

ZeosDBO - Краткий обзор архитектуры



iboxjo

[Подписаться](#)

>>

0

0

0

0 0 0

Категории:

- [IT](#)
- [Лытдыбр](#)
- [Отменить](#)

Всё, наконец то... Толи день сегодня такой, толи роза ветров сложилась неудачно, но я просто задолбался. Сегодня вечером выкладываю последний файл на www.iboxjo.h1.ru в раздел Переводы. Всё это находится в рамках ПРОЕКТ-9. Устал я что-то. С завтрашнего дня временно перехожу к другому проекту... переводить тоже по малясь буду.

Краткий обзор архитектуры объектов базы данных ZEOS

Сергей Серухов

Мерлин Монкьюр

26 ноября 2003 года (обновлено Марком Дземсом 29.05.2008 года)

Оглавление

1. *Общий обзор*
2. *Уровень плоского (простого) API*
3. *Уровень связи с базой данных*
4. *Уровень компонент*
5. *Эпилог*

Прямой доступ к базам данных SQL продолжает являться жизненной технологией даже в современной корпоративной среде. Разрабатываются тысячи двух уровневых приложений типа

клиент-сервер находящих применение в международном бизнес сообществе. Большинство из них использует прикладной интерфейс программирования (API) для получения связанных данных и выполнения SQL операторов. В настоящий момент существует несколько специализированных и широко используемых API для получения доступа к SQL базам данных, таким как ODBC, JDBC и ADO. Кроме того, Borland сформировал собственный промежуточный (middleware) интерфейс для своих средств разработки, называемый Borland Database Engine (BDE). Несмотря на свободное распространение с популярными приложениями Borland, BDE не стал популярен из-за сложностей установки и низкой производительности. Поскольку Delphi стал одним из ведущих инструментов разработки приложений для платформы Windows, некоторые люди и компании предложили интерфейсы альтернативные BDE. Эти "альтернативы" оптимизировали доступ к базам данных путём непосредственного использования собственных драйверов баз данных, обеспечив высокую производительность и функциональность.

Понимая ограниченность BDE, Borland предложил новый тип интерфейса базы данных, названный dbExpress. Этот интерфейс был разработан дабы обеспечить посредничество в доступе между Delphi и практически любой реляционной базой данных, посредством сторонних драйверов. Borland значительно улучшил производительность dbExpress по отношению к BDE, но реализация содержала ряд ошибок и поддерживала ограниченное подмножество SQL, что препятствовало развитой функциональности.

Библиотека компонентов Zeos Database Object (ZeosLib) является одной из самых известных альтернатив BDE. Первоначально, библиотека была разработана для MySQL и PostgreSQL, но скоро была добавлена поддержка других провайдеров. В процессе разработки стали очевидны определённые ограничения оригинального проекта. Эти ограничения стали деформировать архитектуру проекта и группа разработчиков решила пересмотреть принципы проекта. Был создан новый проект, целью которого стало создание расширенного списка функций с рядом новых требований:

1. Поддержка различных компиляторов
2. Система версий драйверов базы данных
3. "Нечувствительная база данных" -проект для кроссплатформенной разработки базы данных
4. Поддержка множества высокоуровневых интерфейсов (TDataset, dbExpress, Midas)
5. Расширяемая система функций для сервера со специфической поддержкой

1. Общий обзор

Чтобы удовлетворить множественные и иногда непоследовательные требования, группа разработчиков должна была полностью заново обдумать новую архитектуру. Это было сложно, но не невозможно. Каждый модуль помещённый в новый проект был тщательно рассмотрен и имел уникальные функции. В этой статье мы попытаемся представить новую архитектуру и объяснить её цели.

