Microsoft

Developing ASP.NET MVC 4 Web Applications

Exam Ref 70-486

William Penberthy

Exam Ref 70-486

Prepare for Microsoft Exam 70-486—and help demonstrate your real-world mastery of developing ASP.NET MVC-based solutions. Designed for experienced developers ready to advance their status, *Exam Ref* focuses on the critical-thinking and decision-making acumen needed for success at the Microsoft Specialist level.

Focus on the expertise measured by these objectives:

- Design the application architecture
- Design the user experience
- Develop the user experience
- Troubleshoot and debug web applications
- Design and implement security

This Microsoft *Exam Ref*:

- Organizes its coverage by exam objectives.
- Features strategic, what-if scenarios to challenge you.
- Includes a 15% exam discount from Microsoft. Offer expires 12/31/2015. Details inside.

Developing ASP.NET MVC 4 Web Applications

About the Exam

Exam 70-486 is one of three Microsoft exams focused on the skills and knowledge necessary to create and deploy modern web applications and services.

About Microsoft Certification

Passing this exam earns you a Microsoft Specialist certification. You also receive credit toward a **Microsoft Certified Solutions Developer** (MCSD) certification that demonstrates your ability to build innovative solutions across multiple technologies, both on-premises and in the cloud.

Exams 70-480, 70-486, and 70-487 are required for MCSD: Web Applications certification.

Exams 70-480, 70-486, 70-488, and 70-489 are required for MCSD: SharePoint Applications certification.

See full details at:
microsoft.com/learning/certification

About the Author

William Penberthy, an application development consultant, has been working in software development for the past 25 years. He specializes in the Microsoft stack, focusing on ASP.NET, ASP.NET MVC, Silverlight, and Windows Presentation Foundation.

microsoft.com/mspress

ISBN: 978-0-7356-7722-7

U.S.A. $39.99
Canada $41.99
[*Recommended*]

Certification/Web Applications

Microsoft

Microsoft

Exam Ref 70-486: Developing ASP.NET MVC 4 Web Applications

William Penberthy

ISBN: 978-0-7356-7722-7

3 4 5 6 7 8 9 10 11 QG 8 7 6 5 4 3

Printed and bound in the United States of America.

Acquisitions Editor: Jeff Riley

Developmental Editor: Kim Lindros

Production Editor: Rachel Steely

Editorial Production: Box Twelve Communications

Technical Reviewer: Andre Tournier and Damien Foggon

Copyeditor: Nancy Sixsmith

Indexer: Angie Martin

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

[2013-12-13]

Contents at a glance

Contents

Introduction

The Microsoft 70-486 certification exam tests your knowledge of designing, developing, and troubleshooting ASP.NET MVC 4 web applications using Microsoft Visual Studio 2012. Readers are assumed to be experienced Microsoft ASP.NET web application developers with two or more years developing MVC-based solutions.

Most books take a very low-level approach, teaching you how to use individual classes and accomplish fine-grained tasks. Like the Microsoft 70-486 certification exam, this book takes a high-level approach, building on your knowledge of lower-level web application development and extending it into application design. Both the exam and the book are so high-level that there is very little coding involved. In fact, most of the code samples this book provides simply illustrate higher-level concepts.

Success on the 70-486 exam will prove your knowledge and experience in designing and developing web applications using Microsoft technologies. This exam preparation guide reviews the concepts described in the exam objectives, such as the following:

- Designing the application architecture
- Designing the user interface
- Developing the user interface
- Troubleshooting and debugging web applications
- Designing and implementing security

This book covers every exam objective, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions themselves and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the links you'll find in text to find more information and take the time to research and study the topic. Great information is available on MSDN, TechNet, and in blogs and forums.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premise and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to *http://www.microsoft.com/learning/en/us/certification/cert-default.aspx*.

Acknowledgments

This book would not have been possible without the patient and loving support of my wife Jeanine, who had to take over much of the responsibility of running a family so I could mutter to myself in the corner and click away on a keyboard. Many thanks also go out to my editor, Kim Lindros, who patiently walked this first-time author through the process of building a book.

Appreciation also goes out to Andre Tournier and Damien Foggon for keeping me on the straight and narrow, and to Jeff Riley from Box Twelve Communications for giving me this opportunity. Finally, I need to acknowledge you, the reader, for your desire to continue your own growth as a developer. Your efforts to improve your skills make us all work to improve ourselves to keep up. Kudos to you, and keep raising the bar!

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/ER70-486/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

Preparing for the exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. While there is no substitution for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. We recommend that you round out your exam preparation plan by using a combination of available study materials and courses. For example, you might use the training kit and another study guide for your "at home" preparation, and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you.

Note that this training kit is based on publically available information about the exam and the author's experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

CHAPTER 1

Design the application architecture

Every application must have an architecture, but plenty of applications have been created with architectures that were not well considered. As a developer, you should design your solution's architecture to fulfill application requirements and create a robust and high-performing application.

Start by determining the most appropriate way to build your application and then decide how and where it will be deployed. After you have narrowed down the deployment plan, whether on- or off-premise or across multiple physical machines, you can decide how best to fulfill your other application needs. Perhaps data must be stored in a database or the client needs to check in regularly with the server. Some applications might need to be distributed on a server farm, have 99.999 percent availability, serve thousands of pages an hour, or support hundreds of concurrent users. You must consider all of this information as you choose and design your application's architecture.

Objectives in this chapter:

- Objective 1.1: Plan the application layers
- Objective 1.2: Design a distributed application
- Objective 1.3: Design and implement the Windows Azure role life cycle
- Objective 1.4: Configure state management
- Objective 1.5: Design a caching strategy
- Objective 1.6: Design and implement a WebSocket strategy
- Objective 1.7: Design HTTP modules and handlers

Objective 1.1: Plan the application layers

An *application* is simply a set of functionality: a screen or set of screens that displays information, a way to persist data across uses, and a way to make business decisions. A *layer* is a logical grouping of code that works together as a common concern. Layers work together to produce the completed application.

In this section, you'll learn about the major aspects of an application's architecture that contribute to the layers of an application, such as data access methods and separation of concern (SoC). One of the essential parts of an ASP.NET MVC application is the architectural design of the Model-View-Controller (MVC) pattern. It is based on providing separation between the appearance of the application and the business logic within the application. The model is designed to manage the business logic, the view is what the user sees, and the controller manages the interaction between the two. Adhering to separation of concern, the model doesn't know anything about the view, and the view doesn't know anything about the controller.

This objective covers how to:

- Plan data access
- Plan for separation of concern
- Appropriate use of models, views, and controllers
- Choose between client-side and server-side processing
- Design for scalability

Planning data access

A key reason for using ASP.NET MVC to meet your web-based business needs is how it connects users to data. As you plan an application, you should evaluate your data requirements early in the process. Will your application access a set of data you already have, or will your data design be managed along with your application design? For example, suppose you want to add a just-in-time (JIT) supplier view to your inventory process so your suppliers can better understand how much of their product you have in stock. Perhaps you already have data and your application will provide access to other data, or maybe you have to design and implement an entirely new database schema.

Data access options

After you determine your data requirements—existing data, new data, or a combination—consider how you need to access the data. The two primary options are:

- **Using an object relational mapper (O/RM)** An O/RM is an application or system that aids in the conversion of data within a relational database management system (RDBMS) and the object model that is necessary for use within object-oriented programming. The O/RM hydrates the object with the data from the database, or creates the SQL statements that will save the object data into the database. Examples of O/RM products that can be used to support ASP.NET MVC 4 are NHibernate, the Entity Framework, and Linq-to-SQL.

- **Writing your own component to manage interactions with the database** Writing your own component implies you will need to manage any conversions to and from your object model. This approach might be preferred when you are working with a data model that does not closely model your object model, or you are using a database format that is not purely relational, such as NoSQL.

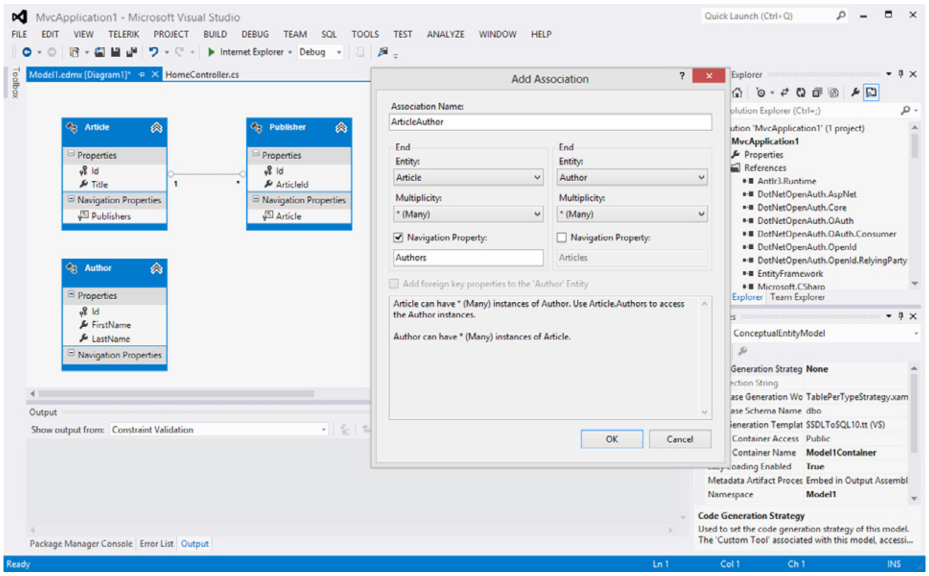
Design approaches

After you have worked through your data considerations and the type of access model you want to work with, you can start to consider the design approach for bringing the two together. The type of access model you will use drives the rest of your conceptual thinking. If you will create your own data access layer by using ADO.NET for access into your database, for example, you will be minimally affected whether the data schema exists or not. If, however, you are using an O/RM, your flexibility will be limited by the tool you use. Linq-to-SQL, for example, works only with pre-existing databases; it offers no support for building the object model and using it to create a database. Entity Framework and NHibernate enable you to write the model as part of your business design process and then create the database from that model.

NOTE **SESSIONS**

You must also consider how you will manage state. If you want to use sessions across multiple servers, you likely need to use Microsoft SQL Server because Microsoft Internet Information Services (IIS) supports it by default. If you plan to maintain state on your own, it needs to become part of your data management design.

Entity Framework supports the Model First, Code First, and Database First design approaches. Model First and Code First each offer a different way to link objects and a database. An architect uses the Model First approach when designing the database and the object model at the same time with Entity Designer in Microsoft Visual Studio. This was one of the most-requested features after the initial release of Entity Framework because new projects tend to need new database schemas. Using a visual modeling tool (see Figure 1-1) helps developers design the appropriate object and data model.



Add Association

Association Name: ArticleAuthor

End	End
Entity: Article	Entity: Author
Multiplicity: * (Many)	Multiplicity: * (Many)
[✓] Navigation Property: Authors	[ ] Navigation Property: Articles

[ ] Add foreign key properties to the 'Author' Entity

Article can have * (Many) instances of Author. Use Article.Authors to access the Author instances.

Author can have * (Many) instances of Article.

ConceptualEntityModel

Generation Strateg	None
ection String	
ase Generation Wo	TablePerTypeStrategy.xam
ase Schema Name	dbo
eneration Templat	SSDLToSQL10.tt (VS)
Container Access	Public
Container Name	Model1Container
oading Enabled	True
Metadata Artifact Proces	Embed in Output Assembl
Namespace	Model1

Code Generation Strategy
Used to set the code generation strategy of this model. The 'Custom Tool' associated with this model, accessi...

FIGURE 1-1 Model First approach to creating both an object and data model in Entity Designer

Entity Framework also supports the design of a new data schema through Code First, a process in which the development team writes the plain old CLR object (POCO) classes, and the Code First generator builds the database from those classes. Doing this enables the development team to design the object structure, in code, that bests suits their application and generate the database from that design. It is done outside of Entity Designer. You can attribute the model properties to control the database configuration, which enables you to control such items as the name of the table or column in the database, maximum length, default values, keys, database-generated IDs, and other characteristics.

As you plan your application design, you must evaluate the current state of your data. If you are working on an upgrade or conversion, we recommend the Database First approach, which enables you to continue using the existing structure with no impact on the database. However, if you are creating a new database schema, you can choose whichever approach best serves your development team. Some teams prefer to use Entity Designer; others prefer to conceptualize the object model using a third-party tool or a white board. Other teams work best when designing the database first. Your considerations at this point will likely be less about the technology and more about your current database design and the preferences and strengths of the team.

There are several things to keep in mind as you consider the life cycle of your implementation. Model First and Code First are both strongest in the creation of the initial database schema. Maintaining the schema is more problematic. Although both tools have improved their capability to manage database upgrades, most teams tend to use the Model First or Code First approach for the initial connection and then take a more Database First approach for upgrades in which you script the database changes and then refresh your .edmx file from the database to capture the updates.

MORE INFO **ENTITY FRAMEWORK**

The MSDN Data Developer Center provides detailed information on Entity Framework at *http://msdn.microsoft.com/en-us/data/ef.aspx*. Because Entity Framework uses an open development model, you can look at the code behind it and even contribute functionality to the project.

Data access from within code

After you select the means by which you will manage your initial database design, you need to consider the approach to access data from within your code. In some respects, the stateless nature of ASP.NET MVC complicates this because Entity Framework relies on the *DBContext* class, which is an abstraction over the database that manages data querying as well as a unit-of-work approach that groups changes and persists them back to the datastore in a single transaction. However, *DBContext* relies on several managed features and flags that keep track of changes in items that have been queried from the datastore. It relies on the flags to determine the best way to persist the information. The stateless nature of ASP.NET MVC prevents the default functionality of Entity Framework from working, however. You have to choose a different method to control data flow into *DBContext* and thus into your database. Because some additional work must be done outside of Entity Framework, you should evaluate whether you want to do this work in your controller(s) or provide a level of abstraction between your controllers and Entity Framework.

