УЧЕБНО-НАУЧНО-ПРОИЗВОДСТВЕННЫЙ КОМПЛЕКС
«МЕЖДУНАРОДНЫЙ УНИВЕРСИТЕТ КЫРГЫЗСТАНА»

«УТВЕРЖДЕНО»
Ректор НОУ УНПК «МУК»
к.т.н., доцент Савченков Е.Ю.
_____
«10» 10 2018 г.

БАКАЛАВРИАТ

Кафедра «Компьютерные информационные системы и управление»

Учебно-методический комплекс дисциплины

WEB-ориентированные информационные системы

Направление: **710100 «Информатика и вычислительная техника»**

Профиль: **Компьютерные информационные системы в бизнесе**

Академическая степень **- бакалавр**

Форма обучения **(очная)**

График проведения модулей 7-семестр

Нед.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Лек.	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Лаб.	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

«РАССМОТРЕНО»
Протокол заседания кафедры
«КИСиУ»
№ 2 от 16.10 2018
Зав. кафедрой д.т.н., проф. Миркин Е.Л.
_____

«СОГЛАСОВАНО»
Проректор по академ. вопросам
проф. Мадалиев М.М.
_____

Составитель _____ к.т.н., и.о., доцента
Нежинских С.С.

Директор Научной библиотеки _____ Асанова Ж.Ш.

БИШКЕК 2018

ОГЛАВЛЕНИЕ

АННОТАЦИЯ 3
УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ (МОДУЛЕЙ) 4
1. Пояснительная записка 4
1.1. Миссия и стратегия 4
1.2. Цель и задачи дисциплины (модулей) 4
1.3. Формируемые компетенции, а также перечень планируемых результатов обучения по дисциплине (модулю) 4
1.4. Место дисциплины (модулей) в структуре ООП ВПО 5
2. Структура и содержание дисциплины (модулей) 5
3. Конспект лекций 6
4. Информационные и образовательные технологии 6
5. Фонд оценочных средств для текущего, рубежного и итогового контролей по итогам освоению дисциплины (модулей) 7
5.1. Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины 7
5.2. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности 7
5.3. Описание показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания 9
6. Учебно-методическое и информационное обеспечение дисциплины 10
6.1. Список источников и литературы 10
6.2. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимый для освоения дисциплины (модулей) 10
7. Материально-техническое обеспечение дисциплины 10
8. Приложения 11

АННОТАЦИЯ

Цель дисциплины (модулей) заключается в подготовке выпускника, владеющего теоретическими и практическими основами современных технологий разработки web-приложений и информационных систем.

На изучение дисциплины отводится 120 часов в седьмом семестре. Рубежный контроль успеваемости проводится на 4, 8, 11, 14 неделях. Формы текущего контроля: опрос, проверка заданий, посещаемость. Форма рубежного контроля — модульная работа. Форма итогового контроля — экзамен.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ (МОДУЛЕЙ)

1. Пояснительная записка

1.1. Миссия и стратегия

Миссия НОУ УНПК "МУК" – подготовка международно - признанных, свободно мыслящих специалистов, открытых для перемен и способных трансформировать знания в ценности на благо развития общества.
Видение НОУ УНПК «МУК»- создание динамичного и креативного университета с инновационными научно-образовательными программами и с современной инфраструктурой, способствующие достижению академических и профессиональных целей.
Стратегии развития - модернизация образовательной деятельности университета – совершенствование образовательного процесса в соответствии с требованиями Болонского процесса.

1.2. Цель и задачи дисциплины (модулей)

Целью дисциплины является подготовка выпускника, владеющего теоретическими и практическими основами современных технологий разработки web-приложений и информационных систем.

Задачи дисциплины:

- получение практических навыков по реализации и функционированию технологии “клиент-сервер”;
- изучение современных веб-технологий и языков веб-разработки;
- формирование навыка работы с учебно-методической и научной литературой по проблематике дисциплины.

1.3. Формируемые компетенции, а также перечень планируемых результатов обучения по дисциплине (модулю)

Дисциплина (модуль) направлена на формирование следующих компетенций:

- инструментальными (ИК):
  - владеть основными методами, способами и средствами получения, хранения и переработки информации, навыками работы с компьютером, как средством управления информацией, в том числе в глобальных компьютерных сетях и корпоративных информационных системах (ИК-5);
- профессиональными (ПК):
  - разрабатывать интерфейсы «человек - электронно-вычислительная машина» (ПК-3);
  - способен разрабатывать модели компонентов информационных систем, включая модели баз данных (ПК-4).

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1. Знать:
   - основные проблемы и способы их решения, связанные с разработкой web-приложений;
   - современные инструменты и технологии разработки веб-приложений для автоматизации бизнес-процессов.
2. Уметь:

- анализировать требования к web-системам;
- проектировать архитектуру веб-приложений для обеспечения требований;
- разрабатывать веб-приложений для автоматизации бизнес-процессов.

3. Владеть:
   - современными подходами и методами разработки веб-приложений.

1.4. Место дисциплины (модулей) в структуре ООП ВПО

Дисциплина (модуль) является частью общенаучного цикла (блока) дисциплин учебного плана по направлению подготовки 710100 «Информатика и вычислительная техника». Для освоения дисциплины (модулей) необходимы компетенции, сформированные в ходе изучения следующих дисциплин и прохождения практик: программирование, базы данных.

2. Структура и содержание дисциплины (модулей)

Структура дисциплины (модулей) для очной формы обучения
Общая трудоемкость дисциплины составляет 4 кредита, 120 ч., в том числе аудиторная работа обучающихся с преподавателем 60 ч., самостоятельная работа обучающихся 60 ч.

№ п/п	*Раздел, Темы Дисциплины*	*Семестр*	*Неделя семестра*	*Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)*				*Формы текущего контроля успеваемости (по неделям семестра) Форма промежуточной аттестации (по семестрам)*
				Лекции	*Сем /лаб*	*СРС*	*СРСП*	
Раздел 1.								
1	Введение в Веб-программирование.	7	1	1	3	3	1	опрос, проверка задания, посещаемость
2	Серверные технологии веб-программирования.	7	2	1	3	3	1	опрос, проверка задания, посещаемость
3	Язык PHP.	7	3	1	3	3	1	опрос, проверка задания, посещаемость
4	Конструкции языка программирования PHP	7	4	1	3	3	1	Модуль 1
Раздел 2.								
5	Базы данных.	7	5	1	3	3	1	опрос, проверка задания, посещаемость

6	Соединение web-приложения и базы данных	7	6	1	3	3	1	опрос, проверка задания, посещаемость
7	Преимущества и недостатки современных баз данных	7	7	1	3	3	1	опрос, проверка задания, посещаемость
8	Эффективное хранение данных	7	8	1	3	3	1	Модуль 2
Раздел 3.								
9	Разработка приложений, основанных на БД.	7	9	1	3	3	1	опрос, проверка задания, посещаемость
10	Клиентские технологии веб-программирования: HTML, Javascript, CSS.	7	10	1	3	3	1	опрос, проверка задания, посещаемость
11	Современная модель веб-приложения.	7	11	1	3	3	1	Модуль 3
Раздел 4.								
12	Системы управления контентом — CMS.	7	12	1	3	4	1	опрос, проверка задания, посещаемость
13	Веб-сервисы. SEO.	7	13	1	3	4	1	опрос, проверка задания, посещаемость
14	Оптимизация веб-страниц.	7	14	1	3	4	1	Модуль 4
15	Консультация	7	15	1	3	0	1	посещаемость

3. Конспект лекций

Конспект лекций можно просмотреть в приложении 1.

4. Информационные и образовательные технологии

№ п/п	Наименование раздела	Виды учебной работы	Формируемые компетенции (указывается код компетенции)	Информационные и образовательные технологии
1	Разделы: 1, 2, 3, 4	Лекция	ИК-5, ПК-3, ПК-4	Лекция-визуализация с применением слайд-проектора,

				Дискуссия, Лекция с разбором конкретных ситуаций
		Лабораторная работа	ИК-5, ПК-3, ПК-4	Дискуссия, Консультирование с разбором абстрактных ситуаций
		Самостоятельная работа	ИК-5, ПК-3, ПК-4	Использование электронного курса лекций, Консультирование и проверка заданий посредством электронной почты

5. Фонд оценочных средств для текущего, рубежного и итогового контролей по итогам освоению дисциплины (модулей)

5.1. Перечень компетенций с указанием этапов их формирования в процессе освоения дисциплины

№ п/п	**Контролируемые разделы дисциплины (модулей)**	**Код контролируемой компетенции (компетенций)**	**Наименование оценочного средства**
1	Разделы №1, №2, №3, №4	ИК-5	опрос, выполнение лабораторных работ
2	Разделы №1, №2, №3, №4	ПК-3	опрос, выполнение лабораторных работ
3	Разделы №1, №2, №3, №4	ПК-4	опрос, выполнение лабораторных работ

5.2. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности

Форма контроля	Срок отчетности	Макс. количество баллов	
		За одну работу	Всего
Текущий контроль: - опрос	1, 2, 3, 4 недели	8 баллов	До 32 баллов
- выполнение лабораторных работ	1, 2, 3, 4 недели	10 баллов	До 40 баллов

- посещаемость	1, 2, 3, 4 недели	2 балла	8 баллов
Рубежный контроль: (сдача модуля)	4 неделя	100%×0,2=20 баллов	
Итого за I модуль			До 100 баллов

Форма контроля	Срок отчетности	Макс. количество баллов	
		За одну работу	Всего
Текущий контроль: - опрос	5, 6, 7, 8 недели	8 баллов	До 32 баллов
- выполнение лабораторных работ	5, 6, 7, 8 недели	10 баллов	До 40 баллов
- посещаемость	5, 6, 7, 8 недели	2 балла	8 баллов
Рубежный контроль: (сдача модуля)	8 неделя	100%×0,2=20 баллов	
Итого за II модуль			До 100 баллов

Форма контроля	Срок отчетности	Макс. количество баллов	
		За одну работу	Всего
Текущий контроль: - опрос	9, 10, 11 недели	7 баллов	До 21 баллов
- выполнение лабораторных работ	9, 10, 11 недели	15 баллов	До 45 баллов
- посещаемость	9, 10, 11 недели	4 балла	12 баллов
Рубежный контроль: (сдача модуля)	11 неделя	100%×0,2=20 баллов	
Итого за III модуль			До 100 баллов

Форма контроля	Срок отчетности	Макс. количество баллов	
		За одну работу	Всего
Текущий контроль: - опрос	12, 13, 14 недели	7 баллов	До 21 баллов
- выполнение лабораторных работ	12, 13, 14 недели	15 баллов	До 45 баллов
- посещаемость	12, 13, 14 недели	4 балла	12 баллов
Рубежный контроль: (сдача модуля)	14 неделя	100%×0,2=20 баллов	

Итого за IV модуль			До 100 баллов
Итоговый контроль (экзамен)	Сессия	ИК = Бср × 0,8 + Бэкз × 0,2	

Полученный совокупный результат (максимум 100 баллов) конвертируется в традиционную шкалу:

Рейтинговая оценка (баллов)	**Оценка экзамена**
от 0 до 54	неудовлетворительно
от 55 до 69	удовлетворительно
от 70 до 84	хорошо
от 85 до 100	отлично

5.3. Описание показателей и критериев оценивания компетенций на различных этапах их формирования, описание шкал оценивания

Текущий контроль (0-80 баллов)

При оценивании посещаемости, опроса и выполнении лабораторных работ учитываются:
- посещаемость (10 баллов)
- степень раскрытия содержания материала (25 баллов);
- изложение материала (грамотность речи, точность использования терминологии и символики, логическая последовательность изложения материала (20 баллов);
- знание теории изученных вопросов, сформированность и устойчивость используемых при ответе умений и навыков (25 баллов).

Рубежный контроль (0 – 20 баллов)

При оценивании контрольной работы учитывается:
- полнота выполненной работы (задание выполнено не полностью и/или допущены две и более ошибки или три и более неточности) – 10 баллов;
- обоснованность содержания и выводов работы (задание выполнено полностью, но обоснование содержания и выводов недостаточны, но рассуждения верны) – 5 баллов;
- работа выполнена полностью, в рассуждениях и обосновании нет пробелов или ошибок, возможна одна неточность – 5 баллов.

Итоговый контроль (экзаменационная сессия) – ИК = Бср × 0,8 + Бэкз × 0,2

При проведении итогового контроля обучающийся должен ответить на 3 вопроса (два вопроса теоретического характера и один вопрос практического характера).

При оценивании ответа на вопрос теоретического характера учитывается:
- теоретическое содержание не освоено, знание материала носит фрагментарный характер, наличие грубых ошибок в ответе (0 баллов);
- теоретическое содержание освоено частично, допущено не более двух-трех недочетов (10 баллов);
- теоретическое содержание освоено почти полностью, допущено не более одного-двух

недочетов, но обучающийся смог бы их исправить самостоятельно (20 баллов);
- теоретическое содержание освоено полностью, ответ построен по собственному плану (30 баллов).

При оценивании ответа на вопрос практического характера учитывается:
- ответ содержит менее 20% правильного решения (0-9 баллов);
- ответ содержит 21-89 % правильного решения (10-39 баллов);
- ответ содержит 90% и более правильного решения (40 баллов).

6. Учебно-методическое и информационное обеспечение дисциплины

6.1. Список источников и литературы

Литература:
- Основная:
  1. Кузнецов М.В., Симдянов И.В., Голышев С.В. PHP 5. Практика разработки Web-сайтов. – СПб: БХВ-Петербург, 2012
  2. Кузнецов, М.В., Симдянов, И.В. PHP. Практика создания Web-сайтов., 2-ое издание – СПб: БХВ-Петербург, 2011
  3. Основы программирования на PHP: курс лекций: учеб. Пособие для студентов вузов, обучающихся по специальностям в области информ. технологий/ Н.В. Савельева. – М.: Интернет – Ун-т информ. технологий, 2011
- Дополнительная:
  1. Дунаев В. Самоучитель JavaScript – СПб.: Питер, 2005
  2. Хольцнер С. PHP в примерах. Пер. с англ. – М.: ООО «Бином-Пресс», 2007

6.2. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимый для освоения дисциплины (модулей)

- //www.intuit.ru
- //habrahabr.ru/blogs/programming/
- //phpclub.ru/
- //www.webscript.ru/
- http://www.iprbookshop.ru/
- http://kyrlibnet.kg/ru/
- http://biblioteka.kg/

7. Материально-техническое обеспечение дисциплины

Для изучения дисциплины, необходимо следующее оборудование: ЭВМ, проектор.