Сверху вниз структура библиотеки разделяется на три логических уровня:

1. Плоский уровень API
2. Уровень связи с базой данных
3. Уровень компонент

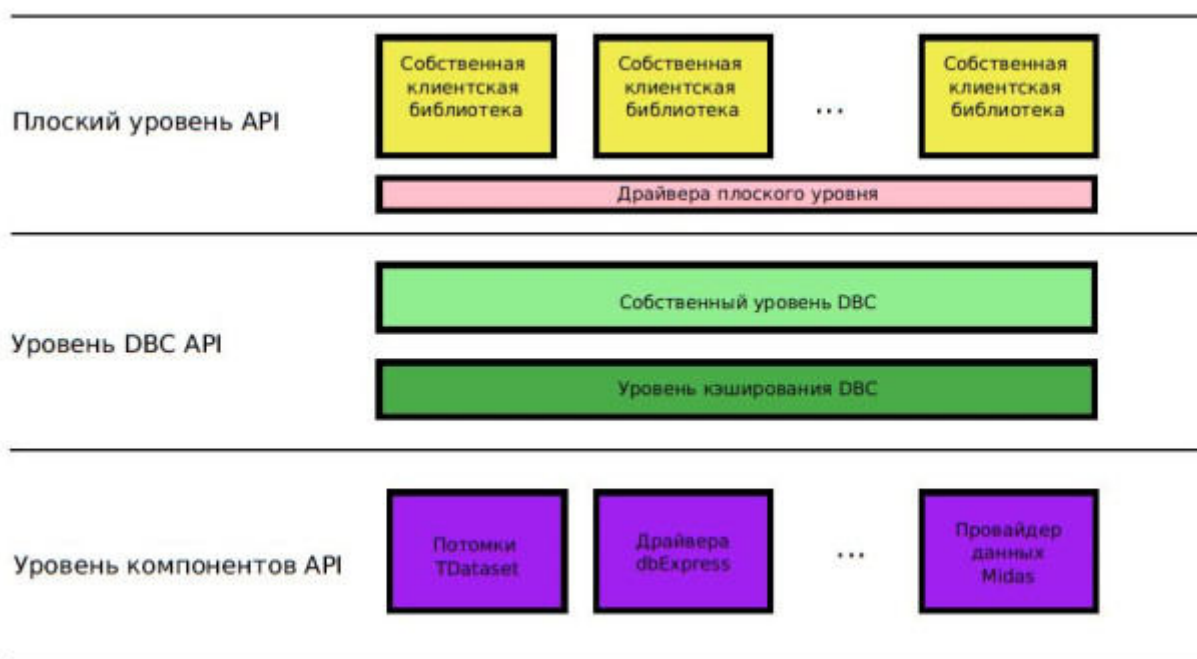


Рисунок 1

Плоский API реализует низкоуровневые функции, специфичные для каждого SQL сервера. API компонентов инкапсулирует основной интерфейс библиотеки, который используется разработчиками приложений. API DBC - промежуточное программное обеспечение, получающее, хранящее и модифицирующее данные для высокоуровневых компонент. Каждый уровень имеет несколько горизонтальных и/или вертикальных подуровня частично зависимых от соседних (смотрите рисунок 1).

Давайте пройдемся через каждый уровень и посмотрим, как всё это работает.

2. Уровень плоского (простого) API

Компоненты ZeosDBO непосредственно не связываются с SQL серверами. Вместо этого, они пользуются родными клиентскими библиотеками поставляемыми с базами данных SQL. Уровень плоского API обеспечивает доступ к функциям, константам и структурам данных родных клиентских библиотек (обычно написанных на C) из Object Pascal или C++. Эта функциональность является отправной для ZeosDBO. Поддержка множества версий клиентских библиотек и SQL серверов являлось основным недостатком более ранних проектов.

Родные вызовы библиотеки (динамические библиотеки Windows и совместно используемые библиотеки Unix) представлены как регулярные функции. Например:

ZPlainMySQL323.pas:

```
function mysql_init(Handle: PMYSQL): PMYSQL; external 'libmysql.dll';
```

Обычно, API базы данных изменяются незначительно, от версии к версии. Но, поскольку вызовы функций не позволяют добавить поддержку полиморфизма для новых версий, Plain API интенсивно кодируется. Трудно кодируемые подходы менее гибкие и подвержены ошибкам, которые ограничивают долгосрочную выполняемость.

```

if Version = 'mysql-3.23' then
ZPlainMySql323.mysql_init(...)
else ZPlainMySql40.mysql_init(...);

