ГЛАВА 35

Создание веб-служб

В течение многих лет разработчики и дизайнеры программного обеспечения пытались создавать программные компоненты, к которым можно было бы подключаться удаленно через локальные сети или Интернет. В результате этого процесса появилось несколько новых технологий и поставляемых с ними патентованных решений. Хотя некоторые из этих технологий работали достаточно успешно, запуская прикладные системы во внутренних сетях, ни одна из них не удовлетворяла требованиям Интернета, представляющей собой огромную, порой ненадежную, сеть компьютеров, которые запускаются на оборудовании и операционных системах самых разнообразных типов.

Вот где в игру вступили веб-службы. Для взаимодействия с веб-службой необходимо просто отослать XML-сообщение через HTTP. Поскольку каждое совместимое с Интернетом устройство поддерживает протокол HTTP, и практически каждый язык программирования имеет доступ к XML-анализатору, ограничений касательно того, какие типы приложений могут использовать веб-службы, почти не существует. На самом деле, многие среды программирования включают наборы инструментов более высокого уровня, которые позволяют подключаться к веб-службе так же просто, как вызывать локальную функцию.

В этой главе представлена общая информация о веб-службах и проблемах, которые они решают. Те, кто вообще не знаком с веб-службами, в этой главе смогут узнать, как создавать веб-службы и использовать их в ASP.NET. Однако никаких подробных сведений о лежащих в основе этих служб протоколах читатель здесь не найдет: об этом речь пойдет уже в следующей главе.

Изменения, касающиеся веб-служб, в .NET 2.0

Тем, кто работал с веб-службами в .NET 1.*x*, наверняка интересно узнать, что же изменилось в версии .NET 2.0. С практической точки зрения изменений удивительно мало: на самом деле, лежащая в основе инфраструктура ничем не отличается от прежней. Однако некоторые полезные изменения все же имеются и многие из них касаются того, как веб-службы работают со сложными типами (специальными классами данных, которые используются для извлечения или отправки информации в веб-метод). Часть их них перечислена ниже.

- *Поддержка для процедур свойств.* Если веб-служба использует специальный тип, .NET создает на клиенте копию такого класса. В версиях .NET 1.*x* такой автоматически генерируемый класс создавался из общедоступных свойств. В .NET 2.0 для этого используются процедуры свойств. Это незначительное изменение не влияет на работу веб-службы, но зато позволяет использовать данный класс в сценариях привязки данных.
- *Совместное использование типов.* .NET теперь распознает, когда две веб-службы используют один и тот же сложный тип, и генерирует только один класс на стороне клиента. Лучше всего то, что поскольку оба прокси-класса используют одинаковые типы, вы можете легко извлекать объект из одной веб-службы и отправлять его в другую.

- *Пользовательская сериализация.* Вы теперь можете включаться в процесс сериализации пользовательских классов и тем самым полностью контролировать их XML-представление. Хотя эта возможность была доступна и в AP.NET 1.*x*, она не была документирована и официально не поддерживалась.
- *Многофункциональные объекты.* Хотите обмениваться сложными объектами так, чтобы их методы и конструкторы оставались неизменными? Это возможно, если вы создадите новый компонент, который называется *импортером схем* (schema importer). Импортер схем проверяет схему веб-службы и сообщает прокси-классу, какие типы должны использоваться.
- *Разработка на основе контракта.* Вы теперь можете создавать в .NET веб-службу на основе какого-нибудь существующего контракта WSDL.

Все эти улучшения более подробно будут описываться в главе 36.

Обзор веб-служб

Если HTML-страницы (или HTML-вывод, генерируемый веб-формами ASP.NET) считываются конечными пользователями, то веб-службы используются другими приложениями. Они представляют собой фрагменты бизнес-логики, к которым можно получать доступ через Интернет. Например, сайты электронной коммерции могут использовать веб-службу компании, занимающейся доставкой и упаковкой, для вычисления стоимости доставки того или иного товара. Сайты новостей могут извлекать заголовки новостей и статьи, составляемые внешними службами новостей, и отображать их на своих собственных страницах в реальном времени. Компания может даже предоставлять в реальном времени значения своих фондовых опционов, считывая их с какого-нибудь финансового или инвестиционного сайта. Все эти сценарии уже реализованы в Web, и ведущие Интернет-компании, такие как Amazon, Google и eBay, уже предлагают свои собственные веб-службы сторонним разработчикам.

Благодаря веб-службам, вы можете посредством всего лишь нескольких строк кода воспользоваться бизнес-логикой кого-нибудь другого вместо того, чтобы воспроизводить ее самостоятельно. Эта технология похожа на ту, которую программисты сейчас применяют для библиотек API-интерфейсов, классов и компонентов. Главное отличие состоит в том, что веб-службы могут располагаться удаленно на другом сервере и управляться другой компанией.

История веб-служб

Хотя веб-службы — это новая технология, из истории их происхождения можно узнать много полезного. Двумя наиболее важными событиями в разработке программного обеспечения за два последних десятилетия стало появление технологии объектно-ориентированного программирования и технологии проектирования программного обеспечения на основе компонентных объектов.

Технология объектно-ориентированного программирования начала широко использоваться в начале 80-х годов прошлого века. Многие рассматривали ее как средство для разрешения кризиса, возникшего в результате роста степени сложности и размеров программных приложений. Разработка проектов занимала все больше и больше времени и все чаще выходила за рамки предусматриваемого для них бюджета, а конечный результат все равно нельзя было назвать надежным. Технология объектно-ориентированного программирования обещала, что путем структуризации кода в объекты разработчики смогут создавать компоненты, которые будет проще использовать многократно, расширять и обслуживать.

В 1990 г. на свет появилась компонентная технология, которая сделала возможным создание приложений путем сборки компонентов. Компонентная технология — это, по сути, расширение объектно-ориентированных принципов за пределами любого одного конкретного языка, в результате чего он становится основной частью инфраструктуры, которой может пользоваться каждый. Если объектно-ориентированные языки позволили разработчикам многократно использовать объекты в своих приложениях, то компонентные технологии позволили им легко обмениваться скомпилированными объектами с *другими* приложениями. Наибольшее распространение получили две таких компонентных технологии: COM (Component Object Model — модель компонентных объектов) и CORBA (Common Object Request Broker Architecture — общая архитектура брокера запросов к объектам). Позже появились и другие компонентные технологии (такие как JavaBeans и .NET), но они представляли собой патентованные решения для определенных сред программирования.

Практически сразу же после создания стандарты COM и CORBA начали применять к распределенным компонентам для того, чтобы приложение могло взаимодействовать с объектами, обслуживаемыми на разных компьютерах в сети. Хотя и COM, и CORBA обладают сложнейшими техническими характеристиками, зачастую их бывает довольно трудно настроить и поддерживать в сетевых средах, и они не могут работать вместе. Эти недостатки значительно усугубились, когда появился Интернет, и разработчики начали применять эти технологии при создании распределенных приложений для более медленных и менее надежных глобальных сетей (WAN). Из-за этих сложностей и несовместимости между собой, ни технология COM, ни технология CORBA так и не смогли стать по-настоящему успешными в этой среде.

Веб-службы — это новая технология, призванная решить упомянутые проблемы путем расширения технологии компонентных объектов так, чтобы та могла работать в Web. По сути, веб-служба — это фрагмент логики приложения (компонент), который может вызываться удаленно через Интернет. Задачи у технологии веб-служб те же, что у технологии компонентных объектов, главными из которых являются: упростить процесс сборки приложений из заготовленной логики, обеспечить возможность совместно использования функциональных возможностей организациями и партнерами и позволить создавать приложения, состоящие из большего числа модулей. В отличие от ранних технологий компонентных объектов, веб-службы разрабатывались *исключительно* для этой цели. Это означает, что при желании обеспечить возможность обмена скомпилированными блоками между .NET-приложениями, вы все равно должны будете использовать модель компонентных объектов .NET. Однако при необходимости обеспечить возможность обмена функциональными возможностями между приложениями, работающими на разных платформах или обслуживаемых разными компаниями, прекрасно подойдут веб-службы.

Также в технологии веб-служб делается намного больший акцент на функциональной совместимости. Все платформы для разработки программного обеспечения, которые позволяют программистам создавать веб-службы, используют одинаковые принципы открытых стандартов XML. Это гарантирует, что вы сможете, например, создать веб-службу, используя .NET, и вызывать ее с клиента Java, или наоборот, создать веб-службу с помощью Java и вызывать ее из .NET-приложения.

На заметку! Веб-службы доступны не только в среде .NET Framework. Их стандарты были определены еще до выпуска .NET и поставляются, используются и поддерживаются также и другими производителями (а не только Microsoft). Среда .NET Framework является особенной, потому что она скрывает весь “рабочий” (низкоуровневый) код, что упрощает выполнение таких задач, как отображение своих собственных служб через Интернет и получение доступа к службам, предоставляемым другими компаниями. Вы увидите, что вам вовсе не обязательно знать

все подробности об XML и SOAP, чтобы успешно программировать веб-службы (хотя, конечно, такие знания совсем не помешают). ASP.NET обобщает все такие подробности и генерирует интерфейсные классы, которые отображают простую объектно-ориентированную модель, позволяющую легко отправлять, получать и интерпретировать SOAP-сообщения.

Распределенная обработка данных и веб-службы

Чтобы полностью осознать степень значимости веб-служб, вы должны понимать, что собой представляет *распределенная обработка данных* и каковы ее требования. Распределенная обработка данных — это разделение логики приложения на блоки, которые будут выполняться на двух или более компьютерах в сети. Идея распределенной обработки данных появилась давно, и для обеспечения возможности распределения и многократного использования логики приложения уже было разработано немало технологий связи.

Причин для распределения логики приложений достаточно много. Некоторые наиболее важные из них перечислены ниже.

- *Высокая масштабируемость.* Разделение логики приложения распределяет нагрузку между несколькими разными машинами. Это, как правило, не приводит к улучшению производительности приложения для отдельных пользователей (а, на самом деле, может даже снизить ее), но зато практически всегда повышает степень масштабируемости приложения, тем самым позволяя ему обслуживать одновременно намного большее количество пользователей.
- *Простое развертывание.* Фрагменты распределенного приложения могут обновляться без обновления всего приложения, а центральный компонент может быть обновлен без обновления сотен (или даже тысяч) клиентов.
- *Более высокая степень безопасности.* Распределенные приложения часто выходят за пределы компании или организации. Например, вы могли бы использовать распределенные компоненты, чтобы позволить торговому партнеру запрашивать каталог продуктов вашей компании. Разрешить ему подключаться непосредственно к самой базе данных компании было бы небезопасным. А так он будет пользоваться запускающимся на ваших серверах компонентом, который вы можете настроить и ограничить так, как вам нужно.

С появлением Интернета степень важности и применимости технологии распределенной обработки данных значительно возросла. Простота и повсеместность Интернета делают его вполне логичным выбором для использования в качестве основы для распределенных приложений.

До появления веб-служб доминирующими протоколами были COM (в случае применения в сети, называемый протоколом DCOM) и CORBA. Хотя протоколы COM и CORBA имеют много общего, в некоторых моментах они сильно отличаются, что делает их взаимодействие практически невозможным. В табл. 35.1 перечислены некоторые основные сходства и различия между CORBA, DCOM и веб-службами, а также приводится целый ряд акронимов с расшифровкой их значения.

И CORBA, и DCOM допускают вызовы удаленных объектов. CORBA использует стандарт, называемый IIOP, а DCOM — стандарт под названием DCERPC (см. табл. 35.1). В CORBA кодировка данных осуществляется на основе формата CDR (Common Data Representation — общее представление данных), а в DCOM — на основе похожего, но несовместимого формата NDR (Network Data Representation — представление данных в сети). Разница между используемыми этими технологиями стандартами существенно усложняет дело.

Таблица 35.1. Сравнение различных распределенных технологий

Характеристики	CORBA	DCOM	Веб-службы
Механизм RPC (Remote Procedure Call — вызов удаленных процедур)	IIOP (Internet Inter-ORB Protocol — Протокол, определяющий передачу сообщений между объектами по TCP/IP)	DCE-RPC (Distributed Computing Environment Remote Procedure Call — Протокол вызова удаленных процедур DCE)	HTTP (Hypertext Transfer Protocol — Протокол передачи гипертекста)
Кодировка	CDR (Common Data Representation — Общее представление данных)	NDR (Network Data Representation — Представление данных в сети)	XML (Extensible Markup Language — Расширяемый язык разметки)
Описание интерфейса	IDL (Interface Definition Language — Язык описания интерфейсов)	IDL (Interface Definition Language — Язык описания интерфейсов)	WSDL (Web Service Description Language — Язык описания веб-служб)
Поиск	Система идентификации имен и служба поиска компромиссов	Реестр	Технология UDDI (Universal Description, Discovery and Integration — Универсальное описание, поиск и взаимодействие)
Может работать с брандмауэрами?	Нет	Нет	Да
Сложность протоколов	Высокая	Высокая	Низкая
Межплатформенная?	Частично	Нет	Да

Также имеются отличия и между языками, которые поддерживают эти протоколы. DCOM поддерживает много различных языков (C++, Visual Basic и т.д.), но в первую очередь использовался в операционных системах от Microsoft. CORBA поддерживает разные платформы, но наибольшее применение нашел с приложениями, основанными на Java. В результате разработчики получили сразу две платформы, которые имели техническую возможность поддерживать системы распределенных объектов, но не могли работать вместе.

Проблемы с технологиями распределенных компонентов

Функциональная совместимость — это лишь одна из проблем протоколов CORBA и DCOM. Имеются еще и другие технические трудности. Оба протокола разрабатывались до того, как появился Интернет, а это значит, что они не рассчитаны на удовлетворение потребностей слабосвязанных, порой ненадежных сетей с большим трафиком. Например, оба эти протокола ориентированы на установление логических соединений. Это значит, что клиент DCOM устанавливает соединение с сервером DCOM каждый раз, когда ему необходимо сделать вызов. Компонент DCOM на стороне сервера может сохранять информацию о клиенте в памяти. В результате получается многофункциональная гибкая модель программирования, но для разработки приложений с высокой степенью масштабируемости, использующих не сохраняющие информацию о состоянии протоколы Интернета, она не подходит. Если клиент просто исчезает, не очистив, как следует, соединение, происходит ненужная трата ресурсов. Точно так же, если тысячи

клиентов попытаются подключиться одновременно, сервер запросто может оказаться перегруженным, исчерпав выделенные запасы памяти или соединений.

Еще одна проблема состоит в том, что оба эти протокола являются чрезмерно сложными. Они объединяют технологию распределенных объектов с функциями для сетевой безопасности и управления. Веб-службы намного легче использовать во многом благодаря тому, что они не имеют такого уровня сложности. Однако это вовсе не означает, что вы не можете создать безопасную веб-службу. Это просто означает, что если вы решите это сделать, вам придется использовать технологию веб-служб *и* еще какой-нибудь стандарт вроде SSL (как реализуемый веб-сервером) или WS Security и XML Encryption (как реализуемый средой программирования).

На заметку! Опасность здесь состоит в том, что разработчики могут просто начать путаться в стандартах, количество которых растет с каждым днем и которые не нужны для создания обычных веб-служб, но необходимы для создания более сложных использующих веб-службы приложений. Однако данная модель все равно является на сегодняшний день самым оптимальным вариантом по критериям сложности и простоты. Ее преимущество заключается в том, что она позволяет разработчикам создавать новые функции для веб-служб (например, поддержку транзакций), не нарушая базового уровня функциональной совместимости, предоставляемого основными стандартами веб-служб.

Преимущества веб-служб

Веб-службы интересны сразу с нескольких точек зрения. С технологической точки зрения веб-службы пытаются решить проблемы, встречающиеся при использовании тесно связанных технологий, таких как CORBA и DCOM. Это проблемы типа прохождения брандмауэров, обработки низкоуровневых транспортных протоколов и интеграции гетерогенных платформ. Веб-службы также интересны с организационной и экономической точек зрения, потому что они открывают двери для новых способов организации бизнеса и интеграции систем между предприятиями.

DCOM и CORBA прекрасно подходят для построения прикладных систем предприятия, в которых все программное обеспечение запускается на одной и той же платформе и в одной и той же локальной сети. Однако они никак не подходят для разработки прикладных систем, работающих сразу на нескольких платформах, которые имеют доступ в Интернет и требуют высокой степени масштабируемости, соответствующей возможностям Интернета. Они просто вообще разрабатывались для другой цели.