Требования к аудитории: компьютерный класс, имеющий ЭВМ в количестве идентичном количеству обучающихся, ЭВМ для преподавателя с подключенным проектором, наличие доски и средств для отображения/удаления информации на доске (мел/ветошь, маркер/губка).

8. Приложения

Приложение 1

Веб-приложения – это специальный вид приложений, которые работают в глобальной сети Интернет по протоколу HTTP (см. "Введение в платформу .NET Framework и ASP.NET"). Как правило, веб-приложения не требуют установки дополнительного программного обеспечения на стороне клиента, а вся логика, в основном, выполняется на стороне сервера. Для отображения пользовательского интерфейса используется браузер – программа, способная распозновать язык разметки HTML (и сопутствующие технологии – таблицы стилей CSS, клиентский скриптовой язык программирования JavaScript и т.д.). Браузер обычно принято называть "тонким клиентом", т.е. клиентом, который содержит минимальное количество бизнес-логики.

Давайте разберемся в том, как функционирует любое веб-приложение. Необходимые компоненты для работы пользователя с веб-приложением – браузер (тонкий клиент), веб-сервер (серверная часть), протокол взаимодействия клиента и сервера (HTTP) и язык разметки для создания документов (HTML). Основу функционирования веб-сервера и протокола HTTP мы подробнее рассмотрим в следующих лекциях, а пока остановимся на концептуальном представлении. Для того, чтобы веб-приложение стало доступно, его необходимо разместить в рамках веб-сервера (специальная программа, которая обрабатывает запросы из сети). После этого приложение получит свой уникальный адрес в рамках протокола HTTP (например, http://www.myapplication.com/page1.html). Используя этот адрес пользователь может обратиться к приложению. Для этого он должен запустить браузер (клиентское приложение) и ввести в адресной строке адрес приложения. В этот момент браузер сгенерирует запрос к серверу и отправит его, используя протокол HTTP. В момент, когда сервер примет этот запрос, он сможет распознать, что именно требуется от него на основе полученного запроса. Используя эти данные он сгенерирует ответ и отправит его обратно клиенту также используя протокол HTTP. Обычно ответ содержит гипертекстовую разметку HTML, содержащую структуру документа, который передается пользователю. После того, как браузер получит ответ в виде HTML-документа, он немедленно отобразит его пользователю. Таким образом, было совершено взаимодействие клиента и сервера. Зачастую документ HTML содержит ссылки на изображение, другие медиа-файлы, таблицы стилей или клиентские сценарии. В этом случае браузер генерирует еще несколько аналогичных запросов к веб-серверу. Однако, в этом случае веб-сервер передает клиенту уже не HTML-документ, а соответствующие запрашиваемые ресурсы (изображения, таблицы стилей, клиентские сценарии и т.д.).

В самом простейшем случае на сервере может быть сохранен готовый документ HTML, который по запросу будет передаваться пользователю. Это – так называемый, статический документ. В этом случае он просто считывается с жесткого диска и передается клиенту. Однако, такой сценарий работы в глобальной сети становится все более редким. Другой подход – генерация кода HTML в процессе обработки запроса от клиента. Этот подход позволяет сделать веб-приложение более интерактивным, отзывчивым на действия клиента и создающее впечатления настоящего приложения, а не простой загрузки HTML-документов. Таким образом, мы, имея возможность генерировать HTML-код страницы, можем влиять на ту информацию, элементы управления и другие аспекты интерфейса, которые увидит пользователь. По сути, задача веб-приложения и заключается в том, чтобы генерировать нужный HTML-код, в зависимости от действий пользователя. Однако, возможности веб-приложений могут не ограничиваться только генерацией разметки HTML – они могут генерировать изображения, клиентские сценарии, таблицы стилей и другие ресурсы, которые могут быть загружены пользователем. Тем не менее, основным сценарием является все-таки

генерация конечного документа HTML.

Как уже говорилось ранее, для взаимодействия клиента и сервера используется протокол HTTP (который более детально будет рассмотрен в последующих лекциях). Сейчас важно понять, что этот протокол работает по схеме "запрос-ответ". В момент, когда клиент хочет обратиться к серверу, он генерирует запрос, который отправляется серверу. Сервер обрабатывает этот запрос и подготавливает ресурсы, которые будут отправлены клиенту. После этого сервер генерирует ответ, в котором содержатся все необходимые данные и отправляет клиенту. Работа веб-приложений заключается в формировании необходимых данных как раз в момент подготовки ресурсов на сервере. Обычно в этот момент запускается некоторый программный код, который содержит определенную бизнес логику. Схему работы типичного веб-приложения схематически можно представить следующим образом (зеленым цветом обозначены действия, которые выполняются на клиентской стороне, а синие – на серверной).

Веб-приложения существенно отличаются от настольных приложений. Последние запускаются на компьютере клиента и выполняют свой код именно там. Поэтому настольные приложения зачастую обладают более богатым и отзывчивым пользовательским интерфейсом и позволяют реализовывать более богатые сценарии. По сравнению с настольными приложениями, веб-приложения обладают более ограниченными возможностями по формированию пользовательского интерфейса и клиентской функциональности. По этой причине за последнее время сложился стереотип о том, что серьезные приложения (например, бизнес-приложения) – это, как правило, настольные приложения. Однако, развитие веб-технологий доказало, что веб-приложения также могут реализовывать богатые сценарии и успешно конкурировать с настольными приложениями. Кроме того, за последние несколько лет очень активно развиваются технологии, позволяющие сделать веб-приложения еще более интерактивными. К ним относятся технология AJAX (будет рассмотрена в рамках курса), которая на основе клиентских сценариев JavaScript может сделать взаимодействие более интерактивным. Также существует ряд технологий, которые добавляют интерактивности приложению за счет внедрения в браузер специальных модулей (плагинов), которые могут отображать специальные типы файлов с более богатыми возможностями. К таким технологиям в первую очередь относятся технологии Silverlight и Flash.

Однако, несмотря на то, что существует ряд технологий, упрощающих создание динамичных веб-приложений, их разработка по прежднему остается довольно трудоемкой задачей. Разработка веб-приложений существенно отличается от разработки настольных систем. Этому есть две главные причины:

Веб-приложения исполняются на сервере. Весь программный код исполняется в рамках веб-сервера, а клиенту доставляется уже готовая разметка HTML, которая отображается внутри браузера.
Веб-приложения не хранят состояния. По-сути, сервер "забывает" про пользователя после того, как обработал его запрос.

Оба этих фактора существенно влияют на процесс разработки веб-приложений. Из-за этого при построении любого веб-приложения приходится решать типичные задачи – способы хранения информации о пользователе, организация сеансов работы пользователя, способы переходов от страницы к странице, механизмы оптимизации эффективности (например, кэширование) и др. При реализации каждого веб-приложения разработчику придется

столкнуться с этими проблемами и решить их. Поскольку набор этих задач является достаточно стандартным и одинаково решается для большинства веб-приложений, то его реализация вынесена в отдельные технологии, которые называются технологиями для разработки веб-приложений. К таким технологиям относятся технология Microsoft ASP.NET, PHP, Ruby on Rails и др. В них, фактически, содержатся все компоненты, необходимые для реализации веб-приложений и учитывающие их специфику. Далее в рамках данного курса мы будем рассматривать разработку веб-приложений с позиции платформы Microsoft ASP.NET.

Однако, несмотря на те преимущества настольных приложений, которые мы рассмотрели ранее, веб-приложения также обладают своими преимуществами. Основное преимущество веб-приложений заключается в процессе развертывания приложения, т.е. установке приложения конечному клиенту. Если настольное приложение необходимо установить на каждое рабочее место где оно будет использоваться, то веб-приложение нужно разместить на сервере и дать ссылку на него всем пользователям. Особенно актуален данный аспект там, где присутствует большое количество рабочих мест. Кроме того, в случае обновления программного кода, веб-приложения также имеют преимущество – для их обновления требуется только обновить код на сервере. При этом настольное приложение потребовалось бы обновлять на каждом рабочем месте.
Краткие итоги

Веб-приложения работают на сервере и исполняются в рамках веб-сервера (специальной программы, обрабатывающей запросы). Взаимодействие клиента и сервера осуществляется про протоколу HTTP в рамках схемы "запрос-ответ". Вся логика веб-приложения размещается на сервере. Из-за этого появляются дополнительные проблемы при разработке веб-приложений. Настольные приложения имеют более богатый пользовательский интерфейс, но веб-приложения легче разворачивать и обновлять (особенно, если имеется большое количество рабочих мест). Пользовательский интерфейс веб-приложений может стать более интерактивным, если использовать дополнительные инструменты – клиентские сценарии JavaScript, а также приложения Silverlight, Flash и др.

Протокол HTTP/HTTPS

Цель лекции: сформировать представление о функционировании протокола HTTP/HTTPS.

HTTP (HyperText Transfer Protocol) – один из наиболее важных протоколов, который обеспечивает передачу данных через интернет. Протокол HTTP находится на седьмом, прикладном уровне модели OSI и работает на основе протокола TCP.

Поскольку протокол HTTP находится на прикладном уровне, прикладные приложения могут использовать непосредственно его для организации сетевого взаимодействия. Кроме того, протокол HTTP является важнейшей частью веб-приложений. В этом случае браузер, используя возможности HTTP, взаимодействует с сервером для получения необходимых данных.

Протокол HTTP предполагает передачу данных в режиме "запрос-ответ". При этом в рамках такого взаимодействия могут передаваться данные практически любого типа – обычный текст, гипертекст (HTML), таблицы стилей, клиентские сценарии, изображения, документы в различных форматах, бинарная информация и т.д.

В рамках протокола HTTP всегда четко выделяется клиент и сервер. Клиент всегда является инициатором взаимодействия. Сервер, в свою очередь, прослушивает все входящие соединения и обрабатывает каждое из них. Поскольку HTTP-взаимодействие функционирует по схеме "запрос-ответ", то для инициации сеанса передачи данных необходимо сгенерировать HTTP-запрос. В рамках этого запроса клиент описывает то, какой ресурс он хочет получить от сервера, а также указывает различные дополнительные параметры. После этого запрос отправляется серверу и тот, в свою очередь, обрабатывает запрос и генерирует HTTP-ответ, в котором содержится служебная информация и содержимое того ресурса, который был запрошен. В целом схематически процесс можно изобразить следующим образом.

HTTP-запрос и HTTP-ответ сходны по своей структуре и называются HTTP-сообщениями. Фактически, все взаимодействие в рамках протокола HTTP сводится к пересылке HTTP-сообщений. Каждое HTTP-сообщение является обычной текстовой информацией, представленной в определенном формате. Давайте поподробнее рассмотрим формат HTTP-сообщения.

Каждое HTTP-сообщение состоит из нескольких строк. Первой строкой всегда идет приветственная строка, она существенно различается для HTTP-запроса и HTTP-ответа. Обычно в ней содержится общая информация о запросе. После первой строки в HTTP-сообщении присутствуют HTTP-заголовки – каждый заголовок с новой строки. HTTP-заголовки присутствуют как в HTTP-запросе, так и в HTTP-ответе. Смысл HTTP-заголовков заключается в уточнении HTTP-сообщения для того, чтобы принимающая это HTTP-сообщение сторона могла наиболее точно обработать входящее сообщение. Количество заголовков HTTP-сообщения является переменным и зависит от конкретного HTTP-сообщения. Если отправляющая сторона считает, что этот HTTP-заголовок необходим в этом HTTP-сообщении, то она добавляет его, если нет – то не добавляет. Каждый HTTP-заголовок начинается с новой строки. HTTP-заголовок состоит из имени и значения, имя заголовка определяет его предназначение. После набора HTTP-заголовков следует пустая строка, после которой идет тело HTTP-сообщения. Таким образом, общую структуру HTTP-сообщения можно представить следующим образом.

HTTP-запрос формируется на клиенте и отправляется на сервер с целью получения информации от него. В нем содержится информация о ресурсе, который необходимо загрузить, а также дополнительные сведения. Первая строка содержит метод запроса (который мы рассмотрим далее в этой лекции), имя ресурса (с указанием относительного пути на сервере), а также версию протокола. Например, вид приветственной строки может быть определен как "GET /images/corner1.png HTTP/1.1". Такой запрос обращается к серверу с требованием выдать методом GET изображение, расположенное в папке "images" и имеющее название "corner1.png". HTTP-заголовки имеют важное значение для HTTP-запроса, поскольку в них указывается уточняющая информация о запросе – версия браузера, возможности клиента принимать сжатое содержимое, возможности кэширования и другие важные параметры, которые могут влиять на формирование ответа. В теле HTTP-запроса обычно содержится информация, которую необходимо передать на сервер. Например, если требуется загрузить файл на сервер, то содержимое файла будет находится в теле HTTP-запроса. Однако, размещение данных в теле HTTP-запроса допускается не для всех HTTP-методов. Например, тело HTTP-запроса всегда пустое, если используется метод GET. Таким образом, стандартный HTTP-запрос может выглядеть следующим образом.

В приведенном HTTP-запросе клиент обращается к серверу "microsoft.com", запрашивает

ресурс "images/corner.png" и указывает, что он способен принимать сжатое содержимое по алгоритму "gzip" или "deflate", его языком является английский язык и указывает версию своего браузера. Как было отмечено ранее, количество и набор заголовков может существенно отличаться. Можно привести другой пример HTTP-запроса.

Этот запрос отличается от предыдущего тем, что в нем используется метод POST, который также загружает данные на сервер. При этом сами данные содержаться в теле HTTP-запроса после пустой строки.

