Раздел 3 WEB OLTP – ТЕХНОЛОГИИ . MS DOT.NEI

3.1

- Широкому использованию Интернет для организации взаимодействия между системами категории B2B и е-коммерции мешали две проблемы:
- Невозможность выполнения полноценных бизнес-транзакций;
- Разнородность интерфейсов RBC и необходимость переформатирования передаваемых данных
- Решение первой проблемы связано с развитием WEBOLTP –технологии, интегрирующей удобный доступ к данным через интернет и выполнение бизнес - транзакций в 3-х звенной (многозвенной) архитектуре «клиент-сервер» Изложение возможных решений начнем с введения в эту архитектуру.

Еа данном рисунке приведены направления развития Интернет.

. .

- WebOLTP — имя, которое было предложено компанией Sybase, для описания приложений, выполняющих транзакции в интернете, интранете или традиционных корпоративных сетях. Отличительные черты WebOLTP при сравнении с OLTP-технологией на мэйнфреймах или в системах клиент/сервер:
  - *Тонкие клиенты* ;
  - *Большие объемы при большом количестве соединений ;*
  - *Непредсказуемые нагрузнагрузки ;*
  - *Короткий жизненный цикл приложения*

В этой новой модели, пользователи находят и запускают приложения, использующие традиционные HTML-страницы и веб-серверы. Но вместо

просто загрузки статической страницы, динамическая «***апплета***» загружается в индивидуальный браузер. К тому же апплета поддерживает высокоскоростные протоколы, которые позволяют ей соединяться непосредственно с ***сервлетами*** («servlets») — ПО, работающим на промежуточном уровне. Обычно сервлеты обеспечивают доступ к одной или более баз данных, реализуют бизнес- логику и возвращают результаты апплете для отображения на клиенте

Апплеты являются динамически загружаемыми программами, которые управляют логикой представления данных. В архитектуре WebOLTP апплеты также содержат часть кода, отвечающего за коммуникации, что позволяет им напрямую соединяться с сервлетами, работающими на промежуточном уровне.

Существует два метода для построения апплетов. Первый основан на использовании Java и JavaBeans, второй — на основе технологии ActiveX

Апплеты реализуют следующие преимущества:

небольшой объем, что обеспечивает быструю загрузку,

большую интерактивность и более дружественный интерфейс по сравнению с обычными HTML-страницами,

легкость в разработке и сопровождении

Протоколы, отличные от HTTP

Ключевые требования для этих новых протоколов включают способность:

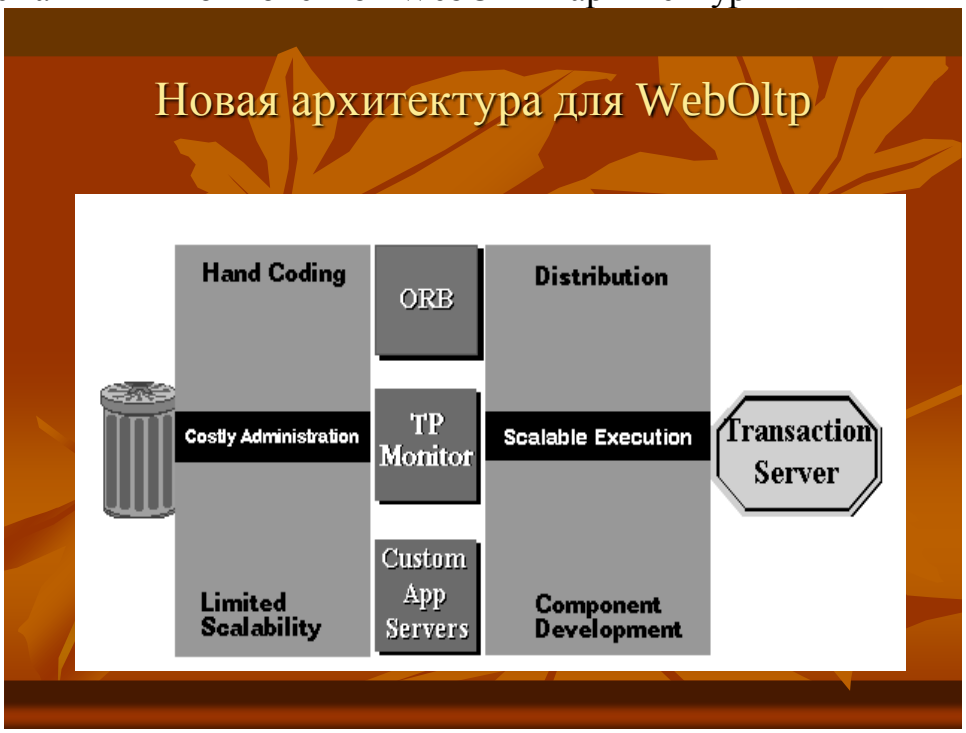
поддерживать информацию о пользователях и транзакциях,

эффективно формировать результирующую выборку и управлять ею,

поддерживать транзакции,

предоставлять надежные механизмы шифрования данных

С появлением многоуровневой архитектуры, на промежуточный уровень (или уровни) была перенесена основная тяжесть прикладной обработки. Этот факт делает промежуточный уровень одним из наиболее критических и проблематичных компонентов WebOLTP архитектуры



Новая архитектура для WebOltp

- основные требования к ПО промежуточного уровня:
- масштабируемость и производительность при работе с большим количеством пользователей, сессий, транзакций и соединений с БД,
- высокопроизводительное соединение браузера и back-end хранилища данных,
- поддержка быстрой разработки и развертывания WebOLTP-приложений на промежуточном уровне,
- поддержка как синхронного, так и асинхронного управления транзакциями

3.2 ОПИСАНИЕ ПЛАТФОРМЫ DOT.NET(MICROSOFT.NET)

Microsoft.Net открытая для языков программирования платформа для построении Web-служб. Некоторые эксперты утверждают, что <.NET> представляет прекрасный прмер того, что станет доминирующей моделью для 3-го поколения Интернет – приложений. Отличительной особенностью .NET является ее независимость от языка программирования. Наряду с наличием собственного языка C# она открыты для множества других языков, таких как Cobol, Effel, Fotran, Pel, Pyon, Smalltalk. Цель данной платформысостоит в том,чтобы представить для профессиональных разработчиков абстрактную машину, подходящую как для традиционных (клиент-серверных, n-звенных), так и для Web – ориентированных приложений.

Архитектура .NET Включает 6 уровней: Web – службы; Оболочки и библиотеки ASP.NET, ADO. NET, Windovs Forms; Стандарты обмена SOAP и WSDL; общие средства разработки Vstudio.NET; Компонентная модель; Объектная модель и спецификация единого языка; единая среда исполнения. Перейдем к последовательному рассмотрению соответствующих уровней архитектурры. *Web – службы* предоставляет пользователю индивидуальные и корпоративные службы для электронной коммерции и приложений категории B2B. *Оболочки и библиотеки ASP.NET* – активные серверные страницы поддерживают разработку интеллектуальных Web – сайтов; *ADO. NET* –дополнение на базе языка XML к Activ X Data Objects для объектно-реляционной обработки и баз данных. *Windovs Forms* служит для работы с графикой. Всего .NET содержи тысячи повторно используемых компонентов. *Стандарты обмена* на базе XML служат в качестве платформенно – независимых средств передачи объектов. Из них наиболее важен SOAP – Simple Object Access Protocol – простой протокол доступа к объектам WSDL – Web Services Description Language - язык описания Web- служб. Среда разработки Visual Studio. NET представляет собой единую среду разработки служб для создания компиляции, просмотра и отладки программ, написанных на различных языках. Данная среда позволяет сторонним разработчикам подключать инструментальные средства и компиляторы для других языков. *Компонентная модель*. До появления . NET уже существовали 3 претендента на роль лидера в области моделей и стандартов для разработки на основе компонентов: Corba, J2EE и COM. С помощью . NET можно создать «сборочные» модули, каждый из которых состоит из нескольких классов с четко определенными интерфейсами. Достоинством данного подхода является простота и отсутствие языка описания интерфейсов IDSL. *Объектная модель* предлагает концептуальную основу, на которой зиждется объектно-ориентированная система типов <.NET> и многое другое. *Единая языковая среда* - базовый набор механизмов для исполнения NET- программ вне зависимости от языка, на котором они были написаны.

Перейдем к более подробному рассмотрению достоинств и функциональных возможностей , предоставляемых *ASP.NET* . Данный продукт представляет собой эволюцию технологии .активных серверных страниц.

ASP позволяет программистам реализовывать алгоритмы динамического создания страниц на IIS, состоящих из статического HTML и кода сценариев. Когда пользователь запрашивает ASP-страницу, IIS должен ее найти и активизировать ASP-процессор. ASP-процессор должен прочитать страницу и, один к одному, скопировать содержащиеся на ней HTML-элементы в выходную страницу. По мере расширения Интернет и увеличения потребностей пользователей Web-разработчикам потребовались совершенствование двух ключевых свойств исполняющей среды: простоты программирования и качества

выполнения. ASP.NET и явилось таким усовершенствованием. ASP.NET похожа на оригинальную ASP, и большая часть кода может быть переведена на нее практически без изменений. Но внутренняя реализация ASP.NET полностью переделана с тем, чтобы задействовать возможности Microsoft.NET . Как написано в MSDN "вы обнаружите, что работает она лучше, рушится меньше, и в ней проще программировать. И это приятно."
Смесь HTML-элементов и сценарного кода может казаться логичной, но её достаточно трудно реализовывать и сопровождать, кроме простых случаев. Поскольку код и данные должны и появляются в любом месте страницы, развитые средства разработки, подобные Visual Basic, не могут обрабатывать такие смеси. ASP.NET отделяет HTML от алгоритмов, создавая фоновый код (code-behind). Вместо того чтобы перемешивать HTML с кодом, код пишется в отдельном файле, на который есть ссылка на ASP-странице. В результате такого разделения Microsoft смогла усовершенствовать среду разработки и отладки Visual Studio.NET так, что можно использовать ее при разработке Web-приложений. ASP.NET поддерживает Web Forms — архитектуру Web-страниц, делающую их программирование похожим на программирование форм настольных приложений. При добавлении на страницу управляющего элемента , можно описывать логику для его обработчика событий так же, как при написании настольного приложения на Visual Basic. Простота использования, сделавшая Visual Basic столь популярным, теперь доступна при построении Web-приложений. Так же, как Visual Basic зависит от элементов управления Windows и элементов ActiveX сторонних производителей, Web Forms зависит от нового типа элементов управления — серверных элементов управления Web Forms (Web Forms Server Controls). Это готовые фрагменты алгоритмов для реализации ввода и вывода, которые разработчик помещает на ASP.NET-страницы так же, как он поступал с формами для Windows-приложений. Эти элементы позволяют абстрагироваться от деталей работы с HTML, как Windows-элементы позволяли абстрагироваться от GDI.