MORE INFO **MODELS, VIEWS, AND CONTROLLERS**

You'll learn details about models, views, and controllers in the "Using models, views, and controllers appropriately" section later in this chapter.

The primary data access pattern in C# is the Repository pattern, which is intended to create an abstraction layer between the data access layer and the business logic layer. This abstraction helps you handle any changes in either the business logic or the data access layer by breaking the dependencies between the two. It also enables the business logic layer to access the repository without knowing the specific type of data it is accessing, such as a Microsoft SharePoint list or a database. What the repository does internally is separate from the business logic layer.

The Repository pattern is also highly useful in unit testing because it enables you to substitute the actual data connection with a mocked repository that provides well-known data. Another term that can describe the repository is *persistence layer*. The persistence layer deals with persisting (storing and retrieving) data from a datastore, just like the repository. When using the Repository pattern, you create the repository interface and class. When you need to use the repository, you instantiate the interface rather than the class. This enables you to use the data connection when doing work on the mock repository during testing. Adding the Unit Of Work pattern enables you to coordinate the work of multiple repositories by creating a single shared class for them all. You have many different ways to implement a repository: You can create a global repository for all the data, a repository for each entity, or some combination. Figure 1-2 shows how the controller, repository, and Entity Framework interact.

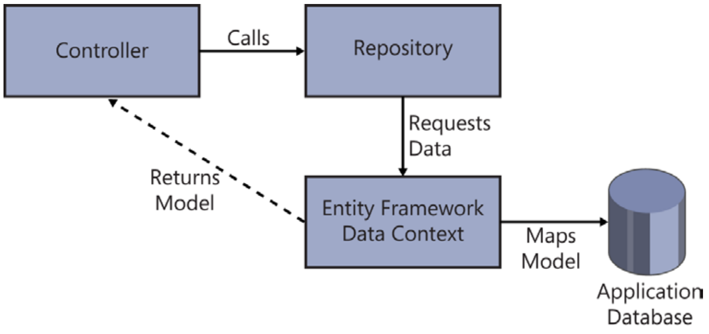


FIGURE 1-2 Repository pattern implementation

MORE INFO **DATA ACCESS**

CodePlex provides references that illustrate how to implement the Repository, Unit of Work, Specification, State, and other patterns using ADO.NET Entity Framework 4.0, as well as the ASP.NET MVC framework, Unity, Prism, and the Windows Communication Framework (WCF) REST Starter Kit. Visit *http://dataguidance.codeplex.com/.*

Planning for separation of concern (SoC)

Separation of concern (SoC) is a software development concept that separates a computer program into different sections, or concerns, in which each concern has a different purpose. By separating these sections, each can encapsulate information that can be developed and updated independently. N-tier development is an example of SoC in which the user interface (UI) is separated from both the business layer and the data access layer.

ASP.NET MVC adds a level of concern due to the client-based nature of web browsing. Supporting JavaScript in the browser means there are two parts of the UI the developer needs to consider: the part of the UI created and rendered on the server and the part affected solely by code on the client side. Although the addition of SoC adds some complexity to the application's design, the benefits outweigh the extra complexity.

A term closely associated with SoC is loose coupling. *Loose coupling* is an architectural approach in which the designer seeks to limit the amount of interdependencies between various parts of a system. By reducing interdependencies, changes to one area of an application are less likely to affect another area. Also, by eliminating interdependencies, you ensure that your application is more maintainable, testable, and flexible, which tends to result in a more stable system.

Using models, views, and controllers appropriately

The appropriate use of models, views, and controllers in an ASP.NET MVC application is critical to having a well-designed application. It is important to remember that ASP.NET MVC is highly convention-driven, in that it uses built-in assumptions about the folders various files might be in, what they are named, and the types and names of the methods within those files. These conventions will be emphasized as the components of the MVC pattern are discussed. Each component has a particular function in the framework; the controller answers the HTTP call and, if necessary, gives the model to the view for display. Figure 1-3 shows the interaction between the model, view, and controller.

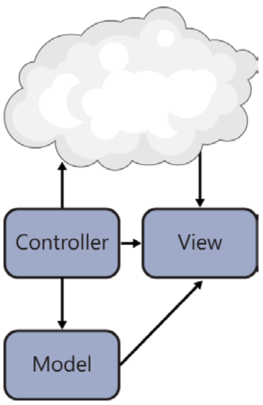


FIGURE 1-3 Default MVC design

Model

The *model* is the part of the application that handles business logic. A model object manages data access and performs the business logic on the data. Unlike other roles in an MVC application, the model does not implement any particular interface or derive from a certain base class. Instead, it is a model because of the role the class plays and where it is located in the folder structure of the application. This is an example of the convention-based aspects of the framework because model classes are traditionally placed in the Models folder. It is also common, however, to store the models in a separate assembly. Storing the models in a separate assembly makes model sharing easier because multiple applications can use the same set of models. It also provides other incremental improvements, such as enabling you to separate model unit tests from controller unit tests as well as reducing project complexity. Controllers typically instantiate the model in its actions and then provide the model to the view for display.

In general, you can build your model, domain, view, or input modeling in different ways. You use a domain model when the object you are using describes the data you work with in the middle tier of that application. If you are using Entity Framework, for example, and present these objects to a view for display, you are using a domain model approach for creating your model.

A view model approach describes the data being worked on in the presentation layer. Any data you present in the view is found within the properties of the view model class, which represents all the data the controller transmits to the view after processing the request. A view model is generally the result of aggregating multiple classes into a single object.

The input model faithfully represents the data being uploaded to the server from the client with each individual HTTP request. The input model approach uses model binding to capture user input. When you consider a typical complex data entry form, you might have one entry form that captures information that would typically span across multiple objects in a domain, such as name, address, employers, phone numbers, and other values. Those objects would get mapped to different domain objects. The use of an input model, however, enables all the work to create and manage these domain objects to stay within a single controller and model.

Model binders are a simple way to map posted form data to a type and pass that type to an action method as a parameter. Once again, this requires approaching the construction with an ASP.NET MVC convention in mind. The *DefaultModelBinder* automatically maps input values to model properties if the names match precisely. The model binder implements the *IModelBinder* interface and contains a *GetValue* method that retrieves the value of a specified parameter or type. You can use existing value providers to evaluate request values or you can create custom value providers for special evaluation.

Model binding is recursive and transverses complex object graphs. ASP.NET MVC enables you to create custom model binders, which is useful because the default model binder does not support abstract classes or interfaces. There are times when that ability is necessary, especially if you want to use dependency injection and inversion of control.

Controllers

Controllers are the part of ASP.NET MVC 4 that handles incoming requests, handles user input and interaction, and executes application logic. A controller calls the model to get the required business objects, if any, and then calls the view, either with or without a model, to create and render the output Hypertext Markup Language (HTML). A controller is based on the *ControllerBase* class and is responsible for locating the appropriate action method to call, validating that the action method can be called, getting values in the model to use as parameters, managing all errors, and calling the view engine to write the page. It is the primary handler of the interaction from the user.

ASP.NET pages raise and handle events between the browser and webpage, whereas ASP.NET MVC applications are organized around controllers and action methods. *Action methods* are typically one-to-one mappings to user interactions. Each user interaction creates and calls a uniform resource locator (URL). The routing engine parses the URL using routing rules to determine the controller and action method that needs to be called, for example, with *http://myurl/Default/Index*. The default convention interprets this by determining the *subpath/Default/Index* and uses it to call the Index method on the *DefaultController* class.

Because action methods map to user interactions, an action method is called every time a user does something that interacts with the server. This is important to remember when you are approaching the design of your application. Historically, traditional web design has taken a paged approach, in which a set of features occurs on an individual page. ASP.NET Web Forms, for example, uses that methodology, in which the implementation logic for a page is handled on that page. Although this design makes some aspects of communicating between pages complicated, it acts as a built-in mechanism for managing the design. If you need to create a page for users to manage a widget, you can do that. Whenever a user needs to create a widget, you would redirect them to that page. With ASP.NET MVC, you need to take a different approach to design because there are no pages, just action methods.

One way to look at a controller is as a way to separate functionality. You could create a large, complex application with dozens of screens using a single controller. You can see a small example of this when you create a new ASP.NET MVC Internet project in Visual Studio (the default integrated development environment for ASP.NET MVC 4). The HomeController that is built as part of this project handles the views for the About, Contact, and Home pages by using an action for each page. A better approach when laying out the controller structure is to have a controller for each type of object with which the user will be interacting on the screen. This enables you to compartmentalize the functionality around the object into a single place, making code management simpler and providing more easily understandable URLs.

The best time to conceptualize your controller structure is when you are building your data model for the application. Although there is generally not a one-to-one match between a controller and the application's data or object model, there is a correlation. You should not follow a specifically data-based approach, however, because the work the user will be doing is an important consideration. If the screens the user will be interacting with do not map to your application's data model, your controllers likely should not, either. Instead, you should consider the use of a separate business layer that more closely matches the business process the user will follow, or a view model approach that enables you to create a specialized object or set of objects as an intermediary between the object model and the user. In either case, you should align your controllers with those objects to provide a sensible separation.

ACTIONS AND ACTION RESULTS

After you map your controllers, you need to work on the actions that will be methods in the controller. Because there is a one-to-one mapping between user interactions and the actions in the application, the initial set of actions you have to create should be clear if you have an understanding of the application flow. You should be able to predict most actions based on the application's requirements, but you might have to add or modify actions later. You will discover other actions that might not necessarily be linked to a user interaction, but instead a system interaction taken by the application on behalf of the user. Examples include a JavaScript timer on the client that calls an action to get an update on the current weather or populating drop-down lists based on a previous selection as the user goes through a data entry form.

Because there are different expectations from an action, there are different types of action results. An *action result* is any kind of outcome from an action. Although an action traditionally returns a view or partial view, it can also return JavaScript Object Notation (JSON) results or binary data, or redirect to another action, among other things. Keep action results in mind as you plan for communication between the client and server; as the action results dictate the client experience.

MORE INFO **ACTION RESULTS**

For more information on action results, see Chapter 3, "Develop the user experience."

Action names are also important. Because the name is part of the URL request, it should be short and descriptive. Do not be so descriptive that you provide too much of the business process in the name, which can result in security issues. Also consider consistency of action names across controllers. Actions that do the same thing to different objects should have the same name. Convention would also have you not reuse the name of the controller in the action name: *http://urlhere/product/edit* versus *http://urlhere/product/productedit*.

ROUTES AND ROUTING

It is difficult to talk about controllers without including routes. The routing table is stored in the Global.asax file. The routing system enables you to define URL mapping routes and then handle the mapping to the right controller and actions. It also helps construct outgoing URLs used to call back to the controller/actions.

ASP.NET provides some default routing. The default routing format is *{controller}/{action}/{id}*. That means an HTTP request to *http://myurl/Product/Detail/1* will look for the *Detail* action on the *ProductController* that accepts an integer as a parameter. The routing engine doesn't know anything about ASP.NET MVC; its only job is to analyze URLs and pass control to the route handler. The route handler is there to find an HTTP handler, or an object implementing the *IHttpHandler* interface, for a request. *MvcHandler*, the default handler that comes with ASP.NET MVC, extracts the controller information by comparing the request with the template values in the routing table. The handler extracts the string and sends it to a controller factory that returns the appropriate controller. The controller factory is easily extendable by creating a custom controller factory that implements *IControllerFactory*.

MORE INFO **ROUTING**

Designing and implementing routing is covered in Chapter 3, “Develop the user experience.”

Controller actions have attributes that provide additional information to the framework. The most-used select attributes are *ActionName*, *AcceptVerbs*, and *NonAction*, which help the framework determine which action to run. Filter attributes enable you to add caching, validation, and error handling through the use of *OutputCache*, *ValidateInput*, and *HandleError*. Because the attributes are part of ASP.NET MVC, they are customizable as well. You can create custom action filters that surround an action with custom logic by overriding the base *ActionFilter* class.

ASYNCHRONOUS CONTROLLERS

One of the major changes in ASP.NET MVC 4 involves asynchronous controllers. ASP.NET MVC 3 uses an *AsyncController* class that needs to be implemented to have asynchronous controllers. ASP.NET MVC 4 brings the concept of asynchronous controllers into the default controller class. Asynchronous action methods are useful for long-running, non-CPU-bound requests because they avoid blocking the web server from performing work while the method request is still pending. When designing your action methods, you need to determine whether to use synchronous or asynchronous processing. You should strongly consider asynchronous methods when the operation is network-bound or I/O-bound rather than CPU-bound. Also, asynchronous methods make sense when you want to enable the user to cancel a long-running method.

Modern computers have processors that have multiple cores, which makes multithreading even more important because it is gaining more support with every computer generation. Being able to do work on multiple threads allows parallel processing, which should result in an increase in performance, especially when multiple long-running processes occur during the same HTTP request. When designing your ASP.NET MVC 4 application, you should look at every process that reaches outside of your domain and consider making them asynchronous. You should do the same for those calls that might be long-running, such as pages that return lists from multiple data sources or that perform intensive business operations, because they could be ideal candidates for the using of asynchronous behavior.

Using asynchronous actions is easy with ASP.NET MVC 4. The key to using the new asynchronous framework is the *Task* framework in the *System.Threading.Tasks* namespace. The purpose of *Task* is to provide a pluggable architecture to increase flexibility and to make multitasking applications easier to write. To create an asynchronous action on a controller, mark the controller as async and change the return from an *ActionResult* into a *Task<ActionResult>*. In the C# code in Listing 1-1, the application is making a call to an external data feed.