```

Для реализации полиморфизма, упрощения исходного кода и обеспечения изоляции от изменений в протоколах SQL сервера, в ZeosDBO был добавлен новый, чрезвычайно тонкий интерфейсный уровень. Этот уровень назван "Plain Drivers" (плоскими или простыми драйверами) и реализован следующим образом:

```

// Общий интерфейс драйвера MySQL
IZMySQLDriver = interface ...
function mysql_init(...)
end;
// MySQL драйвер для версии 3.23
TZMySQL323Driver = class (TInterfacedObject, IZMySQLDriver)
function mysql_init(...)
end;
// MySQL драйвер для версии 4.0
TZMySQL40Driver = class (TInterfacedObject, IZMySQLDriver)
function mysql_init(...)
end;
function TZMySQL323Driver.mysql_init(...)
begin
Result := ZPlainMySql323.mysql_init(...);
end;
function TZMySQL40Driver.mysql_init(...)
begin
Result := ZPlainMySql40.mysql_init(...);
end;

```

Использование такой тонкой обёртки класса возволило легко добавлять новые клиентские интерфейсы. Определённая функциональность, которая требует переопределения, инкапсулируется в плоских драйверах. У остальной части кода теперь есть универсальный метод реализации собственных вызовов базы данных, не требующая специфических знаний о сервере базы данных.

```

// Инициализация плоского драйвера
PlainDriver: IZMySQLDriver;
if Version = 'mysql-3.23' then
PlainDriver := TZMySQL323Driver.Create()
else PlainDriver := TZMySQL40Driver.Create();
// Использование плоского драйвера
PlainDriver.mysql_init(...)

```

С таким подходом становится возможным использовать тот же самый API для различных SQL серверо, независимо от версии.

3. Уровень связи с базой данных

С обеспечением родного доступа к базе данных, компоненты баз данных Delphi могут обеспечивать специфическую функциональность универсальным способом. Однако, каждый SQL сервер имеет различную семантику, которая должна быть создана в компонентах для реализации универсального интерфейса. Основная цель объектов базы данных Zeos - предоставление этого универсального интерфейса разработчику приложений. В более старых версиях ZeosDBO промежуточный интерфейс был реализован как обёртка класса для объектов соединения MySQL и PostgreSQL. Эти две базы данных имеют схожие возможности, таким образом проектирование промежуточного звена API было не слишком сложным.

Однако поддержка других SQL серверов добавляла новые, весьма специфические особенности. После нескольких расширений для поддержки этих особенностей, архитектура класса стала неясной. В конечном счёте, интерфейс нарушал свои правила инкапсуляции, а высокоуровневые компоненты были вынуждены использовать низкоуровневые вызовы базы данных.

Чтобы преодолеть эту трудность в новой версии, требовалось реализовать интерфейс промежуточного звена с абстрактным подходом. Проектирование такого интерфейса не тривиально. Дабы избежать новых ошибок, было решено избегать собственного проектирования и использовать наработки известного API базы данных.

В качестве прототипа для промежуточного звена, группа разработчиков выбрала JDBC 2.0. JDBC - один из последних и наиболее популярный API в сообществе баз данных. Он охватывает различные абстракции, такие как операторы, результирующие наборы, хранимые процедуры, блобы и богатые определения метаданных. API JDBC реализуется через набор интерфейсов JAVA. Компиляторы Borland поддерживают интерфейсы, таким образом непосредственно портируют JDBC из JAVA в Object Pascal. Типы данных и имена методов были сохранены. Перегруженных методов удалось избежать из-за их плохой поддержки в семействе компиляторов C++ Builder. Основные интерфейсы DBC представлены на рисунке 2.