Вот где в игру вступают веб-службы. Веб-службы представляют собой следующий логический шаг в эволюции технологий распределенных компонентных объектов. Некоторые основные их преимущества перечислены ниже.

- *Веб-службы просты.* Это означает, что они легко могут поддерживаться на самых разнообразных платформах.
- *Веb-службы являются слабосвязанными.* Веб-служба позволяет расширять свой интерфейс и добавлять новые методы, не причиняя вреда клиентам, и при этом продолжает предоставлять старые методы и параметры.
- *Веб-службы не поддерживают состояний.* Клиент отсылает запрос веб-службе, веб-служба возвращает результат и соединение закрывается. Никакого постоянного соединения не существует. Это упрощает подключение множества клиентов и позволяет использовать для обслуживания веб-служб сразу группу серверов. Лежащий в основе протокол HTTP, используемый веб-службами, также не поддерживает состояний. Конечно, вы можете предоставить для него какое-нибудь состояние, воспользовавшись дополнительными технологиями наподобие тех, которые вы применяете на веб-страницах ASP.NET, включая cookie-наборы. Однако эти технологии не были стандартизированы.

- *Веб-службы могут работать с брандмауэрами.* Брандмауэры нередко представляют проблему для технологий распределенных объектов. Единственное, что срабатывает практически всегда — это использование протокола HTTP и портов 80 и 443. Поскольку веб-службы используют именно этот протокол, они могут проходить через брандмауэры без явного конфигурирования.

На заметку! Тем не менее, брандмауэр все равно может использоваться для блокировки SOAP-трафика. Это возможно потому, что HTTP-заголовок сообщения веб-службы идентифицирует его как SOAP-сообщение, и администратор может сконфигурировать брандмауэр так, чтобы он не пропускал SOAP-трафик. В сценариях типа “бизнес для бизнеса” брандмауэр может пропускать SOAP-трафик, поступающий только с выбранных диапазонов IP-адресов.

Конечно, простота веб-служб имеет свою цену. В частности, веб-службы поддерживают далеко не все функциональные возможности, которые есть в более сложных технологиях распределенных компонентов. Например, они не поддерживают двусторонний режим связи, а это означает, что веб-сервер не может снова вызвать клиента после того, как тот отключился. В этом отношении тесно связанные протоколы, такие как DCOM и CORBA, мощнее веб-служб. В .NET Framework также имеется новая технология — *удаленный доступ* (Remoting) — которая идеально подходит для обеспечения связи между распределенными .NET-приложениями во внутренней сети. Эта технология является своего рода “последователем” технологии DCOM на платформе .NET. Узнать больше о ней вы можете в книге *Advanced .NET Remoting, Second Edition* (Apress, 2005 год). О том, когда лучше использовать эту технологию, а не веб-службы, рассказывается ниже во врезке “Когда следует использовать веб-службы”.

Когда следует использовать веб-службы

Microsoft называет два основных случая, когда следует отдавать предпочтение веб-службам. При необходимости иметь возможность пересекать границы платформы (например, для обеспечения связи между .NET- и Java-приложением) или границы предприятия (например, для обеспечения связи между двумя компаниями) в применении веб-служб есть большой смысл. Также выбирать веб-службы стоит и в случае, когда требуется использовать встроенные функциональные возможности ASP.NET, такие как кэширование, или функциональные возможности IIS вроде SSL-безопасности и аутентификации Windows. Еще в них есть смысл и тогда, когда вы хотите сохранить возможность интеграции сторонних приложений на будущее.

Однако если необходимо просто позволить двум .NET-приложениям совместно использовать функциональные возможности, веб-службы могут оказаться излишними и даже привести к появлению нежелательных непроизводительных издержек. Например, если вы просто хотите, чтобы данные веб-приложения имели доступ к определенной бизнес-логике, лучше создайте блок библиотеки классов (компилируемый в DLL) и используйте его в обоих приложениях. Это позволит избежать непроизводительных издержек внепроцесснвх или сетевых коммуникаций, что может сыграть очень важную роль.

И, наконец, если вы хотите распространять функциональные возможности так, чтобы к ним можно было получать доступ удаленно, но и клиент, и сервер разрабатывались с помощью .NET Framework, вам стоит отдать предпочтение предлагаемой .NET технологии удаленного доступа (Remoting). Эта технология не обеспечивает такого же уровня поддержки для открытых стандартов, подобных SOAP, но позволяет вам использовать различные типы связи, патентованные типы данных .NET, объекты, кумулятивно изменяющие параметры своего состояния в процессе выполнения по вызовам клиентов, а также более быстрый тип связи, осуществляемой по протоколу TCP/IP. Другими словами, она предлагает больше функциональных возможностей и способов улучшить производительность для .NET-решений.

Зарабатывание денег с помощью веб-служб

Любая новая технология просто обречена на провал, если она не предлагает новых возможностей для людей, заинтересованных в зарабатывании денег. С точки зрения делового человека, веб-службы открывают новые возможности по следующим причинам.

- *Новые схемы оплаты.* Пользователь веб-службы может платить определенную сумму денег за пользование данной службой в целом (например, за получение новостей от Associated Press). Другой вариант — повременная оплата или оплата за определенную услугу. Например, поставщик службы проверки кредита может взимать плату за каждый запрос.
- *Взаимодействие и сотрудничество в реальном времени.* Сегодня данные, как правило, воспроизводятся и используются локально. Веб-службы позволяют запрашивать удаленные данные в реальном времени. Для примера возьмем сайт электронной коммерции, на котором продаются компьютерные игры. Этот сайт может подключаться к хранилищу данных и извлекать оттуда сведения о количестве оставшихся на складе дисков, что значительно повысит его качество обслуживания. Ничто не раздражает больше, чем купить что-нибудь через Интернет и на следующий день узнать, что того, что куплено, на складе больше нет.
- *Агрегированные службы.* Веб-служба может включать в себя другие веб-службы, скопированные веб-сайты, унаследованные компоненты, отображаемые посредством патентованных протоколов, и т.д. Типичным примером агрегированной службы является сравнительная служба, предоставляющая пользователям полную информацию обо всех продуктах. Еще один тип службы — это та, которая объединяет в одну группу связанные между собой службы. Например, предположим, что вы переезжаете в новый дом. Кто-то мог бы помочь вам обновить ваш почтовый адрес, найти компанию, занимающуюся перевозкой мебели, и т.д.

Веб-службы ни в коем случае не являются единственной технологией, которая может предложить такие решения. Многие подобные решения уже реализуются сегодня с помощью существующих технологий. Однако веб-службы имеют и стимул, и стандарты, чтобы сделать такие виды решений доступными повсеместно.

Стек протоколов технологии веб-служб

Ключом к успеху веб-служб является то, что они основываются на открытых стандартах *и* что эти стандарты используются всеми крупными производителями программного обеспечения, такими как Microsoft, IBM и Sun. Тем не менее, использование открытых стандартов не приводит автоматически к функциональной совместимости. Для начала, производители должны реализовать все эти стандарты. Более того, они должны реализовать их так, чтобы они были совместимыми.

При построении веб-служб используются несколько спецификаций. На рис. 35.1 показан стек протоколов веб-служб в том виде, в котором он существует на сегодняшний день.

Более подробно о протоколах SOAP, WSDL и UDDI, которые поддерживают веб-службы, будет рассказано в следующей главе. Однако прежде чем вы начнете создавать и использовать веб-службы, вы обязательно должны получить общее представление о том, какую роль играют эти стандарты. В табл. 35.2 приводится краткое описание всех этих стандартов, а в следующих нескольких разделах — некоторые более подробные сведения о каждом из них. Полную информацию о них вы сможете найти в следующей главе.

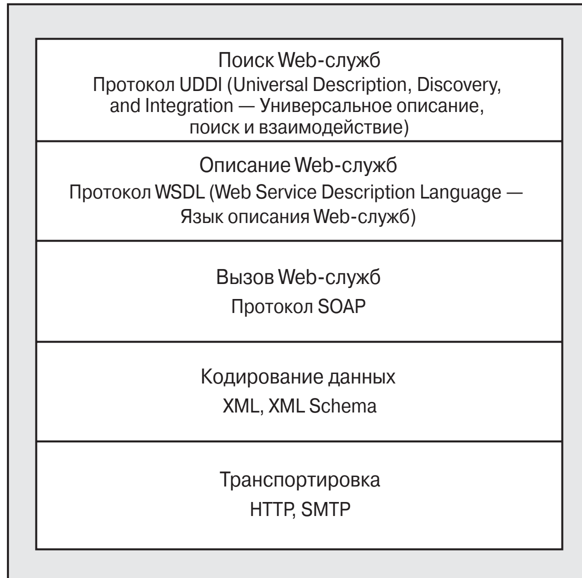


Рис. 35.1. Стек протоколов технологии веб-служб

Таблица 35.2. Стандарты веб-служб

Стандарт	Описание
WSDL	Используется для создания описания интерфейса для веб-службы. WSDL-документ сообщает клиенту, какие методы доступны в данной веб-службе, какие параметры и возвращаемые значения применяет каждый метод, и как с ними следует обращаться.
SOAP	Формат сообщений, используемый для кодирования информации (такой как значения данных) перед ее отправкой веб-службе.
HTTP	Протокол, через который веб-службы непосредственно осуществляют обмен информацией. Например, SOAP-сообщения отсылаются через HTTP-каналы связи.
DISCO	Используется для создания документов поиска, которые содержат ссылки на множество конечных пунктов веб-служб. Этот стандарт совмести только с продуктами Microsoft и со временем будет заменен похожим стандартом, который получил название WS-Inspection.
UDDI	Стандарт для создания бизнес-журналов, фиксирующих имена компаний, предоставляемые ими веб-службы и соответствующие URL-адреса их WSDL-контрактов.

Поиск веб-служб

В случае, когда речь идет о простом приложении, URL-адрес необходимой веб-службы уже может быть известным. Если это так, вы можете просто жестко закодировать его или поместить его в конфигурационный файл. Больше никаких действий вам выполнять не нужно.

В других ситуациях не исключено, что вы сначала захотите выполнить поиск нужной веб-службы. Например, это может быть стандартизированная служба, которую предлагают разные хостинговые компании и которая не всегда доступна по одному и тому же URL-адресу. Или же вам может понадобиться всего лишь какой-нибудь простой

способ найти все веб-службы, предоставляемые данным торговым партнером. И в том, и в другом случае вам придется воспользоваться стандартом DISCO (полное название которого выглядит как *Discovery* (Обнаружение)), чтобы программно установить месторасположение необходимых вам веб-служб.

В принципе в поиске веб-службы помогают две спецификации.

- *DISCO (сокращение от Discovery).* Стандарт DISCO создает один единственный файл, который содержит список связанных между собой веб-служб. Компания может опубликовать этот DISCO-файл со ссылками на все предоставляемые ею веб-службы на своем сервере. В этом случае для поиска всех доступных веб-служб клиентам придется просто запрашивать этот файл. Такой способ удобен, когда клиент уже знает, какая компания предоставляет необходимые ему услуги, и хочет просмотреть список используемых ею для этого веб-служб со ссылками на страницы с их более подробным описанием. Осуществлять поиск новых веб-служб в Интернете не очень выгодно, но может оказаться полезным для локальных сетей, в которых клиент подключается к серверу и может видеть, какие службы ему доступны и где они доступны.
- *UDDI (Universal Description, Discovery, and Integration — Универсальное описание, поиск и взаимодействие).* UDDI — это централизованный каталог, в котором веб-службы опубликованы по именами компаний. Это также место, куда потенциальные клиенты могут обратиться, чтобы отыскать то, что им нужно. Различные организации и группы компаний могут использовать разные UDDI-реестры. Чтобы извлечь информацию из UDDI-каталога или зарегистрировать свои компоненты, следует использовать интерфейс веб-службы.

Discovery — это один из самых новых и наименее разработанных стандартов в стеке протоколов веб-служб. DISCO поддерживается только Microsoft, поэтому в следующих версиях .NET его планируют заменить похожим, но более общим стандартом, который получил название WS-Inspection. Стандарт UDDI подходит для веб-служб, которые необходимо сделать доступными для всех или для какой-то определенной группы (консорциума) компаний или организаций. Он не встроен в .NET Framework, хотя вы можете загрузить отдельный компонент .NET (`http://msdn.microsoft.com/library/en-us/uddi/uddi/portal.asp`) для поиска UDDI-каталогов и регистрации своих компонентов. Поскольку хорошо зарекомендовавших себя UDDI-каталогов пока еще не существует и поскольку многие веб-службы по-прежнему разрабатываются для применения в пределах одной единственной организации или для небольшой группы известных торговых партнеров, скорее всего, большинство веб-служб не будет публиковаться в UDDI.

Описание веб-службы

Чтобы клиент мог получить доступ к веб-службе, ему необходимо знать, какие методы доступны в данной веб-службе, какие параметры использует каждый из этих методов и каким является тип данных каждого из параметров. WSDL (Web Service Description Language — язык описания веб-служб) — это основанный на XML язык, который описывает все эти детали. Он описывает сообщение запроса, которое клиент должен отправить веб-службе, и ответное сообщение, возвращаемое веб-службой. Он также определяет транспортный протокол, который должен использоваться (как правило, это протокол HTTP), и месторасположение веб-службы.

WSDL — сложный стандарт. Но, как будет показано в этой главе, определенные средства используют информацию WSDL и автоматически генерируют вспомогательные классы, которые скрывают низкоуровневый “рабочий” код, требуемый для осуществления взаимодействия с веб-службами.

Формат соединений

Чтобы установить соединение с веб-службой, для начала необходимо как-то создать сообщения запроса и ответа, которые будут анализироваться и восприниматься на любой платформе. Протокол SOAP (полное название которого раньше выглядело как Simple Object Access Protocol (Простой протокол доступа к объектам), но теперь больше не используется) — это основанный на XML язык, с помощью которого должны создаваться эти сообщения.

Важно понимать, что SOAP определяет сообщения, используемые для обмена данными (то есть определяет формат сообщений), но не описывает способ их отправки (за это уже отвечает транспортный протокол). Для веб-служб, создаваемых с помощью ASP.NET, транспортным протоколом является протокол HTTP. Другими словами, чтобы обменяться данными с веб-службой, клиент открывает HTTP-соединение и отправляет SOAP-сообщение.

В .NET также поддерживаются HTTP GET и HTTP POST — два более простых подхода для взаимодействия с веб-службами, которые не являются столь же стандартизированными и не предлагают такой же богатый набор функциональных возможностей. В обоих этих случаях для связи используется канал HTTP, и данные отправляются в виде простой коллекции пар "имя-значение", а не в виде цельного SOAP-сообщения. Единственное место, где вам, возможно, удастся увидеть, как этот более простой подход применяется в .NET-среде — это на простой браузерной странице, которую ASP.NET предоставляет для тестирования ваших веб-служб. На самом деле, по умолчанию файл `machine.config` (в ASP.NET 1.1) допускает получение с локального компьютера только запросов типа HTTP POST и полностью отключает поддержку запросов типа HTTP GET.

На рис. 35.2 в сжатом виде представлен весь жизненный цикл веб-службы. Сначала пользователь веб-службы отыскивает нужную веб-службу либо путем ввода URL-адреса этой веб-службы, либо с помощью UDDI-сервера, либо с помощью DISCO-файла. Далее клиент извлекает WSDL-документ веб-службы, в котором описывается, как следует взаимодействовать с этой службой. Обе эти задачи выполняются во время проектирования. Когда вы запускаете приложение и начинаете фактически взаимодействовать с веб-службой, клиент отсылает SOAP-сообщение для инициализации соответствующего веб-метода.

