УДК 004.65
DOI: 10.36979/1694-500X-2024-24-4-10-17

ОСОБЕННОСТИ И ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ ADO.NET С КЛАССИЧЕСКИМ ADO

Н.К. Аркабаев, Т.М. Мадумарова, О.И. Турдалиева, И.Н. Абдукаримов

Аннотация. Рассматриваются две платформы доступа к данным от корпорации Microsoft – классическая технология ADO и более поздняя реализация ADO.NET, появившаяся в составе .NET Framework в 2002 году. Анализируются ключевые архитектурные и функциональные особенности каждой из платформ, проводится их сопоставление по различным аспектам. Отдельно изучается структура ADO.NET – наборы данных, адаптеры, провайдеры данных, а также возможности по кэшированию, параллельной работе, обеспечению целостности информации. Подробно исследуются преимущества ADO.NET перед ADO в области производительности систем доступа к данным – за счет встроенных механизмов клиентской оптимизации и расширений серверной стороны обеспечивается выигрыш в скорости и масштабируемости решений доступа к данным. Сравнительный анализ и примеры кода направлены на помощь разработчикам ПО в выборе оптимальной стратегии построения подсистемы доступа к хранилищам данных как в уже развёрнутых проектах, так и при проектировании новых систем.

Ключевые слова: ADO; ADO.NET; доступ к данным; работа с данными; кэширование; DataAdapter; LINQ to SQL; SQL Server.

КЛАССИКАЛЫК ADOГО КАРАГАНДА ADO.NETТИ КОЛДОНУУНУН ӨЗГӨЧӨЛҮКТӨРҮ ЖАНА АРТЫКЧЫЛЫКТАРЫ

Н.К. Аркабаев, Т.М. Мадумарова, О.И. Турдалиева, И.Н. Абдукаримов

Аннотация. Бул макалада Microsoft корпорациясынан маалыматтарга жеткиликтүү болуунун эки платформасы – классикалык ADO технологиясы жана 2002-жылы .NET Framework курамында пайда болгон ADO.NETтин кийин ишке ашырылган версиясы каралат. Ар бир платформанын негизги архитектуралык жана функционалдык өзгөчөлүктөрү ар кандай аспектилерде талданат жана салыштырылат. ADO.NET түзүмү өзүнчө изилденет – маалыматтар топтому, адаптерлер, маалымат провайдерлери, ошондой эле кэштөө, параллелдүү иштөө жана маалыматтын бүтүндүгүн камсыз кылуу мүмкүнчүлүктөрү. ADO.NETтин ADOга караганда маалыматтарга кирүү системаларынын иштеши жаатындагы артыкчылыктары кылдат изилденген – кардарларды оптималдаштыруунун жана сервердик кеңейтүүнүн орнотулган механизмдеринин аркасында маалыматтарга жетүү чечимдеринин ылдамдыгынын жана масштабдуулугунун жогорулашы камсыз кылынат. Салыштырмалуу талдоо жана код мисалдары программалык камсыздоону иштеп чыгуучуларга мурунтан эле орнотулган долбооролордо да, жаңы системаларды долбооролоодо да маалымат сактагычка жеткиликтүү болуунун чакан системаларын түзүү үчүн оптималдуу стратегияны тандоого көмөк көрсөтүүгө багытталган.

Түйүндүү сөздөр: ADO; ADO.NET; маалыматтарга жеткиликтүүлүк; маалыматтар менен иштөө; кэштөө; DataAdapter; LINQ to SQL; SQL Server.

FEATURES AND BENEFITS OF USING ADO.NET WITH CLASSIC ADO

N.K. Arkabaev, T.M. Madumarova, O.I. Turdalieva, I.N. Abdukarimov

Abstract. The article provides an in-depth examination of two data access platforms from Microsoft Corporation – the classic ADO technology and the later ADO.NET implementation that appeared as part of the .NET Framework in 2002. The key architectural and functional features of each platform are analyzed and compared across various aspects. The structure of ADO.NET is studied separately – data sets, adapters, data providers, as well as caching, parallel operation, and information integrity capabilities. The article explores in detail the advantages of ADO.NET over ADO in

terms of data access system performance – through built-in client optimization mechanisms and expanded server-side capabilities, gains in speed and scalability of data access solutions are ensured. The comparative analysis and code examples are aimed at helping software developers choose the optimal strategy for constructing a data storage access subsystem both in already deployed projects and when designing new systems.