Изначально ASP слабо поддерживала защиту. В ASP.NET реализовать защиту гораздо проще. Работая в «чистой» Windows-среде, можно аутентифицировать пользователя (проверить, что он тот, за кого себя выдает) автоматически, используя встроенные средства аутентификации Windows. Для большинства конфигураций, содержащих не только Windows-приложения, ASP.NET имеет готовые средства для создания собственных схем аутентификации. ASP поддерживала управление сеансами с помощью простого API. Главными недостатками были невозможность распространить управление сеансами на более чем одну машину и прерывание сеансов при перезапуске процесса. ASP.NET расширяет поддержку, позволяя выполнять обе функции автоматически. Можно настроить ASP.NET для автоматического сохранения и последующего восстановления состояния сеанса на определенной машине (или, если есть необходимость, использовать большее количество машин). ASP.NET имеет также много новых функций периода выполнения.. ASP.NET автоматически компилирует сценарии при их первоначальной установке или при первом обращении, что существенным образом ускоряет работу. Поскольку она применяет компиляцию по требованию, принятую в Microsoft.NET , нет необходимости останавливать сервер для замены компонента или страницы. Она также поддерживает утилизацию процессов. Вместо того чтобы оставлять процесс работающим бесконечно, ASP.NET можно настроить на автоматическое прекращение и повторный запуск серверного процесса после определенного периода времени или числа сбоев (эти значения настраиваются). Это снимает большинство проблем с утечкой памяти в пользовательском коде. Потенциально ASP.NET проще администрировать, так как все параметры хранятся в читабельном виде в самой иерархии каталогов ASP.

Ориентированные на Web элементы управления в ASP.NET реализуют пользовательский интерфейс, аналогичный GUI – средам, не рассчитанным на Web и намного превосходящий все, что предлагает HTML. ASP.NET берет на себя реализацию одного из самых тонких моментов обработки в Web – поддержку состояния клиента.HTTP не сохраняет состояние, но любой реальный Web интерфейс , например корзина для покупки в ТИС, дожжен сохранять клиентскую информацию при переходе от одной отображаемой страницы к другой. ASP.NET поддерживает состояние сеанса, не храня клиентской информации на сервере, освобождая разработчиков от выполнения этой задачи вручную, используя для этого громоздкие технологии.

В паре с оболочкой ADO.NET ASP.NET позволяет настроить часть страницы Web таким образом, чтобы напрямую (автоматически) отображать содержимое таблиц БД. ASP.NET позволяет и многое другое, но **главное** – *она устраняет различие между традиционной для ИТ разработкой программ (для транзакционных бизнес-приложений) и Web – разработкой.*

Ключевым понятием платформы dot.NET является управляемый код (managed code). Управляемый код работает в среде CLR (Common Language Runtime), который поддерживает более богатый набор служб, чем стандартная ОС Win32 (рис. 1.2.)
Любой инструмент разработки, совместимый с CLR, компилирует свой исходный текст в программы на стандартном языке Microsoft Intermediate Language (MSIL или, для краткости, IL) (рисунок. 1.3.). Поскольку

независимо от языка исходного текста все инструменты разработки выдают программы на одном и том же языке — IL, все различия в их реализации исчезают к моменту достижения CLR. Независимо от формы представления, например, строковых переменных в исходном тексте на самом языке программирования, их внутренняя реализация одинакова в любых программах, так как в CLR все они используют объект String. То же самое верно в отношении массивов, классов и всего остального.

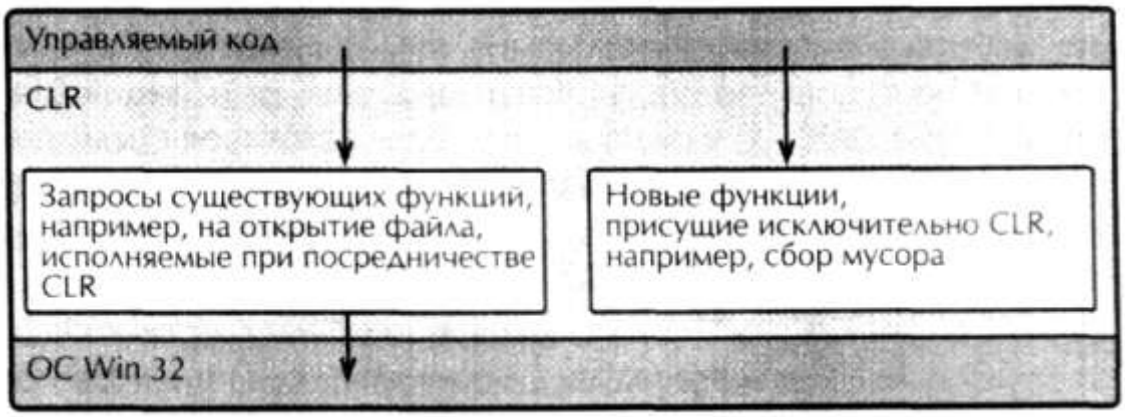


Рисунок 1.2. Управляемое исполнение программы в CLR.

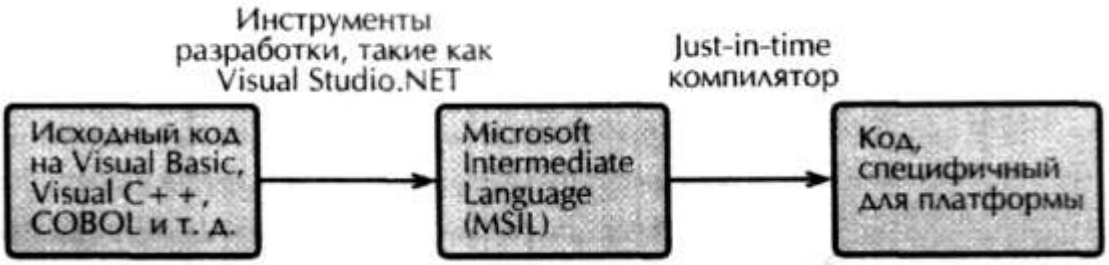


Рисунок 1.3. Исходные тексты на различных языках компилируются в программы на MSIL.

Любая компания может создать язык, совместимый с CLR. В Microsoft Visual Studio.NET входят CLR-совместимые версии Visual Basic, C# (произносится как «Си-шарп»), JScript и C++ . Рассмотрим рис. 1.3. Как видно из рисунка, код на IL, созданный инструментом разработки, не может сразу работать ни на каком компьютере. Для этого требуется следующий шаг - компиляция по требованию (just-in-time, JIT). Утилита компилятор по требованию (just-in-time compiler или JITter) читает исходный текст на IL и производит настоящий машинный код, способный работать на данной платформе. Благодаря этому .NET в некоторой степени независима от платформы, коль скоро для каждой платформы существует свой JITter. Это похоже на обработку бинарного потока данных Java-машиной, но в отличии от Java-машины, JIT пока доступна только для семейства ОС Windows.

. NET Framework поддерживает автоматическое управление памятью с помощью механизма сбор мусора (garbage collection). Программа не обязана явно освобождать выделенную для нее память. CLR определяет, что память больше не используется программой, и автоматически повторно использует ее.

.NET поддерживает управление версиями. Microsoft.NET дает разработчикам серверов стандартизированный способ указания версии (эти сведения находятся в конкретном EXE или DLL-файле) и стандартизированный механизм, позволяющий клиенту указать нужную ему версию. ОС будет поддерживать запросы клиентами сведений о версии с помощью базового набора функций управления поведением, допуская их подмену разработчиком и позволяя в явном виде задавать поведение при управлении версиями. Поскольку в результате компиляции исходных текстов на любом языке получаются тексты на IL, все поддерживающие CLR языки потенциально поддерживают одинаковый набор функций. Microsoft. NET организует функциональность ОС посредством пространства имен System. Все объекты, интерфейсы и функции ОС теперь иерархически организованы, так что искать нужные функции проще. Это также исключает конфликты имен между объектами и функциями разработанными, объектами и функциями самой ОС, и созданными другими разработчиками.

Microsoft.NET поддерживает безопасность доступа к коду. Можно разрешить выполнять определенные действия одной программе и запретить другой. Скажем, программе можно разрешить чтение файлов, но запретить запись. CLR реализует заданные ограничения и блокирует любые попытки выйти за их пределы. Это значит, что к полученному из разных

источников коду можно применять разные уровни доверия. Это позволяет работать с кодом, загруженным, скажем, из Интернет, не беспокоясь за целостность системы.
Microsoft.NET поддерживает бесшовное взаимодействие с COM как в качестве клиента, так и сервера. .NET помещает объект COM в объект-оболочку, в результате первый воспринимается как встроенный объект .NET.

1.2.1. Пространства имён в .NET

В современных 32-разрядных Windows все функции ОС достаточно сложно организовать в алфавитный список. Их очень много , больше 5 000. Для создателей ОС это тоже проблема. Добавляя новую функцию, им приходится выбирать для нее имя, которое должно быть описательным и не конфликтовать с именами уже реализованных функций. Прикладные программисты также должны быть уверены, что имена их глобальных функций не конфликтуют с функциями ОС. Можно сказать, что пространство имен (name-space), набор имён, в пределах которого любое имя должно быть уникальным, достаточно велико. С большими списками легче обращаться, разбив их на меньшие «подсписки», - их легче анализировать. Классическим примером такого подхода может служить кнопка меню Start в Windows. Если бы приложения со всего компьютера были собраны в одно гигантское меню, найти нужное приложение было бы достаточно сложно . Вместо этого меню Start предоставляет относительно короткий список (около 10 пунктов), который легко просматривать в поисках нужного элемента. В каждой группе есть список логических подгрупп, вложенных настолько глубоко, насколько это удобно. В итоге каждая из них раскрывается в короткий список реальных приложений. Чтобы запустить нужное приложение при путешествии по меню, в общей сложности приходится просматривать до 50 элементов, но их никогда не бывает больше десяти (или около того) одновременно. Microsoft.NET предоставляет лучший способ организации функций и объектов ОС. Этот механизм исключает конфликты между именами функций и объектов, написанными другими разработчиками. Он основан на понятии пространства имен. С помощью пространств имен в .NET организована вся функциональность ОС.
Все классы и функции .NET CLR являются частью пространства имен System. Можно сказать, что все объекты и функции, имена которых начинаются с таких символов, «принадлежат к пространству имен System». Конечно, пространство имен System довольно велико, так как оно содержит имена каждого функционального элемента из их богатого набора в ОС. Поэтому оно разделено на несколько подчиненных пространств имен, скажем, System.Console, содержащее все функции, которые имеют дело с вводом-выводом в окно консоли. У некоторых из этих подпространств имен есть еще вложенные «подподпространства» и т. д. вглубь, пока не будет описана вся функциональность. Полностью квалифицированным именем (fully qualified name) функции, которое иногда называют квалифицированным именем (qualified name) или q-именем (q-name), является имя, где перед именем собственно функции стоит полный путь к ней в пространстве имен, или квалификатор (qualifier). Например, System.Console.Write это полностью
квалифицированное имя системной функции, которая выводит информацию в окне консоли. Полностью квалифицированное имя позволяет вызвать функцию из любого места программы. Подобный способ организации функций в логические группы очень удобен и позволяет найти нужную функцию с минимумом затрат времени. Его единственный недостаток в том, что полностью квалифицированные имена могут быть очень длинны. Так, функция System. Runtime. InteropServices. Marshal. ReleaseComObject сразу освобождает заданный объект COM, не выполняя полный сбор мусора. В большинстве .NET-приложений эта функций вообще не нужна, но в некоторых может использоваться многократно. Поэтому аналогично тому, как люди, обращаются друг к другу по имени, CLR позволяет импортировать пространство имен . Импортируя пространство имен, мы как бы говорим компилятору о том, что будем использовать функции из этого пространства имен очень часто, и хотим обращаться к ним по имени. Импорт пространства имен на Visual Basic выполняет ключевое слово Imports, а в C или C# программах - ключевое слово Using.