LISTING 1-1 Calling an external data feed

```
public async Task<ActionResult> List()
{
    ViewBag.SyncOrAsync = "Asynchronous";
    string results = string.Empty;
    using (HttpClient httpClient = new HttpClient()
    {
        var response = await httpClient.GetAsync(new Uri("http://externalfeedsite"));
        Byte[] downloadedBytes = await response.Content.ReadAsByteArrayAsync();
        Encoding encoding = new ASCIIEncoding();
        results = encoding.GetString(downloadedBytes);
    }
    return PartialView("partialViewName", results);
```

Asynchronous programming gives you different ways to solve performance issues where multithreading might help. You can create an action that returns synchronously but uses asynchronous work within the method to get work done faster. (The main thread has to wait only for the longest-running work unit to respond rather than waiting for all the work to occur, one after the other.) This kind of approach makes sense if you are merging the results from multiple service calls into a single model to be passed to the view. Another approach is to use an asynchronous partial view, such as in Listing 1-1. This helps the overall performance of your application by running the work in that partial view in a different thread, enabling the primary thread to continue to process other items. It also helps you avoid thread locking because your MVC4 application parses the action. A third approach is to break content out on the page and load it asynchronously from the client. A typical use case is to create your page normally, but rather than directly calling the action result *@Html.Partial("LeadArticleControl", Model.LeadArticle)* in your .cshtml file, you instead use JavaScript code that calls the server to ask for the partial view result after the page has been rendered on the client side, a traditional AJAX approach.

Views

The view is the part of the application responsible for displaying information to users. It's the only part of the application that users see. Users' initial impressions, and their entire interaction with your application, are through a view. The controller gives the view a reference to the model or the information that needs to be displayed. Technically, a set of messages is sent to the view via a *ViewDataDictionary*, which is wrapped by a *ViewBag*. This means you can set and read values as if the collection were a standard dictionary: *ViewData["UserName"] = User.UserName*. You can also access the data in the *ViewBag* as a wrapper: *ViewBag.UserName = User.UserName*.

The following are additional considerations when working with a view:

- **Strongly-typed views** Eliminates the need for casting in the view by setting the attached model property. The view engine can work with the information through mapped class values rather than through a string-based lookup.

- **View-specific model** An intermediate class for when the display does not map directly to a domain object. The view-specific model gathers all the values that are needed for the view from one or more model objects into a single class specifically designed for that view.
- **Partial view** ASP.NET MVCs version of a user control that can be displayed within a page. The Razor view engine displays it the same as a full view, but without including the *<html>* and *<head>* tags.
- **Master or layout page** A way to share a design across multiple pages. This page is a building block for the application because it contains much of the wrapper HTML code that turns your output into a format understood by web browsers.
- **Scaffold template** A template that creates standard pages as part of the process when creating a project. This ability gives you a quick start on development. Because the default scaffold types are Visual Studio T4 templates, you can alter the existing scaffold types or create a new one.

Figure 1-4 shows how the design of a rendered page might have been built when a layout page is used by a view that also contains a partial view.

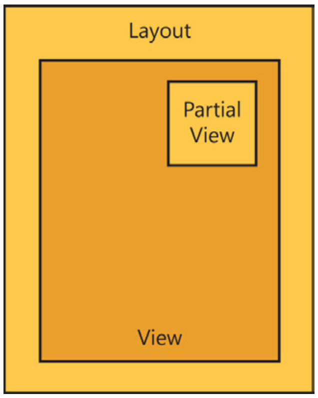


FIGURE 1-4 Rendered page with view relationships

THE RAZOR VIEW AND WEB FORMS VIEW ENGINES

The Razor view engine was introduced in ASP.NET MVC 3 and became the default view engine in ASP.NET MVC 4. The Razor view provides a streamlined, compact, expressive, and fluid format that minimizes the amount of coding required within a view. Razor also supports the concept of layouts, which help maintain a consistent look and feel across multiple views within an application.

The Web Forms view engine was the initial view engine. It is similar to ASP.NET and provides a familiar experience to an ASP.NET developer. The Razor view engine uses the @ code delimiter; Web Forms uses the <% notation.

Neither view engine can understand the syntax of the other. Table 1-1 compares Razor and Web Forms syntax.

TABLE 1-1 Comparisons between Razor and Web Forms syntax

Code expression	Razor	Web Forms
Implicit	*<span>@article.Title</span>*	*<span><%: article.Title %></span>*
Explicit	*<span>Title@(article.Title)</span>*	*<span>Title<%: article.Title %></span>*

Unlike code expressions, which are designed to output content to the user interface, blocks of code are pieces of code executed within the view. You need to avoid doing work that should properly be done in the controller or model. Listing 1-2 shows examples of code blocks that create a variable that can be used throughout the rest of the page.

LISTING 1-2 Code blocks that create a variable

Razor view engine

```
@{
    string title = article.Title.ToUpper();
}
```

Web Forms view engine

```
<%
    string title = article.Title.ToUpper();
%>
```

Sometimes you need to mix plain text with decisions that are made in code. The code samples in Listing 1-3 show how to mix code and plain text.

LISTING 1-3 Incorporating plain text into code blocks

Razor view engine

```
@if (article.HasContent)
{
    <text>some message here</text>
}
```

Web Forms view engine

```
<% if (article.HasContent) { %>
    some message here
<% } %>
```

Finally, sometimes you want to display the output from a generic method. You should carefully consider these cases because this approach makes it easy to do work in the view that should be handled in the controller. The code in Listing 1-4 calls a generic method.

LISTING 1-4 Calling a generic method

Razor view engine

```
@(Html.GenericMethodHere<TheType>())
```

Web Forms view engine

```
<%: Html.GenericMethodHere<TheType>() %>
```

EXTENDING THE VIEW ENGINES

Both the Web Forms and the Razor view engines are derived from the *BuildManagerViewEngine* class, which is derived from the *VirtualPathProviderViewEngin*e class. A common reason for overriding or replacing the default view engine classes is to deviate from the convention-based design the standard view engines must follow. You can also write an HTML helper to help you generate HTML inside views. An *HTML helper* is a class that helps you create HTML controls programmatically. A helper generates HTML and returns the result as a string for inclusion in the response stream. You can create HTML and AJAX-HTML for inclusion in your view, or URL helpers, which help determine the appropriate route or URL that can be accessed from both the view and controller. You can also write a Razor helper using Razor syntax. Razor helpers are one of Razor's unique features. They encapsulate blocks of HTML and server-side logic into reusable page-level methods.

EXAM TIP

SoC is one of the primary reasons why ASP.NET MVC exists because its very nature separates the presentation and business layers. However, the framework's flexibility enables you to easily violate these rules. You should be familiar with the differences between the logic that should take place in a view, in a controller, and within the model. The use of inline code in the view should be strictly limited to those items that affect only the display of information, not the processing of information.

Choosing between client-side and server-side processing

Choosing between client- and server-side processing seems straightforward when you look at SoC concerns. Client-side processing makes the most sense when the work being done stays completely within the client, such as when selecting a value in a drop-down list changes a background color. Unfortunately, you won't encounter many requirements where the interaction is completely client side.

Factors to take into account when considering client-side versus server-side are application performance, user experience, and business requirements. Application performance is important because there will always be some latency when connecting over the Internet. Validation on the client side, for example, enhances performance by eliminating calls across the network for transactions that would fail validation. Heavily used sites can increase performance by lowering the server's load. However, be careful not to sacrifice security for speed. You shouldn't completely replace server-side checking with client-side validation. With only client-side validation, there is still a chance of bad data getting to the server and entering the business process. A best practice is to put validation on both sides—on the client side to provide a responsive UI and lower the network cost, and on the server side to act as a gateway to ensure that the input data is valid.

As you consider client- and server-side processing, remember that it is not one or the other; you can do both on a single user request. Also, some decisions you make on the client side might need to be replicated on the server side as well.

Designing for scalability

Scalability is the capability of a system to handle a growing amount of work. Although usage is minimal during site development, usage can increase greatly after implementation to a production environment. To ensure a positive user experience, you need to consider scalability early in the application planning phase because your scalability decisions affect your architectural design considerations. There are two primary ways that you can scale: horizontally or vertically.

With horizontal scaling, you scale by adding additional nodes to the system. This is a web farm scenario, in which a number of commodity-level systems can be added or removed as demand fluctuates. They are served using a load balancer or other piece of network equipment that determines which server should be called.

MORE INFO **WEB FARMS**

You will learn about web farms in the "Planning web farms" section later in this chapter.

If your application will scale horizontally, you must make various decisions. Depending on the network hardware that will be deployed and how it handles sessions, your session state information will be affected. You also need to determine how multiple servers will affect server caching of information, such as whether to cache rendered HTML that was sent to the client or cache data from a database. Also, if your application will provide file management, consider where those files will be stored to ensure access across multiple servers. Scaling horizontally adds some architectural considerations, but it is a low-cost and effective way to scale, especially because the cost for commodity servers continues to drop. Keep in mind that commodity servers are not necessarily physical servers, but can be virtual machines. It is far less expensive to roll in unused capacity using virtualization from another system than it is to add capacity to a system.

With vertical scaling, you scale by adding resources to a single system. This typically involves adding central processing units (CPUs) or memory. It can also refer to accessing more of the existing resources on the system. Vertical scaling has its own architectural considerations as well. An application that scales on a single system might pay more attention to threading, input/output (I/O), garbage collection, and other design decisions that would help the application take better advantage of the additional memory or CPUs. By definition, however, a vertical scaling solution is limited. Theoretically, you can keep adding systems when scaling horizontally; however, you might run out of physical capability in a vertical solution if usage continues to grow. Also, reliability is negatively affected in a vertical scaling solution because there remains a single point of failure. If the system's motherboard goes down, so does your application.

Although application scalability is a major concern for a software developer, you also need to consider database scalability when determining your data access methods. As a developer, you are not expected to be a database architect. However, you should be familiar with possible database decisions and how they can affect your application. Although many scalability solutions for SQL Server do not affect your connection application, some might. A database design consideration that can affect architecture is when separate servers store different data by object types. For example, the Customer database resides on SQLSRV012, the Product database is on SQLSRV089, and each has a different connection requirement.

Regarding scalability and architectures, consider modern cloud-based hosting systems such as Windows Azure to support your scaling requirements. Windows Azure provides immediate scalability and it offers an Autoscaling feature that increases the resources available to your application as usage grows. Windows Azure also provides highly scalable data storage solutions, both relational and NoSql. If you plan to deploy to a cloud solution, you need to ensure that your architectural design takes this into account by abstracting as many of the items that might change as possible.

When you plan an ASP.NET MVC 4 application with scalability in mind, you should consider all scalability options and how they will affect your architecture decisions. Everything from session management to data access will be affected by the decisions you make about how you will support your application's need to handle users. A web farm might affect how you plan to manage session. A database cluster can affect how you manage data access. The earlier you analyze your need for scalability and understand how you will manage it, the less it will affect your application.

Thought experiment

Implementing a government website

In the following thought experiment, apply what you've learned about this objective to predict how you would design a new application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a consultant helping a municipal government bring some of its services onto the Internet. The first application you will work on enables pet licensing over the web. The initial work was done by a volunteer from the local pet shelter. Although it is an attractive website that was very well received by the public, it provides only downloadable forms that must be filled out and returned manually.

Answer the following questions about your approach to enhancing this website:

1. The client received positive feedback on its current website design and wants to keep it. How would you plan to maintain the look and feel across the new application? What components need to be included in your architecture?
2. The department currently files submitted forms in alphabetical order by pet owner. What could you do with the information so the employees would have real-time access to it?
3. You realize you would be best served by creating a separate business layer. How would you manage this layer?

Objective summary

- The ASP.NET MVC framework provides a certain level of SoC by breaking the application responsibilities down into models, views, and controllers. Many aspects of each can be customized if necessary by overriding the base classes and creating your own.
- A view represents the area of the application that will be seen by the user. When coding your views, do not do anything to directly change the model. There are two view engines included with ASP.NET MVC 4: the Razor view engine and the Web Forms view engine. Each provides different ways to write and manage data within the view. The Razor view engine cannot parse ASPX-style coding, and the ASPX view engine cannot parse Razor syntax.
- A controller handles the incoming HTTP requests and sends commands to the model to update the model's state, and sends commands to its associated view to change the view's presentation of the model. A model is the part of the application that handles the data and business logic. It also manages the persistence layer and data access.
- Client-side processing is ideal for work that is specific to the client. It is also important when it can help remove processing from the server. Server-side processing is

recommended when you might be needing to perform the same processing in multiple views or when you need large amounts of data to do the processing and you do not want to have to transfer this information.

- As you design your application, you should also design for scalability. This might have multiple levels of impact upon other decisions that you might be making around caching, server-side versus client-side processing and data access.
- There are three primary ways to manage the creation of a database when using the Entity Framework. The Database First approach enables you to leverage an existing database schema to create entities. Code First and Model First approaches are intended to be used in scenarios in which you are creating a new database schema as part of your project. Code First enables developers to create the object structure first and then use it to create the database schema, whereas the Model First approach enables designers to work in a tool that enables them to build the object model visually and will use that output to create the database schema. The approach you choose should depend on the current status of your database as well as the preferences and skills of the team implementing the initial version.
- The stateless nature of ASP.NET MVC disables some of the built in features of Entity Framework. This will cause you to have to write additional code to make the best use of the *DBContext* class and its approach to data access. With that in mind, it is best to abstract the data access layer. The Repository pattern is one of the most used patterns for managing data abstraction.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing an application in which a section of the main page will be populated by content from a third-party provider. You do not have control over the responsiveness of the client or how much information will be returned with each request. The call is to a RESTful service and will return the information formatted in Extensible Markup Language (XML). What is the best way to implement this application?

   A. Design a model that handles the data call to populate the model. Create a partial view containing only this display area and put an asynchronous service call that returns this model in the partial view controller.

   B. Put a synchronous service call into the main page controller.

   C. Create a partial view containing only this display area and put a synchronous service call in the partial view controller.