HTTP-ответ генерируется веб-сервером в ответ на поступивший HTTP-запрос. По своей структуре он схож с HTTP-запросом, но имеет определенные отличия. Главное отличие содержится в первой строке. Вместо имени запрашиваемого ресурса и метода запроса в ней указывается статус ответа. Статус указывает на то, насколько успешно выполнился HTTP-запрос. Например, в случае, если документ найден на сервере и может быть выдан клиенту, то статус имеет значение "OK", которое говорит о том, что запрос выполнился успешно. Однако, могут появляться исключительные ситуации – например, документ отсутствует на сервере или у пользователя отсутствуют права на получение ресурса. Набор всевозможных статусных сообщений HTTP-ответа мы рассмотрим далее в этой лекции. Таким образом, первая строка HTTP-ответа может принимать значение "HTTP/1.1 200 OK". HTTP-заголовки в HTTP-ответе также являются важным элементом. Они характеризуют содержимое, которое передается клиенту. Например, в этих HTTP-заголовках может содержаться информация о типе содержимого (HTML-документ, изображение и т.д.), длине содержимого (размер в байтах), дате модификации, режиме кэширования и др. Все эти заголовки влияют на способ отображения данных на клиенте, а также устанавливают правила хранения данных в клиентском кэше. Типичный вид HTTP-ответа может быть следующим.

В приведенном примере сервер указывает, что ресурс найден, его тип – HTML-документ, а также указывает размер и дату модификации. После пустой строки идет содержимое HTML-документа, т.е. по сути то, что запрашивал клиент. Как и в случае с HTTP-запросом, в HTTP-ответе количество заголовков может изменяться на усмотрение веб-сервера.

При рассмотрении структуры HTTP-запроса было затронуто понятие метода HTTP-запроса. Метод HTTP-запроса определяет каким образом будет обрабатываться указанный HTTP-запрос, т.е. в каком-то смысле определяет его семантику. Поскольку HTTP-запросы могут иметь самый разнообразный смысл, то указание метода является важной частью построения HTTP-запроса. HTTP-запросы могут иметь следующие значения: запрос ресурса от сервера, создание или изменение ресурса на сервере, удаление ресурса на сервере и т.д.

Наиболее распространенными методами HTTP-запроса являются следующие типы методов:
GET позволяет получить информацию от сервера, тело запроса всегда остается пустым;
HEAD аналогичен GET, но тело ответа остается всегда пустым, позволяет проверить доступность запрашиваемого ресурса и прочитать HTTP-заголовки ответа;
POST позволяет загрузить информацию на сервер, по смыслу изменяет ресурс на сервере, но зачастую используется и для создания ресурса на сервере, тело запроса содержит изменяемый/создаваемый ресурс;
PUT аналогичен POST, но по смыслу занимается созданием ресурса, а не его изменением, тело запроса содержит создаваемый ресурс;
DELETE удаляет ресурс с сервера.

Кроме указанных методов HTTP, существует еще большое количество других методов,

определенных в спецификации протокола HTTP. Однако, несмотря на это, браузерами зачастую используются только методы GET и POST. Тем не менее, другие прикладные приложения могут использовать HTTP-методы по своему усмотрению.

Как мы увидели ранее, в составе HTTP-ответа содержится статусный код или код возврата. Этот статус показывает состояние HTTP-ответа, которое получено от сервера. Этот механизм является необходимым при функционировании протокола HTTP, поскольку при обработке запроса могут встречаться различные нестандартные ситуации. Все статусные коды являются трехзначными числами. Кроме того, в составе HTTP-ответа может присутствовать текстовое описание состояния. Все статусные коды делятся на пять групп.

Каждая группа статусных кодов идентифицирует ситуацию, в которой оказался запрос. Группа определяется первым разрядом статусного кода. Например, статусные коды группы 2xx говорят об успехе выполнения HTTP-запроса. Наиболее используемые статусные коды приведены в таблице ниже.

Код	Описание
1xx	Информационные коды
2xx	Успешное выполнение запроса
200	Запрос был обработан успешно
201	Объект создан
202	Информация принята
203	Информация, которая не заслуживает доверия
204	Нет содержимого
205	Сбросить содержимое
206	Частичное содержимое (например, при "докачке" файлов)
3xx	Перенаправление (чтобы выполнить запрос, нужны какие-либо действия)
300	Несколько вариантов на выбор
301	Ресурс перемещен на постоянной основе
302	Ресурс перемещен временно
303	Смотрите другой ресурс
304	Содержимое не изменилось
305	Используйте прокси-сервер
4xx	Проблема связана не с сервером, а с запросом
400	Некорректный запрос
401	Нет разрешения на просмотр ресурса
402	Требуется оплата
403	Доступ запрещен
404	Ресурс не найден
405	Недопустимый метод
406	Неприемлемый запрос
407	Необходима регистрация на прокси-сервере
408	Время обработки запроса истекло
409	Конфликт
410	Ресурса больше нет
411	Необходимо указать длину
412	Не выполнено предварительное условие
413	Запрашиваемый элемент слишком велик
414	Идентификатор ресурса (URI) слишком длинный
415	Неподдерживаемый тип ресурса
5xx	Ошибки на сервере

500 Внутренняя ошибка сервера
501 Функция не реализована
502 Дефект шлюза
503 Служба недоступна
504 Время прохождения через шлюз истекло
505 Неподдерживаемая версия HTTP

Эти и другие статусные коды используются для передачи информации о статусе запроса от клиента к серверу.

Отличительной особенностью протокола HTTP является то, что в рамках этого протокола информация передается в виде текста. Это означает, что работать с таким протоколом достаточно просто. Кроме того, инженеры по безопасности даже при строгом режиме безопасности оставляют открытым именно протокол HTTP. Поэтому реализация сетевого взаимодействия в рамках протокола HTTP является одним из перспективных направлений.

Однако, несмотря на простоту протокола, существует проблема утечки передаваемой информации. Поскольку информация передается в виде обычного текста, то перехват такой информации осуществляется достаточно просто. В некоторых ситуациях эта проблема не является критичной. Однако, для веб-приложений, работающих с конфиденциальной информацией это достаточно существенный недостаток.

По этой причине существует модификация этого протокола – HTTPS, т.е. протокол HTTP с поддержкой шифрования.

Как известно, существуют классические криптостойкие алгоритмы шифрования, которые шифруют данные на основе существующего ключа. Для шифрования и расшифровки данных используется один и тот же ключ – если кто-либо знает ключ к зашифрованной информации, то он может расшифровать ее. Ключ – это обычная последовательность бит определенной длины. Чем больше длина ключа, тем сложнее взломать алгоритм шифрования. Таким образом, для того, чтобы защитить свою информацию, необходимо хранить в секрете ключ шифрования. Однако, каким образом это обеспечить в рамках взаимодействия по протоколу HTTP? Ведь если передавать этот ключ в открытом виде, то смысл шифрования пропадает. В этом случае используют дополнительно другой вид шифрования – ассиметричный. В этом случае существует пара ключей – открытый и закрытый. С помощью открытого ключа можно только зашифровать информацию, а с помощью закрытого – расшифровать. Обычно при таком подходе закрытый ключ хранится в секрете, а открытый ключ является общедоступным. Однако, ассиметричный алгоритм работает медленнее, чем симметричный, поэтому его используют для первоначального обмена симметричными ключами. Давайте рассмотрим весь алгоритм работы зашифрованного соединения по HTTP.

При обращении клиента к серверу по защищенному каналу сервер хранит открытый и закрытый ключ. В начальный момент времени сервер передает клиенту открытый ключ ассиметричного шифрования. Клиент случайным образом генерирует ключ симметричного шифрования и шифрует его с помощью открытого ключа, полученного от сервера. После этого клиент отправляет зашифрованный ключ на сервер и в этот момент времени клиент и сервер имеют одинаковые ключи для симметричного шифрования. Далее идет HTTP-взаимодействие, которое шифруется с помощью этого симметричного ключа. Симметричный ключ остается в секрете и не может быть перехвачен, поскольку закрытый ключ (которым можно расшифровать первое сообщение, содержащее симметричный ключ)

остается в секрете на сервере. Таким образом, обеспечивается конфиденциальность и целостность передаваемых данных по протоколу HTTP

Краткие итоги

Все веб-приложения работают на основе протокола HTTP. Протокол HTTP передает текстовую информацию и работает в режиме "запрос-ответ". HTTP-запрос и HTTP-ответ имеют строго определенную структуру – привественная строка, заголовки и тело сообщения. Количество HTTP-заголовков переменное. HTTP-заголовки от тела сообщения отделяет пустая строка. Каждый HTTP-запрос отправляется на сервер в рамках HTTP-метода. HTTP-метод определяет семантику запроса (получить ресурс, добавить, изменить, удалить и т.д.). В HTTP-ответе кроме служебной информации и полезных данных, отправляется статус запроса, который информирует клиента об успешности выполнения запроса. Все статусные коды делятся на группы. Поскольку данные, передаваемые по протоколу HTTP можно перехватить, то он не обеспечивает конфиденциальности передаваемой информации. Если подобный уровень безопасности необходим, то нужно использовать протокол HTTPS, который обеспечивает шифрование передаваемой информации на основе комбинирования симметричного и ассиметричного алгоритмов шифрования.

В предыдущей лекции мы разобрались с функционированием протокола HTTP. Теперь давайте рассмотрим, как работают инструменты, которые делают возможным описанные ранее взаимодействия. В основе функционирования веб-приложений лежит такое понятие как веб-сервер. Веб-сервер – это программа, которая принимает входящие HTTP-запросы, обрабатывает эти запросы, генерирует HTTP-ответ и отправляет его клиенту. Общий алгоритм работы веб-сервера можно представить следующим образом (зеленым цветом помечены действия, которые обрабатываются веб-сервером).

После того, как пользователь обратился к определенному ресурсу по протоколу HTTP, клиент (обычно браузер) формирует HTTP-запрос к веб-серверу. Обычно указывается символическое имя сервера (например, "http://www.microsoft.com") – в этом случае браузер предварительно преобразует это имя в IP-адрес при помощи сервисов DNS. После этого по протоколу HTTP на веб-сервер отправляется сформированное HTTP-сообщение. В этом сообщении браузер указывает какой ресурс необходимо загрузить и всю дополнительную информацию. Задача веб-сервера – прослушивать определенный TCP-порт (обычно порт 80) и принимать все входящие HTTP-сообщения. Если входящие данные не соответствуют формату сообщения HTTP, то такой запрос игнорируется, а клиенту возвращается сообщение об ошибке.

В простейшем случае при поступлении HTTP-запроса веб-сервер должен считать содержимое запрашиваемого файла с жесткого диска, упаковать его содержимое в HTTP-ответ и отправить клиенту. В случае если требуемый файл не найден на жестком диске, то веб-сервер сгенерирует ошибку с указанием статусного кода 404 и отправит это сообщение клиенту. Такой вариант работы веб-сервера принято называть статическими сайтами. В этом случае на стороне сервера не запускается никакой программный код, кроме программного кода самого веб-сервера. Однако подобные сценарии работы все чаще оказываются непригодными, а им на смену приходят полноценные веб-приложения. Отличие таких приложений состоит в том, что HTML-документы и другие ресурсы не хранятся на сервере в виде неизменяемых данных. Вместо этого, на сервере хранится программный код, который способен сгенерировать эти данные в момент обработки запроса. Разумеется, некоторые ресурсы (такие как файлы каскадных стилей, изображения и т.д.) могут храниться как статическое содержимое, но основные страницы HTML генерируют в процессе обработки. В

таком случае веб-сервер при обработке запроса HTTP должен обращаться к программному коду, который должен сгенерировать содержимое. С учетом вышесказанного алгоритм работы веб-сервера будет выглядеть следующим образом.

Одной из наиболее важных задач, которые решаются при построении веб-сервера является задача обеспечения масштабируемости (т.е. возможности увеличения количества обслуживаемых пользователей) и защищенности от внешних атак. Поскольку веб-сервер работает в открытой среде – глобальной сети Интернет – то зачастую доступ к нему может осуществляться откуда угодно. Это делает веб-сервер подверженным большим нагрузкам и потенциальным атакам. Наиболее распространенными атаками на веб-сервер является обращение к веб-серверу с большим количеством запросов и их высокой частотой. В этом случае веб-сервер не сможет быстро обрабатывать все запросы, а это может сказаться на производительности веб-сервера для настоящих пользователей. Особенно остро подобным атакам подвержены веб-сервера, на которых исполняется какой-то внешний программный код за исключением программного кода самого веб-сервера. Обычно для борьбы с подобными атаками блокируются все запросы, которые приходят с определенного IP-адреса. Кроме того, в подобных случаях следует позаботится об оптимизации программного кода приложения, например, использовать кэширование – в этом случае при обработке каждого запроса нагрузка на центральный процессор будет меньше, что может существенно усложнить задачу атакующим.

Нередко на одном и том же веб-сервере располагается множество независимых веб-сайтов. Более того, все эти веб-сайты используют один и тот же IP-адрес. Т.е. веб-сервер, имеющий только один IP-адрес может размещать внутри себя несколько веб-сайтов и при этом каждый такой веб-сайт будет ассоциирован с собственным адресом (например, на одном веб-сервере могут располагаться веб-сайты: "microsoft.com", "gotdotnet.ru", "techdays.ru" и т.д.). Каким образом это становится возможным? Такое явление называется виртуальным хостингом. Для того чтобы понять как это работает, давайте еще раз обратимся к процессу взаимодействия клиента и сервера. Браузер отправляет HTTP-запрос на IP-адрес веб-сервера, который ассоциирован с доменным именем. Разрешение IP-адреса происходит с помощью служб DNS. Однако, несмотря на то, что запрос отправляется, используя полученный IP-адрес, клиент указывает дополнительный HTTP-заголовок "Host", в котором определяется оригинальное имя веб-сайта. Благодаря этой информации веб-сервер может разграничить доступ к нескольким веб-сайтам и при этом использовать один и тот же IP-адрес. Это очень важный момент, поскольку если бы для каждого доменного имени приходилось бы регистрировать отдельный IP-адрес, то адресное пространство протокола IP (v.4) очень быстро бы закончилось, а стоимость размещения веб-сайта в глобальной сети Интернет была бы намного выше. Для того, чтобы было более понятно давайте рассмотрим работу виртуального хостинга на примере. Предположим, имеется веб-сервер с IP-адресом 85.51.210.22. На этом сервере размещено несколько веб-сайтов: mysite1.com, mysite2.com, mysite3.com. Сервера DNS настроены таким образом, что каждое из этих доменных имен указывает на единственный IP-адрес 85.51.219.22. Давайте посмотрим, какие HTTP-запросы браузер будет генерировать при обращении к каждому из сайтов. При обращении к сайту "mysite1.com" HTTP-запрос может выглядеть следующим образом.