1.2.2. Сборки.

В Microsoft.NET для хранения кода, ресурсов и метаданных широко используются сборки (assemblies). Весь код, исполняемый средством поддержки периода выполнения .NET, должен находится в сборке. Кроме того, все функции безопасности, разрешения пространств имен и управления версиями работают для отдельной сборки. Поскольку сборки применяются часто и для самых разных нужд, рассмотрим их подробнее. Сборка — это логический набор, состоящий из одного или нескольких EXE- или DLL-файлов, в которых находятся код и ресурсы приложения. В сборку также входит декларация (manifest) -- метаданные,

описывающие код и ресурсы, которые находятся внутри сборки. Зачастую сборки состоят из единственного EXE- или DLL-файла. При использовании таких инструментов, как Visual Studio.NET, каждому проекту часто соответствует одна сборка. Хотя часто сборка располагается в одном файле, она может быть и логической (в противоположность физической) совокупностью нескольких файлов из одного каталога . Декларация с указанием файлов, составляющих сборку, может располагаться в одном из EXE- или DLL-файлов сборки, содержащих код приложения. Она также может существовать в отдельном EXE- или DLL-файле, который при этом не содержит ничего, кроме декларации.

1.2.3 Наследование.

Методика ООП под названием «наследование» намного облегчает написание объектов программ. Допустим, кто-то где-то на поддерживающем наследование языке написал некоторый класс - Класс. Этот класс поддерживает полезную универсальную функциональность, скажем, чтение байтов из потока и запись их в поток. Необходимо создать другой класс, который не только осуществляет чтение и запись, как базовый класс, но и предоставляет некоторую статистику (например, длину последовательности байтов), взяв за основу функциональность базового класса и добавив несколько методов. Для этого нужно написать фрагмент программы, называемый производным классом, в котором функциональность базового класса некоторым образом модифицирована: часть кода добавлена, другая - - изменена, а часть оставлена как есть. Для этого с помощью синтаксиса языка компилятору дается указание, что наш производный класс наследует функциональность базового класса. В результате с помощью соответствующей ссылки компилятор автоматически включит в производный класс функциональность базового класса. Это можно представить в виде операции вырезания/ вставки, при которой не происходит на самом деле никаких перемещений. Говорят, что производный класс наследует от базового класса, происходит от него или расширяет .
Все виды функциональности .NET , от простого преобразования строк до самых сложных служб, предоставлены с помощью наследования. Большое значение придается способности .NET поддерживать перекрестное межъязыковое наследование. Иначе говоря, класс, написанный на одном языке (скажем, на Visual Basic), может быть производным базового класса, написанного на другом языке (скажем, на C#). В COM такая возможность не поддерживалась из-за слишком больших различий в реализации языков. Однако, стандартизированная архитектура CLR на основе IL позволяет приложениям .NET использовать эту функцию.

1.2.4. Управление памятью в .NET

Одним из главных источников неприятных и трудно обнаруживаемых ошибок в современных приложениях является некорректное использование ручного управления памятью. Старые языки, такие как C++, требовали от программиста удалять созданные объекты вручную. Это вызывало две главные проблемы. Во-первых, создав объект, программисты забывали удалить его по окончании использования. Такие утечки в конечном счете полностью истощали память процесса и вызывали его крах. Во-вторых, удалив объект вручную, программист мог позже по ошибке попытаться вновь обратиться к этому адресу памяти. Visual Basic обнаруживает ссылку на недействительную область памяти сразу, а C++ - как правило, нет. Иногда в той области памяти, где раньше был удаленный объект, все еще остаются какие-то внешне нормальные значения, поэтому программа продолжит работу с испорченными данными. В реальных программах объекты часто создаются в одной части программы, удаляются — в другой, а между ними располагается сложная логика, которая в одних событиях удаляет объект, а в других — нет. Все эти ошибки трудно предотвратить, и еще труднее обнаружить. У современных языков, какими являются Visual Basic и Java, таких проблем нет. В этих языках управление памятью осуществляется автоматически . Это стало одной из причин, по которой программисты выбирают их для разработки ПО. Программисту на Visual Basic 6.0, как правило, не обязательно помнить об удалении созданных объектов. Visual Basic 6.0 подсчитывает ссылки на каждый объект и, когда счетчик ссылок упадет до О, автоматически удаляет объект, а освободившуюся память утилизирует. Microsoft ввела автоматическое управление памятью в .NET CLR, что делает его доступным для любого языка.
Программист создает объект оператором new или createobject и получает ссылку на него. CLR выделяет для него память из управляемой кучи (managed heap) -- части памяти процесса, зарезервированной CLR для этой надобности. С определенной частотой системный поток анализирует все объекты в управляемой куче, чтобы выяснить, на какие из них программа удерживает ссылки. Объект, на который не осталось ни одной ссылки, называется мусором (garbage) и удалятся из управляемой кучи. После этого оставшиеся в управляемой куче объекты дефрагментируются, а имеющиеся у программ ссылки устанавливаются на

новые адреса объектов. Таким образом, вышеуказанные проблемы ручного управления памятью решаются без написания кода. Невозможно забыть удалить объект, так как система «убирает» объекты самостоятельно. Невозможно и обратиться к удаленному объекту через недействительную ссылку, поскольку объект не будет удален, пока на него есть хоть одна ссылка. Очевидно, для сборщика мусора потребуется больше циклов процессора, чем для стандартного диспетчера кучи, даже если он написан так, чтобы не проверять один и тот же объект дважды и распознает циклические ссылки. Как сказано выше, я считаю, что это правильная нагрузка на процессор, поскольку при этом ускоряется разработка и уменьшается число ошибок.
До сих пор автоматический сбор мусора показывал себя замечательно, но кое-что было упущено. Например, освобождение ресурсов при освобождении объекта. Приложения на C++ обычно выполняли освобождение ресурсов внутри деструктора объекта, а классы Visual Basic -- внутри их метода Class_Terminate. Это удачное место для размещения кода, выполняющего освобождение ресурсов, так как клиент не сможет забыть вызвать его. Но как сделать это во время автоматического сбора мусора? Главной задачей освобождения ресурсов, выполняемой в деструкторах C++, было удаление дополнительных объектов, на которые ссылался удаляемый объект. Теперь об этом позаботится сборщик мусора. Но время от времени приходится выполнять освобождение ресурсов без участия механизма локального сбора мусора, скажем, при освобождении соединения с БД . Сборщик мусора CLR поддерживает завершитель (finalizer) — метод объекта, вызываемый при удалении объекта сборщиком мусора. В силу определенных причин завершитель аналогичен деструктору классов C ++ и методу Visual Basic класса Terminate, заменителем которых он является. Однако завершитель существенно отличается от обоих этих механизмов. Когда объект удаляется сборщиком мусора, поток сборщика обнаруживает наличие у объекта метода Finalize и вызывает его, исполняя, таким образом, код освобождения ресурсов.
У завершителей есть несколько недостатков. Ясно, что этот механизм потребляет процессорное время; не стоит его использовать, если очистка не нужна. Невозможно дать гарантии относительно порядка вызова сборщиком мусора завершителей удаляемых им объектов, поэтому необходимо по возможности исключить всякую зависимость от этих действий в программах. Завершители вызываются в отдельном потоке сборщика мусора, поэтому их никак нельзя упорядочить, чтобы навязать собственный порядок вызова этих методов. В противном случае может быть разрушена вся система сбора мусора процесса. Завершители не вызываются при завершении приложений, так как программа «понимает», что при завершении процесса нужно освободить все ее внутренние ресурсы. В ранней версии .NET была функция, позволявшая заставить систему вызывать завершители при закрытии приложений, но из рабочей версии ее убрали. Завершители идеально подходят для задач, при решении которых нам не важно, когда будет выполнена очистка. В этом есть свои плюсы и минусы, если ресурсы, которые должен освободить завершитель, дефицитны и лимитируют работу процесса (например, при соединении с БД). Эвентуальное освобождение не вполне приемлимо, если объект должен быть уничтожен немедленно, чтобы освободить занятые им ресурсы, о которых не знает сборщик мусора. Как сказано выше, можно инициировать немедленный сбор мусора, но для этого потребуется анализ всей управляемой кучи, что довольно расточительно, если не нужна очистка для других объектов. Поскольку объект, подлежащий разборке, известен, нужен способ очистить лишь этот объект. Подобная операция имеет имя: детерминированное завершение (deterministic finalization).
Объекты, которым требуется поддержка детерминированного завершения, реализуют интерфейс Disposable, содержащий единственный метод Dispose. В этом методе можно разместить любой код, освобождающий дефицитные ресурсы. Клиент, вызывая этот метод, дает объекту указание немедленно освободить эти ресурсы. Примером может служить системный класс System.Windows.Forms.DataGrid.

1.2.5. Взаимодействие с COM

Критическими факторами коммерческого успеха любой новой платформы являются ее интегрируемость с имеющимися платформами и одновременная поддержка новых средств разработки лучших приложений. Например, Windows 3.0 поддерживала не только приложения DOS и обеспечивала их многозадачность лучше, чем любой другой продукт в то время, но и предоставляла платформу для создания приложений Windows, которые были лучше, чем любое приложение для DOS. С 1993 г. области взаимодействия между приложениями Windows зависят от COM. COM пронизывает практически весь код среды Windows. Microsoft.NET приходится поддерживать COM, чтобы получить шанс на коммерческий успех. Клиент .NET может использовать сервер COM, и наоборот. Рассмотрим рисунок 1.4. Как видно из рисунка, клиент .NET обращается к серверу COM через вызываемую оболочку периода выполнения (runtime callable wrapper, RCW). RCW является оболочкой объекта COM и работает как посредник между ним и средой CLR, позволяя клиентам .NET воспринимать объект COM просто как встроенный объект .NET, а клиент .NET при этом рассматривается объектом COM просто как стандартный клиент COM. У разработчика

клиента .NET есть несколько способов генерации RCW, если воспользоваться Visual Studio.NET, щелкнув правой кнопкой раздел проекта References и выбрать из контекстного меню Add Reference. При этом в диалоговом окне , в котором предложены на выбор все зарегистрированные объекты COM, найденные в системе, выбрать объект COM, для которого нужно генерировать RCW, Visual Studio.NET выдаст её. Без Visual Studio.NET аналогичную задачу позволяет выполнить Tlblmp.exe (средство импорта библиотек типов) — инструмент командной строки из .NET SDK. Алгоритм, который читает библиотеку типов и генерирует код RCW, на самом деле находится в классе периода выполнения .NET под названием System.Runtime.InteropServices.TypeLib-Converter. Как Visual Studio.NET, так и Tlblmp.exe используют этот класс.