   D. Create a partial view containing only this display area and put an asynchronous service call in the partial view controller.

2. You have been given requirements for a dashboard page that will contain summary information from your order processing system in a single display table. However, this summarization needs to be done by combining data requests from the order system, the shipping system, and the accounting system. The dashboard page will be the only place you use this combined data. What is the best way to implement this requirement?

   A. Make the various data requests and compile the information in the controller for display.

   B. Create an individual model for each of the data requests, and then create a view-specific model that calls those models and merges the data.

   C. Create a model for the summary data and handle the various data requests within that model as well as the merging of the data.

   D. Create an individual model for each of the data requests and then merge the data on the client side for display.

3. A significant change has been requested in an application maintained within your company. The application is a classic ASP application that uses custom Open Database Connectivity (ODBC) drivers to connect to a relational data repository on a mainframe computer. The CIO decided that the company needs to replace this 30-year-old system. The team that worked on the original project is made up of developers who have never worked with an object-oriented approach before. Which approaches would be the best to use when designing your initial schema in Entity Framework? (Choose all that apply.)

   A. Create your own custom design because it's too much work to manage an inexperienced staff.

   B. Use Code First.

   C. Use Model First.

   D. Use Database First.

4. You are designing an application that allows employees to change their human resources (HR) information, such as next of kin and direct deposit information. The requirements state that the application should talk directly to the HR systems' database. However, at a recent company meeting, the CFO announced that the company will be converting to a new HR system over the next two years. They will take an additional year to move employees to the new system, one department at a time. How will this affect your design?

   A. It won't; the requirements state that the application should talk directly to the HR systems' database.

   B. You should ensure your naming convention for the database as clearly as possible so you can rework your data calls with minimal changes.

C. You should implement the Repository pattern with the current HR system being the first repository that is built. When the second system comes online, you implement that data access using the same pattern.

D. You should map the model directly to the database calls, anticipating that you will have to change the model as the new system rolls out.

Objective 1.2: Design a distributed application

A *distributed application* is defined as software that runs on two or more computers. The capability to run on multiple computers is critical for systems that are concerned with performance, availability, scalability, and reliability. A typical non-web system following a distributed application architecture would have the client on one machine, the business layer on another, and the data access layer on a third machine. Designing a distributed application in ASP.NET MVC is similar in that you have the client (or view) in the browser, the business layer (the model), and the data access layer behind the model. However, you can abstract this out more and provide the opportunity for more distribution in your architecture. Add in external cloud services such as Windows Azure and get even more distribution across more nodes.

This objective covers how to:

- Integrate web services
- Design a hybrid application (on premise vs. off premise, including Windows Azure)
- Plan for session management in a distributed environment
- Plan web farms

Integrating web services

A common part of a distributed application is the inclusion of web services. Using web services as your data mechanism enables the ASP.NET MVC 4 application to be a consumer of a set of web services that can serve information to other clients, applications, or processes. Adding those web services to the architectural design can furnish a layer of abstraction to the application between the business layer, model, and data layer. It also enables you to incorporate some shared logic in a level below your web application. The historical Microsoft standard for putting services into the application space has been Microsoft Windows Communication Foundation (WCF). With ASP.NET MVC 4, however, the concept of the Web API was introduced, which enables you to bind data using model binding directly to the output. This gives you additional flexibility as you design your application. Different information on your screen can be called from different services or directly onto the page based on user interactions or on jQuery calls. The potential layering is highly flexible.

You can also use ASP.NET MVC 4 to create Representational State Transfer (REST) services. The ASP.NET Web API comes with its own controller called *ApiController*. Choosing the right controller for the right job is important. For creating REST services, you should use the *ApiController* because it returns serialized data. This controller does not use views, but instead reviews the HTML header to find the *Accepts* property being sent with the header to determine how to send the data back. It chooses to return XML or JSON-formatted data based on the *Accepts* property. A regular controller can be configured to produce XML or JSON, but you have to do the serialization and deserialization, whereas the *ApiController* handles this for you.

ASP.NET Web Services (ASMX) is an older Microsoft technology that enables a developer to quickly roll out a Simple Object Access Protocol (SOAP)–based web service. It also eliminates many configuration issues you encounter with other solutions because it simply enables a consumer to make a call to a function. However, you cannot customize certain critical components, such as transfer protocols, security, and encoders. Although ASMX has been superseded by WCF and Web API, many sites still use ASMX to provide their primary web services.

Consuming a web service in ASP.NET MVC 4 in Visual Studio is as simple as using the Add a Service Reference command. By adding a web service, you can use the proxies created and the exposed object set as your model. To do so, you would use a construct such as the following in your controller to instantiate the model:

```
using (ServiceProxy proxy = new ServiceProxy())
{
    model = proxy.GetData(input);
}
```

This approach expects the presence of a Web Services Description Language (WSDL) at the service you are calling. WSDL is a XML format that describes network services that operate on messages that can contain either data or procedure-oriented information. WSDL describes these messages abstractly and then binds them to a concrete communications stack. This communication stack includes network protocol, message type, and message format; and it is defined as an endpoint. Together, a group of related concrete endpoints makes up abstract endpoints. These abstract endpoints can be extended to allow multiple message formats and/or network protocols. Consuming a REST service requires a different technique, but ASP.NET MVC 4 makes it easy to work with.

Listing 1-5 shows how to use the *HttpService* class to get the output from a REST URL.

LISTING 1-5 Using the *HttpService* class to get output from a REST URL

```
private HttpService _httpService;

        public ArticleRepository()
        {
            _httpService = new HttpService();
        }

        public IQueryable<Article> GetArticle s()
        {
            Uri host = new Uri("http://www.yourdomain.com");
            string path = "your/rest/path";
            Dictionary<string, string> parameters = new Dictionary<string, string>();
            NetworkCredential credential = new NetworkCredential("username",
                "password");
            XDocument xml = _httpService.Get(host, path, parameters, credential);
            return ConvertArticleXmlToList(xml).AsQueryable();
        }

        private List<Article> ConvertArticleXmlToList(XDocument xml)
        {
              List<Article> article = new List<Article>();
              var query = xml.Descendants("Article")
                                 .Select(node =>
                node.ToString(SaveOptions.DisableFormatting));
                foreach (var articleXml in query)
                {

                   article.Add(ObjectSerializer.DeserializeObject<Article>(articleXml));
                }
                return article;
        }
```

MORE INFO **WEB SERVICES IN ASP.NET MVC 4**

You can find additional details on ASP.NET Web API's HTTP services for building RESTful applications on the .NET Framework at *http://www.asp.net/web-api*.

As you look at distributed applications, some of the principal needs are communications and a plan for how the various parts of the application will exchange information. Each method of communication mentioned previously, such as SOAP or RESTful services, have a different impact on how you need to design your application. When planning to distribute your application, whether in-premise, off-premise, or some combination, the method you use to communicate between the pieces is critical. Before using a distributed environment, pieces that "just talked to each other" never need development support. As the application spreads out over multiple areas or servers, the communications between the pieces become more complicated.

The closer the different pieces of your application are to each other from a network design, the simpler the communications flow. The farther the pieces of your application are from each other, the more variables that have to be accounted for. Latency, firewalls, and protocol limitations all have to be considered as you plan application distribution. Distribution gives you many advantages but they come at a cost. By recognizing the costs up front, you can better plan how to minimize the impact.

Designing a hybrid application

A *hybrid application* is an application hosted in multiple places. The term has become popular with the growth of Windows Azure to represent an application in which one part is hosted within the company's network and another part is hosted in Windows Azure. This kind of solution makes sense if the application will access private or sensitive data, runs well but might need additional periodical capacity, or is not designed in a stateless fashion. A hybrid approach to application development and deployment is also a way to implement a good migration or expansion strategy.

NOTE **DEFINITION OF HYBRID APPLICATION**

Before the growth of Windows Azure, the term "hybrid application" was sometimes used to describe a web application that supported both the ASPX and Razor view engines to render content. Microsoft has since emphasized using the term as an application hosted in multiple places.

There are two primary hybrid patterns. The first is a client-centric pattern in which the client application determines where the application needs to make its service calls. This pattern is generally the easiest to code, but it is also most likely to fail. Applications built with this approach are the most fragile because any change to either server or client might require a change to the other part. The second primary pattern is a system-centric approach, in which you take a more service-oriented architecture (SOA) approach. It ideally includes a service bus, such as Windows AppFabric, which will distribute service requests as appropriate whether it is to a service in the cloud, on-premise, or at another source completely such as a partner or provider site. (You will learn about AppFabric in the "Distribution caching" section later in the chapter.) Figure 1-5 shows how this service bus distributes requests.

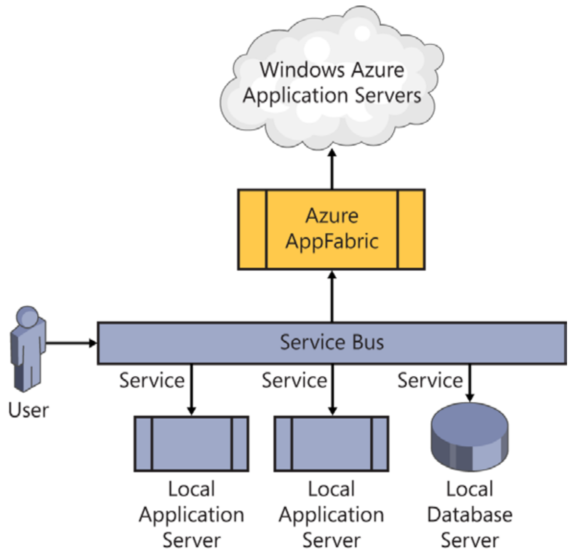


FIGURE 1-5 A hybrid approach using a service bus

When you consider a federated approach, whether to Windows Azure, SQL Azure, or other distributed architectures, there are some factors you need to consider in the planning phase. Connection resiliency becomes a point of concern when building a distributed application. A solution that's all on-premise generally has low latency and good connection properties. You are not guaranteed either when working with a hybrid application. Whether a centralized client or a distributed one, the code needs to be able to handle the riskier nature of the communications and understand the concept of a retry. Authorization and access are also complicated by going to a hybrid solution because you need to manage access into multiple domains. Windows Azure comes with the capability to help you manage authorization and access, but this is something you need to plan for when you design the architecture. Finally, you must plan for consistency and concurrency. In a service-based architecture, you need to plan for sequential message handling and life cycles. Once again, Windows Azure provides tools to manage sequential message handling and life cycles, and this type of management must be a part of your plan.

MORE INFO **HYBRID APPLICATIONS IN WINDOWS AZURE USING THE SERVICE BUS**

The Windows Azure team provides many useful documents and samples on using the service bus in a hybrid application at *http://www.windowsazure.com/en-us/develop/net/tutorials/hybrid-solution/*.

You will deploy your ASP.NET MVC code as a single application. Where that application and its external connections reside will determine how hybrid the application will be. You can take several approaches to building your application as a hybrid application. Consider a few scenarios for using ASP.NET MVC in a hybrid environment. In one, you host your application in your network and access ancillary services in Windows Azure. Or you might host your ASP.NET MVC application in Windows Azure and keep confidential information in your own network. The decision lies with where you think your potential issues might be: whether you are looking at Windows Azure to provide robust and scaling systems on which to deploy your application, whether you are looking at one of Azure's storage options to manage your data, or whether Azure might be hosting an ancillary service on which your ASP.NET MVC application might have dependencies.

One of the primary concerns in cloud-hosted systems is security. Windows Azure has strong standards about how it maintains security, including prevention of data leakage and data exposure. However, if you access data from another location, you might open security holes in your system. To counteract this vulnerability, a traditional on-premise solution can put the database in a protected location from which it does not allow connections from the Internet. However, using a hybrid solution, where the database is hosted elsewhere, makes that impossible. If you are going to accept data from a different network, you will have an increased security footprint.

Scalability, latency, cost, robustness, and security are considerations as you evaluate a hybrid solution. There is no one answer on how best to manage all aspects of your application. You need to analyze each piece of your application and determine where it makes the most sense to be hosted.

Planning for session management in a distributed environment

A session is stored on the server and is unique for a user's set of transactions. The browser needs to pass back a unique identifier, called *SessionId*, which can be sent as part of a small cookie or added onto the query string where it can be accessed by the default handler.

You can approach sessions in ASP.NET MVC 4 in two different ways. The first is to use session to store small pieces of data. The other is to be completely stateless and not use session at all. Because ASP.NET MVC lies on top of ASP.NET, you can access session information and use it throughout the application. The session is available for use in your controllers as needed; however, ASP.NET MVC 4 is designed to run in a stateless manner. It is designed to be able to transfer all the information the application needs each time it makes a call. By being able to call an action on a controller and pass in an object, ASP.NET MVC 4 can control everything it needs every time it makes a call to the server.

Session management in a distributed environment is more complicated than a traditional session management scenario because a single page might get information from multiple domains and servers. Session management through a service bus can also be unreliable. The surest way to manage state in a distributed application is to implement a sessionless design in which you use a query string or hidden input form value to transmit information to the end handler. Regarding a sessionless state solution, the key determination is where the state information will be stored. Because it will not be stored in the session, you need to determine whether it should be maintained on the client side or on the server side.

In a distributed environment, it is important to remember that that requests can be distributed among different servers when using a session. There are three modes of session management available in Microsoft Internet Information Services (IIS): InProc, StateServer, and SQLServer. They each have advantages and disadvantages.

You can configure IIS to manage the *SessionId* either way. InProc mode is the default setting and means that the web sessions are stored in the web server's local memory. This option provides the best performance but is not clusterable. In StateServer mode, session information is stored in memory on a separate server. When configuring the state server in IIS, you need to enter the connection string to the server. All servers that use the same state server have access to the state information. SQLServer mode has the same advantage as StateServer in that the session information is shared across multiple servers. It has a performance impact, however, because there needs to be a call to a SQLServer and it will add latency to the session access.

