в режиме чтения.

Хранение данных на сервере (в Azure Storage) мы организовали следующим образом:

1. Информация об учетных записях пользователей хранится в таблице users (в Table Storage).
2. Файлы, закачиваемые клиентом, записываются в Blob Storage. В Blob Storage хранение данных организовано в виде двух уровней: контейнер и блоб, причем в одном контейнере может храниться несколько блобов, каждый из которых должен иметь уникальное имя; в нашем проекте имя контейнера соответствует имени учетной записи пользователя, имя блоба — имени файла (каталога), который закачивается в блоб.

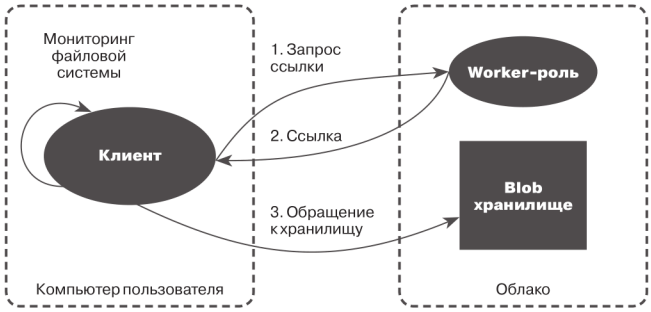
ОРГАНИЗАЦИЯ ДОСТУПА К ХРАНИЛИЩУ

Клиент, выполняя синхронизацию каталога с облаком, взаимодействует с облачным хранилищем, для доступа к которому нужен ключ (primary access key). Хранить ключ в клиенте (даже применяя разные методики шифрования) небезопасно. Решение этой проблемы заключается в использовании технологии Shared Access Signatures (SAS). Она позволяет отделить код, который выполняет аутентификацию в хранилище, от кода, который выполняет управление данными в хранилище. Таким образом, на стороне клиента работа с блоб-хранилищем выполняется без ключа. На рисунке показана схема обращения клиента к хранилищу.

Аналогичным образом мы реализовали алгоритм регистрации пользователя на сервисе: доступ к хранилищу выполняется на стороне сервиса, который хранит ключ доступа.

Создание ссылки на доступ к контейнеру

```
[SecurityCritical]
public string GetLinktoBlob(string userName, string fileName) {
    // Инициализация ссылки на аккаунт хранилища
    CloudStorageAccount storageAccount = ↩
    CloudStorageAccount.Parse(storageConnectionString);
```




```
    CloudBlobClient blobClient = ↩
    storageAccount.CreateCloudBlobClient();

    // Создание ссылки на контейнер, для которого будет
    // создана сигнатура распределенного доступа
    CloudBlobContainer container = ↩
    blobClient.GetContainerReference(userName);

    // Создание контейнера
    container.CreateIfNotExist();

    // Описание прав на доступ к контейнеру
    BlobContainerPermissions containerPermissions = ↩
    new BlobContainerPermissions();

    // Установка свойства приватности контейнера,
    // контейнер недоступен в режиме анонимного доступа
    containerPermissions.PublicAccess = ↩
    BlobContainerPublicAccessType.Off;

    // Применение политик безопасности к контейнеру
    container.SetPermissions(containerPermissions);

    // Создание ссылки на доступ к контейнеру
    string sas = container.GetSharedAccessSignature(new ↩
    SharedAccessPolicy(){
        // Установка времени активности ссылки
        SharedAccessExpiryTime = DateTime.UtcNow.↩
        AddMinutes(30),
        // Установка прав на доступ к контейнеру
        Permissions = SharedAccessPermissions.Write | ↩
        SharedAccessPermissions.Read | ↩
        SharedAccessPermissions.Delete | ↩
        SharedAccessPermissions.List
    });
    return sas;
}
```

Рассмотрим алгоритмы, которые потребуются для реализации синхронизации локального каталога и облачного хранилища. При разработке этих

МОНИТОРИНГ ФАЙЛОВОЙ СИСТЕМЫ

Один из возможных способов отслеживать изменения в каталогах или файлах основан на использовании класса FileSystemWatcher. Данный класс позволяет обрабатывать уведомления файловой системы, возникающие при создании, удалении, переименовании, изменении файлов (каталогов).

WWW

Сорцы ищи не на диске, а на сайте. Просто мы хотим тут кое-что причесать, голых женщин добавить и поэтому в сроки сдачи диска точно не уложимся :)

ТРИАЛЬНЫЙ ДОСТУП К AZURE

Бесплатный 90-дневный доступ к Azure можно получить, перейдя по ссылке goo.gl/ifLwK. Для регистрации потребуется банковская карта. Здесь goo.gl/3XGuC описан способ использования виртуальной карты взамен реальной.

алгоритмов мы учитывали, что одновременно к серверу в рамках одного аккаунта может быть подключено несколько клиентов.

ЗАГРУЗКА ФАЙЛА В ОБЛАКО

Процесс загрузки файла в облако состоит из двух шагов: определения возможности загрузки и собственно загрузки файла.

В общем случае под одним аккаунтом на сервер могут зайти несколько клиентов (примем для определенности, что таких клиентов два: «Клиент А» и «Клиент Б»). В такой ситуации может возникнуть ошибка синхронизации загружаемых файлов. Допустим, оба клиента открыли локально один и тот же файл. После выполнения определенных действий «Клиент А» завершает работу с файлом и сохраняет его на диск. Код, отвечающий за мониторинг изменений файловой системы, обнаруживает изменение и загружает измененный файл в облако. Однако в это же самое время «Клиент Б» продолжает редактировать тот же файл. Поэтому, если не предусмотреть описанную ситуацию, один из клиентов может потерять данные.