При обращении к сайту "mysite2.com" HTTP-запрос будет выглядеть иначе.

При анализе HTTP-запросов хорошо видно, что HTTP-заголовок "Host" отличается в каждом из запросов. Таким образом, становится понятно, что веб-сервер анализирует этот заголовок и отправляет клиенту содержимое соответствующего сайта. Схематически этот процесс

можно представить следующим образом.

Подобную схему виртуального хостинга использует большинство компаний, занимающихся размещением веб-сайтов в Интернет. Поскольку в этом случае на одном физическом сервере могут размещаться большое количество совершенно различных сайтов, то этот способ один из самых дешевых. Однако, в рамках виртуального хостинга обычно запрещено запускать различные службы и сервисы, а также существует ограничение по степени использования центрального процессора. Это означает, что в случае, когда веб-сайт потребляет слишком много серверных ресурсов, то владельцу сайта предлагается либо перейти на более дорогой тариф (с большим количеством выделенных ресурсов), либо при превышении допустимого порогового значения веб-сайт блокируется на некоторое время. Поскольку иногда от сервера требуется большое количество ресурсов или в рамках этого сервера необходимо запускать дополнительные приложения или службы, виртуальный хостинг можно использовать не всегда. В этом случае обычно арендуют выделенный сервер – физический или виртуальный. Однако, это более дорогой вид размещения веб-приложений в сети Интернет, поэтому зачастую используется именно виртуальный хостинг.

Как уже говорилось ранее, самый простой сценарий работы веб-сервера заключается в получении HTTP-запроса, его обработки, считывания нужного файла с жесткого диска, формирование HTTP-ответа и отправки его клиенту. Подобный сценарий является самым простым, однако, в реальности встречается все реже. Дело в том, что при подобном подходе, содержимое, которое передается клиенту, является статическим (т.е. не изменяется от запроса к запросу). Однако если требуется построить веб-приложение, то содержимое HTML-страницы, которое передается клиенту должно изменяться от различных внешних условий (параметров запроса, содержимого базы данных, времени обработки запроса, типа пользователя и т.д.). В этом случае требуется запускать внешний (по отношению к веб-серверу) программный код, реализующий логику веб-приложения. Этот код должен содержаться отдельно от программного кода самого веб-сервера, поскольку код приложения будет различным от одного приложения к другому, а веб-сервер будет один и тот же. Таким образом, программный код, обрабатывающий HTTP-запросы и генерирующий HTTP-ответы можно условно разделить на две части:

- программный код, реализующий служебные функции по взаимодействию через протокол HTTP (программный код самого веб-сервера);
- программный код, реализующий логику конкретного веб-приложения (бизнес-логика, обращение к СУБД и т.д.).

Поскольку программный код веб-приложения обычно упаковывается в отдельные модули и поставляется независимо, то требуются механизмы взаимодействия этих двух частей, т.е. интерфейс взаимодействия. В данном случае под интерфейсом взаимодействия понимается набор правил, по которым веб-сервер и приложение будут взаимодействовать друг с другом. Фактически, схема обработки запроса может выглядеть следующим образом.

Исторически сложилось так, что существует два главных типов интерфейс взаимодействия внешнего приложения и веб-сервера - CGI и ISAPI.

CGI (Common Gateway Interface) – наиболее ранний способ взаимодействия веб-сервера и веб-приложения. Основная идея, которая лежит в основе CGI заключается в том, что при поступлении очередного HTTP-запроса, веб-сервер инициирует создание нового процесса и передает ему все необходимые данные HTTP-запроса. После того, как этот процесс

отработает, он завершается, передав при этом результат обратно веб-серверу. Поскольку веб-сервер и приложение – это разные процессы с точки зрения операционной системы, то для обмена информации между ними используются средства межпроцессного взаимодействия (IPC) – зачастую это переменные окружения, именованные каналы и т.д. Основным преимуществом CGI является то, что процесс веб-сервера и приложения изолированы друг от друга и в случае неполадок в веб-приложении, завершится с ошибкой именно процесс приложения, при этом процесс самого веб-сервера будет продолжать функционировать.

С другой стороны, необходимость создания каждый раз нового процесса влечет за собой дополнительные накладные расходы на создание процесса (создание процесса – дорогостоящая операция с точки зрения операционной системы) и передачи данных через границы процессов. Этот факт является серьезным недостатком и оказывает существенное влияние на масштабируемость веб-приложения (возможность обрабатывать большее количество поступающих запросов).

ISAPI (Internet Server API) – альтернативный способ взаимодействия веб-сервера и веб-приложения. В отличии от CGI, при взаимодействии в рамках интерфейса ISAPI, при поступлении очередного запроса, веб-сервер инициирует создание нового потока в рамках основного процесса, в котором работает веб-сервер. Поскольку с точки зрения операционной системы создание потока – это менее дорогостоящая операция, чем создание процесса, то такие приложения на практике оказываются более масштабируемыми. Кроме того, упрощается взаимодействие веб-сервера и веб-приложения, поскольку в этом случае используется единое адресное пространство в рамках операционной системы (поскольку весь код работает в одном и том же процессе). Однако, в случае серьезных неполадок в веб-приложении, которое взаимодействует с веб-сервером в рамках ISAPI, веб-сервер также потенциально подвергается риску быть завершенным. Поскольку веб-сервер и веб-приложение работают в одном и том же процессе, это действительно так. Поэтому разработчикам программного кода веб-сервера, поддерживающего ISAPI следует уделить этому вопросу особое внимание.

На сегодняшний день наиболее распространенным способом взаимодействия веб-сервера и веб-приложения является интерфейс ISAPI, поскольку обеспечивает наиболее оптимальные показатели по накладным расходам и масштабируемости. Однако, при работе нескольких веб-приложений на одном и том же веб-сервере, в этом случае существует потенциальная опасность влияния одного приложения на другое. Если говорить о компаниях, размещающих веб-приложения на своих серверах, то может случиться такая ситуация, что на одном и том же веб-сервере одновременно размещаются веб-сайты компаний-конкурентов. В этом случае теоретически одна из компаний может намеренно загрузить код, который будет завершать работу веб-сервера с ошибкой и, таким образом, все веб-сайты размещенные на этом веб-сервере окажутся недоступными. Для того, чтобы избежать подобной ситуации используется совмещенный подход – для каждого приложения может создаваться пул приложения (application pool), который представляет из себя отдельный процесс, в котором функционируют потоки для обработки входящих HTTP-запросов от пользователей. В этом случае, если какое-то из приложений будет содержать код, который завершает работу процесса с ошибкой, то будет завершаться процесс только этого приложения. Более того, каждый пул приложения содержит набор заранее созданных и подготовленных потоков. Это необходимо для того, чтобы не тратить время на создание потока в момент поступления входящего запроса. Такой набор заранее созданных потоков называется пулом потоков. Как правило, веб-сервер следит за каждым пулом приложения и если оно завершает свою работу с ошибкой, то веб-сервер перезапускает его процесс.

Кроме приведенных функций и механизмов веб-сервера, в его функции зачастую входят и сопутствующие дополнительные задачи. К этим задачам относится аутентфикация и авторизация пользователя, ведение серверного лога (для отладки работы веб-сервера), поддержка нескольких веб-сайтов на одном сервере (виртуальный хостинг), поддержка безопасных подключений по протоколу HTTPS и др. Эти функции в каждом конкретном случае зависят от реализации веб-сервера.

На сегодняшний день существует большое количество различных реализаций веб-серверов. Одним из наиболее популярных и универсальных веб-серверов является веб-сервер с открытым исходным кодом Apache. Он был создан для работе в среде Linux, также существует реализация для работы в рамках Windows. На его основе были построены другие различные вариации, например, Apache Tomcat для запуска веб-приложений на основе Java. Другим, наиболее серьезным продуктом в этой области является веб-сервер Microsoft Internet Information Services (IIS), который работает в рамках операционной системы Windows. Как правило, в рамках этого веб-сервера работают приложения на базе ASP.NET (и родственных технологий), а также приложения PHP и статические веб-сайты. При создании веб-приложений на базе ASP.NET мы будем использовать именно IIS 7. Наконец, существуют другие, менее масштабные проекты по разработке веб-серверов, например Nginx. Этот проект был разработан одним из разработчиков Rambler с целью оптимизации производительности этой поисковой системы. Впоследствии проект оказался настолько удачным, что нашел применение и для работы в других приложений. Обычно Nginx используют когда необходимо построить высоконагруженную инфраструктуру.
Краткие итоги

Веб-сервер – это программа, которая обрабатывает входящие HTTP-запросы и генерирует HTTP-ответы. В простейшем случае веб-сервер передает клиенту содержимое файлов, которые размещены на жестком диске сервера. Когда необходимо генерировать HTTP-ответы на основе какой-то программной логики, подключается внешний программный код. Для подключения внешнего программного кода используются интерфейсы CGI и ISAPI. В настоящий момент наиболее перспективным считается использование интерфейса ISAPI в силу более высокой масштабируемости. В рамках веб-сервера создается пул приложения (для каждого веб-приложения отдельный процесс в рамках ОС, в составе которого работает несколько потоков для обработки запросов). Существует большое количество реализаций веб-серверов, для приложений ASP.NET обычно используется веб-сервер Microsoft Internet Information Services (IIS).

Microsoft .NET Framework – мощная платформа для построения приложений. Первые упоминания о.NET Framework появились в конце 90-х годов, когда компания Microsoft объявила о своей намеренности работать над этой платформой (в то время этот продукт именовался Next Generation Windows Services – NGWS).

увеличить изображение

После выпуска бета-версии NGWS, в 2000 году была выпущена первая бета-версия .NET Framework 1.0. Финальная версия .NET 1.0 вышла только в январе 2002 года. После этого, в апреле 2003 года, появился .NET 1.1, затем в ноябре 2005 года – 2.0. Через год, в ноябре 2006 года, появился ряд библиотек (под общим названием WinFX), расширяющий возможности .NET 2.0, который впоследствии стал называться .NET 3.0. Через год после этого, в ноябре 2007 года, была выпущена версия .NET 3.5, а в августе 2008 года – ряд обновлений, которые

получили название .NET 3.5 SP1. Эта версия .NET Framework остается актуальной и на сегодняшний день, а выход следующей версии .NET 4.0 запланирован на весну 2010 года. Из приведенной выше схемы видно, что версии .NET 3.0, .NET 3.5 и .NET 3.5 SP1 работают в рамки одной среды исполнения (из .NET 2.0) и просто увеличивают функциональность всей платформы за счет добавления новых библиотек. При этом .NET 1.0, .NET 1.1, .NET 2.0 и .NET 4.0 имеют различную среду исполнения.

Что же такое .NET Framework? Одной из основных идей .NET Framework является совместимость программного кода написанного на различных языках программирования. Так, например, в рамках идеологии .NET Framework приложение, разрабатываемое на языке программирования C#, может с легкостью использовать программный код, который разработан на основе Visual Basic .NET. Точно также программный код на Visual Basic .NET может использовать программный код на Delphi .NET и т.д.

Также .NET Framework решает проблему совместимости версий библиотек. Каждая библиотека .NET (сборка, assembly) содержит сведения о своем имени и версии. Из этой информации складывается строгое имя сборки. Таким образом, когда из одной сборки делается ссылка на другую сборку, указывается не только имя этой сборки, но и её версия. Такой подход позволяет устранить возможные конфликты между разными версиями сборок.

При компиляции приложений .NET на выходе получается не исполняемый машинный код, а промежуточный код на специальном языке, который называется MSIL (Microsoft Intermediate Language) или просто IL (Intermediate Language). Своим внешним видом MSIL напоминает язык Ассемблера, но является более высокоуровневым и по своей природе является объектно-ориентированным. Давайте рассмотрим следующий пример: мы создадим консольное приложение и напишем элементарный код для вывода строки на консоль.

После этого, из полученной сборки получим код на языке IL (путем дизассемблирования). Это можно сделать при помощи утилиты Ildasm.exe, которая поставляется в составе .NET Framework. Код на языке IL для приведенной выше программы будет выглядеть следующим образом.

Такой подход, при котором компиляция осуществляется не в машинный код (в виде инструкций центральному процессору), а в код на промежуточном языке программирования решает несколько задач. Во-первых он позволяет организовать процесс разработки одного и того же приложения одновременно на нескольких языках программирования. Это означает, что в одном проекте может содержаться код на различных языках программирования. Поскольку существует промежуточный язык IL, то код, который получается на этом языке программирования, теоретически не зависит от исходного языка программирования. Второй задачей, которую решает этот подход, является возможность создавать приложения независящие от разрядности операционной системы (32 бита или 64 бита). Как известно, системы команд 32- и 64-разрядных операционных систем имеют свои особенности. По этой причине разработчики программного обеспечения вынуждены создавать две версии приложения. Приложения на базе .NET Framework не нуждаются в этом, поскольку промежуточный код компилируется в машинные инструкции в момент первого запуска ("на лету") и может адаптироваться к той системе команд, которая используется в данный момент (см. ниже). Наконец, третья задача, которую решает приведенный подход – это теоретическая возможность запускать программный код, без перекомпиляции на разных платформах. Это означает, что исполняемый файл содержит только код на промежуточном языке, который может быть преобразован в машинный код для той платформы, на которой

он исполняется в данный момент. Таким образом, тот же самый исполняемый файл можно запустить не только в операционной системе Windows, но и, например, в Linux или MAC OS X.Для этого существует отдельный проект, поддерживаемый компанией Novel, который называется Mono.

Идеология .NET Framework способствует внедрению новых языков программирования в общую инфраструктуру, поскольку разработчику этого языка программирования нужно лишь позаботиться о трансформации языковых конструкций в язык IL – все остальное сделает среда исполнения. Благодаря этому в последнее время появляется огромное количество языков программирования для платформы .NET – F#, Boo, IronPython, IronRuby, Haskell, Nemerle, JScript.NET и др.