Planning web farms

Web farms are groups of servers that share the load of handling web requests. In a simple system design, a single server typically supports all application requests. However, as the number of requests to your server increases, the less capable your server becomes in processing all requests. The most common way to solve this problem is to use multiple servers that host the application together. Doing this enables you to balance the traffic between the available servers rather than relying on a single server to fulfill them all. Figure 1-6 shows a simple web farm.

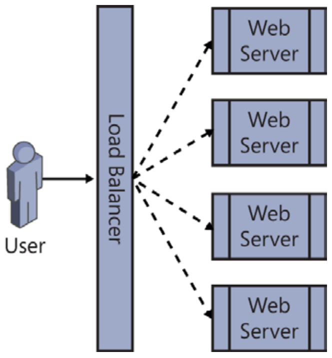


FIGURE 1-6 A web farm

Using web farms with an ASP.NET MVC 4 application gives you some flexibility for deploying the various parts of your application. Because SoC is inherent in the MVC architecture, you can locate components of the application on different servers. You can place views on one server and the model on another, as long as you manage communications between the two. ASP.NET MVC is designed to be flexible, enabling you to run an application with separate parts as well as together as a single application.

There are many advantages of using a web farm, one of which is high availability. If a server in the farm goes down, the load balancer redirects all incoming requests to other servers. A web farm also improves performance by reducing the load each server handles, thus decreasing contention problems. The ability to add in servers to the farm also provides better scalability.

The impact of going to a web farm can be managed in several ways. The biggest change is that the architect cannot just assume that the default session will be available. Although some load balancers can match a particular server to a session, referred to as a "sticky session," it is better to assume that the load balancers cannot ensure that—and plan accordingly. As mentioned previously, the default setting for session mode in IIS is InProc, which stores session data in the memory of the local machine. This makes the information in that session unavailable to the other servers in the farm. In web farm mode, you need to be able to share the session among all the servers in the farm. This can be done by selecting the session mode of *SessionMode OutProc* (*StateServer* or *SQLServer* mode). If you are using sessions in a web farm, an *OutProc* setting enables the load balancer to send connections to a new server and still have the session information available.

Thought experiment

Building a geographically distributed application

In the following thought experiment, apply what you've learned about this objective to predict how you would design a new application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are the new technical architect at a large custom home building firm with offices in North America, South America, and Europe. Your firm is expanding, both by opening new offices and by buying smaller custom home builders. Although each main geographic region stores its own data in its own systems, your CIO wants you to build an application that displays information from each region in different widgets on a dashboard. She does not want SQL queries run from the corporate office where the dashboard will be hosted.

1. **You will have to deploy some software in the various regions. What will the software do?**
2. **What are some primary concerns of building an ASP.NET MVC application that gathers information from such disparate sources?**
3. **What is the benefit of adding Windows Azure AppFabric?**

Objective summary

- Web services are a traditional way to transfer information from one system to another on the Internet. They have been managed in several different ways over the years. ASMX services use WSDL to communicate with consumers about endpoints, protocols, and message formats. WCF is a SOAP-based protocol and is still the primary communications mechanism, but ASP.NET MVC 4 Web API has made advances in RESTful services. Web API also uses the ASP.NET MVC pattern for managing HTTP requests.
- Designing for a distributed environment can be one of the most complex tasks a developer take on. Each part of the application that will be deployed separately needs to be able to manage message sending and receiving. This issue occurs whenever you separate items, such as the database from your ASP.NET MVC application, or when you locate the view on one server and the model on another. Communications between all parts of the application are critical and need to be accounted for while the application is being built.
- Different types of web services can be used in distributed environments. WCF and Web API are two out-of-the-box frameworks that help you design and implement web services.
- A hybrid application is an application that is partially deployed on-premise and partly off-premise. When working in this kind of environment, you need to be aware of the riskier nature of communications and manage the concept of a retry. You can split the application and host the parts in different locations. The web server portion can be on-premise while the data management area is off-premise, or vice versa.
- When you design for a distributed environment, you will find state management to be a point of concern, especially when using sessions. Some design consideration should go into how you will implement sessions or whether you should design the application to be sessionless.
- A distributed environment can improve availability, reliability, and scalability. One of the ways you can do that at the web server level is to use a web farm, in which you have multiple servers working in parallel to manage the various user requests.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are developing an application. One requirement is that part of your data access layer needs to be available to a third party, that wants to get this information from a REST URL in XML. Your company does not have experience with web services, but you have several websites running ASP.NET MVC 4. How could you design and provide these new services? (Choose all that apply.)
   - **A.** Task an individual on staff to learn WCF, and have this individual develop and deploy these new services using WCF.
   - **B.** Use the Web API to create REST services using *ApiController* in which the serialization type is defined by the *Accepts* property of the browser.
   - **C.** Build a basic ASP.NET MVC 4 project in which the view simply passes through the information provided by the controller, and the controller manages the code for serializing the response.
   - **D.** Create an ASP.NET ASMX services file to get, serialize, and return the data.
2. How could you traditionally consume an ASMX web service from your application? (Choose all that apply.)
   - **A.** Generate a proxy by selecting Add Reference In Visual Studio.
   - **B.** Create an *HttpService* and connect using *Get(URL)*.
   - **C.** Generate a proxy by selecting Add A Service Reference in Visual Studio.
   - **D.** Create a WCF proxy class.
3. What are examples of hybrid applications using Windows Azure? (Choose all that apply.)
   - **A.** An application where the local network hosts the IIS server while the database is being run from the corporate IT office
   - **B.** An application where Windows Azure is used to host the IIS process, and Windows Azure SQL is used to store the data
   - **C.** An application where the IIS process is run on a local web server, whereas the data is stored in Windows Azure SQL
   - **D.** An application where the web part of the application is run on Windows Azure, whereas the confidential data is stored in the company's network

Objective 1.3: Design and implement the Windows Azure role life cycle

Windows Azure is a Microsoft cloud computing platform used to build, deploy, and manage applications through a global network of Microsoft-managed data centers. Windows Azure allows for applications to be built using many different programming languages, tools, and frameworks; and makes it possible for developers to integrate their public cloud applications in their existing IT environment.

This objective covers how to:

- Identify startup tasks (IIS configuration [app pool], registry configuration, third-party tools)
- Identify and implement *Start*, *Run*, and *Stop* events

Understanding Windows Azure and roles

Windows Azure provides both platform as a service (PaaS) and infrastructure as a service (IaaS) services, and is classified as the "public cloud" in Microsoft's cloud computing strategy.

NOTE **PAAS AND IAAS**

With PaaS, cloud providers deliver a computing platform, typically including an operating system, a programming language execution environment, a database, and a web server. IaaS offers virtual machines.

One way to conceptualize Windows Azure is as a large data center running offsite. It is managed by Microsoft, so you do not have to worry about typical system administration chores such as upgrades and patching. What it does give you is a highly flexible and scalable computing environment running a familiar operating system. This is especially relevant when you consider the testing and production phases of your ASP.NET MVC application development life cycle.

There are three different types of solutions available in Windows Azure: Virtual Machines, Web Sites, and Cloud Services. Virtual Machines provide the most general solution. Virtual Machines in Windows Azure function like a virtual machine that you might be running in your local environment. Virtual Machines give you the most control over the environment, so they are generally a good choice for development and testing, and for running off-the-shelf applications in the cloud. Because you control the environment, you can set up Virtual Machines that look like your on-premise virtual machines. This enables an Azure Virtual Machine to be used for disaster recovery.

Web Sites is a good choice for simple web hosting, and is a good solution for hosting and running your ASP.NET MVC 4 applications without the overhead of maintaining a full virtual machine. Web Sites enables a scalable experience, with fast deployment and an almost immediate startup, and you can upgrade or downgrade this solution quickly and easily as needed.

Cloud Services, which is a strictly PaaS approach, was the initial deployment model for Windows Azure.

All three Windows Azure execution models have pros and cons. Making the best choice requires understanding the models, knowing what you're trying to accomplish, then choosing the one that's the best fit.

Identifying startup tasks

Windows Azure startup tasks are used to perform actions before a role starts. There are three types of roles in Windows Azure: Web, Worker, and VM. If you plan to run IIS in Windows Azure, you should use a Web role. If you are going to run middle-tier applications without IIS, a Worker role will fulfill your need. If what you want to do in Azure is beyond the scope of the Web or Worker roles, Microsoft gives you complete access to the VM instances themselves—the VM role.

With startup tasks, you can register COM components, install a component, or set registry keys, for example. Startup actions are also commonly used for starting long-running processes. Startup tasks are available only for Web and Worker roles; VM roles cannot manage startup tasks.

Startup tasks are defined in the *Task* element, which is a node in the *Startup* element of the ServiceDefinition.csdef file. A typical startup task is a console application or a batch file that can start one or more Windows PowerShell scripts. You can use one or more environment variables if you need to pass information into the task. When you need to get data from the task, you can store a file containing the information to a well-known location on the file system. Startup tasks run each time a role recycles in addition to when a server reboots. Startup tasks have to end with an error level of zero (0) for the startup process to complete. When startup tasks end with a non-zero error level, the role does not start.

When you consider a Windows Azure deployment, consider the differences between running an application on a remote system in which you do not have full privileges versus running it on a server in which you have full control. Although you are ceding the responsibility for server uptime to Windows Azure, you are also ceding some control over what is happening on the server. Some secondary applications you might be running to support your application or that offer additional functionality might not work the same way. If you need to ensure that secondary applications are running while your application is running, you need to start the applications through a startup task or other process.

The procedure followed by Windows Azure when a role starts is the following:

1. The instance is marked as *Starting*. It will no longer receive traffic.
2. Startup tasks are executed according to their *taskType* attribute:
   A. Simple tasks are executed synchronously.
   B. Background and foreground tasks are started asynchronously. This is in parallel with the startup task.
3. The role host process is started and the site is created in IIS.
4. The task calls the *Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint.OnStart* method.
5. The instance is marked as *Ready* and traffic is routed to the instance.
6. The task calls the *Microsoft.WindowsAzure.ServiceRuntime.RoleEntryPoint.Run* method.

The AppCmd.exe command-line tool is used in Windows Azure to manage IIS settings at startup. The tool enables you to add, modify, or remove settings from both web applications and websites. You need to add the appropriate AppCmd.exe commands to the appropriate task if you plan to run the task at startup.

Remember that a startup task can be run more than once, and misconfiguring AppCmd.exe commands can result in runtime errors. For example, a common error is to add a Web.config section in the startup task. When the task is run again, it throws an error because the section already exists after the initial run. Managing this kind of situation requires that your application monitor both its internal and external statuses. Regarding the Web.config issue, for example, the errorlevel is 183. Your application should monitor for that errorlevel and ensure that, if received, it is handled appropriately, and the startup can continue. There will be times when you need the errorlevel to be elevated to the client when an error has occurred that should be reported. However, there will also be times when you will want to handle the error internally.

MORE INFO **WEB.CONFIG FILE**

See the "Apply configuration settings in the Web.config file" section later in this chapter for information on configuring the Web.config file.

Another consideration is marking a task as Background. Doing so prevents Windows Azure from waiting until the task completes before it puts the role into a Ready state and creates the website. You can set a task as background as shown in the following example:

```
<Startup>
    <Task commandLine="Startup\ExecWithRetries.exe
            &quot;/c:Startup\AzureEnableWarmup.cmd&quot;
            /d:5000 /r:20 /rd:5000 &gt;&gt; c:\enablewarmup.cmd.log
            2&gt;&gt;&amp;1"
        executionContext="elevated" taskType="background" />
</Startup>
```

As you plan your scripts, remember that the names of websites and application pools are not generally known in advance. The application pool is usually named with a globally unique identifier (GUID), and your website is typically named *rolename_roleinstance number*, ensuring that each website name is different for each version of the role. You can use the search functionality in AppCmd.exe to search for the web role name and then use it as a prefix for the name of the site. You can pipe this output to AppCmd.exe to manage the configuration, as follows:

```
> %windir%\system32\inetsrv\appcmd list sites "/name:$=MyWebRoleName*" /xml |
    %windir%\system32\inetsrv\appcmd set site /in /serverAutoStart:true
```

The following example for the application pool lists the site, the apps within that site, and the application pools for those apps; and then sets a property on those application pools:

```
> %windir%\system32\inetsrv\appcmd list sites "/name:$=MyWebRoleName*"
/xml |
 %windir%\system32\inetsrv\appcmd list apps /in /xml |
 %windir%\system32\inetsrv\appcmd list apppools /in /xml |
 %windir%\system32\inetsrv\appcmd set apppool /in /enable32BitAppOnWin64:true
```

The types of objects available through AppCmd.exe are listed in Table 1-2.

TABLE 1-2 Objects available for use in AppCmd.exe

Object	**Description**
Site	Virtual site administration
App	Application administration
VDir	Virtual directories administration
Apppool	Application pools administration
Config	General configuration sections administration
Backup	Management of server configuration backups
WP	Worker process administration
Request	Active HTTP request display
Module	Server module administration
Trace	Server trace log management

AppCmd.exe enables you to manage different aspects of your IIS configuration. However, other common tasks take place within startup tasks, such as managing the registry. Some single-use web servers have various configuration information stored within the Windows registry rather than in configuration files. This keeps the information secure in case someone gets file-level authority to your server, and it offers a faster response time than file-based configuration settings. Because the configuration information in the registry needs to be changed upon a software release, the easiest way to maintain this information is through a script.

Managing the registry is straightforward. You can either create a small executable application that you run from the startup task or create a script that will do the same thing. Running it in a startup task is the same process you use to run AppCmd.exe. Although the registry keys do not exist in the role by default, you should check before attempting to change them.