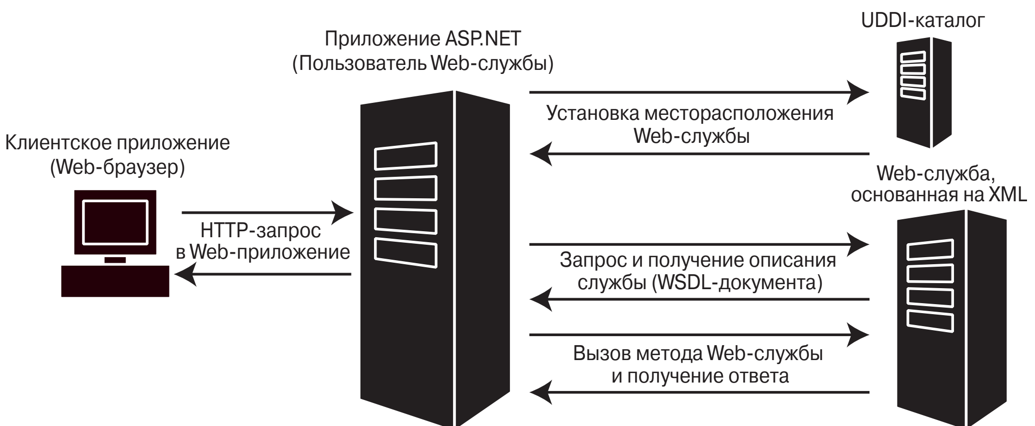


Рис. 35.2. Жизненный цикл веб-службы

Создание базовой веб-службы

В следующих разделах будет показано, как можно создать веб-службу с помощью ASP.NET и протестировать ее в браузере. Мы создадим веб-службу под названием EmployeesService, которая будет подсчитывать и возвращать количество сотрудников в определенном городе (или общее количество сотрудников) путем отправки запросов в таблицу Employees (Сотрудники), находящуюся в поставляемой в качестве образца базе данных SQL Server под названием Northwind.

Класс веб-службы

Создание веб-службы начинается с создания не поддерживающего состояний класса и одного или более методов. Эти методы как раз и будут вызывать удаленные клиенты, чтобы запустить ваш код.

Задачей службы EmployeesService является позволять удаленным клиентам извлекать информацию о сотрудниках компании. Она предоставляет метод GetEmployees(), который возвращает объект DataSet с полным набором данных о сотрудниках. Также она предлагает метод GetEmployeesCount(), который просто возвращает значение общего количества сотрудников в базе данных. Чтобы предоставить эти методы, необходим всего лишь базовый код ADO.NET. Вот как выглядит полный листинг класса.

```
public class EmployeesService
{
 private string connectionString;
 public EmployeesService()
 {
  string connectionString =
    WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
 }
 public int GetEmployeesCount()
 {
  SqlConnection con = new SqlConnection(connectionString);
  string sql = "SELECT COUNT(*) FROM Employees";
  SqlCommand cmd = new SqlCommand(sql, con);
  // Открываем соединение и извлекаем значения.
  con.Open();
  int numEmployees = -1;
  try
  {
       numEmployees = (int)cmd.ExecuteScalar();
  }
  finally
  {
       con.Close();
  }
  return numEmployees;
 }
 public DataSet GetEmployees()
 {
  //Создаем команды и соединения.
  string sql = "SELECT EmployeeID, LastName, FirstName, Title, " +
       "TitleOfCourtesy, HomePhone FROM Employees";
  SqlConnection con = new SqlConnection(connectionString);
  SqlDataAdapter da = new SqlDataAdapter(sql, con);
  DataSet ds = new DataSet();
  // Заполняем набор данных.
```

```
    da.Fill(ds, "Employees");
    return ds;
  }
}
```

`GetEmployess()` и `GetEmployessCount()` являются общедоступными методами, а это означает, что данный класс можно использовать с любой веб-страницы в одном и том же проекте. (Или, если он компилируется в отдельную DLL-библиотеку, его можно использовать в любом проекте, который ссылается на этот блок.) Однако чтобы сделать этот класс готовым для веб-службы, потребуется выполнить еще несколько дополнительных шагов.

Требования веб-службы

Прежде чем преобразовывать класс в веб-службу, следует убедиться в том, что он совместим с требованиями лежащих в основе XML-стандартов (о которых более подробно будет рассказываться в следующей главе). Поскольку веб-службы выполняются ASP.NET, который поддерживает CLR (Common Language Runtime — общеязыковая исполняющая среда), вы можете использовать любой действительный .NET-код. Это означает, что вы можете применять .NET-классы для получения доступа к данным с помощью ADO. NET или проверять достоверность данных с помощью регулярных выражений. Как и в случае с любым прикладным бизнес-компонентом, вы не можете отображать никакого пользовательского интерфейса, кроме того, на который ваш код не имеет никаких ограничений.

Однако ограничения касательно того, какую информацию ваш код может принимать (в виде параметров) и возвращать (в виде возвращаемого значения) все-таки существуют. И связано это с тем, что веб-службы разрабатываются на базе основанных на XML стандартов для обмена данными. А это значит, что набор типов данных, которые могут использовать веб-службы, ограничивается набором типов данных, которые распознаются стандартом XML Schema. То есть вы можете использовать простые типы данных, такие как строки и числа, но не имеете никакого автоматического способа для отправки патентованных .NET-объектов вроде `FileStream` (Файловый поток), `Image` (Изображение) или `EventLog` (Журнал событий). В этом ограничении есть смысл. Очевидно, что другие языки программирования не имеют возможностей интерпретировать такие более сложные классы, поэтому даже если бы вы изобрели какой-нибудь способ для их отправки, не исключено, что клиент просто не смог бы распознать их, что стало бы угрозой функциональной совместимости. (Доступный в .NET компонент Remoting (Удаленный доступ) является примером технологии распределенных объектов, которая *позволяет* использовать специфические типы данных .NET. Однако его недостаток состоит в том, что не .NET-клиентов он поддерживать не будет.)

Все поддерживаемые веб-службами типы данных перечислены в табл. 35.3.

На заметку! В основе типов данных, поддерживаемых веб-службами, лежат типы, определенные стандартом XML Schema. Эти типы также достаточно хорошо отображаются на основной набор типов данных языка C#.

Класс `EmployeesServices` следует всем этим правилам. Единственными типами данных, которые он использует для параметров и возвращаемых значений, являются `int` и `DataSet`, оба из которых поддерживаются. Конечно, некоторые программисты веб-служб предпочитают избегать использования объекта `DataSet` (подробнее об этом далее во врезке "Объект `DataSet` и веб-службы"), но это все равно разумный, широко применяемый подход.

Таблица 35.3. Поддерживаемые веб-службами типы данных для параметров и возвращаемых значений

Тип данных	Описание
Простые типы	Простые типы данных языка C#, такие как целочисленные (короткое целое, целое, длинное целое), целочисленные без знака (короткое целое без знака, целое без знака, длинное целое без знака), нецелочисленные числовые типы (числа с плавающей запятой, числа с плавающей запятой двойной точности, десятичные числа) и некоторые другие смешанные типы (булевский, строковый, символьный, байтовый и тип даты-времени).
Массивы	Вы можете использовать массивы любого поддерживаемого типа. Вы также можете использовать список массивов (`ArrayList`), который просто преобразуется в массив, но вы не можете применять более специализированные коллекции, такие как хеш-таблица (`HashTable`). Вы также можете использовать двоичные данные посредством байтовых массивов. Двоичные данные автоматически кодируются в Base64 так, чтобы их можно было вставить в сообщение веб-службы XML.
Специальные объекты	Вы можете передавать любой объект, который создаете на основе специального класса или структуры. Единственное ограничение состоит в том, что передаются только общедоступные элементы данных и что все элементы и свойства должны использовать один из других поддерживаемых типов данных. То есть, если вы используете класс, который включает специальные методы, эти методы не будут передаваться клиенту и не будут доступны для него.
Перечисления	Типы перечислений (определяемые в C# с помощью ключевого слова `enum`) тоже поддерживаются. Однако веб-служба использует строковое имя значения перечисления (а не лежащее в основе целое число).
`XmlNode`	Объекты, основанные на `System.Xml.XmlNode`, представляют собой часть XML-документа. Вы можете использовать их для отправки произвольных XML-данных.
`DataSet` и `DataTable`	Вы можете использовать объекты `DataSet` и `DataTable` для возврата информации из реляционной базы данных. Другие объекты ADO.NET, такие как `DataColumns` и `DataRows`, не поддерживаются. При использовании объекта `DataSet` или `DataTable` он автоматически преобразуется в XML точно так же, как при применении метода `GetXml()` или `WriteXml()`.

Еще одно требование заключается в том, что веб-службы не должны поддерживать состояния. На самом деле, архитектура веб-службы работает точно так же, как архитектура веб-страницы: в начале запроса создается новый объект веб-службы; этот объект уничтожается сразу же после того, как этот запрос будет обработан и получен ответ. Класс `EmployeesServices` прекрасно соответствует этой модели, потому что он не сохраняет никаких состояний в переменных своих членов. Единственным исключением является переменная `connectionString`, которая инициализируется с подходящим значением всякий раз, когда создается этот класс.

Объект DataSet и веб-службы

Вы заметите, что объект `DataSet` — это один из немногих специализированных классов .NET, который поддерживается веб-службами. А поддерживается он по той причине, что он может автоматически преобразовываться в XML. Однако здесь есть одно "но": несмотря на то, что не .NET-клиенты могут использовать веб-службу, которая возвращает объект `DataSet`, они могут не уметь делать ничего полезного с XML-объектом `DataSet`. Вот почему другие языки не будут автоматически преобразовывать объект `DataSet` в поддающиеся управлению объекты. Вместо этого они будут вынуждены использовать свои собственные поддерживающие XML API-интерфейсы программирования. Хотя в теории эти интерфейсы работают, на практике они могут делать это очень медленно, особенно когда речь идет о сложных, патентованных XML-данных. Именно поэтому разработчики, как правило, стараются избегать применения объекта `DataSet` при создании веб-служб, которые должны поддерживать клиентов на платформах самых разнообразных типов.

Стоит обратить внимание на то, что компания Microsoft могла мы использовать подход, примененный ею к классу `DataSet`, и по отношению ко многим другим классам .NET, чтобы сделать возможным их преобразование в XML. Однако она поступила разумно и не стала добавлять эти функциональные возможности, поняв, что это позволило бы программистам создавать приложения, которые бы использовали стандарты веб-служб, но были бы непригодными для применения в сценариях с платформами разных типов. (Еще не так давно в Microsoft с удовольствием ухватились бы за такую идею, рассчитывая на возможность расширения этих функциональных возможностей в будущем, но, к счастью, они осознали необходимость стремиться к интеграции и широкой совместимости приложений.)

Однако все это никак не объясняет, почему в Microsoft решили поддерживать класс `DataSet` в своем наборе инструментальных средств веб-служб. Причина этого состоит в том, что `DataSet` делает возможным выполнение одной из наиболее часто необходимых задач, а именно — возврат копии информации из реляционной базы данных. Преимущество добавления этой функциональной возможности, похоже, оправдало стоимость потенциальных проблем с совместимостью для разработчиков, которые недостаточно тщательно продумывают архитектуру своих веб-служб.

Поддержка обобщенных типов

Веб-службы поддерживают обобщенные типы. Однако эта поддержка, возможно, не совсем такая, какую вы ожидаете.

Вполне допустимо создать метод веб-службы, который принимает или возвращает обобщенный тип. Например, при желании возвратить коллекцию объектов `EmployeeDetails`, вы могли бы воспользоваться классом обобщенного типа `List`, как показано ниже:

```
public List<EmployeeDetails> GetEmployees()
{ ... }
```

В данном случае .NET интерпретирует коллекцию объектов `EmployeeDetails` точно так же, как массив объектов `EmployeeDetails`:

```
public EmployeeDetails[] GetEmployees()
{ ... }
```

Конечно, чтобы этот код работал, в классе `EmployeeDetails` или классе `List` не должно быть ничего, что нарушает правила сериализации. Например, в случае наличия в этих классах какого-нибудь не поддающегося сериализации свойства, весь объект не может быть сериализован.

Причина, по которой .NET поддерживает обобщенные типы в этом примере, заключается в том, что для .NET не представляет особой сложности определить настоящие

типы классов во время компиляции. Это позволяет .NET определять структуру XML-сообщений, которую будет использовать данный метод, и добавлять информацию в документ WSDL (как вы увидите в следующей главе).

Однако методы обобщенного типа .NET не поддерживает. Например, следующий метод использовать не разрешено:

```
public List<T> GetEmployees<T>()
{ ... }
```

Здесь метод `GetEmployees()` сам по себе является общим. Это позволяет вызывающей программе выбирать тип, который будет использоваться методом. Поскольку теоретически этот метод может применяться с абсолютно любым типом документа, способа заранее документировать его должным образом и определить соответствующий формат XML-сообщения не существует.

Отображение веб-службы

Теперь, когда вы убедились в том, что класс `EmployeesService` готов для Web, пора преобразовывать его в веб-службу. Первое, что вы должны сделать — это добавить атрибут `System.Web.Services.WebMethod` в каждый метод, который хотите представить как часть своей веб-службы. Это заставит ASP.NET сделать этот метод доступным для проверки и удаленного вызова.

Ниже показана измененная версия класса `EmployeesService` с двумя веб-методами.

```
public class EmployeesService
{
 [WebMethod()]
 public int GetEmployeesCount()
 { ... }
 [WebMethod()]
 public DataSet GetEmployees()
 { ... }
}
```

Эти два простых изменения завершают преобразование класса в веб-службу. Однако у клиентов по-прежнему нет точки входа в данную веб-службу: другими словами, у приложений нет возможности запускать данные веб-методы. Чтобы предоставить им такую возможность, вам необходимо создать файл `.asmx`, который будет отображать веб-службу.

На заметку! В этом примере веб-служба содержит код доступа к данным. Однако если вы планируете использовать этот же самый код в веб-приложении, вам не помешает добавить дополнительный уровень для компонентов баз данных. Чтобы реализовать этот проект, вы сначала бы создали отдельный компонент базы данных (как описывалось в части 2 книги), а затем использовали бы его непосредственно в своих веб-страницах и в своей веб-службе.

ASP.NET реализует веб-службы в виде файлов с расширением `.asmx`. Как и в случае с веб-страницей, код для веб-службы может помещаться как прямо в `.asmx`-файл, так и в файл отделенного кода, на который ссылается данный `.asmx`-файл (последний подход применяется в Visual Studio).

Например, вы могли бы создать файл с именем `EmployeeService.asmx` и связать его с вашим классом `EmployeesService`. Каждый `.asmx`-файл начинается с директивы `WebService`, которая объявляет серверный язык, используемый в файле и классе. Она также по выбору может объявлять и другую информацию, такую как имя лежащего

в основе кода файла, и то, должны ли во время компиляции генерироваться символы отладки. В этом отношении она похожа на директиву Page, которая применяется для .aspx-файлов.

Ниже показан пример .asmx-файла для EmployeesService:

```
<%@ WebService Language="C#" Class="EmployeesService" %>
```

В данном случае у вас есть два варианта. Вы можете вставить код класса сразу же после атрибута WebService или скомпилировать его в один из блоков в каталоге Bin. Если вы добавили класс EmployeesService в проект Visual Studio, он будет автоматически скомпилирован в виде части DLL-библиотеки веб-приложения, поэтому вам больше ничего не нужно включать в .asmx-файл.

После этого данный этап можно считать завершенным. Веб-служба создана, доступна и готова к использованию в других приложениях.

Совет. Никаких ограничений по поводу того, сколько веб-служб можно добавлять в одно веб-приложение, не существует, а также вы свободно можете смешивать веб-службы и веб-страницы.

веб-службы в Visual Studio

Если вы используете Visual Studio, то наверняка не будете создавать класс, преобразовывать его в веб-службу и затем добавлять .asmx-файл. Вместо этого вы создадите .asmx-файл и лежащий в основе кода файл одним действием, выбрав в меню Website (веб-сайт) команду Add New Item (Добавить новый элемент). Точно так же, как и при создании веб-страницы, вы сможете выбрать, куда поместить код веб-службы — непосредственно в .asmx-файл или же в отдельный файл отделенного кода.

Вы не знаете еще двух вещей о веб-службах. Во-первых, класс веб-службы наследуется от класса System.Web.Services.WebService, а, во-вторых, к объявлению класса применяется атрибут WebService. Ни одно из этих требований не является обязательным, но вы увидите, какую роль они играют, в следующих разделах.

