**Использование модели briefcase при разработке приложений баз данных**

**Использование средств ADO.**

Михаил Голованов

**Требования бизнеса по обеспечению работы мобильных пользователей**

Внедрение информационных систем для автоматизации деятельности бизнеса требует от разработчиков баз данных реализации все новых возможностей в разрабатываемых ими приложениях. Создание программного обеспечения, позволяющего пользователям работать лишь в пределах офиса, на сегодняшний день становится явно недостаточно. Сотрудникам офиса требуется обеспечить доступ к информационным массивам фирмы в командировке, из дома, из офиса клиента. При этом пользователи хотят не только просматривать данные, но и иметь возможность вносить в них изменения. Немаловажным требованием со стороны администраторов информационных систем является простота установки и настройки клиентских приложений.

Подводя итог вышесказанного, можно выделить основные требования заказчиков к программному обеспечению для мобильных клиентов информационной системы:

Возможность получения данных, сотрудниками вне пределов офиса фирмы.

Возможность внесения мобильным клиентом изменений в данные, которые затем должны быть синхронизированы с центральной БД.

Возможность работы мобильного клиента при отсутствии постоянного канала связи с офисом.

Простота установки, настройки и эксплуатации созданных приложений.

**Технические проблемы и варианты реализации**

При реализации вышеописанных требований заказчиков возникают следующие технические проблемы:

Обеспечение хранения полученной пользователем информации в перерывах между сеансами связи с центральным офисом с возможностью продолжения работы мобильного пользователя. Другими словами пользователь не должен замечать различий в работе приложения в режимах on-line и off-line.

Синхронизация сделанных пользователем изменений с центральной базой. Поскольку время между редактированием записи мобильным пользователем и внесением ее в центральную базу может составлять дни, недели и даже месяцы применение обычного для модели клиент-сервер механизма блокировок записей не имеет смысла. При этом несколько мобильных пользователей могут одновременно редактировать каждый свою копию данных с сервера, тогда при попытке синхронизации данных неизбежно возникновение конфликтов изменений внесенных в одну и ту же запись различными пользователями. Данные конфликты – это конфликты синхронизации. Обеспечение средств разрешения конфликтов синхронизации– это одно из главных требований к технической реализации такого ПО.

Далее мы кратко рассмотрим наиболее часто применяющиеся варианты решения задачи доступа мобильных пользователей к центральной базе данных.

**Использование Internet и Web**

Web изначально проектировалась как территориально распределенная сеть, позволяющая осуществлять доступ к различным информационным ресурсам в режиме on-line. Основным недостатком данного подхода является необходимость быть все время подключенным к сети. Данный недостаток практически не позволяет применять данный подход для работы мобильных пользователей.

**Репликация баз данных**

Репликация – это процесс синхронизации данных между несколькими серверами БД. При применении данного способа работы архитектура системы выглядит следующим образом:

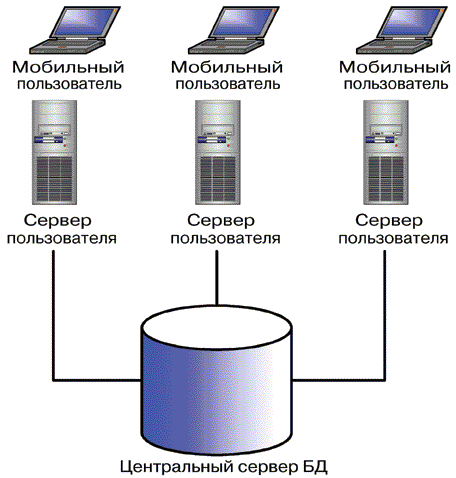


Рисунок 1

Большинство современных серверов БД имеют встроенные средства для поддержки репликации, позволяющие обмениваться данными между несколькими серверами. Клиентское приложение при этом не требует особых модификаций. Главным недостатком такого метода является необходимость установки и обслуживания сервера БД на машине клиента.

Модель работы briefcase

Briefcase модель подразумевает работу клиента с базой данных без поддержки постоянного соединения. Клиент подключается к БД, скачивает требуемые данные, передает сделанные им изменения, и тут же отключается. В Delphi данная модель может быть реализована с использованием возможностей ADO или MIDAS.

При создании приложения, реализующего модель briefcase можно выделить несколько подзадач:

Получение данных с центрального сервера;

Сохранение данных в локальный кэш;

Загрузка данных из локального кэша;

Синхронизация данных с центральным сервером и обработка ошибок синхронизации.