Keywords: ADO; ADO.NET; data access; data manipulation; caching; DataAdapter; LINQ to SQL; SQL Server.

Введение. Технологии доступа к данным играют ключевую роль в разработке современного программного обеспечения. Microsoft предоставляет разработчикам два основных инструмента для работы с базами данных из кода приложений на платформе .NET Framework – классический ADO и более новый ADO.NET [1].

ADO (ActiveX Data Objects) используется уже достаточно давно и хорошо зарекомендовал себя при создании клиент-серверных приложений доступа к данным [2]. Однако выпущенная в 2002 году версия .NET Framework принесла новую концепцию доступа к данным – ADO.NET. Данная технология была полностью пересмотрена с учетом возможностей .NET [3].

ADO.NET предлагает целый ряд преимуществ перед классическим ADO – более простая и гибкая модель программирования, лучшая производительность и масштабируемость, расширенные возможности для объектно-ориентированного дизайна. Данная статья посвящена сравнительному анализу этих двух технологий доступа к данным, рассмотрению их архитектуры и основных компонентов, а также изучению конкретных отличий и преимуществ ADO.NET перед старым добрым ADO.

Технологии доступа к данным играют ключевую роль в разработке современного программного обеспечения. Microsoft предоставляет разработчикам два основных инструмента для работы с базами данных из кода приложений на платформе .NET Framework – классический ADO и более новый ADO.NET.

В данной статье мы подробно рассмотрим архитектуру ADO.NET, ее основные компоненты и принципы работы. Затем сравним с архитектурой и принципами классического ADO. Далее проанализируем преимущества новой технологии – более простую модель программирования, улучшенную производительность, расширенные возможности для ООП дизайна. Также будут рассмотрены такие аспекты как безопасность, транзакционность, работа с разнородными данными. В заключение подведем итоги и сделаем вывод о целесообразности использования ADO.NET взамен устаревшего ADO.

Архитектура ADO.NET и основные компоненты. В основе ADO.NET лежит концепция наборов данных (DataSets) – это отдельные объекты, которые отображают данные из базы данных в оперативной памяти [4]. Набор данных представляет собой отключенную копию данных, к которой можно обращаться независимо от источника.

Основными компонентами ADO.NET являются:

1. DataSet – представляет данные в виде коллекции таблиц, связей, ограничений и схемы данных. Является основным объектом при работе в отключенном режиме [5]. А также является центральным элементом архитектуры, который представляет данные в оперативной памяти. Он может содержать несколько объектов DataTable, а также сведения об отношениях между таблицами, ограничениях и схеме данных. Основные возможности: навигация по таблицам и записям, добавление/редактирование/удаление записей, сериализация содержимого, управление версиями данных при обновлении БД.

2. DataTable – соответствует таблице данных из базы данных, хранит данные в виде коллекции строк и колонок [5]. Он хранит данные в структурированном виде – по строкам и столбцам. Столбцы представлены объектами DataColumn. Для навигации по строкам таблицы используется объект DataRow. В таблицу можно независимо добавлять как строки, так и столбцы.

3. DataView – представляет собой отфильтрованный взгляд на таблицу данных, используется для сортировки и поиска. Он использует ссылку на таблицу данных и может показывать отфильтрованный взгляд этих данных для удобства манипулирования. Позволяет сортировать и искать данные без изменения базовой таблицы.

4. DataAdapter – служит мостом между источником данных и набором данных, управляет потоком информации в обоих направлениях. Этот компонент выполняет функцию связующего звена между DataSet в памяти приложения и источником данных и отвечает за получение данных из источника и заполнение ими DataSet, отслеживание изменений в DataSet и применение этих изменений обратно к источнику данных, вызова хранимых процедур в базе данных. Основными классами DataAdapter являются: SqlDataAdapter – для работы с базами данных SQL Server; OleDbDataAdapter – для работы с источниками данных OLE DB, Oracle, ODBC и т. д.

5. Connection объект – описывает подключение к источнику данных, может находиться в разных состояниях. Объект соединения представляет физическое подключение к источнику данных. Основными возможностями являются: открытие и закрытие соединения, управление транзакциями и выполнение команд к базе данных. Объект Connection может находиться в различных состояниях: открыто, закрыто, ожидание, фоновая операция и т. д.