Windows Azure virtual machines are stateless, which means the local drives are not used when actions are taken on what would normally be persisted information. Thus, saving registry information will not be persisted the next time the role restarts. For the same reason, other applications that you might need to have installed will not be available, either. Perhaps you use a third-party log analysis tool or other application that needs to be installed rather than simply copied over as part of an application deployment. These installations have to be managed the same way as registry or IIS changes.

MORE INFO **WINDOWS AZURE LIFE CYCLE**

Channel 9, which has development-related videos and is part of MSDN, has a two-part series on the Windows Azure life cycle at *http://channel9.msdn.com/posts/Windows-Azure-Jump-Start-03-Windows-Azure-Lifecycle-Part-1* and *http://channel9.msdn.com/posts/Windows-Azure-Jump-Start-04-Windows-Azure-Lifecycle-Part-2*.

Identifying and implementing *Start*, *Run*, and *Stop* events

There are many conceptual similarities between the *OnStart* method and a startup task:

- They both have the same time-out. If you are not out of either function, the execution of role startup continues.
- They both are executed again if the role is recycled.
- You can configure both to process ahead of the role.

Significant differences between the *OnStart* method and a startup task are these:

- A startup task runs in a different process, which enables it to be at a different level of privilege than the primary point of entry. This is useful when you need to install software or perform another task that requires a different privilege level.
- State can be shared between the *OnStart* method and the *Run* method because they both are in the same application domain (AppDomain).
- A startup task can be configured as either a background or foreground task that runs parallel with the role.

After all the configured startup tasks are completed, the Windows Azure role begins the process of running. There are three major events you can override: *OnStart*, *Run*, and *OnEnd*. Figure 1-7 shows the life cycle of the role.

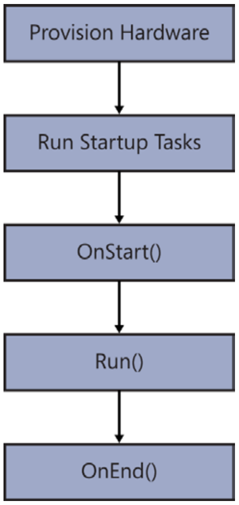


FIGURE 1-7 Flow of Windows Azure processing

If you need to add functionality into the *OnStart* method, you should consider overriding it, which enables you to run code that manages initialization needed to support your role. The following code example shows how you can override the *OnStart* method in a worker role:

```
public class WorkerRole : RoleEntryPoint
{
    public override bool OnStart()
    {
       try
       {
          // Add initialization code here
       }
       catch (Exception e)
       {
          Trace.WriteLine("Exception during OnStart: " + e.ToString());
          // Take other action as needed.
       }
       return base.OnStart();
    }
}
```

When the *OnStart* method is called, Windows Azure sets the role status to Busy. When the role is Busy, it is ignored by any external processes, such as the load balancer. The Boolean value returned from the *OnStart* method determines whether Windows Azure continues the startup process and calls the *Run* method. If *OnStart* returns *true*, Windows Azure assumes the *OnStart* method was successful and allows the role to run. When *OnStart* returns *false*, Windows Azure assumes a problem occurred and immediately stops the role instance.

In Windows Azure, the *Run* method is equivalent to the *Main* method in that it starts the actual application. You do not typically need to override the *Run* method. If you do, make sure your code will indefinitely block because a return from the *Run* method means the application has stopped running and that the process should continue through to shutdown.

After a value is returned from *Run*, Windows Azure raises the *Stopping* event and calls the *OnStop* method. This ensures any necessary shutdown and cleanup processes are completed before the role is stopped and made unavailable. Override the *Run* method to run code for the life of the role instance. Because the *Run* method is void, your override of the *Run* method can run in parallel with the default *Run* method if desired. You might want to do this if you want to have background tasks running throughout the life of your application, such as automated file transfers or other processing. The following code example shows how to override the *Run* method:

```
public override void Run()
{
    try
    {
       Trace.WriteLine("WorkerRole entrypoint called", "Information");
       while (true)
       {
          Thread.Sleep(10000);
          Trace.WriteLine("Working", "Information");
       }
       // Add code here that runs in the role instance
    }
    catch (Exception e)
    {
       Trace.WriteLine("Exception during Run: " + e.ToString());
       // Take other action as needed.
    }
}
```

A Web role can include initialization code in the ASP.NET *Application_Start* method instead of the *OnStart* method. The *Application_Start* method is called after the *OnStart* method.

Override the *OnStop* method to run code when the role instance is stopped. The following code example shows how to override the *OnStop* method:

```
public override void OnStop()
{
    try
    {
       // Add code here that runs when the role instance is to be stopped
    }
    catch (Exception e)
    {
       Trace.WriteLine("Exception during OnStop: " + e.ToString());
       // Take other action as needed.
    }
}
```

When you override the *OnStop* method, remember the hard limit of five minutes that Windows Azure puts on all non-user-initiated shutdowns. This helps ensure that applications that are forced to shut down do so cleanly, without affecting the capability of the role to successfully end. The process is terminated after that period, so if your code has not completed within that time frame, it is terminated. Because of the hard stop, you need to make sure that either your code can finish within that period or that it will not be affected if it does not run to completion. The role will be stopped when the *OnStop* method finishes executing, whether the code completes on its own or it reaches the five-minute limit.

MORE INFO **WINDOWS AZURE WEB ROLE**

You can read an overview of creating a hosted service for Windows Azure and get links to other services in Windows Azure at *http://msdn.microsoft.com/en-US/library/gg432976.aspx*.

Thought experiment

Investigating Windows Azure

In the following thought experiment, apply what you've learned about this objective to predict how the following architecture approach would perform. You can find answers to these questions in the "Answers" section at the end of this chapter.

Your company wants to refactor its mission critical line-of-business (LOB) application to make it more robust and scalable as well as to increase performance. The CIO wants to investigate hosting the item in Windows Azure. Currently, the application has a lot of maintenance processes running in the background, such as processes to ensure that there is no orphaned data by running queries against the database, processes that check website availability with an email sent if a website is non-responsive; and a process that counts how many users logged in during the previous hour and sends an email to the IT support staff.

1. **Where in the Windows Azure process would it make the most sense to put the orphaned data check?**
2. **Would it make more sense to put these processes in a Worker role or within the Web role?**
3. **Assuming that these processes were all C# console programs, do you think it would be difficult to migrate them for use in the cloud? Why or why not?**

Objective summary

- Windows Azure is a cloud-based offering from Microsoft that enables companies and developers to have access to a fully configurable, flexible hosting and services environment. It enables ASP.NET MVC developers to work in a Windows-based system, yet offers the flexibility and scalability of a cloud-based service.

- Azure is a stateless system, so any changes to the system whenever a role is run is not persisted to the next run. Although many applications might not be affected by this consideration, some will be, and consideration has to be given as how to manage this. A traditional server in your data center has any additional needs configured and is available every time that server is restarted. That is not the case for Windows Azure.
- A developer can give a role a set of startup tasks to be run, in a preconfigured order as the system starts up. AppCmd.exe is a flexible Windows Azure-provided tool that enables you to manage your startup tasks. These startup tasks can be batch files, console files, or batch files that run Windows PowerShell scripts. You can use the startup tasks to install any additional software or third-party tool that you might need, make changes to the registry, or handle any other specific needs to support your ASP.NET MVC application.
- After the startup tasks are completed, the *OnStart* method is called. You can override the *OnStart* method to implement other functionality. You need to make sure that you return true from the method, or else the startup will stop with an error.
- After the *OnStart* method has returned, the process calls *Run*. Because *Run* is a void method, you can use the override to have applications start that can run in parallel to the main application.
- Upon shutdown, the process calls the *OnStop* method. This is a void method as well, and would typically be used to close and clean up any ancillary processes you might have started in the *OnStart* or *Run* methods.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. What occurs if an unhandled error is fired on a startup task?
   - **A.** The startup role consumes the error during its load; if there is no event handler configured, it is set to *Handled* as the task completes.
   - **B.** The startup is cancelled and the role does not start.
   - **C.** The *OnStop* method automatically runs.
   - **D.** The startup task goes to the lowest security setting and continues to run, if possible.
2. AppCmd.exe is an application that helps you work with which objects? (Choose all that apply.)
   - **A.** *Site*
   - **B.** *Users*
   - **C.** *Config*
   - **D.** *App*

3. Which of the following are valid reasons for overriding the *Run* method? (Choose all that apply.)
   - **A.** Creating and starting a messaging service that will work in parallel with the Web role to manage queued messages
   - **B.** Creating an always-running service that periodically makes HTTP calls to other websites to determine their availability
   - **C.** Managing error handling for the application
   - **D.** Starting and supporting a logging application for use by the Worker role

Objective 1.4: Configure state management

A software application needs to store information. For example, even remembering the information typed into a text box requires some state to be maintained. In an ASP.NET MVC 4 application, the browser takes care of that part of the application state. You need to determine how you will maintain information from one screen to the next. The primary way of communication over the Internet is HTTP, which is intended to be a stateless protocol. It doesn't know anything about the last request, by design, so any state you need to manage has to be done in code. ASP.NET MVC 4 is designed to comply with the stateless nature of HTTP.

Not only do you need to decide what information you need but you also have to figure out how you want to store this information. Because of the separation between the client and server, you have some choices about where to store user-specific information. Other items that matter are how many servers the application can be deployed on and whether information will need to be shared across them. Performance can also be affected by your choice because adding complexity to state management tends to make the state maintenance process less responsive because you will be going from direct memory calls to calls into another system, whether it is a database or another server.

This objective covers how to:

- Choose a state management mechanism (in-process and out of process state management, *ViewState*)
- Plan for scalability
- Use cookies or local storage to maintain state
- Apply configuration settings in the Web.config file
- Implement sessionless state (for example, *QueryString*)

Choosing a state management mechanism

Your first decision regarding state in ASP.NET MVC is not how you will manage it but whether you will use state at all. HTTP is stateless for a reason, as is ASP.NET MVC 4. By not having to keep an open connection to a requestor or not having to remember anything about a user's last connection, a web server can handle many more concurrent users. Imagine a Fortune 500 company's intranet site with thousands of users using it concurrently. It would take dozens of servers to manage the intranet if each user opened a connection and kept it open throughout the day. The stateless nature of HTTP enables a server to support a connection only until it handles a request and sends a response.

Web Forms supports multiple built-in ways to manage state and does its best to enable state by default throughout the application. The main way that it does this is through the concept of a view state. The *ViewState* is a construct that gathers pertinent information about the controls on a page and stores them on the page in a hidden form field. This ensures that every post request to the server includes the view state; in other words, a Web Forms application has the capability to carry its state around the web application with it, storing information as needed. This was done as a way to circumvent the concept of stateless as defined in HTTP.

ASP.NET MVC, on the other hand, embraces the nature of a stateless application. All it expects when a request comes in is enough information to give it context. This could be a user and the object being manipulated, an identifier to what product should be displayed, or an identifier to a stored shopping cart. Traditionally, a lot of this information is stored in the session so that the application can pull it out as needed. However, much of the information in a session might not be used on every request.

In an ASP.NET MVC 4 application, state information can be stored in the following locations:

- *Cache,* which is a memory pool stored on the server and shared across users
- *Session*, which is stored on the server and unique for each user
- *Cookies*, which are stored on the client and passed with each HTTP request to the server
- *QueryString*, which is passed as part of the complete URL string
- *Context.Items*, which is part of the *HttpContext* and lasts only the lifetime of that request
- *Profile*, which is stored in a database and maintains information across multiple sessions

The *Cache* object provides a broader scope than the other state management objects as the data is available to all classes within the ASP.NET application. The *Cache* object enables you to store key-value pairs that become accessible by any user or page in that application domain. It is in-process in that although it goes across all users and pages, it is confined to that particular application domain on an individual server. If you consider using *Cache* in a web farm setting, you need to ensure that your server has its own copy of the cache. You cannot assume that a value is cached simply because the value was used as part of the last request; the request might be connecting to a different server that never called the value in the first place.

The session was described in Section 1.2. When you are implementing session state, you can use the default stores that come with ASP.NET or you can create your own session-store provider. Inheriting the *SessionStateStoreProviderBase* class enables you to create your own session provider to support situations in which the default session store is inadequate. If your ASP.NET MVC application runs on an Oracle database, for example, there is no built-in support for managing state that is shared by multiple servers. If you want to store the session information in a table in your Oracle database, you need to write a custom provider. Follow these steps to configure the choice in IIS Manager:

1. Open IIS Manager and navigate to the level you want to manage.
2. In Features View, double-click **Session State**.
3. On the Session State page, in the Session State Mode Settings area, click **Custom**.
4. Click **Apply** in the Actions pane.

Cookies are small snippets of information stored on the client side and can persist across sessions. They are individualized to a particular domain or subdomain, so with careful planning you can use cookies across a web farm. Cookie information is sent to the server and returned from the server with every request. The sizing can have an impact and it is always part of the HTTP request. A cookie is available in *HttpContext.Request.Cookies* when reading and *HttpContext.Response.Cookies* when storing the value. A cookie can also be set with an expiration date so that the data stored in the cookies can have a limited time span.

A query string is information that can be used by only one user. Its lifetime is by request unless architected to be managed differently. The query string is appended to the URL, and the interaction between the query string and the routing table is straightforward: The query string is not part of the route data and thus is ignored by the routing engine. You can access the data in the *HttpContext.Request.QueryString["attributeName"]* on the server and from the client side by parsing *window.location.href*. This information is also visible to the end user, so care should be taken about what kind of information is sent. Putting unencrypted personal or secure information in the query string means that, theoretically, anyone can see it because it is not encrypted over HTTPS. However, ASP.NET MVC supports several encryption schemas that enable you to encrypt data as necessary for inclusion into the query string that will make the use of the query string more secure.