Кроме этого, .NET Framework содержит огромный набор классов, реализующих различную базовую функциональность. Этот набор классов называется Framework Class Library (FCL) и поставляется совместно с .NET Framework. В рамках BCL реализована базовая функциональность, которая нужна буквально в каждом приложении (работа со строками, числами, массивами, коллекциями, файлами, графикой, сетевыми подключениями, XML, СУБД и т.д.). Вся функциональность BCL доступна в рамках программного кода независимо от того, какой язык программирования используется в данный момент.

Работа приложений .NET обеспечивается за счет наличия такого механизма как общеязыковая среда исполнения (Common Language Runtime, CLR). Общеязыковая среда исполнения позволяет запускать приложения .NET на разных платформах. В момент запуска приложения происходит преобразование кода IL приложения в соответствии с теми машинными инструкциями, где это приложение запускается. Этот процесс называется "компиляцией на лету" (just-in-time). После компиляции кода IL в машинный код, последний сохраняется в системных временных папках для последующего запуска и используется вплоть до того времени, пока исходный код IL не изменится. Более подробные сведения о компиляции приложений .NET содержаться в следующей лекции "Компиляция приложений .NET".

Другой важной особенностью среды исполнения .NET Framework является наличие встроенного механизма сборки мусора (Garbage Collector). Этот механизм берет на себя заботы по удалению из памяти тех объектов, которые использовались ранее и в данный момент не нужны. Традиционно, забота по освобождению памяти от ненужных данных ложилась на плечи разработчиков приложений и в некоторых ситуациях несли в себе серьезные трудности. Для этого в языки программирования были встроены специальные операторы (например, такие как delete в C++). Если разработчик забывал удалить какие-либо ненужные объекты происходила утечка памяти, т.е. складывалась ситуация, когда приложению уже не требуется какой-либо участок памяти в операционной системе, но он остается занятым вплоть до завершения работы приложения. Подобные ошибки разработчика могут быть достаточно легко допущены в процессе разработки приложения, но обнаружить их крайне трудно. Более того, в этой ситуации разработчики вынуждены каждый раз заботится об очистке памяти, вместо того, чтобы думать над логикой приложения.

Работа сборщика мусора строится следующим образом. Как только объект выходит из текущей зоны видимости, он помечается на удаление. Например, если в методе класса объявляется переменная и ей присваивается новый объект, этот объект будет помечен на удаление, когда исполнение этого метода будет завершено. Кроме того, если в процессе работы метода переменной будет присвоено новое значение, то объект будет также помечен

на удаление (если на этот объект нет ссылок в других переменных). Кроме того, если в коде содержатся блоки, то объект будет существовать только в рамках этого блока. При выходе из этого блока объект будет помечен на удаление.

Давайте рассмотрим несколько примеров. В следующем примере создается новый объект Array, который будет удален при выходе из метода CreatePerson.

В этом примере объект Array будет удален в середине выполнения метода, поскольку переменной list присвоено другое значение.

В следующем примере объект Array будет создан и доступен только внутри блока обработки условия. За рамками этого блока объект Array будет недоступен.

Видно, что сборщик мусора позволяет разработчику думать о логике приложения и не заботиться об очистке памяти при создании объектов.

Таким образом, .NET Framework представляет собой комплексную платформу для разработки приложений, которая позволяет разрабатывать на различных языках программирования, может работать в рамках разных платформ, позволяет оптимизировать код под конкретную платформу, имеет богатую библиотеку базовых классов и автоматически управляет сборкой мусора. Поэтому разработка приложений в рамках .NET Framework зачастую происходит более эффективно, а разработчик может думать о логике приложения, а не об особенностях среды исполнения.
Краткие итоги

Платформа Microsoft .NET Framework – это мощная среда для разработки приложений, которая первый раз увидела свет в 2000 году. В рамках этой платформы существует общеязыковая среда исполнения, которая позволяет запускать программный код на промежуточном языке – MSIL. Этот язык необходим для абстрагирования от конкретной платформы и создания единого приложения, не зависящего от конкретных условий пользователя. В составе .NET Framework содержится мощная библиотека базовых классов, которая реализует наиболее типичные задачи. Также для приложений, работающих в составе .NET Framework выполняется автоматическая сборка мусора, что позволяет разработчику не заботиться об удалении устаревших объектов.

Компиляция приложений .NET

Как мы разобрались в предыдущей лекции, процесс компиляции и исполнения приложений .NET отличается от процесса исполнения традиционных (native) приложений. При компиляции приложения .NET на выходе получается бинарный файл, который содержит команды на языке MSIL, а не инструкции для центрального процессора. Этот факт отличает процесс функционирования приложений .NET. Фактически, приложения .NET компилируются в два этапа:

Исходный код (на языке C#, Visual Basic или др.) преобразуется в MSIL;
Код на языке MSIL преобразуется в машинный код.

Теоретически, программный код, разработанный на разных языках программирования в среде .NET (C#, Visual Basic .NET и др.) и преобразованный в код на языке MSIL, в конечном

счете, будет иметь одинаковый вид. Кроме того, после преобразования программного кода в MSIL будет затруднительно определить исходный язык программирования.

Для чего необходима подобная двухэтапная компиляция приложения? Те, кто разрабатывал на языках низкого уровня знают, что разработка программного кода имеет специфику при разработке для каждой платформы. Это означает, что разработка приложения для 32- и 64-разрядной платформы может иметь существенные различия. При этом при игнорировании особенностей разработки под конкретную платформу могут быть как некритические последствия (например, в виде снижения производительности), так и критические (например, неработоспособность приложения). В связи с этим зачастую можно видеть два типа пакетов распространения приложения (32- и 64-разрядных). Более того, для более эффективной работы приложения нередко требуется оптимизация его кода. Все эти задачи имеют непростую природу и могут отнимать у разработчика приложения много времени. Разработчик в этом случае должен тратить свое время не на улучшение логики программы, а заботиться об инфраструктурных задачах. Однако, несмотря на всю сложность этого процесса, существуют алгоритмы, которые способны выполнить большую часть работы по оптимизации кода под конкретную платформу без участия человека. Именно этой идеологии придерживается платформа .NET Framework.

При запуске кода на языке IL происходит его преобразование в машинный код. При этом весь процесс преобразования происходит в той среде, где приложение должно быть запущено (на компьютере конечного пользователя, где исполняется приложение). Этот факт позволяет оптимизировать конечный машинный код, который получается в процессе преобразования, для той платформы, в рамках которой он будет исполняться. Также при таком подходе нет необходимости иметь две различных сборки для разных сред (32- и 64-разрядных) – конечный машинный код будет получен с учетом специфики платформы. Поскольку преобразование "MSIL – машинный код" требует определенного времени, среда исполнения записывает полученный машинный код в системных папках и при последующем запуске приложения, которое запускалось ранее, используется уже готовый код. Общий процесс компиляции выглядит следующим образом.

Механизм, который занимается преобразованием кода MSIL в машинный код, называется компилятором JIT ("just-in-time"). Этот компилятор преобразует код MSIL в машинный код "на лету". Это позволяет сделать процесс запуска приложений более "прозрачным". При запуске исполняемого файла .NET компилятор автоматически определяет, была ли эта сборка скомпилирована.

Поскольку процесс получения машинного кода является трудоемкой задачей, компилятор JIT работает "по запросу". Дело в том, что JIT-компилятор не пытается получить машинный код для всей сборки в момент запуска приложения – машинный код получается только для той части сборки, которая должна быть исполнена в данный момент. Получение машинного кода для остальных частей сборки происходит в момент запуска этих частей на исполнение.

Несмотря на то, что подобный способ позволяет более гибко подойти к разработке и распространению приложения, он обладает недостатком – для запуска приложения требуется несколько больше времени, нежели для запуска традиционного приложения. Это вытекает из того, что приложению требуется время на получение машинного кода. Этот недостаток компенсируется тем, что при запуске приложение компилируется в машинный код не полностью, а только те части, которые исполняются; также после первого запуска приложения оно будет скомпилировано и время на компиляцию больше не потребуется.

В случае, когда необходимо выполнить предкомпиляцию приложения до его запуска (для предотвращения ситуации компиляции приложения "на лету"), можно воспользоваться утилитой NGen (native image generator), которая поставляется в составе .NET Framework. Утилита NGen использует аналогичные методы компиляции приложения, что и компилятор JIT. Однако, от компилятора JIT эту утилиту отличает три аспекта:

преобразование кода MSIL в машинный код производится перед выполнением приложения, а не в момент исполнения;
преобразовываются сразу вся сборка, а не отдельные методы;
полученный машинный код сохраняется в кэше образа машинного кода в виде файла на жестком диске.

Утилита NGen является консольным приложением и запускается из командной строки. Она позволяет сделать несколько различных действий:

скомпилировать сборку и установить ее в кэш сборок (ключ install);
удалить сборку из кэша сборок (ключ uninstall);
обновить сборку в кэше сборок (ключ update);
отобразить состояние процесса компиляции (ключ display);
управлять очередью выполнения компиляции (ключ queue).

Общий процесс компиляции приложений .NET представлен на следующей схеме.

Как видно, процесс компиляции приложений .NET – сложный и многогранный процесс. Однако, этот процесс придает гибкость и универсальность приложениям .NET, а встроенные механизмы оптимизации кода позволяют получить более эффективный исполняемый код.
Краткие итоги

Все приложения .NET поставляются в виде скомпилированного кода MSIL. Весь процесс компиляции приложения .NET состоит из двух этапов – преобразования исходного кода программы в код на языке MSIL и преобразование кода MSIL в машинный код. Такой процесс позволяет абстрагироваться от платформы исполнения, а также оптимизировать машинный код под конкретную платформу. Преобразование кода MSIL в машинный код происходит в момент запуска программы JIT-компилятором, а результат сохраняется в специальном кэше. Поэтому первый запуск .NET приложения может осуществляться с небольшой задержкой. Однако, этого можно избежать, если использовать утилиту Ngen.

Какие технологии входят в состав .NET Framework?

После изучения основ функционирования платформы Microsoft .NET Framework мы рассмотрим основные типы приложений, которые можно разрабатывать на базе .NET Framework. Несмотря на то, что курс посвящен разработке веб-приложений, современному специалисту необходимо быть в курсе того, какие еще виды приложений можно разрабатывать в рамках платформы.

Поскольку .NET Framework – это мощнейшая платформа для разработки приложений, она дает возможность разрабатывать приложения совершенно различного типа. При этом, важным достоинством .NET Framework является то, что функциональность библиотеки базовых классов доступна во всех типах приложений. Это означает, что разработав

библиотеку для .NET Framework ее можно одинаково удобно использовать как в настольных приложениях, так и в веб-приложениях, в веб-сервисах и т.д. (см. далее).

В целом, все приложения можно разделить на несколько типов:

- настольные приложения (приложения, которые запускаются на локальном компьютере пользователя);
- веб-приложения (приложения, которые работают в рамках веб-сервера и доступны пользователю через браузер в рамках протокола HTTP/HTTPS);
- веб-приложения с богатым пользовательским интерфейсом (приложения, которые доставляются пользователю по протоколу HTTP/HTTPS в рамках браузера и исполняются на клиентской стороне);
- веб-сервисы (программные код, который выполняется на стороне сервера и может быть вызван с клиента для получения каких-либо данных или выполнения операции);
- мобильные приложения (приложения, которые выполняются на мобильных устройствах).

Для каждого из приведенных типов приложений в составе .NET Framework существуют соответствующие технологии, которые позволяют создавать приложения. Кроме того, .NET Framework содержит общие библиотеки, которые можно использовать в разных типах приложений. К таким библиотекам можно отнести библиотеки:

- для работы со строками;
- для работы с математическими функциями;
- для работы с графикой;
- доступа к данным;
- для работы с файлами и другими операциями ввода-вывода;
- для выполнения криптографических операций;
- для организации синхронизации данных между источниками данных;
- и огромное количество других библиотек.

Давайте обзорно рассмотрим каждый тип приложений. Как уже было отмечено выше, для каждого из типов приложений в составе платформы .NET Framework существуют специальные шаблоны проектов.

Настольные приложения отличаются тем, что запускаются непосредственно на компьютере пользователя. Это – наиболее распространенный тип приложений. Настольные приложения обычно имеют доступ к ресурсам компьютера пользователя, таким как жесткий диск, звуковое оборудование и т.д. К достоинствам такого типа приложений можно отнести интерактивность пользовательского интерфейса и возможность работы приложения вне зависимости от подключения к сети Интернет и другим ресурсам. Однако, для работы таких приложений их необходимо устанавливать на локальный компьютер.

Настольные приложения можно разделить на три вида – оконные приложения, консольные приложения и службы Windows. Оконные приложения обладают графическим интерфейсом. Консольные приложения обычно имеют вид командной строки, а интерфейс таких приложений является текстовым, а не графическим. Наконец, службы Windows вовсе не имеют пользовательского интерфейса, а работают в фоновом режиме.

Для построения приложений с оконным графическим интерфейсом в рамках платформы .NET Framework могут использоваться технологии Windows Forms (доступно начиная с .NET

Framework 1.0) и Windows Presentation Foundation (доступно начиная с .NET Framework 3.0). Последняя технология является более новой и перспективной.

На рисунке ниже показано окно создания проекта на .NET Framework. Как видно, доступны все описанные выше типы проектов.

увеличить изображение

Веб-приложения отличаются от настольных тем, что работают удаленно на веб-сервере. Пользователь использует возможности веб-приложений посредством браузера и протокола HTTP/HTTPS. Преимуществом этого типа приложений является то, что нет никакой необходимости устанавливать их на компьютер каждого пользователя – приложение нужно установить на веб-сервер, после чего оно становится доступным для всех пользователей. Однако, недостатком таких приложений является ограниченные возможности построения пользовательских интерфейсов. Это происходит из-за того, что пользовательский интерфейс строится на основе форматов HTML, CSS и JavaScript, которые являются достаточно ограниченными. Поэтому функциональность пользовательских интерфейсов обычно является достаточно ограниченной.

Для создания веб-приложений в рамках платформы .NET Framework используется платформа Microsoft ASP.NET. Это – универсальный и мощный механизм, который позволяет строить высокоэффективные и мощные веб-приложения. Далее этот курс будет посвящен изучению платформы Microsoft ASP.NET и сопутствующим ей технологиям.