В отличие от ADO, где основным объектом является поток данных Recordset, в ADO.NET центральное место занимают отключенные наборы данных. Это коренное архитектурное отличие.

Основным объектом в ADO является Recordset, который представляет набор записей из источника данных. Recordset управляет позицией в наборе записей и позволяет выполнять навигацию и манипулирование данными. Важное отличие в том, что Recordset тесно связан с источником данных и работает только в подключенном режиме. Если соединение разорвано – доступ к данным невозможен.

В ADO.NET же центральным объектом является отключенный DataSet, который представляет копию данных в памяти приложения и обеспечивает доступ к данным даже при разрыве соединения с источником.

Еще одно важное отличие в том, что ADO – это более «плоская» реляционная модель, которая хранит данные в виде двумерных таблиц. В ADO.NET данные хранятся в объектно-ориентированной иерархичной структуре (наборы данных, отношения, ограничения), что обеспечивает более высокий уровень абстракции.

В целом архитектура ADO.NET намного гибче и удобнее для программирования по сравнению с ADO. Еще можно рассмотреть различия в подключении к источникам данных и работе с транзакциями.

В подключение к данным ADO используются соединения (Connection объекты) для непосредственного подключения к источнику данных. Строка подключения определяет провайдера данных. А в ADO.NET используются провайдеры данных .NET, которые обеспечивают более высокий уровень абстракции. Строки подключения также изменились.

Транзакции и параллелизм в ADO транзакции обеспечиваются непосредственно источником данных. А в ADO.NET реализована программная транзакционность на уровне TableAdapter. Это позволяет использовать транзакции даже при отсутствии их поддержки источником данных. ADO.NET лучше подходит для многопоточных приложений и обеспечивает параллельную работу с данными. Поддерживается оптимистичная блокировка.

В ADO основная работа с данными ведется через объекты Recordset, которые получают данные непосредственно из источника через соединение Connection. Таким образом, Connection объект обеспечивает живую связь и передачу данных между источником и приложением. А в ADO.NET же появляется промежуточный слой в виде DataSet – локального кэша данных. DataAdapter выступает в роли посредника между локальным кэшем и источником данных. Он управляет процессом переноса данных в обе стороны и следит за изменениями. Таким образом, в ADO.NET реализована концепция отключенного слоя данных, когда приложение работает в основном с локальной копией данных, а DataAdapter отвечает за синхронизацию этих данных с внешним хранилищем.

Это кардинальное отличие архитектур, которое делает работу с данными в ADO.NET более гибкой и масштабируемой. Приложение меньше зависит от стабильности источника данных.

В целом новая архитектура ADO.NET намного превосходит старую ADO и является шагом вперед.

Особенности работы с базами данных в ADO.NET. В ADO.NET введена концепция провайдеров данных .NET – специальных классов, выполняющих функцию подключения к разным источникам данных и доступа к ним. Например, для работы с Microsoft SQL Server используется провайдер – SqlClient. Он реализует поддержку таких технологий SQL Server как таблицы в памяти, репликация данных, XML данные и позволяет выполнять команды и хранимые процедуры, использовать транзакции. Также поддерживаются пулы соединений, асинхронные методы получения данных.

Строка подключения имеет следующий формат:

Server=myServerAddress;Database=myDataBase;User Id=myUsername;

Password=myPassword;

Для обеспечения подключения к источникам данных, поддерживающим OLE DB интерфейс, используется провайдер OleDbClient. Можно использовать для подключения к источникам Oracle, DB2 и Microsoft Access. Также, OdbcClient позволяет ODBC производить подключение к базам данных. Может использоваться с СУБД, для которых нет специальных ADO.NET провайдеров.

Строка подключения имеет следующий формат:

Data Source=serverAddress;Initial Catalog=dbName;User ID=login;Password=pwd

В целом архитектура провайдеров данных ADO.NET стандартизирует и упрощает доступ к разным СУБД из кода .NET приложений.

При сравнении строк подключения к базам данных в ADO и ADO.NET можно отметить следующие ключевые различия:

1. Провайдер данных. В ADO строки подключения содержат прямую ссылку на OLE DB провайдер или ODBC драйвер. Например,

Provider=SQLOLEDB.1;Data Source=SqlServer;Initial Catalog=Northwind;

Integrated Security=SSPI;

В ADO.NET строки содержат ссылку на соответствующий .NET данных провайдер вместо низкоуровневого OLE DB или ODBC драйвера:

Data Source=SqlServer;Initial Catalog=Northwind;Integrated Security=True

2. Параметры безопасности. В ADO строки подключения используют параметры вроде «SSPI», «udf» и др. для настройки безопасности, а в ADO.NET применяется стандартизованный параметр «Integrated Security=true/false».