Порождение класса веб-службы от класса WebService

Когда веб-служба создается в Visual Studio, класс этой веб-службы автоматически порождается от базового класса WebService, как показано ниже:

```
public class EmployeesService : System.Web.Services.WebService
{ ... }
```

Наследование от класса WebService — это удобная возможность, которая позволяет получать доступ к встроенным объектам ASP.NET (вроде Application, Session и User) так же легко, как и в веб-форме. Эти объекты предоставляются в виде свойств класса WebService, которые ваша веб-служба приобретает путем наследования. Если вам не нужны эти объекты (или если вы предпочитаете получать к ним доступ с помощью статического свойства HttpContext.Current), вы можете не наследовать класс своей веб-службы от класса WebService.

Ниже показан фрагмент кода, демонстрирующий, как вы могли бы получить доступ к состоянию Application в веб-службе, если бы унаследовали ее класс от базового класса WebService:

```
// Сохраняем число в состоянии сеанса.
Sesion["Counter"] = 10;
```

Ниже показан фрагмент кода, который вам следовало бы использовать в таком же случае, если вы не наследовали класс своей веб-службы от класса WebService:

```
// Сохраняем число в состоянии сеанса.
HttpContext.Current.Session["Counter"] = 10;
```

Эта технология на самом деле не будет работать так, как нужно (другими словами, клиент не будет сохранять одно и то же состояние при многочисленных вызовах веб-методов), до тех пор, пока не будут выполнены некоторые дополнительные шаги, как описывается далее в этой главе, в разделе "Свойство EnableSession".

В табл. 35.4 перечислены свойства, которые вы получаете при наследовании от класса WebService.

Таблица 35.4. Свойства класса WebService

Свойство	Описание
Application	Экземпляр класса HttpApplicationState, который обеспечивает доступ к глобальному состоянию веб-приложения.
Context	Экземпляр класса HttpContext для текущего запроса.
Server	Экземпляр класса HttpServerUtility.
Session	Экземпляр класса HttpSessionState, который предоставляет доступ к текущему состоянию сеанса.
User	Объект IPrincipal, который позволяет изучать данные удостоверения и роли пользователя, если пользователь прошел аутентификацию.

Поскольку в .NET Framework поддерживается только одиночное наследование, наследование класса веб-службы от класса WebService означает, что он не сможет наследоваться от других классов. Это, по сути, единственная причина, по которой может оказаться лучше не наследовать класс веб-службы от класса WebService.

На заметку! Интересным моментом в наследовании от класса WebService является то, что класс WebService порождается от класса System.MarshalByRefObject. Последний применяется в .NET-технологии, известной под названием Remoting (Удаленный доступ). А это значит, что, когда вы создаете класс, который порождается от класса WebService, вы получаете возможность использовать этот класс несколькими способами. Вы можете применять его как любой другой локальный класс (и получать доступ к нему прямо на своих веб-страницах), вы можете представить его как часть веб-службы или отобразить его в виде распределенного объекта на главном компьютере, на котором установлен .NET-компонент Remoting. Более подробную информацию об этом компоненте можно найти в книге *Advanced .NET Remoting, Second Edition* (Apress, 2005 г.).

Документирование веб-службы

веб-службы имеют самоописательную природу; это означает, что ASP.NET автоматически предоставляет всю необходимую клиенту информацию о доступных методах и параметрах, которые они требуют. Не обошлось здесь и без основанного на XML стандарта WSDL, о котором мы более подробно поговорим в следующей главе. Однако, хотя документ WSDL и описывает механику веб-службы, цель или значение предоставляемой в каждый метод и возвращаемой из него информации в нем никак не разъясняется. Большая часть веб-служб будет предоставлять такую информацию в отдельных документах разработчика. Тем не менее, вы можете (и должны) включать хотя бы основной минимум таких сведений вместе с веб-службой с помощью атрибутов WebMethod и WebService.

Вы можете добавить описания в каждый метод посредством свойства Description атрибута WebMethod или в веб-службу в целом с помощью свойства Description атрибута WebService. Вы также можете применить к веб-службе описательное имя, воспользовавшись свойством Name атрибута WebService. Ниже показан пример того, как вы могли бы вставить такую информацию в службу EmployeesService.

```
[WebService(Name="Employees Service",
Description="Retrieve the Northwind Employees")]
public class EmployeesService : System.Web.Services.WebService
{
 [WebMethod(Description="Returns the total number of employees.")]
 public int GetEmployeesCount()
 { ... }
 [WebMethod(
 Description="Returns the full list of employees.")]
 public DataSet GetEmployees()
 { ... }
}
```

Эти специальные описания добавляются в документ WSDL, который описывает вашу службу. Они также отображаются на автоматически генерируемой тестовой странице, которую вы будете использовать в следующем разделе.

Вы также должны предоставить еще одну деталь для вашей веб-службы — уникальное пространство имен. Это позволит вашей веб-службе (и генерируемым ею XML-сообщениям) уникально идентифицироваться. О пространствах имен XML мы уже рассказывали в главе 14. По умолчанию веб-службы, создаваемые в ASP.NET, используют стандартное для XML пространство имен http://tempuri.org, которое подходит только для целей тестирования. Если вы не укажете специальное пространство имен, при отображении тестовой страницы появится сообщение, предупреждающее о том, что лучше использовать какое-то более конкретное пространство имен. Обратите внимание на то, что пространство имен XML не имеет ничего общего с концепций пространств имен в .NET. Оно никак не влияет на работу кода или способ, которым клиент использует веб-службу. Вместо этого пространство имен XML просто идентифицирует веб-службу. Пространства имен XML обычно выглядят как URL-адреса. Однако они вовсе не обязательно соответствуют какому-нибудь действительному расположению в Интернете.

В идеале пространство имен, которое вы указываете, должно ссылаться на используемый вами URL-адрес. В большинстве случаев это будет означать, что вы должны включить в указываемое пространство имен имя Интернет-домена вашей компании. Например, если ваша компания использует веб-сайт http://www.mycompany.com, вы могли бы указать для веб-службы Employees Service пространство имен http://www.mycompany.com/EmployeesService.

Пространство имен указывается с помощью атрибута WebService:

```
[WebService (Name="Employees Service",
 Description="Retrieve the Northwind Employees",
 Namespace="http://www.apress.com/ProASP.NET/")]
public class EmployeesService : System.Web.Services.WebService
{ ... }
```

Тестирование веб-службы

Теперь, когда вы увидели, как создается простая веб-служба, вы готовы к ее тестированию. К счастью, создавать клиентское приложение, чтобы протестировать веб-службу, не нужно, потому что .NET включает тестовую веб-страницу, которую ASP.NET отображает автоматически, когда вы запрашиваете URL-адрес .asmx-файла в браузере.

Эта страница использует рефлексию для считывания и отображения информации о веб-службе, такой как имена методов, которые она предоставляет.

Чтобы открыть страницу тестирования, запросите в браузере файл `EmployeesService.asmx`. (В Visual Studio вам необходимо будет просто установить эту страницу как начальную и затем запустить ее.) Страница тестирования, которую вы увидите, показана на рис. 35.3.

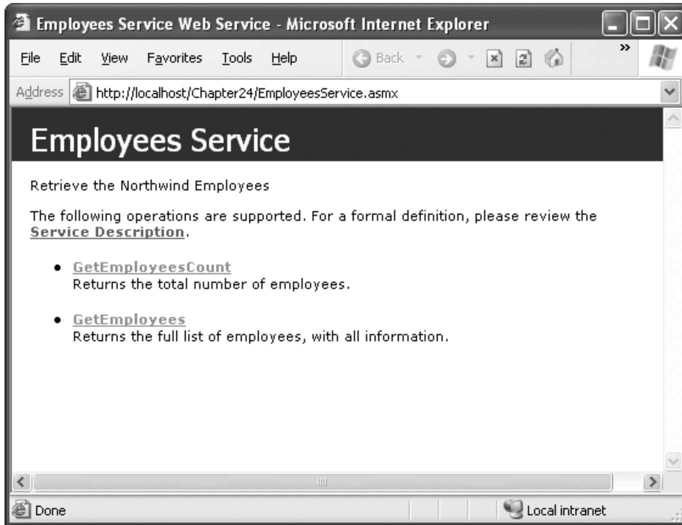


Рис. 35.3. Страница тестирования веб-службы

Обратите внимание, что на этой странице отображаются два веб-метода с описанием, а заголовком страницы служит наименование веб-службы. Если вы щелкнете на одном из методов, появится страница, позволяющая протестировать этот метод (и ввести данные для любого из его параметров). На рис. 35.4 показана страница, позволяющая протестировать метод `GetEmployeesCount()`.

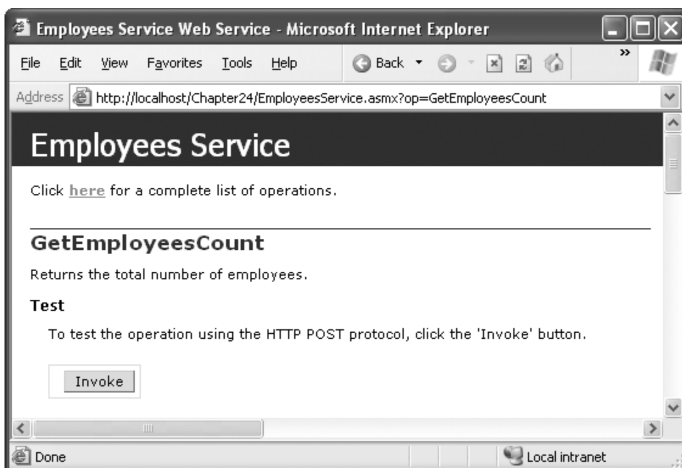


Рис. 35.4. Тестирование веб-метода

При щелчке на кнопке Invoke (Вызвать) появится новая веб-страница, на которой будет отображаться XML-документ, содержащий запрошенные данные. Посмотрев на рис. 35.5, вы увидите, что общее количество записей о сотрудниках составляет 9. Если вы посмотрите на URL-адрес, вы увидите, что он включает имя `.asmx`-файла, за которым следует имя вызывавшегося метода веб-службы.

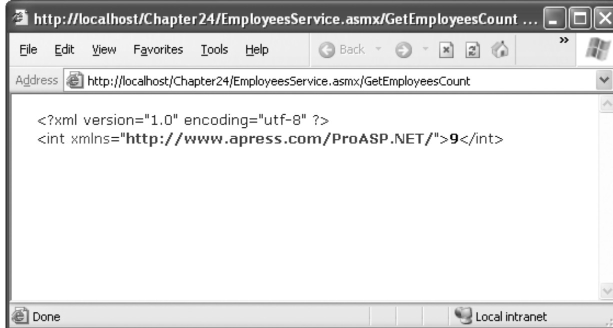


Рис. 35.5. Результаты для метода `GetEmployeesCount()`

Вы можете повторить описанные действия и вызвать метод `GetEmployees()`. В этом случае вы увидите намного более подробный XML-документ, полностью отображающий содержимое возвращаемого набора данных (`DataSet`), как показано на рис. 35.6.

Нетрудно заметить, что благодаря этой вспомогательной странице, тестирование базовой веб-службы представляет собой достаточно простой процесс и не требует создания клиента.

Рис. 35.6. Результаты для метода `GetEmployees()`

Страницы тестирования не являются частью стандартов веб-служб; они — это просто дополнительная возможность, предоставляемая ASP.NET. На самом деле страница тестирования генерируется ASP.NET “на лету” с помощью веб-страницы `c:\[КаталогWindows]\Microsoft.NET\Framework\[Версия]\Config\DefaultWsdlHelpGenerator.aspx`. В некоторых случаях у вас может возникнуть желание изменить внешний вид или поведение этой страницы. Если это произойдет, просто скопируйте файл `DefaultWsdlHelpGenerator.aspx` в каталог своего веб-приложения, внесите в него необходимые изменения, а затем измените файл `web.config` этого приложения так, чтобы он указывал на новую страницу, добавив в него элемент `<wsdlHelpGenerator>`, как показано ниже:

```
<configuration>
 <system.web>
 <webServices>
 <wsdlHelpGenerator href="MyWsdlHelpGenerator.aspx"/>
 </webServices>
 <!-- Остальные параметры были опущены. -->
 </system.web>
</configuration>
```

Эта технология наиболее часто применяется для изменения внешнего вида страницы тестирования. Например, вы могли бы использовать эту технологию, чтобы заменить версию страницы, где отображается логотип компании или информация об авторских правах.

Использование веб-службы

Прежде чем клиент сможет использовать веб-службу, он должен быть способен создавать, отправлять, получать и понимать XML-сообщения. Этот процесс, в принципе, прост, но на практике достаточно утомителен. Если бы вы должны были реализовать его сами, вам пришлось бы писать один и тот же низкоуровневый инфраструктурный код снова и снова.

К счастью, .NET предоставляет решение в виде специального компонента, который называется *прокси-классом* (proxy class) и выполняет наиболее трудную часть работы для вашего приложения. Прокси-класс скрывает вызовы методов веб-службы. Он отвечает за генерацию SOAP-сообщений в корректном формате и управление сообщениями в сети (с помощью протокола HTTP). Когда он получает ответное сообщение, он еще и преобразует результаты обратно в соответствующие типы данных .NET.

На заметку! Чтобы к веб-службе можно было получить доступ с другого компьютера, эта веб-служба должна быть доступной. Это означает, что вы не можете полагаться только на встроенный веб-сервер Visual Studio (который динамически выбирает новый порт каждый раз, когда его запускают). Вместо этого вы должны создать виртуальный каталог для своей веб-службы (как описывалось в главе 18). Выполнив этот шаг, вы должны попытаться, используя имя виртуального каталога, запросить веб-службу в окне браузера, дабы убедиться в том, что она доступна. После этого вы можете добавить ссылку на веб-службу, выполнив шаги, описываемые далее в этом разделе.

На рис. 35.7 этот процесс показан графически. В демонстрируемом примере браузер запускает веб-страницу ASP.NET, которая использует веб-службу с другого находящегося где-то в другом месте сервера. Для установки связи с этой внешней веб-службой веб-страница ASP.NET использует прокси-класс.

Рис. 35.7. Прокси-класс веб-службы

На заметку! Благодаря прокси-классу, вызывать метод в веб-службе можно так же легко, как в локальном компоненте. Конечно, такое поведение не всегда приносит пользу. Веб-службы имеют характеристики, отличающиеся от локальных компонентов. Например, вызов веб-метода занимает неизвестное количество времени, поскольку каждый такой вызов должен преобразовываться в XML и пересылаться через сеть. Опасность состоит в том, что чем больше эта реальность скрывается от глаз разработчиков, тем менее вероятно, что они будут ее учитывать и проектировать свои приложения соответствующим образом.

Существуют два способа создать прокси-класс в .NET.

- Воспользоваться утилитой командной строки `wsdl.exe`.
- Воспользоваться предлагаемой в Visual Studio возможностью для добавления веб-ссылок.

Оба эти подхода, по сути, дают один и тот же результат, поскольку они предполагают использование одних и тех же классов в .NET Framework для выполнения фактической работы. На самом деле вы даже можете жестко закодировать эти классы (которые находятся в пространстве имен `System.Web.Services`), чтобы получить возможность генерировать свои собственные прокси-классы программным путем, хотя такой подход является не очень-то практичным.

В следующих разделах речь пойдет о том, как следует использовать `wsdl.exe` и Visual Studio, чтобы создать прокси-классы, а также о том, как веб-службу можно использовать в клиентах трех разных типов (на веб-странице ASP.NET, в Windows-приложении и на классической ASP-странице).

Совет. Одним отличием подхода, предполагающего применение утилиты `wsdl.exe`, от подхода, предполагающего использование функции для добавления веб-ссылок, является то, что в случае применения функции для добавления веб-ссылок увидеть фактический код прокси-класса будет невозможно (потому что он генерируется позже, во время компиляции). Это означает, что при желании иметь возможность корректировать код прокси-класса или просто просматривать его, когда создаете веб-клиент, вы должны использовать утилиту `wsdl.exe`. Это ограничение не распространяется на клиентов других типов. Они не используют модель компиляции ASP.NET, поэтому код прокси-класса добавляется прямо в проект.

Генерирование прокси-класса с помощью `wsdl.exe`

Утилита `wsdl.exe` берет веб-службу и генерирует исходный код прокси-класса либо в VB.NET, либо в C#. Название утилиты "WSDL" происходит от названия стандарта веб-служб (Web Services Description Language — язык описания веб-служб), который используется для описания предоставляемых веб-службой функциональных возможностей. Более подробно о стандарте WSDL будет рассказываться в следующей главе.