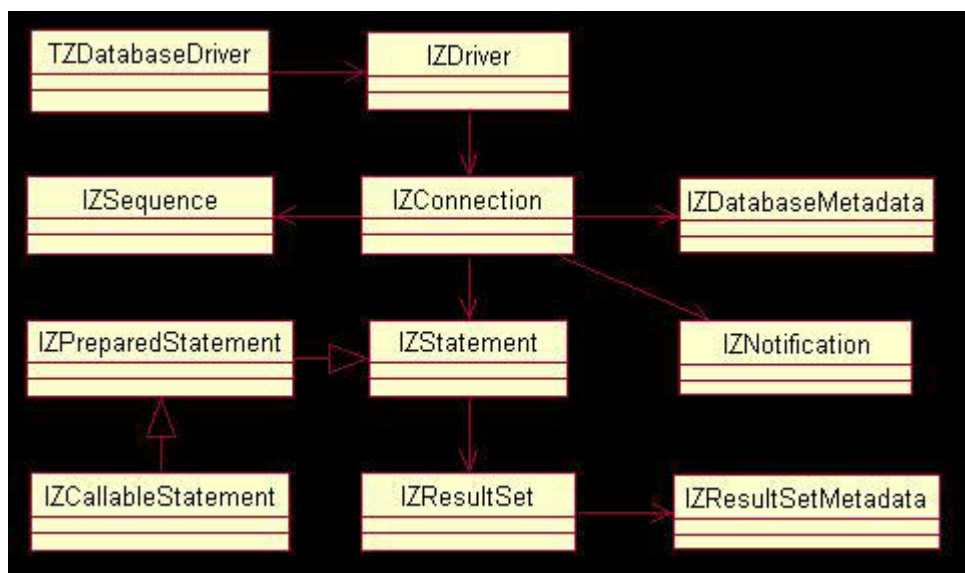


Рисунок 2 Интерфейсы DBC

Стандартные интерфейсы JDBC реализуют универсальный клиентский API. Для реализации функциональности определённой для различных SQL серверов, были выбраны два общих подхода:

1. Разработчики могут инициализировать объект соединения с базой данных по списку строковых параметров. Каждый параметр переключает специфические настройки сервера. Для JDBC этот метод не нов.

Пример: параметры могут быть определены в URL соединения:

```
zdbc:mysql://localhost/database?compress=true
```

или параметры можно передать методу соединения:

```
Params.Values['compress'] := 'true';  
Connection := DriverManager.CreateConnectionWithParams(Url, Params);
```

Дополнительно, в ZeosDBO разработчики расширили другие интерфейсы DBC, дабы инициализировать другой объект с определёнными параметрами, особо объект оператор (Statement):

```
Params.Values['oidasblod'] := 'true';  
Statement := Connection.CreateStatementWithParams(Params);
```

2. В Object Pascal каждый класс в состоянии реализовать одновременно множество интерфейсов. Мы использовали эту возможность, чтобы расширить стандартный интерфейс взаимодействия JDBC новыми методами, определёнными для каждого конкретного SQL сервера. Теперь, каждый класс, знает не только о стандартном интерфейсе, но и о специфическом интерфейсе сервера:

```
IZMySQLConnection = interface (IZConnection)  
function Ping(...);  
function Kill(...);  
end;  
TZMySQLConnection = class (TInterfacedObject, IZConnection, IZMySQLConnection);  
...  
end;
```

Следующий шаг в портировании JDBC в Object Pascal стал реализацией кэшируемого доступа к данным. Фактически, многие серверы реализуют поддержку только для последовательного доступа к данным. Кэширование данных на стороне клиента является важным элементом реализации случайного доступа к данным для получения набора результатов. С другой стороны, многие высокоуровневые интерфейсы баз данных используют ещё более сложные алгоритмы кэширования. Таким образом реализация универсальных алгоритмов кэширования в одном месте может стать компактным и эффективным решением.

Уровень кэширования DBC имеет несколько классов:

1. TZRowAccessor - организует хранение и доступ к полям одной единственной кэшируемой записи в наборе результатов (шаблон Flyweight)
2. TZChachedResultSet - является кэшируемым набором результатов со случайным доступом к данным. Он работает поверх другого родного не кэшируемого набора результатов с

последовательным доступом к данным (шаблон Decorator).

3. TZCachedResolver - интерфейс для обработки специальной логики отправки изменённых данных обратно SQL серверу (шаблон Delegator).

Удобство интерфейсов DBC в том, что они настолько универсальны, что легко позволяют реализовать, дополнительно к стандартным драйверам SQL специальные адаптеры для других интерфейсов баз данных, таким например, как ADO.