Для наших примеров в качестве сервера БД я использовал MS SQL сервер. На нем была создана база данных ParamsHolder, содержащая всего одну таблицу Params со следующими полями:

|  |  |  |
| --- | --- | --- |
| Поле | Тип | Описание |
| ParamID | Int | Первичный ключ, identity |
| ParamName | Varchar(50) not null | Имя хранимого параметра |
| ParamValue | Varchar(50) | Значение параметра |

Каркас главной формы приложения приведен на рисунке. Я не буду подробно описывать каркас, при необходимости Вы можете обратиться к прилагаемым примерам.

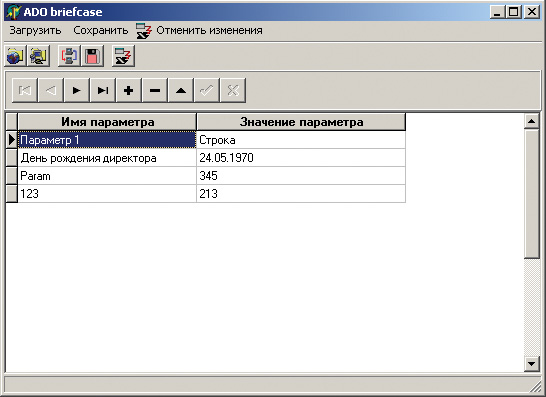


Рисунок 2

Отметим лишь, что компонент подключения к серверу назван ParamConns, а компонент доступа к данным ParamsCS. Сосредоточимся на реализации вышеперечисленных подзадач создания приложения briefcase. Все перечисленные подзадачи реализованы с помощью Action-нов.

Реализация модели briefcase средствами ADO

Поскольку компоненты доступа к данным через ADO имеют возможность сохранять и загружать данные в/из файл, они пригодны для разработки приложений briefcase.

Получение данных с центрального сервера

Код для реализации получения данных с центрального сервера, для дальнейшего обсуждения строки кода пронумерованы:

|  |
| --- |
| procedure TForm1.act\_RemoteConnectExecute(Sender: TObject);  begin  1 try  2 try  3 with ParamsCS do  4 begin  5 Close;  6 CommandType:=cmdText;  7 CommandText:=sqlText;  8 Connection:=ParamsConn;  9 Open;  10 end;  11 act\_SaveLocal.Execute;  12 except  13 on E:Exception do  14 MessageDlg(Format(msgServerConnectError, [E.Message]), mtError, [mbOk],0);  15 end;  16 finally  17 ParamsConn.Connected:=false;  18 act\_ConnectLocal.Execute;  19 end; |

Задача данного кода подключиться к центральному серверу, получить данные и сохранить их в локальный кэш для дальнейшего использования.

Блок try … finally (строки 1, 12-15) позволяет нам вне зависимости от успешности подключения к серверу отключиться от него и отобразить пользователю данные из локального кэша. Код для непосредственно подключения к серверу и загрузки данных содержится в строках 2-10. Блок try except обеспечивает обработку ошибок получения данных с сервера. При возникновении ошибки пользователю отображается сообщение о невозможности подключения. Код, непосредственно реализующий получение данных, это строчки 5-9. В этих строках мы настраиваем компонент класса TADODataset (ParamsCS) на работу с сервером и открываем. Вы спросите: зачем это делать каждый раз. Делать это нужно потому, что при открытии локального кэша (с помощью метода TADODataset.LoadFromFile) датасет сам перестраивает свои свойства CommandType и CommandText. Метод LoadFromFile вызывается внутри акции act\_ConnectLocal. После получения с сервера мы сохраняем выборку в локальный кэш, вызвав соответствующий Action (строка 11).

Сохранение данных в локальный кэш

Для обеспечения возможности работы с данными без постоянного подключения к серверу (и постоянно загруженной программы) необходимо сохранять полученные данные и сделанные пользователем изменения. Компоненты ADO (Наследники TCustomADODataset) имеют возможность сохранять выборку данных в файл, используя метод SaveToFile. Метод имеет два параметра. Первый – имя файла, второй формат сохранения данных. Поддерживаются два формата сохранения данных:

XML

ADTG (Advanced Data Tablegram)

По умолчанию сохранение происходит в формате ADTG, хотя лично я предпочитаю сохранение в формате XML, так как он более удобен для восприятия данных человеком.

|  |
| --- |
| ПРИМЕЧАНИЕ  Если имя файла имеет расширение XML, данные сохраняются в формате XML, игнорируя второй параметр метода SaveFile. |

Код сохранения данных в локальный кэш состоит из лишь вызова метода ParamsCS.SaveFile.