Файл `wsdl.exe` можно найти в каталоге .NET Framework, путь к которому обычно выглядит примерно так: `c:\Program Files\Microsoft Visual Studio 2005\SDK\v2.0\Bin` (что зависит от установленной версии Visual Studio). Этот файл представляет собой утилиту командной строки, поэтому с ним легче всего работать в окне командной строки.

В ASP.NET запросить документ WSDL можно, указав URL-адрес веб-службы плюс параметр `?WSDL`. (В качестве альтернативного варианта, можно также указать и другой URL-адрес или даже имя файла, содержащего WSDL-данные.) Минимальный синтаксис для генерации класса выглядит так:

```
wsdl http://localhost/WebServices1/EmployeesService.asmx
```

По умолчанию класс генерируется на языке C#, но это поведение можно изменить, добавив параметр `/language`, как показано ниже:

```
wsdl /language:VB http://localhost/WebServices1/EmployeesService.asmx
```

По умолчанию генерируемому файлу также присваивается такое же, как и у веб-службы, имя (которое указано в свойстве `Name` атрибута `WebService`). Вы можете изменить его, добавив в команду `wsdl.exe` параметр `/out`, а также можете воспользоваться параметром `/namespace` и изменить для генерируемого класса пространство имен, например, следующим образом (показанный ниже код был разбит на две строки, поскольку иначе он не умещался в колонку):

```
wsdl /namespace:ApressServices /out:EmployeesProxy.cs
http://localhost/WebServices1/EmployeesService.asmx
```

В табл. 35.5 перечислены все поддерживаемые параметры.

Таблица 35.5. Параметры `wsdl.exe`

Параметр	Описание
`<url или путь>`	URL-адрес или путь к контракту WSDL, схеме XSD или документу `.discomap`.
`/nologo`	Подавляет отображение баннера.
`/language:<язык>`	Язык, который должен использоваться для генерируемого прокси-класса. Выберите один из доступных языков (`CS`, `VB` или `JS`) или укажите полностью уточненное имя класса, реализуя `System.CodeDom.Compiler.CodeDomProvider`. По умолчанию будет использоваться язык C#. Сокращенная версия этого параметра выглядит как `/l`.
`/server`	Генерирует абстрактный класс для реализации веб-службы на основе контрактов. По умолчанию генерирует прокси-классы клиента.
`/namespace:<пространство имен>`	Пространство имя для генерируемого прокси или шаблона. По умолчанию используется глобальное пространство имен. Сокращенная версия этого параметра выглядит как `/n`.

Окончание табл. 35.5

Параметр	Описание
/out:<имя файла>	Имя файла для кода генерируемого прокси. По умолчанию это имя порождается от имени службы. Сокращенная версия этого параметра выглядит как /o.
/protocol:<протокол>	Позволяет перекрыть протокол, реализуемый по умолчанию. Допустимыми значениями являются: SOAP (для SOAP 1.1), SOAP12 (для SOAP 1.2), HTTP-GET, HTTP-POST или какой-то специальный протокол, указанный в конфигурационном файле.
/username:<имя пользователя> /password:<пароль> /domain:<домен>	Данные удостоверения, которые следует использовать при установке соединения с сервером, требующим прохождения аутентификации. Сокращенные версии этих параметров выглядят как /u, /p и /d.
/proxy:<URL-адрес>	URL-адрес прокси-сервера, который должен использоваться для HTTP-запросов. По умолчанию будут применяться системные настройки прокси.
/proxyusername:<имя пользователя> /proxypassword:<пароль> /proxydomain:<домен>	Данные удостоверения, которые следует использовать при подключении к прокси-серверу, требующему прохождения аутентификации. Сокращенные версии этих параметров выглядят как /pu, /pp и /pd.
/appsettingurkey:<ключ>	Конфигурационный ключ, который должен использоваться при генерации кода для считывания значения, присваиваемого свойству URL по умолчанию. По умолчанию считывание значения из конфигурационного файла выполняться не будет. Сокращенная версия этого параметра выглядит как /urlkey.
/appsettingbaseurl:<базовый URL-адрес>	Базовый URL-адрес, который должен использоваться при вычислении фрагмента URL-адреса. Также обязательно должен указываться и параметр appsettingurlkey. Фрагмент URL-адреса — это результат вычисления относительного URL-адреса из appsettingbaseurl и URL-адреса в документе WSDL. Сокращенная версия этого параметра выглядит как /baseurl.
/fields	Если установить этот параметр, все сложные типы, используемые веб-службой, будут состоять из общедоступных полей, а не общедоступных свойств. О том, как сложные типы работают с веб-службами, более подробно будет рассказываться в главе 36.
/sharetypes	Позволяет добавлять ссылку на две или более веб-служб, которые используют такие же сложные типы. Эта технология более подробно будет описываться в главе 36.
/serverinterface	Генерирует интерфейс только с теми методами, которые представлены в документе WSDL. Вы можете реализовать такой интерфейс для создания своей веб-службы. Эта технология будет подробно описываться в следующей главе.

Как только этот файл будет создан, его потребуется скопировать в каталог `App_Code` — это сделает его доступным для страниц в веб-приложении. В случае, когда создается многофункциональное клиентское приложение (такое как приложение Windows Forms), этот файл следует не копировать в каталог `App_Code`, а добавить прямо в проект, что гарантирует его компиляцию в окончательный EXE-файл.

Генерирование прокси-класса с помощью Visual Studio

В Visual Studio прокси-класс создается путем добавления в проект клиента соответствующей веб-ссылки. Веб-ссылки похожи на обычные ссылки, но вместо того, чтобы указывать на блоки с помощью обычных типов данных .NET, они указывают на URL-адрес веб-службы с помощью WSDL-контракта.

Чтобы создать веб-ссылку, выполните следующие действия.

1. Щелкните правой кнопкой мыши на имени нужного проекта клиента в окне проводника Solution Explorer и в появившемся контекстном меню выберите команду Add Web Reference (Добавить веб-ссылку).
2. Появится диалоговое окно Add Web Reference (Добавить веб-ссылку), как показано на рис. 35.8. Это диалоговое окно предлагает опции для поиска веб-служб и ввода URL-адреса, а также содержит ссылку, которая позволяет просмотреть веб-службы, имеющиеся на локальном компьютере, или выполнить поиск в UDDI-каталоге.

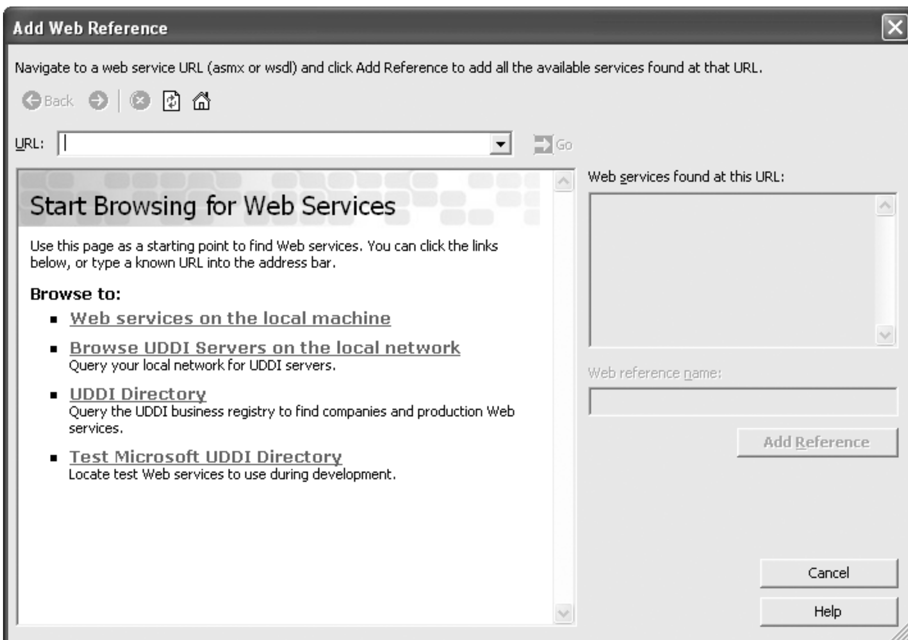


Рис. 35.8. Диалоговое окно Add Web Reference (Добавить веб-ссылку)

3. Чтобы сразу перейти к веб-службе, в поле URL (URL-адрес) введите URL-адрес, указывающий на соответствующий `.asmx`-файл. В окне появится тестовая страница (как показано на рис. 35.9), а кнопка Add Reference (Добавить ссылку) станет доступной.

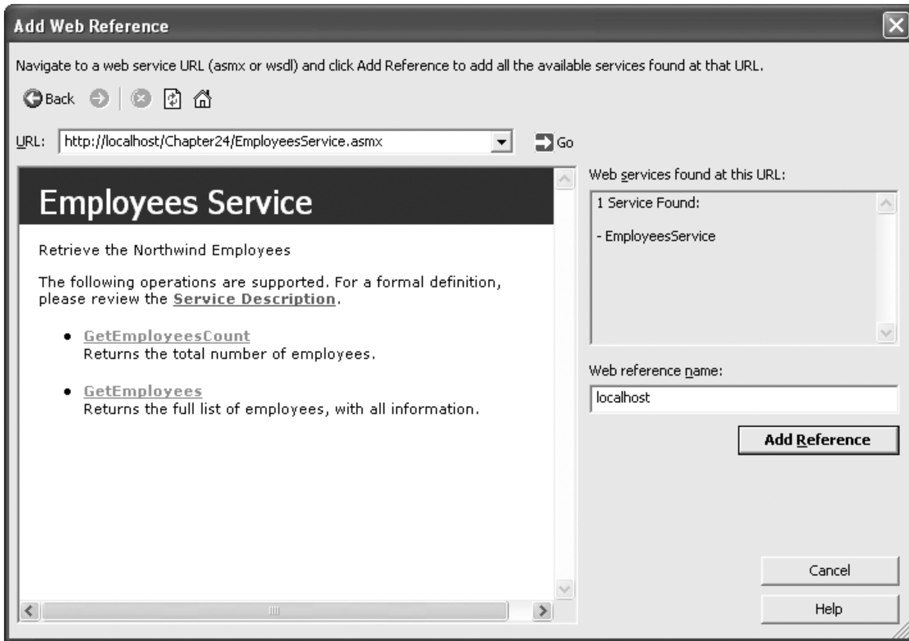


Рис. 35.9. Добавление веб-ссылки

4. Если хотите, укажите в поле Web Reference Name (Имя веб-ссылки) другое пространство имен, в котором должен быть сгенерирован прокси-класс.
5. Чтобы добавить ссылку на эту веб-службу, щелкните на кнопке Add Reference (Добавить ссылку) в нижней части окна.
6. После этого веб-ссылка появится в окне проводника Solution Explorer в разделе Web References (веб-ссылки) соответствующего проекта.

При создании веб-ссылки используют WSDL-контракт и ту информацию, которая существует на момент, когда они добавляются. В случае изменения веб-службы, прокси-класс придется обновить, щелкнув правой кнопкой мыши на веб-ссылке и в появившемся контекстном меню выбрав команду Update Web Reference (Обновить веб-ссылку). В отличие от локальных компонентов, веб-ссылки не обновляются автоматически при перекомпиляции приложения.

Совет. При разработке и тестировании веб-службы в Visual Studio зачастую легче всего добавить веб-службу и клиентское приложение в одно и то же решение. Это позволит тестировать и изменять оба этих компонента в одно и то же время. Вы даже сможете воспользоваться интегрированной программой отладки, чтобы установить контрольные точки и просматривать код как клиента, так и сервера, как будто бы они действительно являются одним приложением. Чтобы выбрать приложение, которое Visual Studio должен запускать при щелчке на кнопке Start (Пуск), щелкните правой кнопкой мыши на имени нужного проекта в окне проводника Solution Explorer и в появившемся контекстном меню выберите команду Set As StartUp Project (Установить в качестве стартового проекта).

При добавлении веб-ссылки Visual Studio сохраняет в проекте копию WSDL-документа. Где именно сохраняется эта информация, зависит от типа используемого проекта.

В веб-приложении Visual Studio создает папку App_WebReferences (если она еще не существует) и затем уже в ней создает папку с именем, соответствующим имени веб-ссылки (которое было указано в диалоговом окне Add Web Reference (Добавить веб-ссылку)). И, наконец, Visual Studio помещает в эту папку все файлы веб-службы. Однако прокси-класс Visual Studio *не* генерирует. Этот класс создается и помещается в кэш на одном из этапов процесса компиляции ASP.NET. Такое поведение является новым в версии ASP.NET 2.0 (если сравнивать с версиями ASP.NET 1.*x*).

В приложениях любого другого типа Visual Studio создает папку WebReferences, после чего создает в ней папку с именем, совпадающим с именем веб-ссылки. В эту папку он помещает все файлы поддержки, которые можно видеть в веб-приложении (наиболее важным из них является файл, содержащий копию WSDL-документа). Он также создает файл с именем Reference.cs (при условии, что речь идет о C#-приложении), который содержит исходный код прокси-класса, как показано на рис. 35.10. По умолчанию этот файл является скрытым. Чтобы увидеть его, необходимо выбрать в меню Project (Проект) команду Show All Files (Показать все файлы).

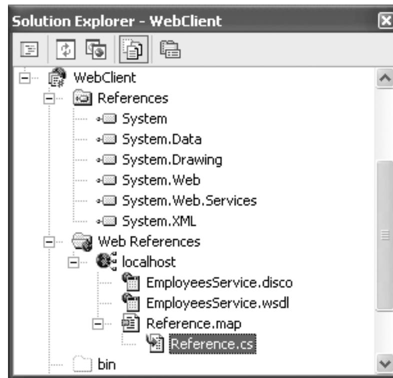


Рис. 35.10. Файл контракта WSDL и файл прокси-класса

Динамические URL-адреса

В предыдущих версиях .NET URL-адрес веб-службы (по умолчанию) жестко кодировался в конструкторе класса. Однако когда веб-ссылка создается с помощью Visual Studio 2005, данные о местонахождении веб-службы всегда сохраняются в конфигурационном файле. Это удобно, поскольку позволяет изменять месторасположение веб-службы во время разворачивания приложения без необходимости повторной генерации прокси-класса.

Точное месторасположение отвечающего за местонахождение веб-службы параметра зависит от типа приложения. Если клиентом является веб-приложение, эта информация добавляется в файл web.config, как показано ниже.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <appSettings>
  <add key="localhost.EmployeesService"
   value="http://localhost/WebServices1/EmployeesService.asmx"/>
 </appSettings>
 ...
</configuration>
```

Совет. Visual Studio несколько хитро именует конфигурационные файлы. В среде проектирования конфигурационный файл будет иметь имя `App.config`. Однако при сборке приложения этот файл будет скопирован в соответствующий каталог и получит другое имя (отвечающее требованиям исполняемого файла). Единственным исключением является случай, когда клиентское приложение представляет собой веб-приложение. Все веб-приложения используют конфигурационный файл с именем `web.config`, независимо от того, какие имена файлов используете вы. Это вы тоже увидите в среде проектирования.

При желании иметь возможность контролировать имя этого параметра, вам следует воспользоваться утилитой `wsdl.exe` и указать параметр `/appsettingurlkey`. Например, вы могли бы использовать такую командную строку:

```
wsdlhttp://localhost/WebServices1/EmployeesService.asmx /appsettingurlkey:WsUrl
```

В данном случае ключ будет сохранен с именем `WsUrl` в разделе `<appSettings>`.

Прокси-класс

После создания прокси-класса не помешает более внимательно изучить сгенерированный код, чтобы понять, как он будет работать.

Прокси-класс имеет то же имя, что и класс веб-службы. Он наследуется от класса `SoapHttpClientProtocol`, который имеет такие свойства, как `Credentials`, `Url` и `Timeout`, более подробно описываемые в следующих разделах. Вот как выглядит объявление прокси-класса, который обеспечивает связь с веб-службой `EmployeesService`:

```
public class EmployeesService :
 System.Web.Services.Protocols.SoapHttpClientProtocol
{ ... }
```

Прокси-класс содержит копию каждого метода веб-службы. Однако версия в прокси-классе не содержит бизнес-кода. (На самом деле, клиент не имеет никакой возможности извлекать какую-либо информацию о внутренней работе кода веб-службы: если бы она у него была, это была бы серьезная брешь в системе безопасности.) Вместо этого прокси-класс содержит код, необходимый для отправки запросов в удаленную веб-службу и преобразования результатов. Например, вот как выглядит в прокси-классе метод `GetEmployeesCount()`:

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute()]
public int GetEmployeesCount()
{
 object[] results = this.Invoke("GetEmployeesCount", new object[0]);
 return ((int)(results[0]));
}
```

Этот метод вызывает базовый метод `SoapHttpClientProtocol.Invoke()`, чтобы создать SOAP-сообщение и начать ожидать ответ. Вторая строка кода преобразует возвращаемый объект в целое число.