4. Уровень компонент

Последний верхний уровень библиотеки ZeosDBO реализует dbware компоненты. Эти компоненты используются для разработки в Delphi, C++Builder и Kylix. В настоящий момент компиляторы Borland поддерживают несколько стандартов для dbware компонент:

1. Компоненты потомка TDataset
2. Драйверы dbExpress
3. Источники данных для многоуровневой технологии Midas

Во время написания этой статьи библиотека ZeosDBO поддерживала только компоненты потомки TDataset. Реализация других типов компонент планируется в будущих версиях. Диаграмма классов компонент представлена на Рисунке 3:

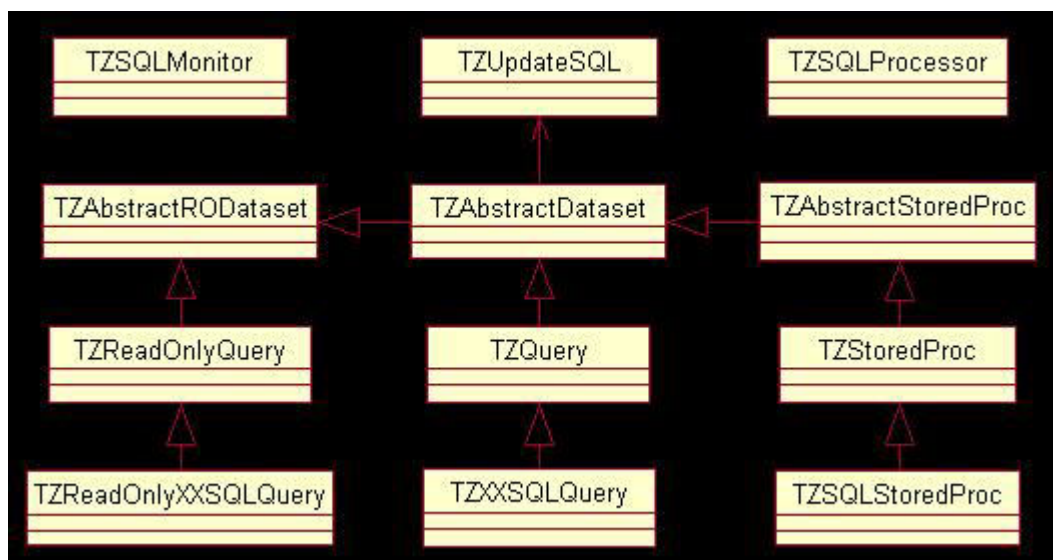


Рисунок 3 Диаграмма классов компонент

1. TZAbstractRODataset, TZAbstractDataset, TZAbstractStoredProc - абстрактные классы для компонента потомка TDataset
2. TZReadOnlyQuery, TZQuery, TZStoredProc - универсальные компоненты потомки TDataset
3. TZReadOnlyXXSQLQuery, TZXXSQLQuery, TZXXSQLStoredProc - TDataset специфичный для каждого поддерживаемого SQL сервера. Путь которым сервер распространяет специфические функции описан в предыдущей главе об уровне DBC.
4. TZSQLMonitor, TZUpdateSQL, TZSQLProcessor - универсальные вспомогательные компоненты.

TDataset использует максимум функциональности уровня DBC для чтения, модификации и хранения данных. Дополнительно он реализует функции фильтрации, поиска и сортировки данных,

соединение с визуальными компонентами и множество других.

5. Эпилог.

В этой статье мы описали основные идеи новой архитектуры библиотеки ZEOS для нативного доступа к базе данных. Эта архитектура была представлена в версии 6.0 и продемонстрировала высокую гибкость и эффективность. В версии 6.1. код был серьёзно пересмотрен и оптимизирован, но основные принципы архитектуры не изменялись.

Функциональность и гибкость архитектуры адаптируется к новым требованиям в будущем. Будет добавлена поддержка новых SQL серверов и доступ к определённой функциональности SQL через интерфейсы dbExpress и Midas. Чтобы более полно ознакомиться с возможностями библиотеки следует посетить сайт проекта <http://zeos.firmos.at> и страницу разработки на SourceForge <http://www.sourceforge.net/projects/zeoslib>

Проект 9нЭмного ПА РусскЕ

0

  [Подписаться](#)  