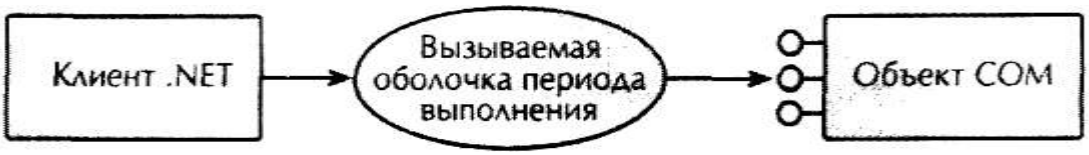


Рисунок 1.4. Взаимодействие .NET с COM через вызываемую оболочку периода выполнения.

Сгенерировав RCW, следуя описанию в предыдущем абзаце, можно импортировать его пространство имен в клиентскую программу оператором Imports. Это позволит ссылаться на объекты по их сокращенному имени. Объект RCW можно создать просто оператором new, как и любой другой объект .NET. При создании RCW вызывает встроенную функцию COM CoCreate-lnstance. Таким образом, создается объект COM, для которого генерируется оболочка. После этого клиентская программа .NET может вызывать методы RCW так, как если бы она была встроенным объектом Microsoft.NET. RCW автоматически приводит каждый вызов в соответствие с правилами вызовов COM, например, конвертирует строки .NET в строки BSTR, необходимые COM, и направляет их объекту. Прежде чем передать клиенту результат, который вернул объект COM, RCW конвертирует его во встроенные типы COM.

1.2.6. Использование объектов .NET из COM

Допустим, у нас есть клиент, уже использующий COM, который надо заставить использовать объект Microsoft.NET. В силу некоторых причин это менее распространенный сценарий, чем предыдущий так как он предполагает разработку новых объектов COM в мире .NET. Но нетрудно предвидеть ситуацию, когда у нас есть клиент, использующий 10 объектов COM, при этом требуется добавить 11-й набор функциональности, который существует только в виде объекта .NET. При этом клиент должен их все воспринимать одинаково из соображений единообразия. Посредством COM callable wrapper (CCW) .NET может справиться и с этой ситуацией . CCW служит оболочкой объекта .NET и играет роль посредника между ним и средой CLR, заставляя клиент COM воспринимать объект .NET так, как если бы это бы объект COM.

1.2.7. Транзакции в .NET

Транзакции нужны для защиты целостности данных в распределенных системах. Допустим, разрабатывается приложение, которое принимает оплату счетов в диалоговом режиме. Для оплаты телефонного счета нужно снять некую сумму с моего счета, который находится в некоторой БД, и перевести ее на счет телефонной компании, который, возможно, расположен в другой БД на другой машине. Если операция списания со счета проходит успешно, но по какой-то причине зачисление на счет оканчивается неудачей, надо отменить списание со счета, иначе деньги будут потеряны, а целостность данных в системе нарушена. Требуется уверенность в успехе или неудаче обеих операций. Именно этого можно достичь, объединив выполнение обеих операций в одной транзакции. Если обе операции завершаются успешно, транзакция фиксируется, и новые значения счетов сохраняются. Если хоть одна операция терпит неудачу, транзакция отменяется, и суммы на всех счетах откатываются к исходным значениям. У COM + и его предка, Microsoft Transaction Server, была автоматическая поддержка, которая облегчала написание объектов, участвующих в обработке транзакций. Программист в административном порядке помечает свои объекты как требующие участия в транзакции. Затем, при активизации объекта, COM+ автоматически использовала транзакцию.
Объекты пользовались менеджером ресурсов COM + Resource Managers, с помощью которого такие программы, как Microsoft SQL Server, поддерживающие обработку транзакций в стиле COM+, вносили изменения в БД. После этого объект сообщал COM+, "удовлетворен" ли он полученными результатами.

Если все объекты, участвующие в транзакции, оказывались "удовлетворены", COM+ подтверждала транзакцию и давала Resource Managers указание сохранить все внесенные изменения. В противном случае COM+ отменяла транзакцию, приказывая Resource Managers отменить результаты всех операций над объектами и откатывая систему в исходное состояние. Встроенные объекты .NET тоже могут принимать участие в транзакциях. Поскольку имеющаяся система обработки транзакций, разработанная Microsoft, основана на COM, объекты .NET могут делать это благодаря своей способности к взаимодействию с COM, описанной в предыдущем разделе. Необходимо зарегистрировать объект .NET как сервер COM. После этого через COM+ Explorer нужно установить этот компонент в приложение COM + и настроить его требования к транзакциям точно так, как если это был встроенный объект
COM. А инструментом командной строки Regsvcs.exe можно одновременно выполнить регистрацию объекта и настройку приложения COM+. Есть альтернативный путь: подобно встроенным объектам COM, которые иногда указывают свои требования к транзакциям в их библиотеках типов, можно задать требования объектов .NET к транзакциям через метаданные этих объектов. Рассмотрим фрагмент кода :
Public Class < TransactionAttribute (TransactionOption.Required) > Classl
Данный фрагмент демонстрирует написанный на Visual Basic класс .NET с атрибутом, указывающим, нужны ли объекту этого класса транзакции.
Если этот объект реализован посредством C#, все было бы точно так же, кроме того, что вместо угловых скобок были бы использованы квадратные. Рассмотрим фрагмент кода :
<%@ page Transaction= "Required" %>
Страница ASP.NET указывает требования размещенного на ней кода к транзакциям через приведённый выше атрибут.

.ASP.NET

Изначально Интернет служила для передачи статических страниц с текстом и графическими изображениями, что было реализуемо. Нужно было просто принять URL , идентифицирующий файл, взять этот файл с диска сервера и передать этот файл клиенту. Рассмотрим рисунок 1.5

Рисунок 1.5 Сервер передает статические Web-страницы.

Из рисунка видно, что даже такая простая архитектура позволяет многое. Однако, такой подход был приемлем на первоначальном этапе развития сети Интернет, когда все данные в Интернет были статическими (отображая только созданное автором содержание без программной логики и возможности вмешательства со стороны пользователя) и открытыми (доступными всем, кто знал или мог найти адрес). Но со временем пользователям Интернет потребовалась большая функциональность, некоторые из пользователей, например, желали иметь возможность просматривать свои банковские счета. Но при статическом подходе это неосуществимо. Банк не может каждый день создавать новую страницу для любого возможного представления каждого счета — страниц окажется слишком много. Вместо этого пользователю нужно ввести некоторую информацию, такую как номер счета, а банковский компьютер должен по запросу создать HTML-страницу, отображающую баланс клиентского счета. (см. рис. 1.6)
Серверному приложению, динамически генерирующему страницы для клиента, требуется:
1. Способ увязки программной логики с запросом страницы. Когда пользователь запрашивает страницу, сервер не может просто извлечь ее с диска -- до этого запроса ее просто нет. Вместо этого сервер должен выполнить некий алгоритм для генерации страницы.
2. Браузер *посылает* запрос серверу.

Рисунок 1.6. Сервер динамически генерирует Web-страницы на основе полученной от клиента информации.

В примере с банком , вероятно, потребуется просмотреть банковскую БД и найти текущий баланс пользовательского счета и последние транзакции. Желательно реализовать этот алгоритм легко и быстро, с применением уже знакомых языков и инструментов. И хотелось бы, чтобы это работало достаточно эффективно. Кроме того, Web-серверу нужен способ передачи вводимой информации серверному алгоритму и возврата выходной информации от этого алгоритма пользователю. Пользовательский браузер передает серверу HTML-форму, содержащую элементы управления для ввода данных, которые определяют интересующую его информацию, например, номер счета и период времени, за который пользователь хочет просмотреть транзакции. Извлечение данных с этой HTML-страницы достаточно трудоёмкий процесс, поэтому необходимо иметь готовый способ брать передаваемые параметры быстро и легко.
Далее , поскольку по крайней мере некоторые данные теперь являются частными, наш Web-сервер должен знать, кем является пользователь, чтобы позволять ему просматривать и делать только то, что ему разрешено. Необходимо чтобы пользователь имел возможность видеть свой счет и использовать этот счёт в своих нуждах. Написание такого кода — дело сложное и дорогое, и убедить недоверчивых заказчиков в его 100% надёжности очень сложно. Вывод- нужна инфраструктура с готовыми средствами защиты. И, наконец, Web-серверу нужен механизм управления пользовательскими сеансами. Для пользователя взаимодействие с Web-сервером — это не последовательность запросов, но диалог, «сеанс». Он ожидает, что Web-сайт может запоминать, что ему было передано некоторое время назад. Скажем, на сайте электронного магазина он хочет складывать товары в корзину так, чтобы они хранились там до расчета. Чтобы этого добиться, нужен соответствующий код -- запрос к отдельной странице не способен поддержать такую функциональность. Это опять же неотъемлемая часть большинства Интернет-приложений, так что необходимо иметь готовую, простую в использовании реализацию. В идеале это должно работать в многосерверной среде и быть устойчивым к сбоям. Таким образом, нашему серверу нужна исполняющая среда, предоставляющая готовые решения проблем, которые возникают при программировании большинства Web-серверов. Хотелось бы, чтобы она упрощала программирование, администрирование и развертывание. Нужно чтобы ее можно было распространять хотя бы на несколько, а лучше - на множество серверов.
В конце 1997 г. Microsoft реализовала относительно простую среду периода выполнения для Web — Active Server Pages (ASP) как часть сервера Internet Information Server, включенного в Windows NT 4 Option Pack. IIS обслуживает Web-страницы, запрашиваемые пользователем. ASP позволяет программистам реализовывать алгоритмы динамического создания страниц на IIS, состоящих из статического HTML и кода сценариев. Когда пользователь запрашивает ASP-страницу, IIS должен ее найти и активизировать ASP-процессор. ASP-процессор должен прочитать страницу и, один к одному, скопировать содержащиеся на ней HTML-элементы в выходную страницу. Рассмотрим фрагмент кода: <html style="color:#OOOOFF; "><body> The time is: <% =time %> on <% =date %><body></html>
В данном простом фрагменте атрибут style устанавливает голубой цвет текста. При этом также интерпретируются элементы сценариев, расположенные между ограничителями <% %>. Этот код должен выполнять алгоритм, выдающий в качестве результата HTML-строки, которые ASP-процессор должен скопировать в выходную страницу в те места, где были элементы сценария. Результирующая страница, собранная из статических HTML-элементов и HTML, динамически сгенерированного сценарием, должна быть передана клиенту. Для простых задач ASP применять относительно легко, что является признаком качества этой технологии. Рассмотрим рисунок 1.7.