Реализованный способ разрешения описанного конфликта выглядит следующим образом. Клиент, загружающий файл, должен с помощью сервиса обратиться к другим клиентам, выполняющимся в рамках того же аккаунта (если такие клиенты вообще есть), и определить, доступен ли в их локальном каталоге на запись файл, который он пытается загрузить:

- если доступ есть — файл загружается в хранилище;
- если доступа нет — файл загружается в хранилище с другим именем: <имя_файла>_<хеш_клиента>.расширение.

Описанный подход позволяет сохранить копии файла, созданные на каждом клиенте. Данный подход можно использовать при разрешении конфликтов для любого количества клиентов в рамках одного аккаунта.

Процесс загрузки файла в облако состоит из двух шагов.

1. Загрузка файла в блоб, имя которого формируется с помощью префикса UPD_ и имени файла;

```
Загрузка файла в облако
using (var fileStream = System.IO.File.OpenRead(fileName)) {
    blob.UploadFromStream(fileStream);
}
```

2. Переименование блоба после окончания загрузки (из имени блоба удаляется префикс UPD_).

В Windows Azure не реализован алгоритм переименования блоба, в связи с этим операция его переименования состоит из двух шагов: копирование существующего блоба в блоб с новым именем и удаление старого блоба (еще пара блобов в этом предложении — и мой мозг бы взорвался. — Прим. ред.).

```
Переименование блоба
CloudBlob existBlob = container.GetBlobReference(names[0]);
CloudBlob newBlob = container.GetBlobReference(names[1]);

// Копирование блоба
newBlob.CopyFromBlob(existBlob);

// Удаление блоба
existBlob.Delete();
```

ГРАНТ ОТ МАЙКРОСОФТ

Майкрософт предоставляет для образовательных учреждений грант на получение бесплатного доступа к Windows Azure. Для этого нужно перейти по ссылке goo.gl/186FJ и заполнить небольшую анкету.

Описанный процесс позволяет создать защиту от ошибок, которые могут произойти при загрузке файла в облако, поскольку в рассматриваемом алгоритме загрузка считается завершенной только после того, как из имени блоба будет удален префикс UPD_.

При загрузке файла в блоб учитывается, что в хранилище может находиться блоб с таким же именем, но помеченный на удаление. Поэтому блоб, помеченный на удаление, перед закачкой файла в облако нужно будет удалить.

ЛОГ ОПЕРАЦИЙ

Клиент, загрузив или удалив файл из облака, должен сообщить об этом другим клиентам. Для этого создаётся лог операций. Хранится лог в таблице, которая размещается в Azure SQL DataBase. Структура таблицы содержит информацию о клиенте, который выполнил операцию, объекте операции, а также служебную информацию.

ВЫГРУЗКА ДАННЫХ ИЗ ОБЛАКА

Алгоритм выгрузки из облака на диск разработан с учетом того, что до выгрузки необходимо создать структуру каталогов для хранения файлов.

```
Выгрузка файла на диск
using (var fileStream = System.IO.File.OpenWrite(filename)) {
    blob.DownloadToStream(fileStream);
}
```

СРАВНЕНИЕ СОДЕРЖИМОГО ДИСКА И ОБЛАКА

Периодически сравнивая (с помощью таймера) содержимое синхронизируемого каталога и облачного хранилища, клиентское приложение генерирует четыре списка файлов (каталогов):

1. файлы (каталоги), которые нужно удалить с диска (данные файлы помечены в облаке флагом удаления);
2. файлы (каталоги), которые нужно удалить из облака (файлы такого типа отсутствуют на диске, но присутствуют в облаке);
3. файлы (каталоги) для загрузки в облако (дата последнего изменения файла на диске позже даты последнего изменения файла в облаке);
4. файлы (каталоги), которые нужно выгрузить из облака на локальный диск (дата последнего изменения файла в облаке позже даты последнего изменения файла на диске).

После формирования списков запускается синхронизация каталога и облака, которая представляет собой последовательность операций загрузки и удаления файлов (каталогов).

```
Описание таймера
System.Timers.Timer mainTimer = new System.Timers.Timer();
// Описание метода, вызываемого при срабатывании таймера
mainTimer.Elapsed += new ElapsedEventHandler(OnTimerEvent);
// Установка времени срабатывания таймера
mainTimer.Interval = TimerInterval;
mainTimer.Enabled = true;
// Использование метода KeepAlive для предотвращения
// обработки таймера механизмом сборки мусора
GC.KeepAlive(mainTimer);
```

РАЗРЕШЕНИЕ КОНФЛИКТОВ ДОСТУПА

Разрабатывая многопользовательское приложение, мы учитывали возможность возникновения ситуаций, в которых несколько клиентов попытаются одновременно выполнить загрузку или удаление одного и того же файла в хранилище. Механизм разрешения конфликтов доступа у нас сейчас реализован на основе сессий, которые создаются при регистрации клиента на сервере.

Предлагаемый механизм заключается в следующем. Клиент при выполнении операции загрузки или удалении файла должен проанализировать список сессий на сервере. Если хотя бы в одной из сессий встречается имя файла, который клиент собирается удалить или загрузить, выполнение операции над файлом откладывается. Если же имя файла не найдено в списке сессий, клиент обновляет информацию о своей сессии, устанавливая имя файла, с которым он выполняет работу. После выполнения операции с файлом информация об активном файле удаляется.

ЗАКЛЮЧЕНИЕ

Описанный проект включает весь необходимый функционал для организации облачного хранилища файлов. Конечно, ты можешь улучшить наш проект, например повысить производительность сервера за счет увеличения количества экземпляров worker-ролей. Но помни, что в этом случае нужно будет реализовать алгоритм синхронизации данных между этими экземплярами, так как клиенты, подключенные к разным экземплярам worker-ролей, по умолчанию взаимодействовать не могут.