Context.Items contains information that is available only during a single request. Typically, it is used to add information to the request through the use of an HTTP module in which you can add some information to the request that will be available to the other modules and to the handler. An example of this is authentication, which is handled by a module. It authenticates the user of the request, and the results of the authentication request are made available for use through the rest of the request-response process.

Profile information is stored in a database by user name and can be accessed through *HttpContext.Profile["miscellaneousData"]*. The profile is part of the Membership and Roles provider, and you need to configure a provider in the Web.config file. The use of a profile means you have to be using the ASP.NET membership provider because it is based on information stored in the membership.

As you approach your ASP.NET MVC application design and consider state management, you have to evaluate the amount of data you want to keep and where you will keep it. If your application requires most of its state to be accessed on the client side, ASP.NET MVC does not offer any special advantages. But because state is almost always needed on the server to support business needs, the flexibility of ASP.NET MVC enables you to take advantage of most of the state management processes described in this section.

If the state information is for display purposes only, you can maintain the information on both the client and server. Caching state information on the client eliminates the need to send it back as part of the rendered HTML with every call and increases performance. Keeping it on the client side can also enable manipulation to occur without a server call until the process finishes, such as the use of a wizard in which the application has a three- to four-step process to gather data from a tabbed data entry form. Keeping this kind of state on the client side until final submission will enhance the user experience by enhancing responsiveness. Keeping it on the server side enables you to use ASP.NET MVC to work with the data; however, you have to make the state part of the HTTP response, and you have to perform initial server requests on all the state changes.

Although keeping state information on the client side has its advantages, there are also some drawbacks. Consider when multiple individuals might be editing a particular item. As more work and management is done on the client without communication back to the server, the more likely collisions will occur when saving the data. You have to manage this risk in the software, such as by locking an item after someone requests it for editing. You can also ignore the risk, knowing that the last save always overwrites previous saves.

After you determine where the state will be used, you then need to determine how to store it. If you will use the information mainly on the client, you should look at local storage. (Local storage is covered in the "HTML5 Web Storage" section of this chapter.) If you will maintain state on the server, you need to evaluate the scope that the state covers and the size of the dataset you will maintain. If the scope of the state is limited to an individual user, your solution will be different from where the scope is for the application. The status of the application, from a system wellness point of view, would be a good candidate for having state maintained in a location that is across the application, such as the *Cache*.

The size of the information to be maintained is another consideration because some of the potential maintenance locations have size limitations. If you need to maintain a large amount of information, cookies and query strings might not work for you. If, however, you need to store a few snippets of information, perhaps 30 to 40 fields in a data entry form, cookies can work well because the user doesn't see the information. If visibility isn't a concern, or is even a bonus, the query string is a good choice. The session is the most commonly used method for storing information between requests and has many built-in facilities for managing it from both a server administrative perspective and when developing the application. Although it can't handle an unlimited amount of data by default, it can be configured to store information in an SQL database, which allows more flexibility for the amount of data you might need to keep track of while maintaining state. It also has a relatively small footprint when the information is going through the request-response process because it does not transfer the information, just a reference ID so the server can find the data as needed.

Planning for scalability

ASP.NET MVC has several characteristics that make it a valid choice when concerned with the scalability of your application. Its very nature enables the creation of clean and simple HTML without additional and unused information. This is especially noticeable when comparing the typical ASP.NET MVC output to ASP.NET Web Forms pages output. This gives additional opportunity for load-balancer caching and other downstream scalability support. It also means there will be less time processing the page and less bandwidth used to transfer the page to the client, all of which will help with scalability concerns.

When planning for scalability, you need to understand what kind of state information you will need to maintain. An e-commerce application might need to maintain only a few pieces of information. Other solutions, however, might need to maintain hundreds of pieces of information in a complicated set of object graphs. Each of these needs indicates a different solution. At a minimum, you should assume you will need minimal scaling and plan accordingly.

NOTE **ACHIEVING SCALABILITY**

In the web world, scalability is usually achieved by adding additional servers across the breadth of the web application layer so that each server handles less of the overall demand. Although this enables your application to support more users, it can also cause a lot of trouble if you haven't correctly architected for the ability. The default settings for state management assume a one-to-one relationship between the client and the server and will lead to an inability to manage scalability and reliability as required.

You can use an OutProc, a StateServer, or a SQLServer *session* or a sessionless solution.

As long as all servers in a web farm are configured to use the same state server or SQL Server, using an OutProc session to access state information stored in a session should get consistent responses, regardless of the server calling the information and serving the page.

You can manage a sessionless state solution in several ways. The key determination is where the state information will be stored, whether on the client side or on the server side. Storing it on the client and sending the information to the server as needed is one solution. Additional coding is required on the client side, but going sessionless while still needing state implies extra coding somewhere. You also have to check browser versions. A client-side state storage system requires the use of local storage or cookies, and some browser versions do not support all the client-side storage mechanisms.

An example of using sessionless connectivity while maintaining state on the server is through the use of a profile. It implies the user has logged in to the system and is recognized as authenticated. This enables you to piggyback off the HTTP Authorization header to get the information you need for state management. If you use a different approach, you still have to pass some kind of identifier back and forth between requests for the server to properly identify the requestor. The identifier can be set as part of the query string, as part of the URL, as a hidden input value on the form, or as a cookie value; and you must code both the client and server correctly. When replacing the session framework, you also have to ensure that your identifier is guaranteed to be unique across all the servers in the web farm.

ASP.NET MVC 4 offers many features that support scalability. ASP.NET MVC 4 is also independent of any of the mechanisms you might select to maintain session. It offers sessionless support by default through the use of routing and model capture, and you can split the various layers into their separate components and put the models on separate servers from the controllers. Section 1.2 offers additional information on considerations on how ASP.NET MVC supports scalability.

Using cookies or local storage to maintain state

Cookies and HTML5 Web Storage are related. Cookies are the predecessor to the Web Storage API. As mentioned previously, cookies are sent back and forth with every request scoped to that cookie. If the information will be used only on the client side, extra bandwidth is consumed by passing cookies. Cookies are also limited in size to 4 kilobytes (KB). For those instances where the data can be kept only on the client during page load, HTML5 introduced the Web Storage API. The purpose of the API is to keep easily retrievable JavaScript objects in the browser memory for use on client-side operations.

Cookies

When you are considering the structure of your ASP.NET MVC application, you might determine that some information needs to be used by multiple requests. Ideally, this information would fit into the model you are using on your strongly-typed view. However, if multiple requests are necessary, it is likely the information is independent of the model being transferred. This gives you two options, neither of which is ideal.

Create a base class for all your models that contain this information so it is available as part of every model you are using in a view, or find some other way to store and transfer this information. This is where cookies come into play. Because of the stateless nature of ASP.NET MVC, you either have to store this information on the server or transfer it with every request, which is what cookies were designed to do. You don't have to provide additional code to use cookies—they are a standard part of server/client communication. An additional reason to use a cookie in this case is if you want the value to be available on the client side or if you want it to persist between site visits. Any site information you might need persisted on the client side, such as login credentials when the user selects Remember Me, will have to be saved as a cookie.

HTML5 Web Storage

HTML5 Web Storage can choose to use either the *sessionStorage* or *localStorage* object. Each option provides a different feature set. The *sessionStorage* scope enables you to use set and get calls on different pages as long as the pages are from the same origin URL. Objects in *sessionStorage* persist as long as the browser window (or tab) is not closed. *localStorage* provides another option that increases scope because *localStorage*'s values persist beyond window and browser lifetimes, and values are shared across every window or tab communicating with the same origin URL.

The HTML5 Web Storage API also allows for events. If a user has two windows or tabs open—for example, a product listing page and a product detail page—each page can be notified when information is added or changed in *localStorage* if the pages have attached an event listener. Although none of this information will be sent to the server automatically, you can place some values into a page variable and send them to the server. Every other state management mechanism is concerned about maintaining state between the client and the server. HTML5 Web Storage API is concerned only with maintaining state information on the client. If you want state information to be used server-side, you have to write the code to send it back as needed.

Browser compatibility is an issue, however. Not all browsers can handle the HTML functionality involved with the use of *localStorage* and *sessionStorage*. Make sure you have browser check code in place. You can put this browser check code on the server as well as on the client. If you perform the check on the server, such as by using *System.Web.HttpBrowserCapabilities browser = Request.Browser*, you can send back a different view based on the browser version. You could have one view based on HTML5 and the other not using HTML5, and send the appropriate one back to the client. An example of how you can check for *localStorage* in JavaScript is:

```
if(window.localStorage){window.localStorage.SetItem('keyName','valueToUse');}
```

You can also use this code:

```
window.localStorage.keyName = 'valueToUse';
```

This code sets an event listener:

```
window.AddEventListener('storage', displayStorageEvent, true);
```

The event listener code pertains to any storage event, either *localStorage* or *sessionStorage*, and that it should call the function *displayStorageEvent*. The *eventListener* fires when there is any change in storage, either *localStorage* or *sessionStorage*.

ASP.NET MVC 4 does not offer specific methods for handling local storage. However, the jQuery library that ships with Visual Studio is an excellent tool for handling the client-side scripting required to manage *localStorage* access. Although ASP.NET MVC 4 does offer good cookie support, there are a few limitations in that the maximum cookie size is 4 KB, and that this information is transmitted to and from the server with each request-response.

Applying configuration settings in the Web.config file

Many choices related to state management can be maintained through the primary Web.config file in the root directory of the project. Sessions can be enabled in the Web.config file through the use of a <sessionState> node. The following is an example of an InProc configuration:

```
<system.web>
    <sessionState mode="InProc" cookieless="false" timeout="20"
        sqlConnectionString="data
        source=127.0.0.1;Trusted_Connection=yes"
        stateConnectionString="tcpip=127.0.0.1:42424"
    />
</system.web>
```

A StateServer configuration for configuring sessionState is as follows:

```
<system.web>
    <sessionState mode="StateServer"
       stateConnectionString="192.168.1.103:42424" />
</system.web>
```

You can also configure the provider if you are going to use the ASP.NET Membership provider:

```
<profile defaultprovider="DefaultProfileProvider"
    inherit="MyApplication.Models.CustomProfile"/>
```

All other session mechanisms are either always on or available only on the client side. These configuration items can also be added at a lower part of the configuration stack including the Machine.config file, which is the lowest configuration file in the stack and applies to all websites on that server.

There might be other necessary information to support your state management process that could be stored in the Web.config file. If you have written a custom state management mechanism, you might need to store supporting items in the configuration file, such as connection strings, service endpoints, or special identifiers. You might also need to configure HTTP modules or HTTP handlers in the configuration file if that is how your custom state is handled. There is more information on the configuration and usage of HTTP modules and handlers in Section 1.7 later in this chapter.

MORE INFO **ASP.NET CONFIGURATION**

Microsoft Support has an informative set of articles on the details of configuration within the ASP.NET system at *http://support.microsoft.com/kb/307626*.

Implementing sessionless state

Sessionless state is a way to maintain state without supporting any session modes. There are several considerations that have to be taken into account when planning to implement sessionless state. The first is that when state-type information is necessary in your application, you have to pass some kind of unique identifier from one server call to the next so that the application can recognize the connection. Performance is another consideration if you will be managing state-type information in custom functionality because the current session management technology has been greatly optimized.

Determining when to use sessionless state in your ASP.NET MVC application requires a deeper look into the mechanics of how sessions interact with the controller. The design of session state, as implemented in ASP.NET, implies that only one request from a particular user's session occurs at a time. This means that if you have a page that includes multiple, independent AJAX callbacks happening at once, these calls will be processed in serial fashion on the server. If your application is sessionless, it can also handle AJAX callbacks in parallel rather than requiring that the work be performed in serial, which enables you to perform multiple, simultaneous AJAX calls from the client. If your application will be best-suited by the use of extensive AJAX calls on the client to continuously work with sections of your page content, and you need state, you would likely be best served to not use session. Requests that use session where there is an overlap in the server calls will be queued up and responded to one at a time. This can affect user perception of performance, especially during the initial set of calls when a page is first rendered, and all the AJAX calls start at the same time. In these situations, you should either go sessionless or ensure that the initial response to the client does not cause simultaneous AJAX calls upon the load of the page.

If you determine that your application will be best served by sessionless state, you need to determine how you will pass the unique identifier from request to request. There are a lot of mechanisms available in ASP.NET MVC 4 to help you do this:

- Create the identifier on the server the first time the user visits the site and continue to pass this information from request to request.
- Use a hidden form field to store and pass the information from one request to the next. There is some risk in this because a careless developer could forget to add the value, and you will lose your ability to maintain state.
- Because the Razor view engine supports the concept of a layout or master page, you can script the unique identifier storage in that area so that it will render on every page.
- Add JavaScript functionality to store the unique identifier on the client side in a *sessionStorage* or *localStorage* and make sure that it is sent back to the server when needed. That way, you don't have to worry about losing the information; you just need to make sure that you include it when necessary.

- Add the unique identifier to the query string so that it is always available whenever a *Request* object available.
- Add the unique identifier to the URL and ensure that your routing table has the value mapped accordingly.

Finally, consider whether you need your application to maintain any special state information. Do you really need to store all of the information that's automatically put into state, whether using session or going sessionless? User information, for example, is already available on the HTTP request, so there isn't necessarily any need for that to be in session. Many decisions you make about what to put in session or the state model is based on whether you'll use the information in the next request. Does that need to be stored in state or will the use of caching (covered in Section 1.5) eliminate the need for the session altogether?

EXAM TIP

ASP.NET MVC was designed to support the stateless nature of HTTP. The use of sessionless state is a natural addendum to that approach because it minimizes the overhead added by the server when managing state. You should be comfortable with the concept of maintaining state information within your application and understand the potential ramifications of each solution, including the risks of passing the state identifier between the client and the server, such as when using cookies and query strings.