На заметку! У прокси также имеются и другие методы, которые поддерживают асинхронные вызовы веб-методов. Более подробную информацию об асинхронных вызовах и практические примеры их использования вы найдете в главе 37.

Прокси-класс завершает прокси-код для метода `GetEmployees()`. Вы заметите, что этот код отличается от кода, который используется для метода `GetEmployeesCount()`, только именем метода, которое передается в метод `Invoke()`, и тем, что возвращаемое значение преобразуется в набор данных (`DataSet`), а не в целое число.

```
[System.Web.Services.Protocols.SoapDocumentMethodAttribute()]
public System.Data.DataSet GetEmployees()
{
 object[] results = this.Invoke("GetEmployees", new object[0]);
 return ((System.Data.DataSet)(results[0]));
}
```

Создание ASP.NET-клиента

Теперь, когда у вас естьвеб-служба и прокси-класс, вы без особого труда можете создать простое клиентское приложение типа веб-страницы. Если вы используете Visual Studio, первый шаг — это создать новый веб-проект и добавить в него веб-ссылку на веб-службу. Если вы используете какое-то другое средство, вам сначала придется скомпилировать прокси-класс с помощью утилиты `wsdl.exe` и затем поместить его в каталог `Bin` нового веб-приложения.

В следующем примере создается простая веб-страница с кнопкой и элементом управления `GridView`. При щелчке на кнопке эта веб-страница отправляет запрос, создает прокси-класс, извлекает из веб-службы набор данных (`DataSet`) о сотрудниках и затем отображает результат путем привязывания этого набора к сетке.

Прежде чем добавлять этот код, не помешает импортировать пространство имен прокси-класса. В Visual Studio таким пространством имен автоматически становится пространство имен текущего проекта плюс пространство имен, указанное в диалоговом окне Add Web Reference (Добавить веб-ссылку) (в котором по умолчанию указывается пространство имен `localhost`). При условии, что проект называется `WebClient`, веб-служба находится на локальном компьютере и никакие изменения в окне Add Web Reference (Добавить веб-ссылку) не вносились, это пространство имен будет выглядеть так:

```
using WebClient.localhost;
```

Теперь мы можем добавить код, использующий прокси-класс для извлечения данных:

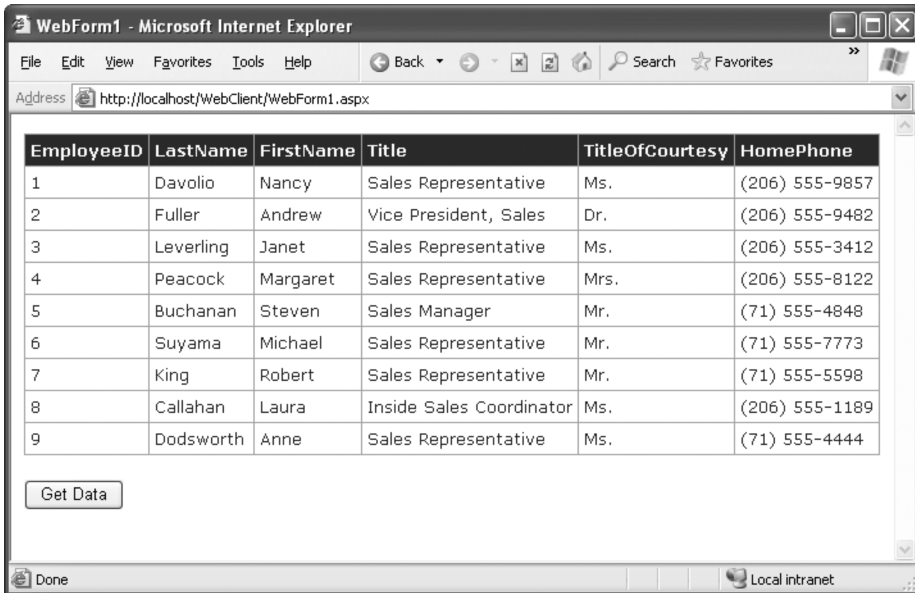
```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
  // Создаем прокси.
  EmployeesService proxy = new EmployeesService();
  // Вызываем веб-службу и извлекаем результаты.
  DataSet ds = proxy.GetEmployees();
  // Привязываем результаты.
  GridView1.DataSource = ds.Tables[0];
  GridView1.DataBind();
}
```

Поскольку прокси-класс имеет такое же имя, как и класс веб-службы, когда клиент создает экземпляр прокси-класса, создается впечатление, будто клиент на самом деле создает экземпляр веб-службы. Чтобы подчеркнуть отличие между ними, прокси присваивается имя объектной переменной.

Если запустить эту страницу, появится страница, показанная на рис. 35.11.

Интересно, что выполнять привязку данных вручную не нужно. Это можно сделать с помощью объекта `ObjectDataSource` (о котором подробно рассказывалось в главе 9), никакой специальный код не требуется:

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
 SelectMethod="GetEmployees" TypeName="localhost.EmployeesService" />
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="True"
 DataSourceID="ObjectDataSource1"/>
```



EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	HomePhone
1	Davolio	Nancy	Sales Representative	Ms.	(206) 555-9857
2	Fuller	Andrew	Vice President, Sales	Dr.	(206) 555-9482
3	Leverling	Janet	Sales Representative	Ms.	(206) 555-3412
4	Peacock	Margaret	Sales Representative	Mrs.	(206) 555-8122
5	Buchanan	Steven	Sales Manager	Mr.	(71) 555-4848
6	Suyama	Michael	Sales Representative	Mr.	(71) 555-7773
7	King	Robert	Sales Representative	Mr.	(71) 555-5598
8	Callahan	Laura	Inside Sales Coordinator	Ms.	(206) 555-1189
9	Dodsworth	Anne	Sales Representative	Ms.	(71) 555-4444

Рис. 35.11. Отображение данныхвеб-службы на веб-странице

С точки зрения кода веб-страницы, между вызовом веб-службы и использованием обычного не поддерживающего состояний класса нет никакой разницы. Однако вы должны помнить, что веб-служба, которая на самом деле реализует бизнес-логику, может находиться на веб-сервере на другом конце света. Поэтому вы должны сократить количество вызовов к ней и быть готовыми обрабатывать исключения, появляющиеся в результате возникновения проблем в сети или ошибок связи.

Тайм-ауты

Прокси-класс включает свойство `Timeout`, позволяющее указывать максимальное количество времени, которое может затрачиваться на ожидание ответа, в миллисекундах. По умолчанию его значение равняется 100 000 миллисекунд (10 секунд).

При использовании свойства `Timeout` в коде обязательно должна предусматриваться и обработка ошибок. В случаях, когда период тайм-аута истек, а ответ так и не был получен, будет выдаваться исключение, дающее вам шанс уведомить пользователя о проблеме.

Ниже показан код ASP.NET-клиента типа веб-страницы из предыдущего примера, в котором теперь также указывается, что период тайм-аута должен составлять 3 секунды:

```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
  // Создаем прокси.
  EmployeesService proxy = new EmployeesService();
  // Это значение тайм-аута будет применяться к вызовам
  // всех методов веб-службы.
  proxy.Timeout = 3000; // 3 000 миллисекунд - это 3 секунды.
  DataSet ds = null;
  try
  {
```

```
        // Вызываем веб-службу и извлекаем результаты.
        ds = proxy.GetEmployees();
    }
    catch (System.Net.WebException err)
    {
        if (err.Status == WebExceptionStatus.Timeout)
        {
            lblResult.Text = "Web service timed out after 3 seconds.";
        }
        else
        {
            lblResult.Text = "Another type of problem occurred.";
        }
    }

    // Привязываем результаты.
    if (ds != null)
    {
        GridView1.DataSource = ds.Tables[0];
        GridView1.DataBind();
    }
}
```

Свойству `Timeout` также может быть присвоено значение `-1`, указывающее, что период ожидания ответа может длиться столько, сколько потребуется. Однако это недопустимо замедлит работу веб-приложения при попытке выполнить ряд операций с веб-службой, которая не отвечает ни на какие запросы.

Подключение через прокси

Прокси-класс также имеет некоторую встроенную логику, которая позволяет изменять маршрут его HTTP-сообщений с помощью специальных параметров настройки Интернета. По умолчанию прокси-класс использует параметры настройки Интернета, установленные на текущем компьютере. В некоторых сетях это может оказаться не лучшим подходом. Вы можете перекрыть эти параметры настройки, воспользовавшись свойством `Proxy` прокси-класса веб-службы.

Совет. В данном случае слово "прокси" применяется в двух значениях: в значении прокси-класса, который отвечает за связь между клиентом и веб-службой, и в значении прокси-сервера в вашей организации, который отвечает за связь между компьютером и Интернетом.

Например, при необходимости подключаться через компьютер с именем `ProxyServer` и порт 80, вы могли бы использовать следующий код, прежде чем вызвать какие-либо методы веб-службы:

```
// Создаем прокси веб-службы.
EmployeesService proxy = new EmployeesService();
// Указываем прокси-сервер для сетевой связи.
WebProxy connectionProxy = new WebProxy("ProxyServer", 80);
proxy.Proxy = connectionProxy;
```

У класса `WebProxy` есть много других опций, которые позволяют конфигурировать соединения и указывать данные для аутентификации в более сложных сценариях.

Создание клиентского приложения Windows Forms

Одним из главных преимуществ веб-служб является способ, которым они позволяют делать локальные приложения, такие как многофункциональные клиентские при-

ложения, пригодными для работы в Web. Используя веб-службу, вы можете создать настольное приложение, которое будет извлекать с веб-сервера самые последние данные. Этот процесс является почти полностью прозрачным. На самом деле, поскольку высокоскоростной доступ становится все более распространенным явлением, вы можете даже не знать, какие элементы функциональных возможностей зависят от Интернета, а какие — нет.

Функциональные возможности веб-служб могут быть использованы в Windows-приложении точно так же, как и в ASP.NET-приложении. Сначала с помощью Visual Studio или утилиты `wsdl.exe` создается прокси-класс. Затем добавляется код для создания экземпляра прокси-класса и вызова веб-метода. Единственным отличием является интерфейс, который использует такое приложение.

Если вам до этого не доводилось создавать настольные приложения в .NET, вас обрадует известие о том, что вы здесь сможете применить большую часть знаний, которые получили во время разработки ASP.NET-приложений. Многие веб-элементы управления (такие как метки, кнопки, текстовые поля и списки) очень напоминают свои .NET-эквиваленты для настольных приложений, и код, который пишется для взаимодействия с ними, чаще всего может быть перенесен из одной среды в другую со всего лишь несколькими изменениями. На самом деле самым главным отличием между программированием настольных приложений и программированием веб-приложений являются дополнительные шаги, которые приходится выполнять в веб-приложениях для обеспечения сохранности информации после ее отправки на сервер и при переходе пользователя с одной страницы на другую.

Чтобы начать создавать клиентское Windows-приложение в Visual Studio, первым делом создайте новый проект Windows-приложения и затем добавьте в него веб-ссылку. Все веб-проекты начинаются с одной активизируемой при запуске формы, проектирование которой выполняется во многом подобно проектированию веб-страницы. Например, для рассматриваемого нами случая необходимо просто перетащить в эту форму с панели Toolbox (Панель инструментов) элементы управления `Button` (Кнопка) и `DataGridView` (Сетка данных).

Далее импортируйте необходимое пространство имен в начало файла класса формы, как делали это при создании ASP.NET-страницы:

```
using WindowsClient.localhost;
```

Затем добавьте код, обрабатывающий события для кнопки. Этот код будет извлекать набор данных (`DataSet`) и отображать его на форме. Код привязки данных для Windows-приложения выглядит несколько иначе: например, вызывать явный метод `DataBind()` после указания источника данных не требуется. Также этот код содержит одно уточнение: он явно указывает приложению использовать курсор в форме песочных часов, пока обрабатывается вызов веб-службы так, чтобы пользователь знал, что операция находится на стадии выполнения. Во всем остальном — все точно так же.

```
private void cmdGetData_Click(object sender, System.EventArgs e)
{
 this.Cursor = Cursors.WaitCursor;
 // Создаем прокси.
 EmployeesService proxy = new EmployeesService();
 // Вызываем веб-службу и извлекаем результаты.
 DataSet ds = proxy.GetEmployees();
 // Привязываем результаты.
 dataGridView1.DataSource = ds.Tables[0];
 this.Cursor = Cursors.Default;
}
```

На рис. 35.12 показаны результаты, которые вы увидите, когда запустите это клиентское Windows-приложение и щелкнете на кнопке, чтобы извлечь данные веб-службы.

EmployeeID	LastName	FirstName	Title	TitleOfCourtes	HomePhone
1	Davolio	Nancy	Sales Represe	Ms.	(206) 555-98
2	Fuller	Andrew	Vice Presiden	Dr.	(206) 555-94
3	Leverling	Janet	Sales Represe	Ms.	(206) 555-34
4	Peacock	Margaret	Sales Represe	Mrs.	(206) 555-81
5	Buchanan	Steven	Sales Manage	Mr.	(71) 555-484
6	Suyama	Michael	Sales Represe	Mr.	(71) 555-777
7	King	Robert	Sales Represe	Mr.	(71) 555-559
8	Callahan	Laura	Inside Sales C	Ms.	(206) 555-11
9	Dodsworth	Anne	Sales Represe	Ms.	(71) 555-444

Рис. 35.12. Отображение данных веб-службы в Windows-форме

Конечно, при разработке Windows-приложений доступно и множество других возможностей, которые подробно описываются во многих других хороших книгах. Наиболее интересным для вас может оказаться тот факт, что клиентское Windows-приложение может взаимодействовать с веб-службой точно так же, как и ASP.NET-приложение. Это открывает множество новых возможностей для интегрируемых Windows- и веб-приложений. Например, вы могли бы расширить данное Windows-приложение так, чтобы оно позволяло пользователю вносить изменения в данные о сотрудниках. Затем вы могли бы добавить в веб-службу `EmployeesService` методы, позволяющие клиенту отправлять измененные данные и применять изменения в конечной базе данных.

Важно понимать, что то, что вы делаете, чтобы использовать свой пробный образец веб-службы — это в точности то же самое, что бы вы делали, чтобы использовать веб-службу любого другого стороннего производителя. Поставщикам веб-служб нет необходимости распространять свои прокси-классы, потому что платформы для программирования, подобные .NET, включают средства, генерирующие их автоматически.

Совет. При желании попробовать использовать какие-нибудь веб-службы, отличные от .NET, вы можете воспользоваться каталогом веб-служб, который доступен на веб-сайте `http://www.xmethods.com`. При желании попрактиковаться с какой-нибудь действительно полезной веб-службой, попробуйте предоставляемую Microsoft веб-службу `MapPoint` (`http://msdn.microsoft.com/library/en-us/dnanchor/html/anch_mappointmain.asp`), которая позволяет получать доступ к картам и другим географическим данным высокого качества. Microsoft также предоставляет и такую интересную веб-службу, как `TerraService` (`http://terraservice.net/webservices.aspx`), которая основана на очень популярном сайте под названием `TerraServer`, где пользователи могут просматривать топографические карты и сделанные с помощью спутника фотографии земного шара. Пользуясь этой службой, вы можете запрашивать информацию о самых различных местах земного шара и даже загружать фрагменты сделанных со спутника фотографий конкретных регионов.

Создание ASP-клиента с помощью MSXML

Также не менее интересно увидеть, как веб-служба может вызываться унаследованным приложением любого типа и платформы. Следующий пример иллюстрирует базовый подход к отображению данных на унаследованной ASP-странице.