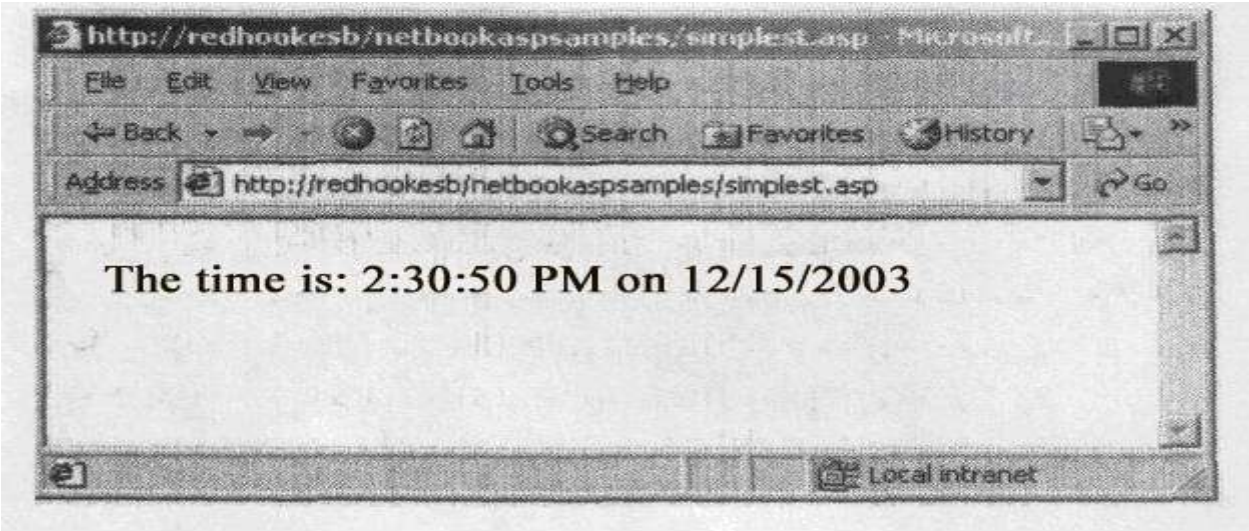


Рисунок 1.7. Web-страница, созданная ASP после обработки фрагмента, приведённого выше.

Данный рисунок иллюстрирует собой страницу созданную посредством комбинирования HTML тегов и серверного сценария.
По мере расширения Интернет и увеличения потребностей пользователей Web-разработчикам потребовались совершенствование двух ключевых свойств исполняющей среды: простоты программирования и качества выполнения. ASP.NET и явилось таким усовершенствованием. ASP.NET похожа на оригинальную ASP, и большая часть кода может быть переведена на нее практически без изменений. Но внутренняя реализация ASP.NET полностью переделана с тем, чтобы задействовать возможности Microsoft.NET . Как написано в MSDN "вы обнаружите, что работает она лучше, рушится меньше, и в ней проще программировать. И это приятно."
Смесь HTML-элементов и сценарного кода может казаться логичной, но её достаточно трудно реализовывать и сопровождать, кроме простых случаев. Поскольку код и данные должны и появляются в любом месте страницы, развитые средства разработки, подобные Visual Basic, не могут обрабатывать такие смеси. Это значит, что писать ASP-код сложнее, чем другие виды кода, например приложения с пользовательским интерфейсом на базе форм Visual Basic. ASP.NET отделяет HTML от алгоритмов, создавая фоновый код (code-behind). Вместо того чтобы перемешивать HTML с кодом, код пишется в отдельном файле, на который есть ссылка на ASP-странице. В результате такого разделения Microsoft смогла усовершенствовать среду разработки и отладки Visual Studio.NET так, что можно использовать ее при разработке Web-приложений. В оригинальной ASP реализация ввода и вывода представляла сложности из-за невозможности отвлечься от HTML. Программист часто вынужден бороться с довольно неприглядными языковыми конструкциями HTML, вместо того чтобы думать о логике своей программы, а это не лучшее использование ресурсов. Скажем, получение данных из HTML-форм требует больше усилий, чем в настольных приложениях. Генерация выходной информации требует, чтобы программист собирал отдельные фрагменты HTML, а это опять же не то, на что программист должен тратить время. ASP.NET поддерживает Web Forms — архитектуру Web-страниц, делающую их программирование похожим на программирование форм настольных приложений. При добавлении на страницу управляющего элемента , можно описывать логику для его обработчика событий так же, как при написании настольного приложения на Visual Basic. Простота использования, сделавшая Visual Basic столь популярным, теперь доступна при построении Web-приложений. Так же, как Visual Basic зависит от элементов управления Windows и элементов ActiveX сторонних производителей, Web Forms зависит от нового типа элементов управления — серверных элементов управления Web Forms (Web Forms Server Controls). Это готовые фрагменты алгоритмов для реализации ввода и вывода, которые разработчик помещает на ASP.NET-страницы так же, как он поступал с формами для Windows-приложений. Эти элементы позволяют абстрагироваться от деталей работы с HTML, как Windows-элементы позволяли абстрагироваться от GDI. Например, нет необходимости вникать в синтаксис HTML для установки основного цвета и цвета фона для строки текста. Вместо этого используется элемент управления «надпись» и код для работы с его свойствами, как это делается на любом современном языке программирования. Изначально ASP слабо поддерживала защиту. В ASP.NET реализовать защиту гораздо проще. Работая в «чистой» Windows-среде, можно аутентифицировать пользователя (проверить, что он тот, за кого себя выдает) автоматически, используя

встроенные средства аутентификации Windows. Для большинства конфигураций, содержащих не только Windows-приложения, ASP.NET имеет готовые средства для создания собственных схем аутентификации. ASP поддерживала управление сеансами с помощью простого API. Главными недостатками были невозможность распространить управление сеансами на более чем одну машину и прерывание сеансов при перезапуске процесса. ASP.NET расширяет поддержку, позволяя выполнять обе функции автоматически. Можно настроить ASP.NET для автоматического сохранения и последующего восстановления состояния сеанса на определенной машине (или, если есть необходимость, использовать большее количество машин). ASP.NET имеет также много новых функций периода выполнения. Первоначальная ASP выполнялась медленнее, так как код сценариев интерпретировался во время выполнения. ASP.NET автоматически компилирует сценарии при их первоначальной установке или при первом обращении, что существенным образом ускоряет работу. Поскольку она применяет компиляцию по требованию, принятую в Microsoft.NET , нет необходимости останавливать сервер для замены компонента или страницы. Она также поддерживает утилизацию процессов. Вместо того чтобы оставлять процесс работающим бесконечно, ASP.NET можно настроить на автоматическое прекращение и повторный запуск серверного процесса после определенного периода времени или числа сбоев (эти значения настраиваются). Это снимает большинство проблем с утечкой памяти в пользовательском коде. Потенциально ASP.NET проще администрировать, так как все параметры хранятся в читабельном виде в самой иерархии каталогов ASP.

2 Сравнительный анализ ASP и ASP.NET.

ASP и ASP.NET - две похожие среды исполнения, и это неудивительно, так как ASP.NET является усовершенствованной версией ASP. ASP.NET , являясь частью новой платформы dot.net унаследовала лучшие её свойства Так например ASP и ASP.NET поддерживают VBScript и jscript, два языка программирования, предназначенные для разработки серверных сценариев. Но существуют и принципиальные отличия ASP и ASP.NET, которые делают ASP.NET более гибкой и удобной в использовании. Например, - элементы управления.

2.2.1. Элементы управления.

Операции ввода/вывода повторяются очень часто . Сделать их более простыми в использовании — обернуть в оболочку, которую сможет использовать любой программист, — отличная идея. Благодаря этому, интерфейс каждой программы будет более согласованным и, следовательно, более понятным. Так в Windows 3.0, в силу отсутствия стандартного интерфейса , программист должен был заниматься разработкой элементов интерфейса , что отнимало много времени. В ASP.NET элемент управления — это любой алгоритм, связанный с пользовательским интерфейсом, который способен:
а) показывать свою программную модель среде разработке, такой как Visual Studio.NET, и б) переводить свое представление в HTML для отображения в любом стандартном браузере (примерно так же, как элемент управления Windows переводит свое представление в последовательность вызовов Windows GDI). Большинство пользователей Интернет знают, что в языке разметки гипер текста (HTML) они также присутствуют, например, кнопки, флажки и ссылки. Серверные сценарии ASP работают именно с ними , принимают их значения и реализуют определённую бизнес-логику в зависимости от принятых значений. HTML поддерживает порядка десяти элементов управления, которые являются частью языка, но здесь есть жесткие ограничения. Во-первых, они не столь многочисленны и возможности их не так богаты. Многим разработчикам хотелось бы иметь гораздо больше функциональности, чем предоставлено существующими элементами управления HTML. Во-вторых, для них трудно писать код. Возможность взаимодействия с ними в период выполнения ограничена. Среда элементов управления (Web Forms) в ASP.NET, в которой существуют элементы, имеет управляемую событиями программную модель, подобную Visual Basic. Их гораздо проще программировать, они лучше поддерживаются средами разработки и позволяют абстрагироваться от многих различий между разными браузерами. Эти элементы более функциональны и программируются значительно проще. ASP.NET поставляется с набором основных следующих элементов управления, которые приведены в таблице 2.2.

Таблица 2.2. Элементы управления ASP.NET.

Функция	Элемент	Описание
Отображение текста	Label	Отображает текст, который пользователь не может редактировать.

Редактирование текста	TextBox	Отображает текст, введенный в период разработки, который может быть отредактирован пользователем в период выполнения или изменен программно.
Выбор из списка	DropDownList	Позволяет пользователю выбрать значение из списка.
Отображение графики	Image Ad Rotator	Выводит последовательность (предопределенную или случайную) изображений.
Установка значения	CheckBox RadioButton	Отображает окошко, щелкнув которое, пользователь устанавливает или сбрасывает значение. Кнопка-переключатель, может быть установлена или сброшена.
Установка даты	Calendar	Отображает календарь, позволяя пользователю выбрать дату.
Команды	Button LinkButton ImageButton	Применяется для выполнения той или иной задачи. Работает как Button(кнопка), но выглядит как гиперссылка. Работает как Button, но вместо текста содержит изображение.
Управление навигацией	HyperLink	Создает навигационную гиперссылку.
Таблицы	Table TableCell TableRow	Создает таблицу. Создает отдельную ячейку в строке таблицы. Создает строку в таблице.
Группирование других элементов	CheckBoxList Panel RadioButtonList	Создает набор элементов CheckBox. Создает на форме панель без рамки. применяемую в качестве контейнера для других элементов управления Создает группу кнопок-переключателей. Внутри этой группы может быть выбран только один переключатель.