На рисунке ниже показано окно создания проекта веб-приложения на .NET Framework

увеличить изображение

Поскольку веб-приложения обладают серьезным недостатком – ограниченными возможностями по формированию интерфейса пользователя – появился новый тип приложений, который называется богатые интернет-приложения или интернет-приложения с богатым пользовательским интерфейсом (Rich Internet Applications, RIA). Идеология этих приложений состоит в том, что в браузер интегрируется специальное дополнение (plugin), которое способно отображать дополнительный тип содержимого. После этого, когда пользователь отрывает страницу в браузере, на сторону клиента передается программный код, который работает в рамках этого дополнения. Поскольку в этом случае весь код отрабатывает на стороне клиента, у разработчиков появляется больше возможностей по формированию пользовательских интерфейсов.

Для построения подобного типа приложений существует несколько технологий. Одна из наиболее известных технологий – это технология Adobe Flash. Кроме того, существует также технология Microsoft Silverlight, которая позволяет наиболее удобно интегрироваться с остальными технологиями в рамках .NET Framework.

Зачастую возможности по построению приложений RIA комбинируют с построением обычных веб-приложений.

Другим типом приложений являются сервисы (или веб-сервисы). Сервисы – это некий программный код, который расположен на сервере и запускается по запросу от пользователя. Например, можно создать набор сервисов по работе с данными из базы данных, хранящейся

на сервере. Обычно, другие типы приложений (настольные приложения, веб-приложения и т.д.) обращаются к сервисам с целью выполнить какую-либо операцию на сервере или получить с сервера данные.

В рамках платформы .NET Framework существует ряд технологий, которые позволяют создавать сервисы. Наиболее старой технологией является ASP.NET Web Services. Она позволяет создавать простые веб-сервисы, которые работают по протоколу HTTP/HTTPS. Дальнейшим развитием стала технология Windows Communication Foundation (WCF). Эта платформа является наиболее мощным и гибким инструментом, которая поддерживает различные типы каналов (HTTP, TCP, именованные каналы и др.) и существенно расширяет возможности разработчика по созданию сервисов.

Также существуют дополнительные ответвления проекта WCF – это WCF Syndication Extensions (WCF REST) и ADO.NET Data Services. Эти проекты разработаны для того, чтобы можно было удобно строить сервисы доступа к данным.

Наконец, последним типом приложений являются мобильные приложения. Мобильные приложения работают в рамках мобильных устройств на базе операционной системы Windows Mobile. Для мобильных устройств также существует реализация подмножества возможностей .NET Framework, которая называется .NET Compact Framework.

Таким образом, весь спектр технологий в рамках платформы .NET Framework можно представить следующей схемой.

увеличить изображение

В этой лекции мы рассмотрели возможности платформы .NET Framework для создания различных типов приложений. Видно, что платформа содержит набор инструментов для создания приложений практически для любой ситуации. Возможность создания всевозможных типов приложений и делает .NET Framework очень мощной платформой.
Краткие итоги

Платформа .NET Framework является мощным инструментом для разработчика, поскольку содержит большое обилие технологий для создания приложений. Основные типы приложений, которые можно создавать на базе .NET Framework – настольные приложения, веб-приложения, интернет-приложения с богатым пользовательским интрефейсом, веб-сервисы, мобильные приложения и др. Для каждого типа приложений .NET Framework содержит набор инструментов и технологий.

Модель компиляции ASP.NET

Как нам стало известно ранее, платформа .NET Framework имеет особую, двухэтапную модель компиляции приложений. Эта модель подразумевает получения промежуточного кода на языке MSIL, который впоследствии преобразуется в машинный код.

Поскольку приложения ASP.NET работают в рамках общей среды исполнения платформы .NET Framework, они также работают на основе этого механизма. Несмотря на это, платформа ASP.NET имеет свои особенности компиляции приложений.

Вообще, если рассматривать способы исполнения веб-приложений, то можно выделить два

направления:

- интерпретируемые приложения;
- компилируемые приложения.

Интерпретируемые приложения исполняют исходный текст программы построчно. Интерпретатор в этом случае считывает очередную строку из текста программы и исполняет её, после чего переходит к следующей строке. Общий алгоритм подобного подхода можно описать следующей схемой.

К основным недостаткам подобного подхода относится невысокая скорость исполнения программного кода. Также, при подобном подходе нет возможности проверить весь листинг кода на наличие ошибок. Например, если исходный код программы состоит из 100 строк и в строке 50 содержится ошибка, то первые 49 строк будут успешно исполнены, а на строке 50 будет выдана ошибка. Такое поведение не всегда желательно, поэтому данный способ мало популярен в современной разработке приложений.

Платформа ASP.NET использует альтернативный подход к исполнению приложений, а именно компиляцию исходных файлов в бинарный исполняемый файл перед выполнением. В этом случае перед запуском приложения исходный файл преобразуется в исполняемый файл. Это позволяет отследить часть ошибок на этапе компиляции (например, синтаксические ошибки). Кроме того, в момент исполнения приложения не требуется выполнять проверку и разбор синтаксических конструкций языка, что увеличивает производительность таких приложений относительно интерпретируемых приложений.

Схема исполнения компилируемых приложений выглядит следующим образом.

Как мы увидим в последующих лекциях (когда будем подробно разбирать механизмы работы страниц ASP.NET), каждая страница состоит из разметки (HTML и ASPX кода) и программного кода. Особенностью платформы ASP.NET является то, что все элементы веб-приложения (страницы, элементы управления, обработчики и т.д.) компилируются, а не интерпретируются.

Однако, если с компиляцией программного кода на языке C# (который является частью страницы ASP.NET) все понятно, то каким же образом компилируется разметка страницы? Ведь разметка страницы, по сути, представляет собой обычный текст на языке HTML, который должен обрабатываться бразуером, но для сервера он является обычной последовательностью символов. Давайте разберемся в этом вопросе более подробно.

Если рассмотреть страницу ASP.NET с точки зрения объектной модели, то каждая страница является классом, который в свою очередь является наследником базового класса Page. С другой стороны в структуре проекта страница ASP.NET представляется двумя файлами – файлом разметки (.aspx) и файлом с программным кодом (.cs или .vb).

Если более внимательно рассмотреть файл содержащий программный код для страницы, то в нем можно заметить одну особенность – класс страницы (наследник класса Page) при определении содержит ключевое слово partial.

Ключевое слово partial при определении класса говорит компилятору о том, что описание данного класса может содержатся в нескольких файлах. Действительно, при компиляции

страницы ASP.NET на основе файла разметки создается вторая часть этого класса. При компиляции такого класса он собирается в одно целое и компилируется в язык MSIL.

Схема процесса компиляции страницы ASP.NET в программный код может выглядеть следующим образом.

увеличить изображение

Каким образом разметка преобразуется в программный код? В общем случае разметка трансформируется в вызовы методов "Response.Write()", которые записывают информацию в выходной буфер, который пересылается клиенту. Таким образом, получается полностью скомпилированная страница, которая может запускаться на исполнение.

Для ускорения работы все скомпилированные страницы хранятся в специальной системной папке "C:\Windows\Microsoft.NET\[version]\Temporary ASP.NET Files\" (где [verison] – версия .NET Framework). Эта папка имеет следующую структуру.

Для каждого сайта создается своя папка, в которой присутствуют скомпилированные страницы. В каждой такой папке содержаться временные файлы и информация об исходной странице. Эта информация необходима для отслеживания изменения исходной страницы.

Поскольку существует возможность размещения на веб-сервере проекта в виде исходных файлов, то зачастую компиляция страниц ASP.NET производится в момент первого обращения к этой странице по протоколу HTTP. Из-за этого при первом обращении к страницам веб-приложения могут быть небольшие задержки. Для того, чтобы этого избежать можно выполнить предкомпиляцию приложения.

Предкомпиляция приложения – это процесс предварительной компиляции всех страниц веб-приложения с целью получения исполняемых сборок. Предкомпиляция веб-приложения может выполняться с целью исключения ситуации с задержкой ответа при первом обращении к странице, а также распространение веб-приложения в виде бинарных исполняемых сборок, а не исходных кодов.

Существует два варианта предкомпиляции веб-приложений:
предкомпиляция приложения с возможностью последующего обновления в этом случае исходный код разметки страниц (.aspx) остается в неизменном виде и при изменении этой страницы на сервере она перекомпилируется при первом обращении;
предкомпиляция приложения без возможности последующего обновления в этом случае все файлы компилируются в бинарные сборки и изменение страниц на веб-сервере невозможно.

Предкомпиляцию приложения можно выполнить двумя способами – с помощью Visual Studio и отдельной утилитой aspnet_compiler.

Для предкомпиляции приложения средствами Visual Studio необходимо выбрать пункт меню "Publish Web Site" и в появившемся окне задать адрес публикации и необходимые параметры.

увеличить изображение

Утилита aspnet_compiler поставляется в составе .NET Framework и может быть использована в случае, если Visual Studio в данный момент недоступна (например, требуется выполнить предкомпиляцию приложения прямо на сервере) или требуется автоматизировать процесс предкомпиляции.

Другой важной особенностью компиляции приложений ASP.NET является возможность разграничения области исполнения различных приложений. Поскольку в рамках одного сервера одновременно может исполняться несколько приложений, то разграничение работы приложений является важным моментом. Такое разграничение производится за счет доменов приложений.

Домен приложения – это .NET-аналог процесса в операционной системе. Основной задачей доменов приложения является разграничение зоны действия для различных веб-приложений. Такое разграничение необходимо для обеспечения безопасности. Например, при таком подходе невозможен сценарий, при котором некачественный код одного приложения выведет из строя другое веб-приложение.

Для каждого отдельного каталога на сервере создается отдельный домен приложения. Использование такой модели позволяет разграничить память, выделяемую каждому приложению. Кроме того, в рамках одного домена приложения совместно используются одни и те же ресурсы, находящиеся в памяти – глобальные данные приложения (Application), данные сеансов пользователей (Session), кэшированные данные (Cache) и др. Эта информация не является доступной напрямую другим приложениям, работающим за рамками данного домена приложения. Кроме того, все страницы ASP.NET и другие ресурсы совместно используют одни и те же конфигурационные настройки, которые наследуются из корневого файла web.config.

Каждое веб-приложение исполняется в рамках своего домена приложения. Для того чтобы приложение могло исполняться необходимо каким-то образом создать домен приложения. Для этого используется механизм отложенной инициализации. Этот механизм подразумевает то, что домен приложения не будет создан при старте сервера или в какое-то другое время. Домен приложения создается в момент первого обращения к нему. Такая модель используется для эффективного обновления приложения.

Время жизни приложения может зависеть от различных факторов. Домен приложения может быть закрыт в силу различных причин. Основные причины прекращения работы домена приложения:

- домен приложения закрывается при остановке или перезапуске web-сервера;
- домен приложения закрывается при длительном отсутствии активности в отношении данного приложения;
- домен приложения может закрываться при возникновении необработанной глобальной ошибки, которую не удалось отловить;
- домен приложения закрывается при обновлении ресурсов в рамках данного приложения;
- домен приложения можно закрыть принудительно, вызвав статический метод HttpRuntime.UnloadAppDomain().

Закрытие домена приложения при отсутствии активности обусловлено оптимизацией использования ресурсов операционной системы. При закрытии домена приложения освобождаются ресурсы, занимаемые этим доменом, например оперативная память.

Одной из важных особенностей ASP.NET является модель обновления страницы. Эта модель подразумевает, что для обновления какой-либо страницы не нужно останавливать работу веб-сервера. При изменении, добавлении или удаления какого-либо ресурса инфраструктура ASP.NET автоматически создает новый домен приложения, в который загружаются обновленные ресурсы и в котором обрабатываются все последующие запросы. Старый домен приложения существует до тех пор, пока не будут обработаны все старые запросы. Таким образом, в один момент времени может существовать два домена одного и того же приложения, один из них будет обрабатывать старые запросы, другой – новые.

В момент исполнения программного кода той или иной сборки, среда исполнения CLR блокирует эту сборку на время исполнения кода. По этой причине ASP.NET не размещает исполняемые файлы в папке с приложением, а вместо этого во время компиляции использует теневое копирование. Для этого в момент компиляции все исполняемые файлы помещаются в папку "C:\Windows\Microsoft.NET\[version]\Temporary ASP.NET Files\". Очень важно, что наряду с бинарными исполняемыми файлами в этой папке хранятся XML-описания сущностей ASP.NET. Это позволяет обнаружить изменения в исходных файлах и, в случае необходимости, перекомпилировать их.
Краткие итоги

Компиляция страниц ASP.NET имеет свою специфику по сравнению с компиляцией остальных типов приложений. Все сущности ASP.NET (страницы, элементы управления и т.д.) компилируются в бинарные исполняемые сборки. При этом существует специальная системная папка, в которой расположены все исполняемые сборки. Компиляция страницы ASP.NET в исполняемую сборку происходит в момент первого обращения к этой странице. Из-за этого могут возникать небольшие задержки генерации страницы в момент первого обращения. Для того чтобы этого избежать существует механизм предкомпиляции приложений. Важную роль в компиляции и обновлении приложений ASP.NET играют домены приложения – они позволяют разграничить исполнение различных приложений, которые расположены на одном сервере.

Глобальные события приложения

Процесс обработки запроса любого веб-приложения нетривиален. Перед тем, как будет запущен программный код генерации результата, будет пройдено еще несколько этапов. Среди этих этапов – подготовка к обработке запроса, проверка системы безопасности, проверка наличия страницы в кэше, восстановление состояния и др. После генерации результата и перед отправкой результата клиенту также выполняется набор действий – сохранение страницы в кэше (если это необходимо), сохранение состояния сеанса и др. В некоторых ситуациях в разрабатываемых приложениях требуется отследить момент выполнения того или иного действия. Для этого была создана модель глобальных событий ASP.NET.

увеличить изображение

Модель событий ASP.NET содержит два типа глобальных событий, которые можно обработать: события, которые возникают каждый раз при обработке запроса и события, которые возникают не всегда, а только при определенных условиях.