```
<script language="VBScript" runat="Server">
Option Explicit
Dim URL
URL = "http://localhost/WebServices1/EmployeesService.asmx/GetEmployeesCount"
Dim objHTTP
Set objHTTP = CreateObject("Microsoft.XMLHTTP")
' Отправляем команду HTTP_POST по URL-адресу.
objHTTP.Open "POST", URL, False
objHTTP.Send
' Считываем и отображаем значение корневого узла.
Dim numEmp
numEmp = objHTTP.responseXML.documentElement.Text
Response.Write(numEmp & " employee(s) in London")
</script>
```

Этот код просто устанавливает URL-адрес, указывающий на веб-метод в веб-службе. Затем он использует класс `Microsoft.XMLHTTP` (из анализатора Microsoft XML Parser, представляющего собой COM-компонент, который предоставляет классы для работы с XML-данными, отправки HTTP-команд и получения соответствующих ответов) для открытия HTTP-соединения и отправки команды синхронным образом. В данном случае код получает доступ к службе посредством команды HTTP POST, которую веб-службы ASP.NET поддерживают только на локальном компьютере. После возврата из метода `Send` текст ответа сохраняется в свойстве `responseXML`. Оно предоставляется в виде объекта `MSXML2.DOMDocument` вместе со свойством `documentElement`, которое указывает на корневой узел возвращаемых XML-данных. Используя этот объект, можно перемещаться по XML-данным ответа. В данном случае, поскольку результатом является целое число, для считывания возвращаемого значения было использовано свойство `Text`. Результаты выполнения этого кода показаны на рис. 35.13.

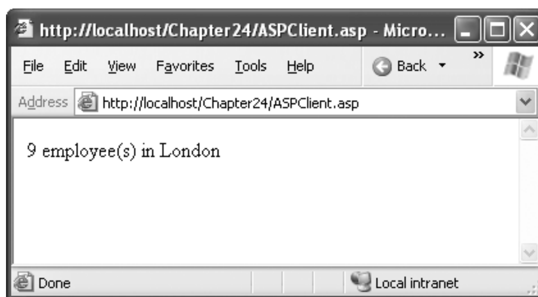


Рис. 35.13. Отображение данных веб-службы на ASP-странице

Интересным моментом в этом примере является то, что он предполагает использование библиотеки Microsoft XML, в которой имеются классы для отправки команд через HTTP, получения ответного текста и анализа XML. Это все, что вам нужно. Если вы до сих пор еще не установили этот компонент, можете загрузить его по адресу: `http://msdn.microsoft.com/library/en-us/xmlsdk/html/xmmscxmlinstallregister.asp`. Обратите внимание на то, что вы можете использовать показанный выше код, с незначительными изменениями, в любом VBScript- или VBA-приложении, в приложении

для создания макросов Microsoft Office и клиенте WSH (Windows Scripting Host — сервер сценариев Windows) включительно.

Теперь давайте рассмотрим более сложный пример — пример веб-метода `GetEmployees()`, который возвращает полный набор данных (`DataSet`). Для взаимодействия с этими данными вам придется прогрузиться в структуру XML-ответа. Ниже показан фрагмент кода, который в цикле просматривает дескрипторы `<Employees>` и извлекает определенные элементы данных для создания списка.

```
<script language="VBScript" runat="Server">
Option Explicit
Dim URL
URL = "http://localhost/WebServices1/EmployeesService.asmx/GetEmployeesCount"
Dim objHTTP
Set objHTTP = CreateObject("Microsoft.XMLHTTP")
' Отправляем команду HTTP_POST по URL-адресу.
objHTTP.Open "POST", URL, False
objHTTP.Send
' Извлекаем XML-ответ.
Dim Doc
Set Doc = objHTTP.responseXML
' Погружаемся в структуру набора данных (DataSet) в XML-ответе.
' Пропускаем один узел, чтобы обойти схему. Затем переходим на один
' уровень ниже к корневому элементу набора данных (DataSet).
' И, наконец, в цикле просматриваем содержащиеся в наборе данных
' дескрипторы, каждый из которых представляет сотрудника.
Dim Child
For Each Child In Doc.documentElement.childNodes(1).childNodes(0).childNodes
 ' Первый узел — это идентификационный номер пользователя (ID).
 Response.Write(Child.childNodes(0).Text + "<br>")
 ' Второй узел — это имя пользователя.
 Response.Write(Child.childNodes(1).Text)
 ' Третий узел — это фамилия пользователя.
 Response.Write(Child.childNodes(2).Text + "<br><br>")
Next
</script>
```

Создание ASP-клиента с помощью инструментального набора Microsoft SOAP Toolkit

В предыдущем примере был показан минимальный набор операций, которые необходимы для вызова веб-службы — это отправка сообщения через HTTP и синтаксический анализ возвращаемых XML-данных вручную. Однако многие клиенты имеют доступ к более надежным наборам инструментальных средств, которые напрямую поддерживают протокол SOAP. Одним из таких наборов является инструментальный набор Microsoft SOAP Toolkit. Он представляет собой COM-компонент и может использоваться для вызова любого типа веб-службы, которая предоставляет действительный WSDL-документ. Следовательно, этот набор SOAP Toolkit поддерживает веб-службы, создаваемые как в .NET, так и на других платформах. С помощью этого набора вы можете использовать веб-службы в основанных на COM приложениях, таких как те, которые были созданы в Visual Basic 6, Visual C++6 и ASP. Загрузить самую последнюю версию SOAP Toolkit можно по адресу `http://msdn.microsoft.com/webservices/_building/soaptk`.

Чтобы использовать Microsoft SOAP Toolkit, вы должны знать, где находится WSDL-документ требуемой вам веб-службы. С помощью этого WSDL-документа SOAP Toolkit будет динамически генерировать прокси. Увидеть этот прокси вы не сможете (потому

что он создается во время выполнения), но вы сможете использовать его для получения доступа к высокоуровневой модели веб-служб.

Ниже показан код ASP-страницы из предыдущего примера, который теперь использует SOAP Toolkit. В этом случае WSDL-документ извлекается непосредственно с веб-сервера. Для улучшения производительности рекомендуется сохранить локальную копию этого WSDL-файла и использовать ее для конфигурирования объекта `SoapClient`.

```
<script language="VBScript" runat="Server">
Option Explicit
Dim SoapClient
Set SoapClient = CreateObject("MSSOAP.SoapClient"
' Генерируем прокси.
Dim WSDLPath
WSDLPath = "http://localhost/WebServices1/EmployeesService.asmx?WSDL"
SoapClient.MSSoapInit WSDLPath
' Считываем данные о количестве сотрудников.
Dim numEmp
numEmp = SoapClient.GetEmployeesCount()
Response.Write(numEmp & " employee(s) in London")
</script>
```

Обратите внимание, что в этом примере не приходится считывать никаких XML-данных. Вместо этого клиент вызывает метод `GetEmployeesCount()` напрямую с помощью объекта `SoapClient`. Однако такой подход не позволит работать с набором данных (`DataSet`), поскольку COM-эквивалента для данного класса не существует. Следовательно, при необходимости делать это придется прибегнуть к синтаксическому анализу XML-данных, как было показано в предыдущем примере.

Конечно, Java, C++, Delphi и т.д. имеют свои собственные компоненты, библиотеки и API-интерфейсы для установки HTTP-соединений, отправки команд и синтаксического анализа XML-текста, но в целом подход будет таким же.

Настройка веб-службы

До сих пор мы показывали, как можно создать базовую веб-службу с парой методов и как создать использующую ее веб-страницу и Windows-приложение. Однако о многочисленных функциональных возможностях веб-служб еще ничего не говорилось. Например, веб-метод может помещать данные в кэш, использовать состояние сеанса и выполнять транзакции. В оставшейся части главы речь пойдет именно о том, как вы можете использовать все эти технологии.

Секрет применения этих функциональных возможностей заключается в атрибуте `WebMethod`. Во всех приводившихся до этого примерах атрибут `WebMethod` использовался для обозначения методов, которые необходимо было отобразить как часть веб-службы, и для присоединения описания (с помощью свойства `Description`). Тем не менее, у атрибута `WebMethod` имеется еще несколько дополнительных свойств, которые перечислены в табл. 35.6.

Свойство `CacheDuration`

Как уже рассказывалось в главе 11, в ASP.NET имеется встроенная поддержка для двух типов кэширования: кэширования вывода и кэширования данных. Веб-службы могут использовать обе эти формы кэширования, как вы увидите в следующих разделах.

Таблица 35.6. Свойства атрибута WebMethod

Аргумент	Описание
Description	Описание метода.
MessageName	Псевдоним для метода, используемый в случае перегрузки версий метода или при потребности отобразить метод под другим именем. Эта технология устарела.
CacheDuration	Количество секунд, в течение которых ответ метода будет храниться в кэше. По умолчанию этому свойству присваивается значение 0, что означает, что ответ метода вообще не будет помещаться в кэш.
EnableSession	Извлекает или устанавливает значение, указывающее, может ли данный метод получать доступ к данным в коллекции Session.
BufferResponse	Извлекает или устанавливает значение, указывающее, должен ли ответ данного метода копироваться в буфер. По умолчанию это свойство имеет значение true, которое следует изменить на false, если известно, что запрос будет выполняться долго, и необходимо, чтобы фрагменты данных начинали поступать раньше.
TransactionOption	Извлекает или устанавливает значение, указывающее, поддерживает ли данный метод транзакцию и, если да, то какого типа. Допустимыми значениями являются Disabled, NotSupported, Supported, Required и RequiresNew. Из-за того, что веб-службы не поддерживают состояний, они могут принимать участие в транзакции только в качестве корневого объекта.

Кэширование вывода

Самый простой вид кэширования, используемый в веб-службах — это кэширование вывода. Функция кэширования вывода работает в веб-службах точно так же, как она работает и на веб-страницах: идентичные запросы (в данном случае имеются в виду запросы одного и того же метода с одними и теми же параметрами) будут получать идентичные ответы из кэша до тех пор, пока не истечет срок хранения помещенной в кэш информации. Это может существенно улучшить производительность на сайтах с большим трафиком, даже если ответ будет храниться в течение всего лишь нескольких секунд.

Функцию кэширования вывода следует использовать только для простых операций по извлечению информации или обработке данных. Ее не стоит использовать в методе, требующем выполнения других операций, таких как изменение элементов сеанса, применение регистрации или внесение изменений в базу данных. Это связано с тем, что последующие вызовы кэшируемого метода будут получать помещенные в кэш результаты, и код веб-метода выполняться не будет.

Для включения функции кэширования применяется свойство CacheDuration атрибута WebMethod. Ниже показан пример использования этого свойства с методом GetEmployees():

```
[WebMethod(CacheDuration=30)]
public DataSet GetEmployees()
{ ... }
```

В этом примере набор данных (DataSet) о сотрудниках помещается в кэш на 30 секунд. Любой пользователь, который вызовет метод GetProduct() в течение этого периода времени, получит тот же самый набор данных (DataSet), прямо из кэша вывода ASP.NET.

Функция кэширования вывода становится даже еще более интересной, если представить, как она работает с методами, которые требуют параметров. Ниже показан пример, в котором веб-метод `GetEmployeesByCity()` помещается в кэш на 10 минут:

```
[WebMethod(CacheDuration=600)]
public DataSet GetEmployeesByCity(string city)
{ ... }
```

В данном случае ASP.NET поступает даже еще немного более разумно, используя повторно только те запросы, которые предоставляют одно и то же значение города. Например, вот как обрабатывались бы три следующих запроса к данной веб-службе.

1. Клиент вызывает метод `GetEmployessByCity()`, указывая для параметра города значение `London` (Лондон). Веб-метод вызывается, подключается к базе данных и сохраняет результат в кэше веб-службы.
2. Клиент вызывает метод `GetEmployessByCity()`, указывая для параметра города значение `Kirkland` (Киркленд). Веб-метод вызывается, подключается к базе данных и сохраняет результат в кэше веб-службы. Предыдущий помещенный в кэш набор данных (`DataSet`) не используется, потому что значение параметра города отличается.
3. Клиент вызывает метод `GetEmployessByCity()`, указывая для параметра города значение `London` (Лондон). При условии, что с момента запроса, описанного в пункте 1, десять минут еще не прошло, ASP.NET автоматически использует сохраненный в кэше результат. Никакой код не выполняется.

Принятие решения о том, стоит помещать в кэш эту версию метода `GetEmployeesByCity()` или нет, на самом деле зависит от объема получаемого веб-службой трафика и количества предусмотренных в ней названий городов. Если допустимых значений городов всего несколько, в применении описанного подхода есть смысл. Если таких значений множество и объем памяти веб-службы ограничен, очевидно, что такой подход будет неэффективен.

Кэширование данных

ASP.NET также поддерживает функцию кэширования данных, которая позволяет сохранять в кэше целые полнофункциональные объекты. Воспользоваться этой функцией можно посредством объекта `Cache` (который доступен в коде веб-службы через свойство `Context.Cache`). Этот объект может временно хранить информацию, создание которой обходится слишком дорого, так, чтобы веб-метод мог повторно использовать ее для других вызовов, поступающих от других клиентов. На самом деле эти данные могут даже быть использованы в других веб-службах или веб-страницах в одном и том же приложении.

Кэширование данных имеет большой смысл в версии веб-службы `EmployessService`, предлагающей два метода `GetEmployees()`, один из которых принимает в качестве параметра название города. При желании иметь оптимальную производительность и отсутствии необходимого объема кэш-памяти, вы можете поместить в кэш один единственный объект — полный набор данных (`DataSet`) о сотрудниках. Тогда, в случае вызова клиентом той версии метода `GetEmployees()`, которая в качестве параметра требует указать название города, вам необходимо будет просто отфильтровать строки для города, запрошенного клиентом.

Показанный ниже фрагмент кода демонстрирует эту схему в действии. Первым делом создается индивидуальный метод `GetEmployeesDataSet()`, который использует кэш. Если в кэше доступен объект `DataSet`, метод `GetEmployeesDataSet()` использует его и

не делает запрос в базу данных. В противном случае он создает новый объект DataSet и заполняет его полным набором записей о сотрудниках. Вот как это выглядит:

```
private DataSet GetEmployeesDataSet()
{
 DataSet ds;
 if (Context.Cache["EmployeesDataSet"] != null)
 {
   // Извлекаем набор данных из кэша.
   ds = (DataSet)Context.Cache["EmployeesDataSet"];
 }
 else
 {
   // Извлекаем набор из базы данных.
   string sql = "SELECT EmployeeID, LastName, FirstName, Title, " +
      "TitleOfCourtesy, HomePhone, City FROM Employees";
   SqlConnection con = new SqlConnection(connectionString);
   SqlDataAdapter da = new SqlDataAdapter(sql, con);
   ds = new DataSet();
   da.Fill(ds, "Employees");
   // Отслеживаем, когда был создан набор данных (DataSet). Вы можете
   // извлечь эту информацию в своем клиенте, чтобы удостовериться
   // в том, что функция кэширования работает.
   ds.ExtendedProperties.Add("CreatedDate", DateTime.Now);
   // Сохраняем его в кэше на 10 минут.
   Context.Cache.Insert("EmployeesDataSet", ds, null,
   DateTime.Now.AddMinutes(10), TimeSpan.Zero);
 }
 return ds;
}
```

Индивидуальный метод GetEmployeesDataSet() может использовать как метод GetEmployees(), так и метод GetEmployeesByCity(). Разница состоит в том, что метод GetEmployeesByCity() просматривает записи в цикле и вручную удаляет каждую запись, которая не соответствует указанному названию города. Ниже приводятся версии обоих методов:

```
[WebMethod(Description="Returns the full list of employees.")]
public DataSet GetEmployees()
{
 return GetEmployeesDataSet();
}
[WebMethod(Description="Returns the full list of employees by city.")]
public DataSet GetEmployeesByCity(string city)
{
 // Копируем набор данных (DataSet).
 DataSet dsFiltered = GetEmployeesDataSet().Copy();
 // Удаляем строки вручную.
 // Это хороший подход (по сравнению с тем, когда применяется
 // метод DataTable.Select()), потому что он устойчив к атакам
 // типа внедрения SQL.
 foreach (DataRow row in dsFiltered.Tables[0].Rows)
 {
   // Выполняем сравнение без учета регистра.
   if (String.Compare(row["City"].ToString(), city.ToUpper(), true) != 0)
   {
      row.Delete();
   }
 }
```