3. Синтаксис. Строки подключения ADO.NET имеют более простой и единообразный синтаксис при обращении к разным СУБД. Например, параметры Server и Database используются всегда, независимо от конкретной БД.

Таким образом, благодаря абстракции ADO.NET провайдеров данных, строки подключения стали проще и унифицированы по сравнению со старым ADO подходом.

Наборы данных (DataSet) являются ключевым компонентом ADO.NET, который обеспечивает мощные и гибкие возможности для манипулирования данными. По сравнению с ADO, где основным объектом является Recordset, возможности значительно расширены. Например, в навигации и редактировании данных в ADO используются методы MoveNext, MovePrevious и т. д., а в ADO.NET можно просто обращаться к строкам данных по индексу как к элементам массива.

```
// Получаем строку по индексу
DataRow row = dataset.Tables[0].Rows[0];
// Изменяем значение в строке
row[«Column1»] = newValue;
// Проходим циклом по строкам
foreach (DataRow r in dataset.Tables[0].Rows)
{
 // действия
}
```

При добавлении и удалении записей в DataSet можно независимо выполнять действия как строки данных (DataRow), так и целые таблицы (DataTable), что сложно реализовать в ADO recordset.

```
// Добавляем строку
DataRow newRow = table.NewRow();
// Задаём значения столбцов
newRow[«column1»] = value1;
table.Rows.Add(newRow);
```

Связи между таблицами DataSet позволяют описывать отношения между таблицами данных, ссылки на родительские строки и т. д. Это существенно расширяет возможности.

```
DataRelation rel = new DataRelation(“rel_name”,
 parentTable.Columns[“id”],
 childTable.Columns[“parent_id”]);
dataSet.Relations.Add(rel);
```

Просмотры данных DataView позволяют отфильтровывать таблицу данных и использовать виртуальные гриды.

```
DataView view = new DataView(table);
// Фильтруем по условию
view.RowFilter = «id > 10»;
// Сортируем
view.Sort = “name DESC”;
```

Удобство для разработчика. Интуитивно понятная работа с коллекциями строк и таблиц в памяти, легкая сериализация. Таким образом, ADO.NET DataSet предоставляет разработчику мощный и гибкий инструментарий для работы с данными.

В архитектуре ADO классические объекты вроде Recordset тесно связаны с источником данных и не предназначены для отключенной работы. Однако в ADO.NET данные заполняются в локальный кэш – DataSet, который представляет собой отключенную копию данных из источника. Этот кэш данных может использоваться в автономном режиме – для чтения, манипулирования, передачи между уровнями приложения и так далее.

Кроме того, в ADO.NET реализована поддержка версионности данных при обновлении базы данных из приложения. При отправке изменений DataAdapter автоматически выполняет проверку на конфликты версий с помощью механизма оптимистичной блокировки.

В целом возможности кэширования и отключенной работы в ADO.NET значительно превосходят старую технологию ADO. Это важное преимущество при построении современных многозвенных клиент-серверных приложений доступа к данным.

Помимо базовых функций кэширования данных в локальном DataSet, в ADO.NET реализованы и более продвинутые механизмы:

1. Управление изменениями данных. DataAdapter может отслеживать изменения в DataSet и при синхронизации эффективно обновлять только модифицированные данные в источнике.
2. Пакетные команды. Для повышения производительности при обновлении базы данных Adapter умеет отправлять изменения пакетами с помощью технологии батчей.
3. Отключенные таблицы (DataTable.RemotingFormat). Позволяют сериализовать локальную таблицу данных и передавать между App Domain, процессами или узлами.
4. Синхронизация данных (Sync Services). Компонент для синхронизации отключенных DataSet между базой данных и другими источниками (файл, web-сервис и т. п.).
5. Интеграция с XML. Возможность сохранять и загружать данные в форматы XML, эффективная работа с XML.

Таким образом, расширенные средства кэширования данных делают ADO.NET еще более гибким и мощным инструментом.