Загрузка данных из локального кэша

Для загрузки данных из файла наследники TCustomADODataSet имеют метод LoadFromFile. Перед загрузкой из файла свойство Connection у ParamsCS необходимо установить в nil, так как в ходе загрузки осуществляется попытка подключиться к серверу БД. Код представлен ниже:

|  |
| --- |
| procedure TForm1.act\_ConnectLocalExecute(Sender: TObject);  begin  ParamsCS.Connection:=nil;  ParamsCS.LoadFromFile(ExtractFilePath(Application.ExeName)+ParamFile);  end; |
| ПРИМЕЧАНИЕ  Вызов LoadFromFile автоматически меняет тип команды датасета (св-во CommandType) на cmdFile и в свойство CommandText сохраняет имя файла, откуда была произведена загрузка. | | |

Синхронизация данных с сервером

Синхронизация включает в себя передачу сделанных пользователем изменений и получение с сервера обновленных (обновления от всех пользователей) данных. Получение данных с сервера мы уже рассмотрели и здесь остановимся на проблеме передачи изменений в центральную базу. Задача передачи изменений может быть разделена на две непосредсвтенно передачу и обработку ошибок синхронизации.

Передача изменений осуществляется вызовом метода UpdateBatch. Как мы уже говорили, причиной ошибок синхронизации является одновременное редактирование одной записи несколькими пользователями. По умолчанию запись на сервере отыскивается по ключевым полю и полям, в которых пользователь сделал изменения. При этом если другой пользователь успел сделать в тех же полях этой записи изменения и внести их в базу, запись не может быть обнаружена. Возникает ошибка синхронизации. Алгоритм поиска записи контролируется свойством Update Criteria объекта ADO RecordSet. Update Criteria может принимать следующие значения:

|  |  |
| --- | --- |
| AdCriteriaAllCols | Поиск по совокупности всех столбцов. Наиболее «жесткий» режим. |
| AdCriteriaKey | Поиск только по ключевым полям. Наиболее «мягкий» режим. Конфликт возникает лишь при удалении записи из базы. |
| AdCriteriaTimeStamp | Если в таблице есть поле типа TimeStamp для синхронизации будет использовано оно |
| AdCriteriaUpdCols | Поиск по совокупности ключевых полей и полей, содержащих изменения данных |

При обнаружении ошибок синхронизации генерируется исключительная ситуация класса EOleError c сообщением о невозможности сохранить изменения. Обработка ошибок синхронизации поддерживается в ADO, начиная с версии 2.7. При этом алгоритм разрешения конфликтов, приведенный в MSDN, следующий:

Свойство Filter объекта Recordset ADO установить равным adFilterConflictingRecords. При этом будут отображены только конфликтные записи.

Вызвать метод Resync того же объекта с параметром AffectRecords равным adAffectGroup, параметр ResyncValues равным adResyncUnderlyingValues, при этом будут получены обновленные данные о состоянии конфликтных записей с сервера. Актуальные значения полей записей рекордсета хранятся в свойстве UnderlyingValue объекта Field, начальные в OriginalValue, а измененные пользователем в Value.

Отобразив пользователю набор конфликтных записей и значения их полей мы даем ему возможность отредактировать конфликтные записи и устранить конфликты.

Записать в БД изменения пользователя можно вызвав UpdateBatch с параметром adAffectGroup.

Обработку ошибок я вынес в отдельный модуль ADOReconcileError. В нем определена процедура HandleADOReconcileError, отвечающая за поддержку обработки ошибок синхронизации. Сам же код синхронизации выглядит так:

|  |
| --- |
| try  ParamsConn.Connected:=true;  ParamsCS.Connection:=ParamsConn;  ParamsCS.UpdateBatch;  except  on E:EOleException do begin  HandleADOReconcileError(ParamsCS);  end else raise;  end;  act\_RemoteConnect.Execute; |

Отмена внесенных изменений

Еще одной особенностью использования ADO является возможность пользователю отмены сделанных им изменений. Данная возможность реализуется методом CancelBatch. При вызове c параметром arAll (параметр по умолчанию) отменяются все сделанные изменения. При вызове с параметром arCurrent будут отменены изменения, внесенные в текущую запись датасета.

В следующей части статьи будет рассмотрена реализации модели briefcase с помощью средств MIDAS.