```
 // Удаляем эти строки безвозвратно.
 dsFiltered.AcceptChanges();
 return dsFiltered;
}
```

В принципе выбор количества времени, в течение которого информация будет храниться в кэше, должен делаться на основании того, насколько долго лежащие в основе данные будут оставаться действительными. Например, в случае, когда речь идет о показателях котировки акций, выбираемое количество секунд, скорее всего, будет меньшим, чем в случае, когда речь идет об информации по прогнозу погоды. В случае же, когда речь идет об информации, которая редко изменяется, такой как результаты ежегодного опроса населения, к этому вопросу вообще следует подходить несколько иначе. В таком случае информация является практически постоянной, но объемы возвращаемых данных будут намного превышать объемы кэша вывода ASP.NET. В этой ситуации целью будет сократить время хранения данных в кэше настолько, чтобы в нем хранились только наиболее популярные запросы.

Конечно, решения, касающиеся кэша, должны приниматься также и на основании того, сколько времени будет уходить на воссоздание информации, и сколько клиентов будут пользоваться данной веб-службой. Не исключено, что вам придется выполнить серьезное тестирование веб-службы в реальных условиях и соответствующую настройку, чтобы достичь наилучшего результата. Более подробную информацию о кэшировании данных можно найти в главе 11.

Совет. Кэш данных является глобальным для всего приложения (на одном веб-сервере). Это означает, что вы можете сохранять информацию в кэше в веб-службе и извлекать ее на веб-странице в том же веб-приложения, и наоборот.

Свойство `EnableSession`

Наилучшим вариантом для создаваемых в ASP.NET веб-служб является отключение состояния сеанса. На самом деле, по умолчанию веб-службы не поддерживают состояние сеанса. Чтобы достичь высокой степени масштабируемости, веб-службы должны проектироваться как не поддерживающие никаких состояний. Однако иногда возможность управления состояниями может пригодиться, например, для сохранения информации о пользователе или оптимизации производительности в каком-нибудь специфическом сценарии. В таком случае следует использовать свойство `EnableSession`, как показано ниже:

```
[WebMethod(EnableSession=true)]
public DataSet StatefulMethod()
{ ... }
```

Что происходит, когда есть веб-служба, которая включает функцию управления состоянием сеанса для некоторых методов, но отключает ее для других? По сути, отключение функции управления сеансом просто указывает ASP.NET игнорировать любую хранящуюся в памяти информацию о сеансе и устранять коллекцию `Session` из текущей процедуры. Это не приводит к удалению существующей информации из коллекции (эта информация удаляется только после завершения сеанса). Единственное преимущество в плане производительности, которое вы получаете в этом случае, состоит в том, что вам не приходится искать информацию о сеансе, когда в ней нет необходимости. Выполнять подобные шаги, чтобы разрешить коду использовать состояние `Application`, не нужно — эта коллекция глобального состояния доступна всегда.

Обработка состояния сеанса не является частью спецификации SOAP. Поэтому полагаться следует только на поддержку лежащей в основе инфраструктуры. Для поддержки состояния сеанса ASP.NET применяет cookie-наборы HTTP. Cookie-набор сеанса хранит идентификатор сеанса, а ASP.NET использует этот идентификатор сеанса для ассоциирования клиента с состоянием сеанса на сервере. Однако когда используется поддерживающая состояния веб-служба, гарантии, что клиент будет поддерживать cookie-наборы, нет. На самом деле поддерживать их не будут многие клиенты. Если клиент не поддерживает cookie-наборы, ASP.NET-функция управления состоянием работать не будет, и при каждом новом запросе будет создаваться новый сеанс. К сожалению, определить условие для такой ошибки в коде невозможно.

Чтобы попробовать поработать с состоянием сеанса (и посмотреть, какие потенциальные проблемы при этом возникают), вы можете создать простую веб-службу, которая показана ниже. Эта служба сохраняет один единственный фрагмент персонализированной информации (имя пользователя) и затем позволяет извлекать его.

```
public class StatefulService : System.Web.Services.WebService
{
 [WebMethod(EnableSession=true)]
 public void StoreName(string name)
 {
   Session["Name"] = name;
 }
 [WebMethod(EnableSession=true)]
 public string GetName()
 {
   if (Session["Name"] == null)
   {
      return "";
   }
   else
   {
      return (string)Session["Name"];
   }
 }
}
```

Если вы протестируете веб-методы `StoreName()` и `GetName()` с помощью описанной ранее в этой главе страницы тестирования ASP.NET, то увидите то поведение, которое и ожидали. Если вы вызовете метод `GetName()`, то получите ту строку, которую указывали тогда, когда в последний раз вызывали метод `StoreName()`. Это связано с тем, что веб-браузеры без проблем поддерживают cookie-наборы.

По умолчанию прокси-класс не имеет такой возможности. Чтобы увидеть эту проблему, добавьте ссылку на службу `StatefulSerivice` в Windows-клиенте. А затем добавьте новую кнопку с помощью следующего кода обработки события:

```
private void cmdTestState_Click(object sender, System.EventArgs e)
{
 // Создаем прокси.
 StatefulService proxy = new StatefulService();
 // Указываем имя.
 proxy.StoreName("John Smith");
 // Пытаемся извлечь имя.
 MessageBox.Show("You set: " + proxy.GetName());
}
```

К сожалению, как вы уже наверняка догадались, этот код работать не будет. Если вы его запустите, то увидите пустую строку, как показано на рис. 35.14.

Чтобы решить эту проблему, вы должны явно подготовить прокси веб-службы к принятию cookie-набора с идентификатором сеанса, создав для этого cookie-набора специальный контейнер (т.е. экземпляр класса System.Net.CookieContainer).

Чтобы исправить приведенный код, вы можете создать такой контейнер в виде переменной уровня формы, как показано ниже. Это гарантирует, что он будет существовать столько же, сколько и содержащий класс (форма), и сможет быть использован во множестве методов этой формы без утери текущего сеанса веб-службы.

```
public partial class Form1 : System.Windows.Forms.Form
{
 private System.Net.CookieContainer cookieContainer =
   new System.Net.CookieContainer();
 ...
}
```

Далее вы должны просто присоединить этот контейнер для cookie-набора к прокси-классу, прежде чем вызывать какой-либо веб-метод:

```
private void cmdTestState_Click(object sender, System.EventArgs e)
{
 StatefulService proxy = new StatefulService();
 proxy.CookieContainer = cookieContainer;
 proxy.StoreName("John Smith");
 MessageBox.Show("You set: " + proxy.GetName());
}
```

Теперь для вызовов обоих методов будет использоваться один и тот же сеанс, и в окне сообщений будет отображаться имя пользователя, как показано на рис. 35.15.

Рис. 35.14. Давшая сбой служба с поддержкой состояний

Рис. 35.15. Успешно выполнившаяся служба с поддержкой состояний

Контейнер для cookie-набора должен быть всегда под рукой до тех пор, пока остается необходимость в cookie-наборе сеанса. Например, если клиентом является веб-приложение и необходимо иметь возможность выполнять операции с веб-службой после каждой отправки данных без завершения сеанса, cookie-набор придется сохранить в состоянии сеанса текущей страницы. Обратите внимание на то, что состояние сеанса веб-приложения отличается от состояния сеанса веб-службы. Они являются не только отдельными приложениями, но также, возможно, запускаются на совершенно разных веб-серверах.

Теперь вам, наверное, уже понятно, насколько сложным процессом может оказаться использование сеансов с веб-службами. Поскольку веб-службы уничтожаются после каждого вызова метода, они не предоставляют естественного механизма для хранения информации о состоянии. Это ограничение может быть компенсировано с помощью коллекции Session, но такой подход, в свою очередь, чреват следующими сложностями.

- Состояние сеанса будет исчезать при завершении сеанса. Клиент не имеет возможности узнавать, когда завершается сеанс, а это означает, что поведение веб-службы может оказаться непредсказуемым.
- Состояние сеанса привязывается к определенному пользователю, а не к определенному классу или объекту. Это может привести к появлению проблем, если один

и тот же клиент попытается использовать одну и ту же веб-службу двумя разными способами или создать два экземпляра прокси-класса одновременно.

- Состояние сеанса поддерживается, только если клиент сохраняет cookie-набор сеанса. Функция управления состояниями, используемая в веб-службе, не будет работать, если клиент не выполняет этих условий.

По этим причинам веб-службы и функция управления состояниями не могут взаимодействовать естественным образом.

Свойство BufferResponse

Свойство BufferResponse позволяет выбирать, когда возвращаемые из веб-службы данные должны отправляться клиенту. По умолчанию этому свойству присваивается значение true. Это означает, что результаты будут отправляться клиенту только после того, как они будут полностью упорядочены. Если для этого свойства установить значение false (как показано ниже), ASP.NET будет возвращать и упорядочивать выходные данные по мере их поступления.

```
[WebMethod(BufferResponse=false)]
public byte[] GetLargeStreamOfData()
{ ... }
```

Метод веб-службы будет всегда заканчивать выполняться до возращения каких-либо результатов. Значение свойства BufferResponse применяется к процессу упорядочения, который происходит *после* выполнения метода. При отключенной функции буферизации сначала упорядочивается и отправляется клиенту первая часть результатов, затем упорядочивается и отправляется следующая часть и т.д.

Устанавливать для свойства BufferResponse значение false имеет смысл только в том случае, если веб-служба возвращает слишком большое количество данных. И даже в таком случае это редко когда сильно меняет дело, потому что автоматически генерируемый в .NET прокси-класс не имеет возможности обрабатывать возвращаемые данные по частям. Это означает, что этот прокси-класс все равно будет дожидаться, когда будет получена вся информация, прежде чем передавать ее в клиентское приложение. Однако вы можете изменить это поведение, взяв под свой непосредственный контроль процесс обработки XML-сообщений с помощью интерфейса IXmlSerializable, который описывается в следующем разделе.

Свойство TransactionOption

Веб-службы, как и любой другой фрагмент .NET-кода, могут инициировать транзакции ADO.NET. Кроме того, веб-службы могут принимать участие в транзакциях COM+. Транзакции COM+ интересны, так как они позволяют выполнять транзакцию, которая охватывает множество разных источников данных (например, базы данных SQL Server и Oracle). Еще транзакции COM+ фиксируются или откатываются автоматически. Однако такие дополнительные возможности и удобства имеют свою цену: поскольку транзакции COM+ используют двухэтапный протокол фиксации, они всегда выполняются медленнее, чем транзакции ADO.NET, инициируемые клиентом, или транзакции хранимых процедур.

Поддержка для транзакций COM+ в веб-службе тоже имеет определенные ограничения. Из-за того, что протокол HTTP не поддерживает состояний, методы веб-службы могут выступать только в качестве корневого объекта в транзакции. Это означает, что метод веб-службы может запускать транзакцию и использовать ее для выполнения ряда связанных между собой задач, но сгруппировать в одну транзакцию сразу несколько веб-служб нельзя. Поэтому при создании транзакционной веб-службы не помешает до-

полнительное продумывание деталей. Например, создавать финансовую веб-службу с отдельными методами DebitAccount() и CreditAccount() не имеет смысла, потому что они не смогут быть сгруппированы в одну транзакцию. Вместо этого лучше будет сделать так, чтобы обе эти задачи выполнялись как одно целое, воспользовавшись транзакционным методом TransferFunds().

Чтобы использовать транзакцию в веб-службе, вы должны сначала добавить ссылку в блок System.EnterpriseServices. Чтобы сделать это в Visual Studio, щелкните правой кнопкой мыши на элементе References (Ссылки) в окне проводника Solution Explorer, в появившемся контекстном меню выберите команду Add Reference (Добавить ссылку), после чего выберите значение System.EnterpriseServices. Далее импортируйте соответствующее пространство имен так, чтобы необходимые типы (TransactionOption и ContextUtil) были у вас всегда под рукой:

```
using System.EnterpriseServices;
```

Чтобы создать транзакцию в методе веб-службы, установите значение для свойства TransactionOption атрибута WebMethod. TransactionOption — это свойство типа перечисления, предоставляющее несколько значений, которые позволяют указывать, использует или требует ли данный компонент кода транзакции. Поскольку веб-службы могут выступать только в роли корневого объекта в транзакции, многие из этих значений для них не подходят. Чтобы создать метод веб-службы, запускающий транзакцию автоматически, используйте следующий атрибут:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public DataSet TransactionMethod()
{ ... }
```

Транзакция будет автоматически фиксироваться по завершении выполнения данного веб-метода. Откат транзакции будет выполняться, если появится какое-нибудь необрабатываемое исключение или если вы явно укажете, что выполнение транзакции должно прекращаться, с помощью следующего кода:

```
ContextUtil.SetAbort();
```

Большинство баз данных поддерживают транзакции COM+. При использовании этих баз данных в транзакционном веб-методе они будут автоматически перечисляться в текущей транзакции. В случае если произойдет откат транзакции, все выполненные с этими базами данных операции (вроде добавления, изменения или удаления записей) будут автоматически отменены. Однако некоторые операции (такие как запись файла на диск) в действительности не являются транзакционными. Это означает, что эти операции не будут отменяться, если во время выполнения транзакции вдруг произойдет сбой.

Теперь давайте рассмотрим другой веб-метод. Этот веб-метод выполняет два действия: он удаляет записи в базе данных и затем пытается считать данные из файла. Однако если операция с файлом не удается и это исключение никак не обрабатывается, будет выполнен откат всей транзакции и все удаленные записи будут восстановлены. Вот как выглядит код этого метода:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public void UpdateDatabase()
{
 // Создаем объекты ADO.NET.
 SqlConnection con = new SqlConnection(connectionString);
 SqlCommand cmd = new SqlCommand("DELETE * FROM Employees", con);
 // Применяем обновления. Это операция регистрируется как часть транзакции.
 using (con)
 {
   con.Open();
   cmd.ExecuteNonQuery();
 }
```

```
 // Пытаемся получить доступ к файлу. Это приводит к генерации
 // необрабатываемого исключения.
 // Выполнение веб-метода будет прервано, и все изменения будут отменены.
 FileStream fs = new FileStream("does_not_exist.bin", IO.FileMode.Open);
 // (Если ошибки не возникают, внесенные в базу данных изменения применяются
 // здесь, после того как завершиться выполнение метода.)
}
```

Другой способ обработать данный код — это перехватить ошибку, выполнить требуемую очистку и затем, при необходимости, явно выполнить откат транзакции:

```
[WebMethod(TransactionOption=TransactionOption.RequiresNew)]
public void UpdateDatabase()
{
 // Создаем объекты ADO.NET.
 SqlConnection con = new SqlConnection(connectionString);
 SqlCommand cmd = new SqlCommand("DELETE * FROM Employees", con);

 // Применяем обновления.
 try
 {
  con.Open();
  cmd.ExecuteNonQuery();
  FileStream fs = new FileStream("does_not_exist.bin",
  IO.FileMode.Open);
 }
 catch
 {
  if (con.State != ConnectionState.Closed) con.Close();
  ContextUtil.SetAbort();
 }
 finally
 {
  con.Close();
 }
}
```

Должна ли веб-служба использовать транзакции COM+? Все зависит от ситуации. При наличии необходимости в обновлении многих данных в отдельных хранилищах данных использование этих транзакций может оказаться полезным для гарантии целостности данных. С другой стороны, если значения изменяются только в одной единственной базе данных (такой как SQL Server 2000), пожалуй, лучшим вариантом будет использовать встроенные транзакционные возможности поставщика, как описывалось в главе 7.

На заметку! В будущем другие новые стандарты, такие как XLANG и WS-Transactions, возможно, заполнят все эти пробелы за счет определения межплатформенного стандарта, который позволит различным веб-службам принимать участие в одной транзакции. Однако до этого еще слишком далеко.

Резюме

В этой главе было рассмотрено, что собой представляют веб-службы, и почему они так важны для бизнеса. Мы также вкратце описали, как можно создавать и использовать веб-службы в .NET, а также тестировать их с помощью браузера. В следующих двух главах мы более подробно расскажем о лежащих в основе стандартах и том, как можно расширить инфраструктуру на базе веб-служб.