События, которые возникают при обработке каждого запроса генерируются в следующем

порядке:
BeginRequest генерируется в начале выполнения каждого запроса;
AuthenticateRequest генерируется непосредственно перед моментом как будет выполнена аутентификация. Это стартовая точка для создания собственной системы аутентификации;
AuthorizeRequest генерируется непосредственно перед моментом авторизации;
ResolveRequestCache генерируется в момент проверки локального кэша на наличие текущей страницы в кэше;
AcquireRequestState генерируется в момент, когда для клиента будет получена информация, специфичная для сеанса (Session);
PreRequestHandlerExecute генерируется непосредственно перед тем, как управление будет передано HTTP-обработчику;
PostRequestHandlerExecute генерируется сразу после того, как запрос будет обработан HTTP-обработчиком;
ReleaseRequestState генерируется в момент, когда информация, специфичная для сеанса сериализуется из коллекции Session в строку, чтобы стать доступной для следующего запроса;
UpdateRequestCache генерируется в момент добавления информации в кэш выходных данных;
EndRequest генерируется в конце запроса перед тем, как объекты будут уничтожены.

Также есть ряд специфичных событий, которые генерируются при определенных условиях:
Start генерируется в момент, когда впервые запускается приложение и создается домен приложения;
SessionStart генерируется всякий раз, когда начинается новый сеанс;
SessionEnd генерируется при завершении сеанса;
Error генерируется всякий раз, когда в приложении возникает необработанное событие;
End генерируется в момент завершения работы приложения;
Disposed генерируется после завершения работы приложения, когда сборщик мусора .NET готов к восстановлению занимаемой памяти.

Для использования модели событий ASP.NET необходимо переопределить класс приложения HttpApplication. Для этого нужно добавить в проект файл приложения global.asax.

увеличить изображение

Для того чтобы подписаться на глобальные события, необходимо в файл приложения добавить методы, имена которых соответствуют именам событий, причем к имени события необходимо добавить название "Application_". Например, чтобы подписаться на событие Start, необходимо создать метод "Application_Start". Аналогичным образом можно создать обработчики и для других событий. Пример файла приложения с созданными обработчиками глобальных событий приведен ниже.

Аналогичным образом на глобальные события можно подписаться из другого места, например, HTTP-модуля ASP.NET (этот механизм будет рассмотрен далее). Для этого следует воспользоваться классом HttpApplication.

Глобальные события нужны далеко не в каждом приложении ASP.NET. Если приложение выполняет простые операции, использующие готовые алгоритмы, то глобальные события

могут не понадобиться. Однако, в более сложных ситуациях использование глобальных событий ASP.NET является очень удобным. Например, при разработке собственной системы безопасности разработчику не требуется думать о том, в какой момент лучше проверять права доступа у данного пользователя – он может успешно подписаться на соответствующее событие, а среда исполнения ASP.NET вовремя сгенерирует нужное событие. Аналогично, если требуется улучшить систему кэширования, то можно подписаться на нужные события и встраивать свою логику кэширования в существующую инфраструктуру.

Таким образом, глобальные события ASP.NET позволяют встроить дополнительную логику в существующую инфраструктуру, обеспечивая тем самым гибкими возможностями по расширению.

Краткие итоги

При обработке HTTP-запроса веб-приложение выполняет ряд действий, скрытых от разработчика приложений. Для того чтобы вмешаться в обработку этого процесса ASP.NET имеет модель событий. Разработчик может подписаться на любое событие модели событий ASP.NET. Это можно сделать, используя глобальный файл приложения global.asax или использовать класс HttpApplication.

Служебные объекты ASP.NET

В некоторых ситуациях в процессе работы приложения требуется получать специфические данные о контексте исполнения. Например, необходимо работать с данными HTTP-запроса и HTTP-ответа, получать серверную информацию или оставлять отладочные данные. Обычно в других платформах для построения веб-приложений использовались переменные окружения. Для работы с этими данными в ASP.NET существует ряд служебных объектов ASP.NET, позволяющих выполнять указанные операции более удобно и безопасно.

Основными служебными объектами ASP.NET являются:

- HttpContext;
- Request;
- Response;
- Server;
- Trace.

Объект HttpContext является "входной точкой" для получения доступа ко всем остальным объектам. Фактически, через этот объект можно получить всю информацию о текущем контексте исполнения. Обычно объект HttpContext используется за пределами страницы, т.к. страница также позволяет получить доступ к этим объектам. Для того, чтобы получить доступ к текущему контексту достаточно обратиться к статическому свойству Current класса HttpContext: "HttpContext.Current". При получении доступа к текущему контексту становятся доступны все остальные объекты с одноименными названиями свойств.

Объекты Request и Response отвечают за содержимое HTTP-запроса и HTTP-ответа.

Объект Request содержит все параметры URL, HTTP-заголовки и другую информацию, отправляемую клиентом. Кроме того, в этом объекте инкапсулируется информация о браузере клиента.

Объект Response описывает HTTP-ответ, который впоследствии отправляется клиенту. Используя объект Response можно модифицировать HTTP-ответ – выставлять коды возврата, перенаправлять пользователя на другую страницу, задавать заголовки HTTP-ответа или просто формировать содержимое, которое будет отправлено клиенту. Для помещения данных в выходной буфер используется метод Write класса Response. Таким способом пользуются при разработке собственных элементов управления или HTTP-обработчиков.

Например, давайте создадим фрагмент кода, который считывает адрес, по которому обратился пользователь и количество HTTP-заголовков в HTTP-запросе и сохраним их в HTTP-заголовках HTTP-ответа. Для этого у объектов Request и Response существует коллекция Headers.

Этот пример показывает, каким образом можно работать с объектами Request и Response.

Объект Server предоставляет вспомогательные методы и свойства, необходимые для функционирования веб-приложения. Основные методы и свойства объекта Server:
MachineName позволяет узнать имя компьютера;
HtmlEncode() и HtmlDecode() заменяет обычную строку строкой допустимых символов HTML, и наоборот;
UrlEncode() и UrlDecode() заменяет обычную строку строкой допустимых символов URL, и наоборот;
MapPath() преобразует относительный путь к файлу в рамках веб-приложения в абсолютный путь к файлу на диске (к примеру из "~/images/pic1.png" получается физический путь, например "C:\inetpub\wwwroot\images\pic1.png");
Transfer() передает исполнение другой веб-странице в текущем приложении.

Наконец объект Trace является универсальным объектом трассировки. Он позволяет записывать отладочную информацию в отладочный журнал на уровне страниц. Для отображения отладочной информации на уровне страницы необходимо задать значение атрибута Trace в директиве страницы равным значению True. Для этого необходимо открыть шаблон страницы (.aspx) и сделать необходимые изменения.

После выполнения этой операции при выполнении каждого запроса к этой странице после содержимого самой страницы будет отображаться отладочная информация. Пример отображения отладочной информации представлен ниже.

увеличить изображение

В составе этой отладочной информации содержится детальная информация о запросе, информация о трассировке, дерево элементов управления на странице, состояние сеанса (Session), состояние приложения (Application), содержание коллекции Cookie, HTTP-заголовки и другая полезная информация.

Также видно, что в рамках этой информации содержится раздел о выполняющихся событиях. Он позволяет отследить последовательность выполнения событий в рамках обработки запроса. Если нужно добавить свое событие в этот список, то можно воспользоваться объектом Trace. Для этих целей этот объект содержит два метода – Write и Warn. Оба этих метода добавят запись о своем выполнении. Отличие их состоит в том, что метод Write сделать обычную запись о выполнении события, а метод Warn также выделит эту запись красным шрифтом.

Например, можно написать следующий фрагмент кода, который добавляет информацию в список трассировки в момент создания объекта страницы и в момент обработки события Page Load.

Запустив страницу на исполнение при включенной трассировке можно увидеть следующий набор сообщений.

увеличить изображение

Таким образом, служебные объекты ASP.NET необходимы для получения и изменения служебной информации о текущем контексте исполнения.

Краткие итоги

В рамках среды исполнения ASP.NET существует ряд служебных объектов, которые позволяют получить доступ к различной служебной информации. К такой информации относятся данные об HTTP-запросе и HTTP-ответе, информация о сервере, трассировке и др. Доступ ко всем объектам можно получить либо через одноименные свойства страницы, либо через служебный объект HttpContext.

Зарезервированные папки

При разработке веб-приложения иногда требуется разместить специальные служебные файлы, которые не должны быть доступны для скачивания. Например, к таким файлам относятся файлы данных СУБД, файлы тем, файлы с программным кодом, файлы ресурсов и др. Если разместить подобные файлы за пределами папки с веб-приложением, то потребуется произвести ряд настроек для обеспечения прав доступа к этим папкам. Более того, такой подход не очень удобен с точки зрения администрирования такого приложения. Гораздо удобнее – размещать все необходимые файлы в папке с самим приложением. Однако, если пользователь будет иметь возможность скачать эти файлы по протоколу HTTP – это несомненная уязвимость в безопасности веб-приложения.

Для таких случаев можно создать специальные служебные папки в составе веб-приложения и запретить к ним доступ по протоколу HTTP. Это можно сделать, используя стандартные средства безопасности ASP.NET. Однако по неосторожности некоторые разработчики могут забыть это сделать, и это сделает приложение уязвимым (например, если пользователи скачают файл данных, потенциально они могут получить доступ к паролям всех пользователей веб-приложения). Поэтому в структуре проекта приложения ASP.NET существует ряд стандартных зарезервированных папок.

Зарезервированными папками являются обычные папки в составе проекта ASP.NET, которые, однако, имеют заранее предопределенные имена и содержат строго специфичную информацию. Основным отличием зарезервированных папок от остальных является невозможность загрузить их содержимое напрямую по протоколу HTTP.

К зарезервированным папкам относятся следующие папки:

Bin содержит все предварительно скомпилированные сборки .NET, используемые web-приложением ASP.NET;

App_Code содержит файлы исходного кода, которые не соотносятся с конкретными страницами;

App_GlobalResources содержит файлы глобальных ресурсов, которые доступны всем

страницам web-приложения;
App_LocalResources содержит файлы локальных ресурсов, которые специфичны для каждой страницы web-приложения;
App_Data содержит файлы данных. Например, файлы SQL Server, текстовые файлы, XML и проч.;
App_Browsers содержит определения браузеров, которые определяет разработчик конкретного web-приложения;
App_Themes содержит файлы тем для оформления сайта

Для того, чтобы создать зарезервированную папку в составе проекта ASP.NET следует выбрать пункт меню "Add ASP.NET Folder" во всплывающем меню приложения.

После создания зарезервированной папки в структуре проекта ASP.NET, не требуется выполнять никаких дополнительных действий по ограничению доступа к этим папкам.
Краткие итоги

В структуре проекта ASP.NET можно создать ряд служебных папок, в которых будет содержаться информация служебного характера. К такой информации относятся файлы данных, программного кода, ресурсы, темы и др. Основным отличием зарезервированных папок является то, что их содержимое не доступно напрямую по протоколу HTTP.

Конфигурация приложений

Разрабатывая приложения, можно столкнуться с ситуацией, когда некоторые значения не следует сохранять в листинге исходного кода программы, а эти значения зависят от той среды, где приложение исполняется. В случае сохранения этих настроек в исходном коде, приложение пришлось бы перекомпилировать всякий раз, когда оно переносится в другую среду, например, на другой компьютер.

Поэтому очень часто такие значения выносят за рамки исходного кода программы и сохраняют их во внешнем файле, который впоследствии можно изменить – файл конфигурации.

Исторически сложилось, что конфигурирование приложения - одна из наиболее классических задач, с которой сталкиваются разработчики приложений. Эта задача не специфична только для веб-приложений и решается в различных платформах по-разному.

В более ранних инструментах разработки приложений не содержалось встроенных средств для работы с настройками приложения, поэтому разработчикам каждый раз приходилось этот механизм разрабатывать заново. Формат настроек не был стандартизирован и каждый разработчик хранил в том формате, в котором считал нужным (например, в текстовом файле или в бинарном файле с собственной структурой). Подобный подход создавал ряд неудобств при разработке и использовании приложений. При разработке приложений приходилось постоянно возвращаться к вопросу о том, как хранить настройки приложения. При использовании приложения никогда не было стандартизированного формата конфигурационного файла, который можно использовать.

Платформа .NET Framework решает этот вопрос – в рамках платформы существует стандартный способ хранения конфигурации. В качестве формата для хранения настроек используется формат XML. Использование XML в качестве формата файла конфигурации

обладает рядом преимуществ:

- он имеет иерархическую структуру, что, несомненно, удобно для хранения настроек приложения;
- файл XML, по сути, является текстовым файлом, что означает, что настраивать приложения .NET можно используя любой текстовой редактор.

Все конфигурационные файлы в рамках платформы .NET имеют расширение ".config". Например, для настройки веб-приложений используются файлы "web.config", а для настольных – "app.config".

Существует целая иерархия файлов конфигурации. Схема хранения файлов настроек показана ниже.

увеличить изображение

Система конфигурации .NET Framework разделена на две части – глобальные настройки и настройки приложения. Настройки, заданные в глобальных файлах конфигурации доступны для каждого приложения, т.е. глобальные настройки наследуются в каждом приложении. На уровне приложения определяется главный файл конфигурации (он расположен в корневой папке приложения), в котором задаются основные настройки приложения. Для каждой подпапки в рамках проекта может быть определен собственный файл конфигурации. В этом случае для этой подпапки существует собственная конфигурация, которая наследуется из основного файла конфигурации и уточняется файлом конфигурации в данной папке. Такая иерархия позволяет определить глобальные настройки и постепенно, где это необходимо, уточнять их.

Глобальные файлы настроек расположены в папке "C:\Windows\Microsoft.NET\Framework*версия+\CONFIG". Как видно из приведенной выше схемы в этой папке содержаться два файла – machine.config и web.config. Оба этих файла содержат конфигурационные настройки и имеют идентичный формат. Разница этих файлов заключается в том, что параметры, определенные в файле "machine.config" нельзя переопределить на уровне приложения, а параметры, определенные в файле "web.config" можно переопределять на любом уровне иерархии. Например, давайте рассмотрим параметр, который отвечает за конкурентный доступ к веб-сервисам на основе платформы WCF. Если этот параметр определить в глобальном файле "web.config", то он будет доступен для каждого приложения. При этом нет необходимости определять его для каждого приложения – он уже определен. Однако, если требуется изменить этот параметр для конкретного приложения, то это можно легко сделать переопределив этот параметр в файле "web.config" для приложения. Однако, если переместить определение этого параметра из глобального файла "web.config" в глобальный файл "machine.config", то переопределить параметр на уровне приложения уже будет нельзя.