При разработке клиент-серверных приложений в ADO.NET реализован ряд важных механизмов для обеспечения безопасности и надежности работы с данными:

1. Управление транзакциями: транзакции позволяют объединить последовательность операций в единый неделимый блок; поддерживаются локальные (внутри приложения) и распределённые транзакции; реализованы средства программного управления транзакциями (класс Transaction).

2. Уровни изоляции транзакций: позволяют избегать конкурентного доступа к данным в многопоточных приложениях.

3. Безопасность на уровне данных: проверка подлинности и авторизация через провайдеры данных; шифрование конфиденциальных данных; управление доступом на основе ролей.

В локальных транзакциях могут использоваться управления транзакциями в самом ADO.NET, что гарантирует атомарность операций в границах одного соединения. Например,

```
connection.Open();
var transaction = connection.BeginTransaction();
// транзакционные операции
transaction.Commit(); // или Rollback
```

Рассмотрим конкретный пример использования транзакции для управления целостностью данных в ADO.NET. Представим ситуацию: в нашем приложении есть метод, который переводит средства с одного банковского счета на другой. Требуется гарантировать, что или перевод выполнится полностью, или не выполнится вообще (чтобы избежать промежуточных состояний в базе данных). В ADO.NET можем реализовать это следующим образом:

```
using(var connection = new SqlConnection(connString))
{
 SqlCommand debitCommand = // ...
 SqlCommand creditCommand = // ...
 connection.Open();
 SqlTransaction transaction = connection.BeginTransaction();
 try
 {
  debitCommand.Transaction = transaction;
  debitCommand.ExecuteNonQuery();
  creditCommand.Transaction = transaction;
  creditCommand.ExecuteNonQuery();
  // Фиксируем транзакцию если всё ок
  transaction.Commit();
 }
 catch (Exception ex)
 {
  // Откат в случае ошибки
  transaction.Rollback();
 }
}
```

Давайте теперь кратко рассмотрим возможности параллельности и синхронизации в многопоточных приложениях в ADO.NET.

Параллельность и синхронизация в многопоточных приложениях связана с одновременным доступом к общим данным из нескольких потоков и корректной работой с этими данными. Параллельность – это возможность одновременного выполнения частей программы (нитей, потоков) на многоядерных процессорах или в многопроцессорных системах.

Многопоточное приложение создаёт и запускает несколько потоков для выполнения различных задач. Например, один поток может загружать данные, другой отображать интерфейс и т. д.

Синхронизация нужна для согласованной работы потоков при обращении к общим данным. Без синхронизации возможны состояния данных, когда часть операций произведена, а часть нет, что приводит к ошибкам. Для синхронизации используются различные инструменты – в ADO.NET это оптимистичная блокировка. Она позволяет избежать ошибок целостности данных в многопоточном доступе.

ADO.NET предоставляет развитые возможности для организации параллельной работы с данными в многопоточных приложениях. Рассмотрим основные моменты:

1. Потокобезопасность классов ADO.NET. Все основные классы (Connection, Command, DataAdapter) являются потокобезопасными, их можно использовать одновременно из разных потоков.
2. Асинхронные методы. Поддерживается асинхронное, неблокирующее выполнение команд и получение данных, что важно для многопоточности и отзывчивости интерфейса приложения.
3. Оптимистичная блокировка. Для целостности данных при параллельном доступе используются механизмы сверки версий записей и контроля за конфликтами. Не происходит пессимистической блокировки записей в базе данных.
4. Множественные одновременные открытые соединения. Пули соединений позволяют масштабировать доступ к данным из многочисленных потоков.
5. Параллельная обработка изменений. При синхронизации изменений с базой данных (с помощью адаптера) возможна параллельная отправка пакетов данных.

ADO.NET демонстрирует более высокую производительность по сравнению с более ранней технологией ADO при работе с базами данных из кода приложений на платформе .NET Framework. Это обусловлено несколькими архитектурными решениями, например, встроенное кэширование данных. ADO.NET использует локально отключенные наборы данных (DataSet), которые представляют собой кэш записей в оперативной памяти приложения. Это позволяет сократить число обращений к базе данных. Данные выбираются в кэш единожды, затем многократно используются из памяти. Имеется также отсоединенный режим работы. Благодаря наличию локального кеша данных в виде DataSet, приложение ADO.NET может работать в отключенном режиме от базы данных – обращаться к копии данных. Это снижает нагрузку на БД.