	Repeater DataList DataCrid	Отображает информацию из набора данных, используя набор указанных HTML-элементов и элементов управления для каждой записи из набора данных. Подобен Repeater, но с большими возможностями форматирования и позиционирования включая возможность отображения информации в таблице. Позволяет также определить возможности редактирования. Отображает информацию в табличной форме. Обычно имеет привязку к данным.

Если расположить элементы управления на форме, то среда Visual Studio.NET сгенерирует на ASPX-странице соответствующие операторы..

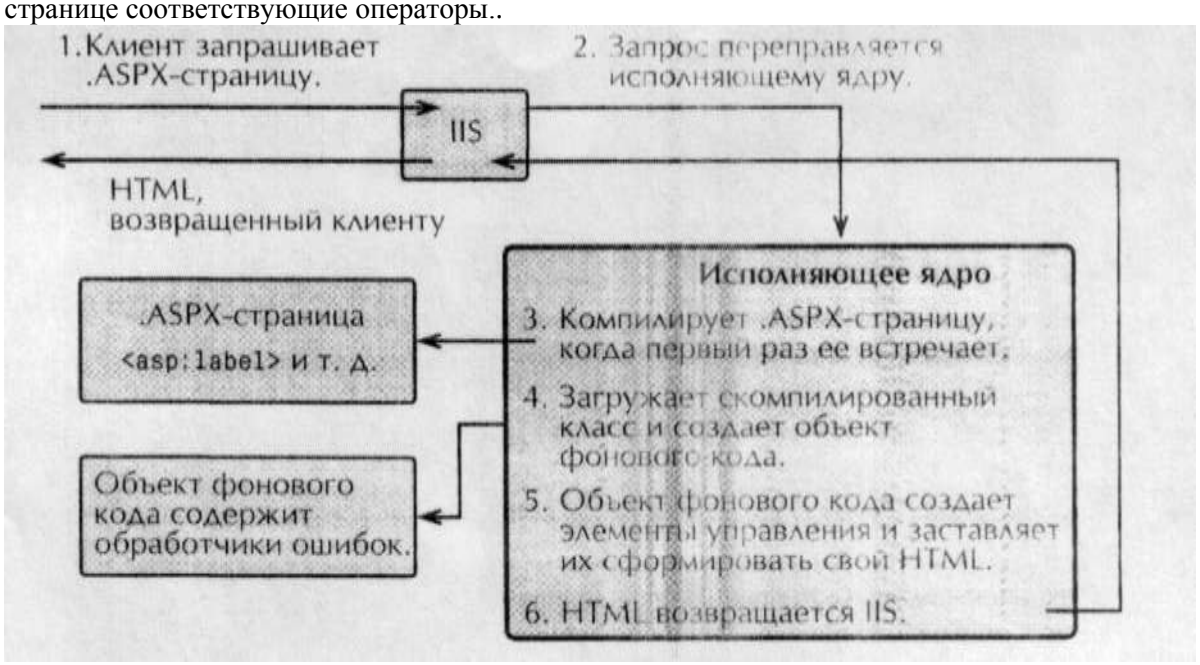


Рисунок 2.2. Процесс обработки.ASPX - сценария ядром.

ASP.NET, как и ASP, поддерживает стандартные элементы управления HTML и может принимать и обрабатывать их значения.

Другая отличительная возможность элементов управления для ввода (списков, полей редактирования, флажков, кнопок-переключателей и т. д.) — это их способность автоматически запоминать состояние, в котором они были до передачи на сервер. В Microsoft называет эту возможность регистрацией ответных данных (postback data). Допустим, сервер посылает браузеру страницу, содержащую сброшенный флажок. Затем человек, работающий с браузером, устанавливает этот флажок и отправляет форму серверу. Если сервер обрабатывает форму и ее же отправляет клиенту, флажок автоматически не запомнит свое состояние. Разработчику, который использует ASP

придется писать код, который будет обеспечивать соответствие состояния флажка тому, что было при передаче от клиента. Элементы управления в ASP.NET не требуют написания такого кода — они автоматически запоминают свое предыдущее состояние в пределах сеанса (боле подробный анализ сеансов в ASP и ASP.NET будет приведён далее). На самом деле, данные хранятся в выделенном поле заголовка HTTP. Это автоматическая функция, которую при желании можно отключить, установив свойство MaintainState элемента управления в False.
Отображающие элементы управления в ASP.NET, такие как надписи, списки и сетки данных, поддерживают собственную версию фиксации своих свойств состояние отображения (viewstate). Хотя пользователи и не могут установить их параметры, они все равно запоминают состояние, в котором их оставила программа при предыдущей передаче. Среда ASPX-страниц автоматически помещает скрытый элемент для ввода _VIEWSTATE на каждую страницу.
Элементы управления имеют также возможность определения конкретного браузера, на котором пользователь будет смотреть страницу, что позволяет им формировать HTML, использующий преимущества различных браузеров таких как Internet Explorer и Netscape. Разработка новых элементов управления в принципе несложна, поскольку .NET содержит готовые базовые классы, от которых можно унаследовать нужную элементу инф-раструктуру. Но этот вопрос выходит за рамки данного дипломного проекта.
Выше я упоминал о том , что каждый элемент управления обладает своими свойствами (что, в общем-то и понятно, так как эти элементы являются экземплярами классов -объектами). Эти свойства могут быть программно изменены в ответ на некоторое событие , например, нажатие кнопки, в соответствии с логикой некоторого фрагмента серверного сценария , например функцией.

2.2.2. События и фоновый код.

Понятие событие неразделимо от элементов управления. Однако не все события присущи элементам управления . Примером такого события является загрузка страницы , к которой можно привязать различную программную логику.
Но меня, в первую очередь интересуют элементы управления. Событийная модель не нова , она уже достаточно долго используется в средах визуально-компонентной разработки, в таких, например, как C Builder или Visual Basic. Элементы управления в ASP.NET являются своего рода аналогами тех компонентов, которые используются в разработке обычных Win32 приложений. На мой взгляд, это очень грамотный и коммерчески выгодный ход корпорации Microsoft. Несложно также предположить и то, что это первая серьёзная попытка стандартизировать , привести к общему знаменателю разработку обычных Win32 приложений и Интернет приложений. К каждому событию каждого элемента управления может быть привязана некоторая логика , например, навели мышь на кнопку (элемент управления button) - изменилась надпись на этой кнопке , щёлкнули - произвели запрос к базе данных. И так с каждым элементом. Каждое событие может быть связано с некоторым фрагментом кода функцией, посредством которой реализуется программная логика. Обе среды исполнения и ASP и ASP.NET поддерживают механизм включения в сценарий других сценариев при помощи директивы include (#include file=" имя файла"), что очень удобно. Можно включать отдельные файлы в зависимости от условий и, таким образом, производить как бы сшивание (объединение) логики всего "приложения" в одном файле. В оригинальной версии ASP обязательно было смешивать серверный сценарий с HTML, что не всегда удобно. Если "приложение" достаточно большое (т.е. используются серверные сценарии, где много кодов), то в процессе разработки программисту становится всё сложней разбираться в этой смеси. В ASP.NET это проблема решена. Серверный скрипт , который соответствует HTML шаблону, можно размешать в отдельном файле. Данный файл получает название фонового кода, а затем производит их связку. В заголовке HTML шаблона для этого необходимо указать путь к этому файлу . На практике, многие разработчики используют комбинирование.
Интересной особенностью ASP.NET по сравнению с ASP является использование языка программирования C# для реализации серверной части интернет "приложения"

2.2.3. Язык программирования C#.

Данный язык программирования стал доступен с выходом Visual Studio.NET , он предназначен как для разработки Win32 приложений, так и для написания серверной части Интернет приложений.
C# и C++ - эти два языка программирования во многом похожи , оба реализуют принципы объектноориентированного программирования : инкапсуляцию, полиморфизм , наследование. Но

имеются и отличия : в C# меньше типов данных , имеется возможность автоматического распределения памяти и некоторые другие возможности.

2.2.4. Работа с базами данных в ASP.NET.

На сегодня самый популярный интерфейс Microsoft для связи с БД ADO. Он располагается поверх OLE DB и таким образом подключается к базе данных или другом источнику данных , для которого существует провайдер OLE DB.

ADO представляет собой семейство классов COM, поддерживающие двойные интерфейсы. Таким образом , ADO могут быть доступны как из сценариев, так и из языков программирования с ранним связыванием. Это означает, что ADO можно использовать в ASP и, следовательно, работать на среднем уровне web-приложений , обеспечивая средство обмена информацией с уровнем данных т.е. использовать COM компоненты из ASP. Рассмотрим рисунок 2.3.

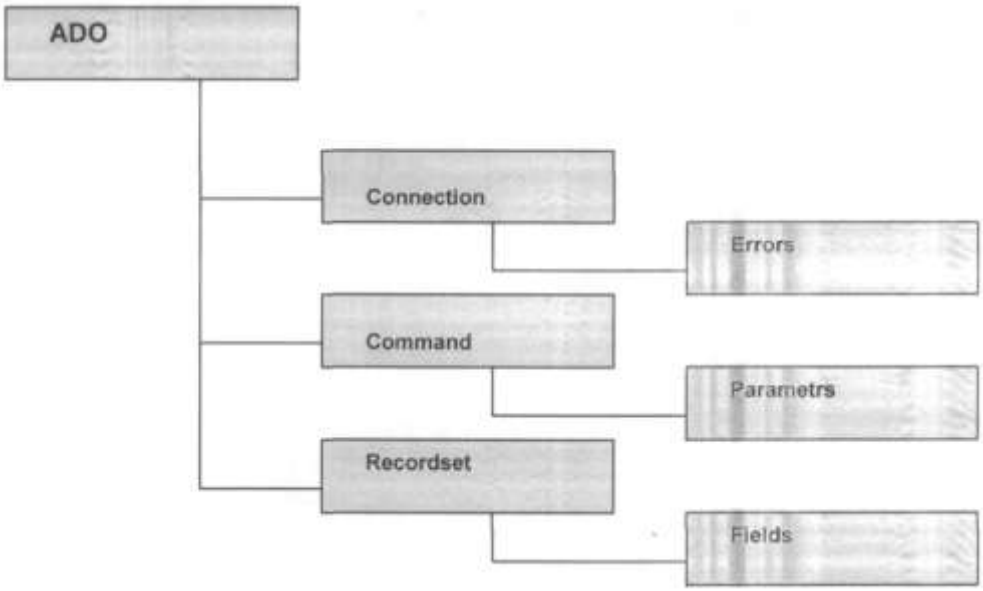


Рисунок 2.3. Упрощенная объектная модель ADO.