Возможность такого "жесткого" задания конфигурационных параметров позволяет явным образом задавать политики сервера, в рамках которых исполняются множество приложений. При этом каждое приложение не сможет установить настройки сервера так, как ему это необходимо.

Как уже было сказано ранее, конфигурационные файлы построены на базе формата XML. Пример конфигурационного файла приведен ниже.

Как видно, файл конфигурации содержит ряд настроек в иерархическом виде. Каждый узел в файле настроек определяет отдельный аспект функционирования веб-приложения – настройки кеширования, настройки компиляции и отладки, строки соединения с СУБД, параметры безопасности и т.д. Для каждого узла определен собственный класс-обработчик этих настроек. Узел не может быть добавлен в конфигурационный файл, если с ним не ассоциирован никакой класс-обработчик. В противном случае при запуске приложения будет сгенерирована ошибка.

Система конфигурации .NET Framework предполагает использование файлов конфигурации и для хранения собственных параметров конфигурации приложения, а не только общих настроек. Для того, чтобы добавить свою секцию конфигурации в файл настроек, для начала следует определить эту секцию специальным образом. Для этого существует стандартная секция "configSections". Определить секцию конфигурации можно следующим образом.

В приведенном примере мы определяем секцию с именем "MySection". При этом должен быть создан класс "MyConfigSection", который является обработчиком параметров этой секции. После в конфигурационном файле можно определить и саму конфигурационную секцию и задать нужные параметры.

Количество параметров, уровень вложенности и весь внешний вид секции конфигурации определяет класс-обработчик для этой конфигурационной секции. Этот класс является наследником базового класса "ConfigurationSection", который содержит базовые механизмы для работы с данными конфигурационных файлов. В наиболее простой ситуации можно определить обработку свойств секции, в данном случае свойства "MyParam1". Для этого необходимо создать в классе-обработчике новое публичное свойство и разметить его соответствующим атрибутом "ConfigurationProperty".

Такое определение позволит среде исполнения .NET Framework корректно интерпретировать конфигурационную секцию.

Следует отметить, что все конфигурационные секции, доступные по умолчанию в .NET Framework объявлены подобным образом (используя секцию "configSections"). Эти определения сделаны в глобальном файле "machine.config".

увеличить изображение

Наконец, для получения доступа к существующим конфигурационным секциям используется объект WebConfigurationManager. Для этого следует воспользоваться статическим методом "OpenWebConfiguration" для получения объекта Configuration, а затем методом "GetSection" для получения доступа к конкретной секции. После этого, эту секцию можно привести к классу-обработчику этой секции и, используя публичные свойства последнего, работать с настройками конфигурации.

Таким образом, подсистема конфигурации .NET Framework является мощным механизмом, который позволяет, как переопределять стандартные настройки приложения, так и определять собственные параметры конфигурирования.
Краткие итоги

Типичная проблема при разработке любого приложения – способ хранения параметров

приложения. В .NET Framework существует стандартный механизм, который позволяет решить эту проблему. Все конфигурационные файлы приложения хранятся в формате XML и имеют расширение ".config". При этом существует иерархия файлов конфигурации. На глобальном уровне существует два файла – machine.config и web.config. Настройки, определенные в файле machine.config нельзя переопределять на уровне приложения, а настройки, определенные в файле web.config – можно. На уровне приложения файлы конфигурации также могут наследоваться. Для создания собственной секции настройки необходимо создать класс-обработчик для этой секции и объявить секцию в разделе "configSections". Доступ к настройкам приложения осуществляется на основе объекта WebConfigurationManager.

Сохранение состояния

Одной из наиболее важных проблем при разработке веб-приложений является специфика взаимодействия клиента и сервера по протоколу HTTP. Как известно, взаимодействие клиента и сервера по протоколу HTTP происходит в режиме "запрос-ответ", при этом сервер удаляет все данные, специфичные для этого взаимодействия после того, как отправит ответ. Веб-сервер вынужден делать подобную очистку из-за того, что он может обслуживать большое количество клиентов.

Однако, несмотря на это в большинстве случаев веб-приложению требуется сохранять состояние приложения между выполняющимися запросами. Например, приложению может потребоваться сохранить имя пользователя и пароль для дальнейшего взаимодействия с сайтом, или сохранить состояние формы для ее корректной обработки в будущем. Эту специфику учитывает платформа Microsoft ASP.NET и разработчику не требуется изобретать собственных механизмов по сохранению состояния.

Платформа ASP.NET содержит целый ряд инструментов, которые позволяют сохранять состояние между HTTP-запросами:

- строка запроса (QueryString);
- состояние вида (ViewState);
- состояние сенаса (Session);
- состояние приложения (Application);
- профили;
- и другие.

Как видно, доступно множество способов сохранения состояния. Они отличаются друг от друга контекстом сохранения состояния или, иначе говоря, временем жизни данных, сохраненных между запросами.

Строка запроса (QueryString) позволяет сохранять состояние при обращении к текущей или внешней странице сайта. При этом значение состояние передается в строке запроса в качестве параметра GET. Например, такой запрос может выглядеть следующим образом:

http://localhost/Default.aspx?StateValue=J1HKJH912IRWYQOICNOUQW

или

http://localhost/SomePage.aspx?Data=NIWQ7DWIU9WQ

При таком способе сохранения состояния, после перенаправления пользователя на новую (или ту же самую) страницу, в GET-параметрах передается ее состояние. При этом не имеют значения названия ключей или типы страниц.

Подобный подход удобен, когда необходимо передавать небольшие по объему, текстовые данные, которые не являются секретными. Если требуется работать с большим объемом данных или передаваемые данные не являются публичными, то нужно отказаться от этого способа сохранения состояния в пользу других методов. Нужно четко понимать, что если передавать с помощью строки запроса набор данных, то эти данные изменяют вид адреса страницы; если этих данных будет слишком много, то адрес страницы будет не читаем, что иногда неудобно.

По этим причинам подобный способ сохранения состояния хоть и используется в практике разработки веб-приложений, но его популярность очень мала. Вместо этого зачастую используются другие способы сохранения состояния.

Состояние вида (ViewState) позволяет передавать состояние между HTTP-запросами в рамках одной и той же страницы.

Платформа ASP.NET Web Forms (см. далее) содержит специальный способ обработки формы, который называется Postback. Идея этого механизма состоит в том, чтобы отправить данные на сервер с помощью HTTP-метода POST, при котором все поля формы также отправляются на сервер и при этом обращение осуществляется к той же самой странице. После обращения к веб-странице посредством механизма Postback форма как правило совершает определенный цикл обработки и модифицирует содержимое или внешний вид формы.

Использование механизма ViewState тесно связано с механизмом Postback. Механизм ViewState обеспечивает передачу данных между HTTP-запросами на основе скрытого поля, которое хранится в HTML-коде страницы. Если странице требуется сохранить состояние между запросами Postback, она сохраняет эти значения в скрытом поле HTML. При следующем Postback-запросе эти данные снова будут отправлены на сервер, где будут обработаны. Так процесс может повторяться несколько раз, пока происходят обращения Postback к данной форме. Однако, состояние ViewState теряется при переходе к другой странице.

Скрытое поле ViewState в коде HTML выглядит следующим образом.

Однако, если в состояние вида ViewState добавить какие-то данные, то оно видоизменится и может стать таким, как показано ниже.

Как видно, данные во ViewState хранятся в закодированном виде (не зашифрованном!) по алгоритму base64. При обработке Postback-запроса среда исполнения анализирует содержимое скрытого поля и помещает данные из него в объект ViewState (коллекция "ключ-значение"), который доступен разработчикам. В процессе обработки страницы содержимое объекта ViewState может изменяться, после чего в конце цикла обработки страницы этот объект трансформируется в строку (сериализуется), кодируется при помощи алгоритма base64 и помещается в скрытое поле. Таким образом, для работы с этим способом сохранения состояния необходимо использовать объект ViewState как показано ниже.

Как правило, при разработке приложений редко приходится напрямую пользоваться объектом ViewState. Обычно, объект ViewState используют элементы управления для сохранения своего состояния. Для управления использованием механизма ViewState у элементов управления есть свойство EnableViewState, которое по умолчанию включено. Если разработчику не требуется подобное поведение для элемента управления, он может отключить использование ViewState для него.

Приведенный способ сохранения состояния является достаточно удобным, но порождает ряд дополнительных проблем.

Во-первых, при частом использовании ViewState размер скрытого поля страницы быстро разрастается и может достигать огромных размеров. Это увеличивает размер страницы и, несомненно, сказывается на времени передачи страницы от сервера клиенту.

Во-вторых, у этого механизма существует потенциальная угроза перехвата и подмены данных в поле ViewState. Для решения проблемы с подменой данных можно включить механизм дополнительной защиты – хеш-код. При включении этого механизма в состоянии вида хранится надежно зашифрованная контрольная сумма. Это позволяет отследить подмену состояния вида и среагировать на эту ситуацию. Для включения этого механизма защиты необходимо установить свойство текущей страницы EnableViewStateMAC в значение True. Однако даже если использовать хеш-коды, данные состояния вида все равно остаются доступными для чтения и существует угроза перехвата этих данных. Чтобы исключить возможность перехвата данных можно включить шифрование состояния вида. Для этого необходимо установить свойство ViewStateEncryptionMode у текущей страницы в значение Always.

В целом, состояние вида ViewState используется для передачи данных между запросами Postback и в некоторых ситуациях может быть полезным.

Другим способом сохранения состояния является состояние сеанса (Session) или просто сессия. Этот способ сохранения состояния позволяет сохранять данные на протяжении всего сеанса работы пользователя с веб-приложением. В этом случае нет ограничения на сохранение состояния в рамках одной страницы – состояние сеанса доступно на протяжении работы с каждой страницей.

Каким образом работает этот механизм? Состояние сеанса базируется на встроенной в браузер возможности сохранять Cookie – небольшие данные, которые хранятся на стороне клиента. Если серверу необходимо сохранить какие-то данные в браузере пользователя, он генерирует специальный HTTP-заголовок, который включается в состав HTTP-ответа. При получении ответа браузер анализирует HTTP-ответ на наличие подобных заголовков и, в случае если такой заголовок обнаружен, сохраняет представленные данные на локальном компьютере в специальной папке. После этого при каждом обращении браузера к данному веб-приложению, браузер будет добавлять специальные HTTP-заголовки к своим HTTP-запросам. Таким образом, у сервера есть возможность сохранить на клиенте какие-то данные и получать к ним доступ при каждом обращении клиента к серверу.

Механизм Cookie сам по себе в какой-то степени является способом сохранения состояния. Однако, если его использовать как средство сохранения состояния для большого объема данных, то объем передаваемых данных от клиента к серверу и обратно существенно

возрастает. Кроме того, существует потенциальная угроза получения данных из Cookie на стороне клиента, поскольку браузер никак не защищает эти данные. Именно поэтому был разработан механизм сохранения состояния сеанса.

Основная идея состояния сеанса заключается в следующем. При первом обращении браузера к серверу (когда Cookie чистые) сервер создает специальную область памяти, которой назначает уникальный идентификатор. Этот идентификатор сервер записывает в Cookie браузера. После этого браузер каждый раз будет сообщать этот идентификатор серверу. Теперь на сервере можно использовать выделенную для каждого пользователя область памяти для сохранения туда различных данных. При этом каждый пользователь будет иметь доступ к своему участку памяти. В данном случае этот идентификатор является идентификатором сессии. Поскольку пользователи не могут знать идентификаторов сессии других пользователей, сессия в каком-то смысле является более защищенной. Более того, в этом случае в сессии можно сохранять большие объемы информации без необходимости увеличения объема передаваемой информации между клиентом и сервером.

Однако, в таком случае при длительной работе сервера существует опасность, что память сервера закончится. Для того, чтобы этого избежать существует время жизни сессии. Это означает, что если пользователь с определенным идентификатором сессии не проявлял активность в течении определенного времени, то данные, ассоциированные с его идентификатором сессии удалятся с сервера. Это позволяет вовремя очищать сервер от лишних данных.

Как правило, в сессии может храниться идентификатор пользователя, его настройки, а также другая специфичная для каждого пользователя информация. Для работы с состоянием сеанса используется объект Session. По аналогии с ViewState, объект Session является коллекцией "ключ-значение", а работать с ним можно следующим образом.

Важным отличием состояния сеанса Session от состояния вида ViewState заключается в том, что в Session можно сохранять даже те объекты, которые не поддаются сериализации (преобразованию к текстовому представлению). Например, к таким объектам относится объект подключения к SQL Server и др.

Наконец, последним способом сохранения состояния является состояние приложения Application. От предыдущих способов сохранения состояния его отличает то, что Application содержит глобальную информацию, доступную всем пользователям одновременно в рамках всех страниц веб-приложения. Это означает, что при сохранении информации в объект Application одним пользователем, другой пользователь, обратившись к приложению, также может получить доступ к этой информации. Обычно за сохранение состояния приложения отвечает объект приложения HttpApplication, доступный по имени Application.

Работа с объектом Application осуществляется аналогичным образом, как и два предыдущих способа. Объект Application – это коллекция "ключ-значение". Для работы с этим объектом может использоваться следующий код.

Однако, к объекту Application (как и к любым другим глобальным данным) следует относиться осторожно, поскольку глобальные состояния могут вносить дополнительную сложность в программный код.

Таким образом, видно, что платформа ASP.NET обладает богатыми возможностями по

сохранению состояния между HTTP-запросами. В данной лекции мы рассмотрели основные способы сохранения состояния и в каждой конкретной ситуации стоит выбирать именно тот способ, который наиболее удобен в этой ситуации.

Краткие итоги

Сложность разработки веб-приложений состоит в том, что взаимодействие по протоколу HTTP не предполагает сохранения состояния между обращениями. Однако, это бывает очень необходимо в большом числе ситуаций. Для этого в ASP.NET существует ряд инструментов для сохранения состояния. Состояние вида ViewState позволяет сохранять состояние между обращениями клиента к одной и той же странице. Состояние сеанса Session позволяет сохранять состояние для конкретного пользователя для всего веб-приложения. Наконец, состояние приложения Application позволяет сохранять глобальное состояние, доступное каждому пользователю в рамках приложения. Все эти способы доступны для работы через коллекции в стиле "ключ-значение" и очень просты в использовании.