При обновлении базы данных из приложения, ADO.NET передает на сервер только измененные или добавленные строки, а не весь объем данных. Это экономит сетевой трафик. На время обращения к БД в ADO используется синхронная модель. При пересылке больших объемов данных между сервером и клиентом может использоваться эффективная XML сериализация.

В целом, архитектура ADO.NET более оптимизирована для производительности, чем предыдущая технология доступа к данным ADO. Правильное применение практик кэширования и асинхронной обработки запросов позволяет добиться существенного прироста быстродействия приложений.

Есть несколько дополнительных аспектов, которые позволяют ADO.NET демонстрировать более высокую производительность по сравнению с классическим ADO:

1. Оптимизация сетевого трафика с помощью пакетной отправки изменений. При синхронизации данных между DataSet в памяти приложения и таблицами базы данных используется механизм пакетной отправки transactions (batching). Изменения накапливаются и отсылаются серверу пачками, что снижает накладные расходы сети.
2. Откомпилированные хранимые процедуры. В ADO.NET широко используется подход с откомпилированными хранимыми процедурами на стороне сервера. Это повышает быстродействие за счет предварительной оптимизации запросов, минимизации сетевого трафика.
3. Создание индексированных представлений. Для ускорения чтения данных наиболее оптимальным решением является предварительное создание индексированных представлений над часто запрашиваемыми таблицами. Это возможность СУБД, не зависящая от ADO.NET, но важная для общей производительности.

4. Использование пулов соединений. Пулы подключений к базе данных позволяют многократно использовать уже установленные соединения, экономя накладные расходы на инициализацию новых подключений. Это оптимизация важна для масштабируемости.

Заключение. Технология доступа к данным ADO.NET, появившаяся в 2002 году в составе платформы .NET Framework, стала новым этапом эволюции инструментов работы с базами данных в приложениях от Microsoft. По сравнению со своим предшественником – технологией ADO – архитектура ADO.NET была полностью пересмотрена с учетом современных требований к масштабируемости, гибкости и производительности систем доступа к данным.

Как показал наш обзор, ADO.NET предлагает целый ряд существенных преимуществ перед классическим ADO – более простую и интуитивную модель программирования с использованием отключенных наборов данных, улучшенную поддержку транзакций и параллельной работы, расширенные средства оптимизации производительности на стороне клиента. Благодаря этому ADO.NET демонстрирует лучшие характеристики в области масштабируемости, быстродействия и эффективности использования сетевых ресурсов. Появление LINQ to SQL дополнительно модернизировало технологию доступа к данным в .NET.

Таким образом, несмотря на сохраняющуюся поддержку классического ADO в современных версиях .NET Framework, при проектировании новых систем рекомендуется отдавать предпочтение более совершенной технологии ADO.NET, которая лучше отвечает вызовам сегодняшнего дня.

Компонентная архитектура с отключаемым набором данных, расширенным средствам оптимизации запросов и кэширования, поддержка параллельности ADO.NET позволяет создавать высокопроизводительные системы работы с базами данных. Помимо этого, появление дополнительных технологий таких как LINQ to SQL, еще более модернизировало стек ADO.NET, добавив удобные объектно-ориентированные абстракции для работы с данными.

Поступила: 27.02.24; рецензирована: 12.03.24; принята: 14.03.24.

Литература

1. [Microsoft Docs]: [сайт]. URL: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ (дата обращения: 05.03.2023).
2. *Мэтью Макдональд*. Профессиональное программирование на ADO.NET 2.0 на C# / Макдональд Мэтью; пер. с англ. М.: ООО «И.Д. Вильямс», 2006. 1056 с.
3. *Прайс Джейсон.* ADO.NET 3.5 Cookbook: Solutions for ADO.NET 3.5 Programmers = Поваренная книга ADO.NET 3.5: решения для программистов ADO.NET 3.5 / Прайс Джейсон, Хилл Дэвид. М.: ООО «И.Д. Вильямс», 2009. 544 с.
4. *Гилберт Д.* Microsoft .NET – Архитектура приложений для корпораций / Д. Гилберт; пер. с англ. М.: Русская редакция, 2004. 544 с.
5. *Эндрю Троелсен*. Язык программирования C# 2010 и платформа .NET 4 / Э. Троелсен; пер. с англ. М.: ООО «И.Д. Вильямс», 2011. 1392 с.