Данный рисунок представляет собой упрощенную модель объектов ADO.
Рассмотрим эту модель более подробно.
Connection:
Для подключения к источнику данных необходим объект подключения Connection. Самое важное свойство этого объекта - ConnectionString. Это свойство используется провайдером OLE DB для подключения к источнику данных. Далее этот параметр передаётся методу open объекта и ADO выполняет подключение к заданному источнику данных или выдаёт сообщение об ошибке.
ADO выполняет объединение подключений в пул. Это означает, что если происходит подключение к источнику данных, а затем уничтожение этого подключения, то при последующем подключении ADO не устанавливает соединение заново, а использует существующее, взяв его из пула подключений. Будучи созданным, объект Connection может быть передан объекту Recordset или Command.
Command:
Данный объект используется для единичного запроса к БД и не возвращает записей .
Recordset:
В ADO , как и в других интерфейсах баз данных Microsoft , для получения данных из БД используется набор записей(recordset) , который может состоять из множества строк и множества столбцов Таким образом объект recordset содержит коллекцию Fields объектов Field.
Рассмотрим фрагмент кода:

```
<%@ language="vbscript" %>
<html><body>
<%   set con=server.createobject("adodb.connection"):con.open "'
set rec=server.creteobject("adodb.recordset"):sql_="select * from customer" rec.open sql_,con %>
<h1>Клиенты</h1>
```

<% do while(not rec.eof):response.write rec("customerid")+"
":rec.movenext:loop </body></html>
В данном примере в начале создаётся объект con класса connection , затем производится подключение к базе данных с названием shop. После создаётся объект rec класса recordset и стринговая переменная sql_ со значением select * from customer (где customer наименование таблицы в БД shop). Далее, при помощи метода open объекта rec выполняется запрос к БД. Параметром метода open является стринговая переменная sql_. Объект rec теперь содержит все записи удовлетворяющие sql запросу. И наконец, производится вывод значения столбца customerid таблицы customer для каждой записи посредством итерации до последней записи в объекте rec. При этом, на каждом шаге итерации производится вставка тега
. ASP.NET взаимодействует со своей версией ADO и называется она ADO.NET. Рассмотрим таблицу 2.3.

.

ADO.NET Объект	Описание.
DataReader	Обеспечивает, поток данных с источника данных , с жёстким порядком вывода записей (т.е записи выводится с первой до последней по порядку и никак иначе.)
DataSet	Обеспечивает доступ к реляционным данным. DataSet независим от любых источников данных и может работать сразу с несколькими источниками данных включая базы данных , XML файлы и другими данными. Причём эти данные могут находиться в нескольких таблицах базы данных. В DataSet можно одновременно работать с несколькими записями , в отличии от ADO где допускалась работа только с одной записью в момент времени. Кроме того DataSet может содержать в нутрии себя несколько наборов записей и связи между ними. В отличии от ADO Recordset где допускался только один набор записей.
DataAdapter	Обеспечивает связь между источником данных и DataSet.

Таблица 2.3. Объекты ADO.NET
.

В таблице приведены только три объекта ADO.NET , на практике их больше. Два из них являются своего рода аналогами ADO Recordset , но обладают более совершенной функциональностью. В ASP для вывода данных из БД необходимо было обращаться не посредственно к объекту recordset с указанием имён полей , в ASP.NET записи невозможно вывести без специальных элементов управления. Рассмотрим рисунок 2.4.

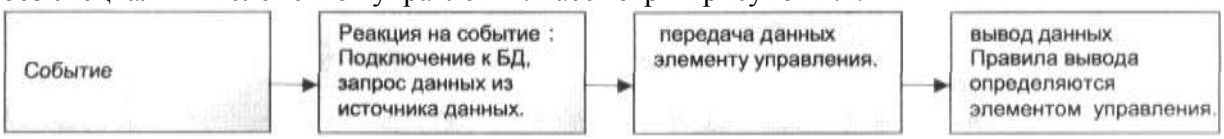


Рисунок 2.4. Процесс вывода данных из БД в ASP.NET

Как видно из рисунка, запрос к данным происходит только после некоторого события (например, загрузка страницы Page_load). Далее, данные передаются элементу управления (к примеру, DataGrid) который описывает, как именно нужно производить их вывод. После этого данные переводятся в набор HTML тегов. Обязательное наличие элемента управления для вывода данных является принципиальным отличием ASP и ASP.NET. в аспекте взаимодействия с БД.

2.2.5 Управление и настройка Интернет-приложений в ASP.NET

Мощная исполняющая среда ASP.NET содержит гораздо большее количество готовых служб, чем традиционная ASP. Любое отдельное Интернет-приложение, в смысле настройки и сопровождения, имеет множество параметров. Механизмы настройки среды важны не меньше самих алгоритмов, лежащих в ее основе.

Существовавшие до сих пор исполняющие среды использовали для хранения конфигурационной информации централизованные хранилища. Так, классическая ASP использует для настроек сервисы MS. В ASP.NET применяется децентрализованный подход. Каждое приложение хранит управляющую конфигурационную информацию ASP.NET в файле web.config в собственном каталоге приложения. Каждый файл web.config содержит конфигурационную информацию в формате XML-файла.

Рассмотрим фрагмент кода:

<?xml version="1.O" encoding="utf-8" ?>
<configuration>
<system.web>

<!- включить компиляцию ->
compilation defaultlanguage="c#" debug="true" />
</system.web>
<configuration>
Данный фрагмент описывает использование по умолчанию языка C# и включения параметра компиляции для целевого каталога.
Файлы web.config в ASP.NET могут располагаться в разных подкаталогах приложения. Каждый файл web.config управляет операциями со страницами в данном каталоге и нижних подкаталогах, если там он не переопределяется Записи, сделанные в web.config нижнего уровня, переопределяют записи на предыдущем уровне. Главный файл, определяющий умолчания для всей системы ASP.NET, называется machine.config и лежит в каталоге в который установлен .NET Framework. Записи, сделанные в файлах web.config в корневых каталогах отдельных Интернет-приложений переопределяют записи, сделанные в главном файле. Записи файлов web config в подкаталогах приложения переопределяют записи в корневом каталоге. Все разделы файла web.config настраиваемы на всех уровнях подкаталогов. На данный момент, эту схему конфигурирования использовать достаточно трудно. Простой XML-файл можно редактировать любым редактором или с помощью Visual Studio.NET, где для этой цели есть специальная утилита. Но ей далеко до других административных утилит Windows 2000, таких как COM+ Explorer. В оригинальной версии ASP все настройки Интернет-приложения производятся посредством административных утилит web-сервера IIS , при помощи этих утилит можно конфигурировать "приложение" в целом . Это проще , но при этом возможностей для различных настроек гораздо меньше, чем в ASP.NET.

2.2.6. Сессии в ASP.NET.

По умолчанию запросы файлов с сервера не зависят друг от друга. Если я запрашиваю с сервера файл (страницу) X.aspx, содержимому этого сценария неизвестно, просматривал ли я перед этим файл Y.aspx. Такой механизм хорошо работает для простых систем с запросами только на чтение. "Приложение" , реализующий такой механизм, легко написать, так как не нужно хранить данные от запроса к запросу. Однако по мере усложнения взаимодействия с Интернет, такая конструкция перестает удовлетворять потребностям пользователей. Мои действия на странице X.aspx должны оказывать воздействие на содержимое страницы Y.aspx. То есть, возникает потребность отделять данные одного пользователя от данных других пользователей. Решением этой проблемы является управлением сеансом.
Старая ASP обеспечивает простой механизм управления сеансами, который поддержан и расширен в ASP.NET. При каждом обращении к .ASPX-странице ASP.NET создает внутренний объект Session (сеанс). Это набор данных, который существует на сервере и привязан к конкретному активному пользователю. Разработчик может хранить в объекте Session любые интересующие его данные, например, идентификатор пользователя. Для этого необходимо создать, так называемую, серверную переменную (серверные переменные одинаково назначаются как в ASP, так и в ASP.NET) и присвоить ей некоторое значение. При помощи значения этой переменной исполняющая среда ASP.NET знает, что пользователем являюсь именно я, а не кто -то другой. Объект Session может хранить произвольное число строк для каждого пользователя, но поскольку все они используют серверную память, имеет смысл хранить ограниченный минимально необходимый объем данных, связанных с сеансом. Это легко реализуемые и потому популярные функции в обеих средах исполнения.
Рассмотрим фрагмент:
language="C#" runat=server %>session("customerfullname")=ФИО пользователя;
response.write[session("customerfullname")] ;
Данный фрагмент иллюстрирует использование переменной session("customerfullname") . Переменной присваивается значение фамилия имя отчество пользователя, далее осуществляется вывод значения этой переменной на клиентский браузер, для каждого клиента индивидуальное. Альтернативой сессионных переменных являются так называемые Cookie , оптимальным вариантом является их совместное использование. Если сервер поддерживал бы постоянный непрерывный сеанс для каждого пользователя, ресурсы памяти быстро бы истощились , так как каждая сессионная переменная требует порядка 10кб оперативной памяти сервера. Механизм сеансов разработан для управления состоянием только активных пользователей, т. е. тех, кому требуется, чтобы сервер управлял их состоянием. ASP.NET автоматически удаляет объект Session пользователя, сбрасывая его содержимое, если пользователь не проявляет активности в течение некоторого времени. Этот интервал устанавливается в разделе <sessionstate> файла web.config . Можно также самим сбросить сеанс, вызвав метод Session Abandon. В простой ASP механизм сеансов имеет некоторые недостатки, препятствующие его распространению в крупномасштабных системах. Во-первых, информация о сеансе сохраняется в рабочем процессе, который выполняет сценарии

и обращается к коду пользовательских объектов. Объекты с неадекватным поведением могут (и зачастую это происходит) разрушать этот процесс, уничтожая сеансы всех обслуживаемых в данный момент пользователей, а не только тех, кто вызвал разрушение. В ASP.NET эта проблема решена благодаря управлению сеансом в отдельном процессе, работающем как системная служба , так что неадекватный пользовательский код уничтожить его не может. Это значит, что рабочие процессы не нарушают состояния сеанса. При этом доступ к сеансовым значениям замедляется, так как приложениям для получения сеансовых значений надо преодолевать границы процесса, но надежность того стоит. Для включения этого режима атрибут mode в файле web.config устанавливается в значение stateserver.
В старой ASP сеанс всегда хранился на серверной машине, на которой он был создан. Такая архитектура плохо масштабируется на многосерверные комплексы, в которых запросы потенциально могут обрабатываться разными серверами. ASP.NET позволяет каждому приложению определить машину, на которой будет храниться сеанс. Это делается установкой атрибута stateConnection-String в файле web config.

2.2.7. Безопасность в ASP.NET.

Безопасность жизненно важна при программировании распределенной системы любого вида. Рассмотрим возникающие в этой области проблемы и то, как эти вопросы решены в ASP и ASP.NET.

Первая проблема безопасности — аутентификация. Понятие аутентификации можно определить как некоторый процесс, в результате которого системы защиты убеждаются в том что перед ней тот пользователь, за которого он себя выдаёт.
Сложность и важность аутентификации заставили встроить эту функцию практически во все современные операционные системы . ASP и ASP.NET поддерживает три механизма аутентификации:

- None: Аутентификация не используется.
- Windows: Стандартная аутентификация Windows через IIS.
- Forms: ASP.NET требует, наличие специальных cookie-файлов на стороне клиента, сформированных сервером.

ASP.NET, кроме трёх вышеприведенных механизмов аутентификации, поддерживает инициативу Microsoft Passport. Данный механизм похож на Forms, за исключением того, что идентификационные данные пользователя сохраняются и cookie-файлы формируются внешней аутентификационной службой Microsoft Passport. Рассмотрим более подробно приведенные выше механизмы аутентификации.
ASP.NET и ASP поддерживает аутентификацию на базе Windows, что по сути означает делегирование процесса аутентификации IIS — базовой Web-серверной архитектуре, над которой надстроена как старая ASP так и её новая версия. IIS можно настроить так, чтобы он выводил диалоговое окно в пользовательском браузере и принимал идентификатор и пароль пользователя. Эти идентификационные данные должны соответствовать пользовательской учетной записи Windows в домене, к которому принадлежит IIS. Другая возможность предполагает наличие Microsoft Internet Explorer 4 или более поздней версии Windows-системы и отсутствие прокси-сервера. Windows-аутентификация хорошо работает в интрасетях, где используется единая платформа -Windows, но не удобна для разработки систем, предназначенных для внешних пользователей.
Большинство проектировщиков Интернет-приложений предпочитают использовать аутентификацию на основе форм, или cookie-аутентификацию.
Рассмотрим рисунок 2.5.

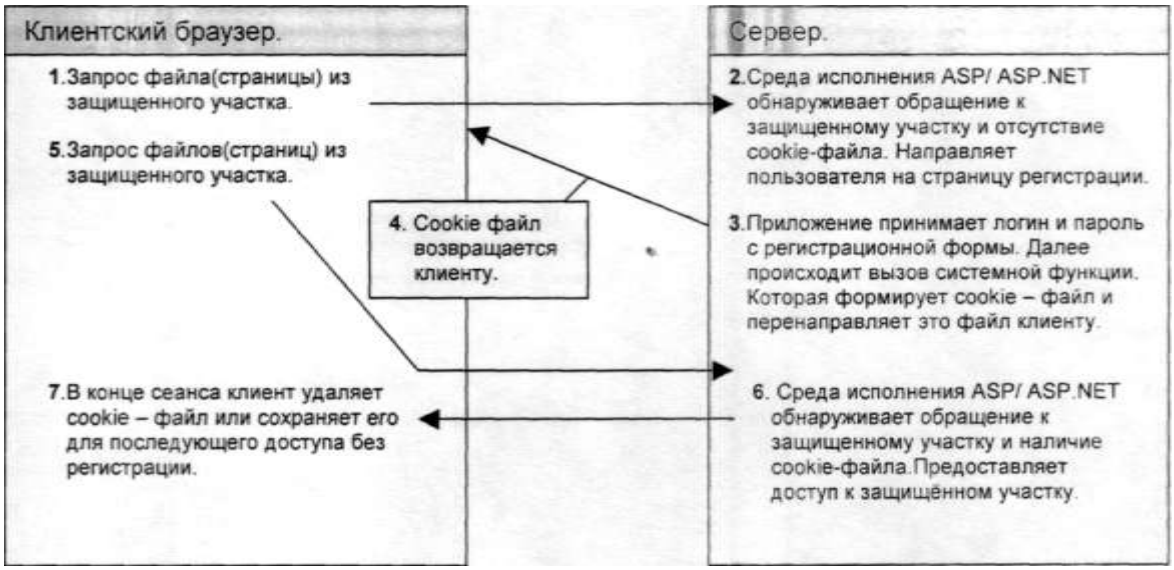


Рисунок 2.5. Аутентификация на основе форм.

Из рисунка 2.5 видно, что когда пользователь первый раз пытается обратиться к защищённому участку сервера (шаг 1),сервер производит запрос на наличие cookie-файла на стороне клиента (шаг 2.). Не обнаружив искомого он предлагает пользователю
пройти регистрацию (шаг 3). Пользователь регистрируется, после чего сервер формирует уникальный cookie-файл и отсылает его браузеру (шаг 4.). Этот уникальный файл содержит идентификационные данные пользователя в зашифрованном виде. Браузер будет автоматически посылать этот cookie-файл в разделе заголовка всех запрашиваемых после этого файлов (страниц) (шаг 5), чтобы сервер имел возможность идентифицировать пользователя (шаг 6). Если cookie-файл был создан автоматически при объявлении сессионной переменной , то по завершению сеанса он будет удалён. В случае явного создания, для этих целей в ASP и ASP.NET существуют специальные методы, он будет сохранён на жёстком диске клиента.
В отличии от ASP , где параметры аутентификации настраиваются посредством IIS, в ASP.NET для использования аутентификацию на основе форм нужно сделать соответствующие записи в файле web.config.
Passport-аутентификация -- новый механизм аутентификации доступный только в ASP.NET. Microsoft Passport — попытка обеспечить универсальную безопасную одношаговую процедуру регистрации. При этом применяется механизм аналогичный аутентификации на основе форм, но хранением и проверкой идентификаторов и паролей занимаются Web-серверы Microsoft Passport. Пользователь должен зайти на сайт службы Passport и создать паспорт — по сути идентификатор и пароль, хранящиеся на серверах Microsoft. Когда пользователь запрашивает файл(страницу) с сервера , применяющего Passport-аутентификацию, этот сервер ищет в запросе cookie-файл службы Passport. Если он его не находит, запрос перенаправляется на собственный сервер Microsoft Passport, где пользователь вводит свой идентификатор и пароль и получает cookie-файл. Последующие запросы браузера к любым серверам, применяющим Microsoft Passport,
содержат этот cookie-файл, пока пользователь не выйдет из системы .
Рассмотрим рисунок 2.6.

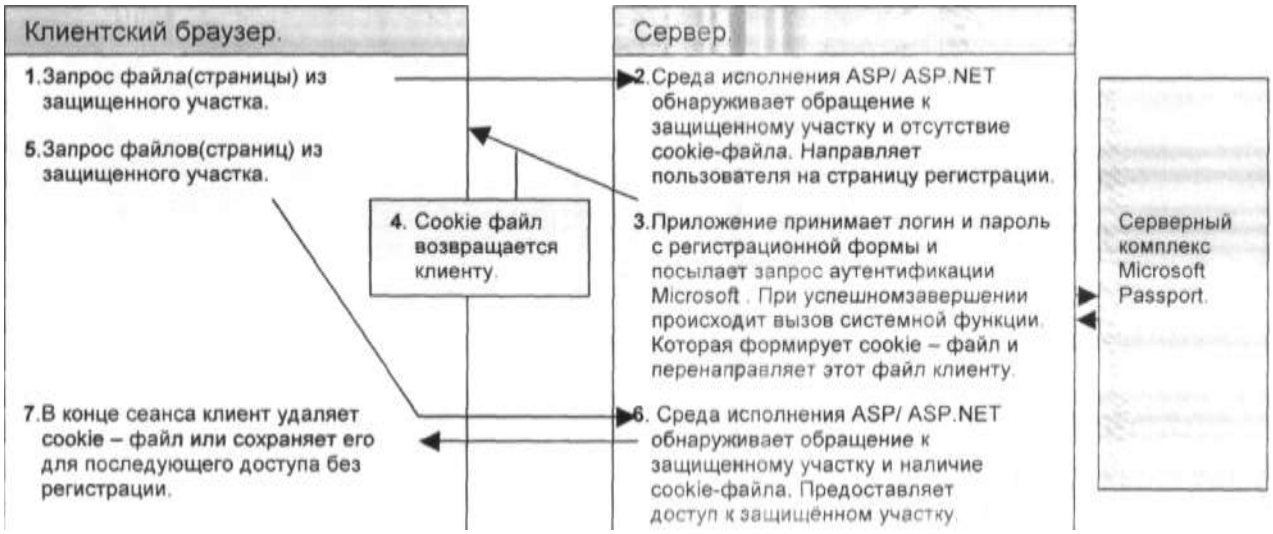


Рисунок 2.6. Аутентификация на основе форм с использованием Microsoft Pasport.

Как видно из рисунка, схема аутентификации с использованием Microsoft Passport аналогична схеме метода с использованием форм и отличается только наличием запроса аутентификации Microsoft.
После завершения процесса аутентификации становится понятным какого рода пользователь пытается получить доступ, и необходимо определить, имеет ли данный пользователь права доступа к файлу, который он запрашивает . Это процесс называется авторизацией.
В ASP авторизация реализуется при помощи настроек IIS. ASP.NET имеет гораздо более гибкую поддержку контроля доступа к сценариям и использует для этого опять же web.config.
.
Общее и различия двух сред исполнения ASP и ASP.NET представлены в таблице 2.3

Таблица 2.3. Общее и различие двух сред исполнения ASP и ASP.NET

.

Элемент сравнения.	ASP	ASP.NET
Элементы управления.	Нет.	Есть.
Смешение кода сценариев с HTML.	Есть	Код сценариев не смешивается с HTML и прописывается отдельно от HTML или подключается отдельным файлом (фоновым кодом).
Событийная модель.	Нет.	Есть.
Поддержка сессий.	Есть.	Значительным образом усовершенствована .
Наличие C#, языка программирования серверных сценариев.	Нет.	Есть.
Автоматическое запоминание состояния элементов управления после пересылки значений на сервер.	Разработчик вынужден программно устанавливать значение элементов в состояние до отсылки значений на сервер.	Есть.
Взаимодействие с БД.	Используется ADO. Значения записей выводятся посредством итерации цикла смеси HTML и отдельных полей таблицы БД.	Используется ADO. NET. Вывод записей производится при помощи специальных элементов управления.

Управление и настройка.	Осуществляется посредством IIS.	Определяется в файле конфигурации. Существует возможность переопределять настройки на каждом уровне приложения.
Безопасность.	Настраивается при помощи IIS, используются аутентификации на основе форм и cookie.	Регулируется в файлах конфигурации. Существует возможность переопределения на каждом уровне приложения. Используется аутентификация на основе форм , cookie и с использованием Microsoft Passport.