Thought experiment

Designing an architecture for a process management system

In the following thought experiment, apply what you've learned about this objective to predict how you would design a new application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are an architect for a Fortune 1000 company that wants to create an internal task and process management system. The concept of the system is to give visibility to daily tasks that are performed throughout the company. Every employee is expected to use the system and be diligent about inputting tasks and to manage the statuses of the tasks as they move through the process. Your team was instructed to design an architecture that would support hundreds of simultaneous users. You have been given five physical servers and an old load balancer to run the system. You also have several licenses for SQL Server.

1. **How would you provision these servers for maximum reliability, safety, and performance?**
2. **How would you manage the state in this situation?**
3. **If two servers shut down at the same time because of hardware issues, and the problem took several days to resolve, how would your application be affected?**

Objective summary

- State management can be an important part of a software application. It is complicated in web applications because, by definition, HTTP is a stateless transfer protocol. ASP.NET MVC 4 offers multiple ways to maintain state. Decisions about maintaining state need to take into account considerations such as whether state information will be just used on the server or in the client as well, latency, and amount of data that is being stored.
- The most common way to maintain state is through a session. The session can be configured to be stored in a SQL Server or separate state server and can also be configured to put the session ID in either a cookie or as part of the query string.
- The query string is also a place where you can put a limited amount of information to pass back and forth to the server. The information is not secure, however, and is not unlimited because there are size limits on requested URLs. The query string is easy to access from ASP.NET MVC 4.
- There is also the capability to completely store state information on the client side if that best serves the application requirements using HTML5 Web Storage API. You need to ensure that the browser adequately handles HTML5, but. ASP.NET MVC 4 does not have any default handlers to work with the client-side information other than the jQuery library.
- Scalability is a major concern when determining how best to manage state. Creating a scaleable architecture will immediately rule out some of the available choices, as having an indeterminate server process the request is problematic because that server might not have access to the state information if it is stored on a single server. ASP.NET MVC 4 supports stateless protocols for scalability as well.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing an ASP.NET MVC 4 application that uses an Oracle database for persistence. What session configuration choices enable you to deploy your application on a web farm? (Choose all that apply.)

   A. InProc

   B. SQLServer

   C. StateServer

   D. Custom session provider

2. You are creating an ASP.NET MVC 4 web application that will be accessed by a large number of traditional consumers. If you need to be able to access state information on the client side in JavaScript/jQuery, where can you store it? (Choose all that apply.)
   - **A.** *localStorage*
   - **B.** *QueryString*
   - **C.** *ViewState*
   - **D.** Cookies
3. As you design a sessionless state management system, what do you need to ensure that your application manages? (Choose all that apply.)
   - **A.** Access to the state management system, whether it is a database, a web service, or other type of system
   - **B.** The HTTP headers
   - **C.** The session setting within the Web.config file
   - **D.** An identifier used by the server to identify the request

Objective 1.5: Design a caching strategy

Caching is a basic application development strategy to help improve performance. You have likely found that a significant amount of time is spent accessing data. Although it might only be milliseconds at a time, it adds up and can have a dramatic impact on overall performance. Caching is a mechanism for storing frequently used information and within high-speed memory. This seemingly small change will reduce access time and increase response time.

As in managing state, there are several places in which you can implement data caching, and each has different ramifications regarding ease of implementation, flexibility, and performance. For example, relatively static data can be marked so that multiple requests will return the same rendered page and forego the expense incurred by re-creating the page content. Data caching provides much of the same advantage by caching information at the data access layer and removing the need for some of the calls into the persistence system.

This objective covers how to:

- Implement page output caching (performance oriented)
- Implement data caching
- Implement application caching
- Implement HTTP caching

Implementing page output caching

Caching is an important part of developing highly scalable web applications. The web browser can cache any HTTP GET request for a predefined period, which means the next time that user requests the same URL during that predefined period, the browser does not call the server but instead loads the page from the local browser cache. ASP.NET MVC enables you to set the predefined period by using an action filter:

```
[OutputCache(Duration=120, VaryByParam="Name", Location="ServerAndClient")]
Public ActionResult Index()
{
    Return View("Index",myData);
}
```

This code sets the response headers so the browser will know to go to its local cache for the next 120 seconds. The *Duration* setting represents the time, in seconds, that the page output should be cached. Due to the *Location* setting in the attribute, any other browser call going to this URL will also get this same server-cached output. Imagine how much work this could remove from a heavily used server if it had to create page content only once every two minutes rather than several thousand times per minute.

There might be times when you want to disable caching, which you can do by using *Duration=0*. Other most commonly used options available in the *OutputCacheAttribute* are *VaryByParam*, *Location*, and *NoStore*. *VaryByParam* stores a different version of the output based on a different parameter collection that was sent in for the action call. The *Location* qualifier gives direction to where caching takes place; *NoStore* is used when caching should be switched off. The default value is *Any*, but *Client*, *Downstream*, *Server*, and *ServerAndClient* are other options available when setting the cache location.

Donut caching

The *OutputCache* attribute works well for caching an entire page. You might need a more flexible approach and to cache parts of the page content while continuing to generate other parts of the page. For example, part of the starting page for an online store includes user information. You want to cache the top and bottom toolbars, but you do not want to cache any personalization areas. *OutputCache* does not work in this case with the default setup, but donut caching is a good solution. *Donut caching* is a server-side technology that caches an entire page other than the pieces of dynamic content—the donut holes.

Although ASP.NET Web Forms supports donut caching through the *Substitution* control, the Razor Engine does not offer support for donut caching. However, because ASP.NET MVC 4 is built on top of ASP.NET, you can still use the Substitution APIs through the *HttpResponse.WriteSubstitution* method by creating an MVC helper. This enables you to cache an entire page on the server except for a particular reference.

Donut hole caching

Where donut caching caches the entire page other than a few sections, donut hole caching takes the other approach and caches only select portions of the page while keeping the rest of the page dynamic. Donut hole caching is also different from donut caching because it is well supported in ASP.NET MVC by using child actions. To perform donut hole caching, create the partial view that will be cached. You also need to add to add the child action that will display the view:

```
[ChildActionOnly]
[OutputCache(Duration=60)]
public ActionResult ProductsChildAction()
{
    // Fetch products from the database and
    // pass it to the child view via its ViewBag
    ViewBag.Products = Model.GetProducts();

    return View();
}
```

Finally, you need to put the reference into the parent view using the Razor command *@Html.Action("ProductsChildAction")*. Using this approach will enable the server to generate this part of your page content no more than once per minute due to the *Duration=60* setting in the attribute.

You can also assign the caching attribute to a controller. Setting *OutputCache* at a controller level automatically configures all actions that accept a GET request to use the same caching settings as if the attribute were put on the individual actions. The caching at a controller level does not affect any actions that accept POST, PUT, or DELETE request types.

Distribution caching

In general, the output caching strategies just discussed work when there is a connection between the client and one server. If you think about a web farm, or where availability requirements demand flexible session switchover, you will find that you lose many of the gains as each server would have to rerun the page to add it to their local cache. To get past this issue, you need the ability to create data on one application server and share it with the other servers. This is called *distribution caching* and is the most complex of all caching techniques. A solution for this is Windows Server AppFabric. By providing a set of extensions to Windows Server, AppFabric enables developers to create faster, more scalable, and more manageable applications. Windows Server AppFabric includes AppFabric Caching Services, which increases responsiveness to frequently used information, including session data.

The main component of AppFabric Caching Services is a cache client that communicates with a cluster of cache servers. Your ASP.NET MVC 4 application is an example of a cache client. Each cache server your application communicates with runs an instance of AppFabric Caching Services, and each maintains a portion of the cached data. AppFabric Caching Services also provides software that can enable each client to keep its own local cache.

When an application needs some information, it initially calls its own local store. If the information is not there, the client asks the cache cluster. If the cache cluster does not have the information, the application must go to the original data source and request the information. All the information in the various caches, local and cluster, is stored under a unique name. The client does not care which physical server holds the information, only whether it can be found in the cache. The process of looking for the value is transparent to the client. It just knows to ask for an item, and AppFabric Caching Services handles the rest of the process.

The item being cached in AppFabric Caching Services can be any serialized .NET object. It is also controlled by the client application. The cached version of the object can be deleted or updated as the application requires. This gives you a chance to fulfill any custom data validation requirements for your application; for example, if object A expires or changes, all versions of object B have to expire as well.

Cached items are also maintained by the cache, which can expire items based on a configurable timeout or to delete items to make room for more commonly accessed items. Timeouts affect both local and cluster caches, and can be coordinated so that timeouts can synchronize between the local caches and the server. Timeout synchronization is especially important when multiple servers (a web farm) handle web requests because each application server can have its own local cache. Synchronization can add a lot of network traffic and can raise some security concerns as well because this data is being exchanged in the background between the caches. To mitigate the security risk, all data sent between the clients and servers can be digitally signed and encrypted. Access to the cache can also be limited by the user. It is important that each of the cache clients trust each other and the cache cluster because they can all access the same data.

One particular benefit of using AppFabric is that the service enables session maintenance. Setting a configuration item enables the *Session* object to be stored in the cache without any additional programmatic support required. The use of AppFabric in this manner enables another OutProc session storage type and replaces the need to set up a shared state server or SQL Server provider to manage shared sessions throughout a web farm.

The throughput and responsiveness of a web application are major concerns because they directly affect an application's usability. Adding distributed caching to your ASP.NET MVC application, especially if you are already deploying your application in a distributed environment, could create measureable performance gains. Windows Azure AppFabric can add a shared caching service that will be available throughout your deployed system. The caching will not add any performance gain on the first server's initial call for a piece of data, but it will enhance the responsiveness of each additional request for that same piece of data from all servers connected to that cache cluster.

Implementing data caching

Another form of caching that can occur at the server side is by using the new .NET 4 Caching Framework. The default implementation uses the *ObjectCache* and *MemoryCache* objects that are within the *System.Runtime.Caching* assembly. When you create your cache, you can set an expiration period just as in output caching. Don't forget that this cache is used by all users on the server. Generally, you create a *CacheProvider* class that implements the *ICacheProvider* interface, used as an intermediate layer between the business layer and the data access layer. Figure 1-8 illustrates all the layers for caching.

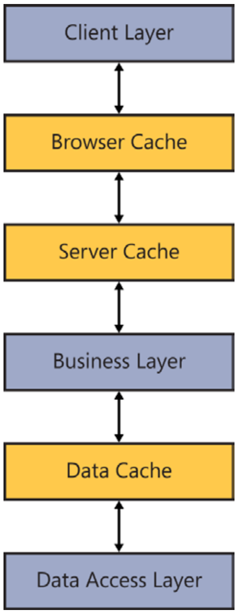


FIGURE 1-8 Fully cached request route

Data caching is an important form of caching that can decrease the load on your database and increase application responsiveness. As you plan your ASP.NET MVC application, you should consider the demand you will be putting on your database and the amount of static database queries your application might require. Static queries, in which the data is unlikely to change often, are excellent candidates for implementing data caching. Best practices in ASP.NET MVC 4 would put the calls to the caching service in the model because the model contains the primary business logic. Designing and implementing a caching subsystem will add additional work during your applications development cycle, but if designed correctly can significantly improve usability. Introducing a caching layer on top of the persistence layer, for example, can improve performance if your application requeries the same data.

MORE INFO **.NET CACHING FRAMEWORK**

There is an informative set of articles on MSDN about caching in .NET Framework applications that includes data caching, services caching, output caching, and how you can extend caching at *http://msdn.microsoft.com/en-us/library/dd997357(v=VS.110).aspx*.

Implementing application caching

The HTML5 specification defines an Application Cache API (AppCache) to give developers access to the local browser cache. To enable the application cache in an application, you must create the application cache manifest, reference the manifest, and transfer the manifest to the client.

Create the application cache manifest

A simple version of the application cache manifest is provided in the following example. The key sections are CACHE, NETWORK, and FALLBACK. The CACHE represents the resources that should be cached on the client, NETWORK defines those items that are never cached, and FALLBACK defines the resources that should be returned if the corresponding resources are not found.

```
CACHE MANIFEST

    # Cached entries.
    CACHE:
    /favicon.ico
    default.aspx
    site.css
    images/logo.jpg
    scripts/application.js

    # Resources that are "always" fetched from the server.
    NETWORK:
    login.asmx

    FALLBACK:
    button.png offline-button.png
```

Reference the manifest

You reference the manifest by defining the manifest attribute on the *<html>* tag from within the Layout.cshtml or Master.Page file:

```
<html manifest="site.manifest">
```

Transfer the manifest

The main thing to remember about transferring the manifest is to set the correct MIME-type, which is *"text/cache-manifest"*. If you are doing this through code, use *Response.ContentType="text/cache-manifest"*. Without this MIME-type specified, the browser won't recognize or be able to use the file. When the application cache is enabled for the application, the browser will fetch resource information in only three cases:

- When the user clears the cache
- When there is any change in the manifest file
- When the cache is updated programmatically via JavaScript

Implementing HTTP caching

HTTP is generally used in distributed systems, especially the Internet. The HTTP protocol includes a set of elements that are designed to help caching. Cache correctness is one of those elements. An HTTP server must respond to a request with the most up-to-date response held by the cache that is equivalent to the source server; meets the freshness case; or is an appropriate 304 (Not Modified), 305 (Proxy Redirect), or error (4xx or 5xx) response message.

Another element is the expiration model that provides a server-specified or heuristic expiration, and the HTTP protocol has multiple rules around calculating expiration. The protocol also has a validation model in which the client sends information to the server so the server can detect whether any changes need to be returned. Actually, the server sends a special status code, usually a 304 (Not Modified) response without an entity-body, when there has been no change in the output; otherwise, the server transmits the full response including the entire body. This gives the server the chance to respond with a minimal message if the validator matches; a chance to stop a full round trip requery by sending the correct information. With HTTP caching, everything happens automatically as part of the request stack and there is little programmatic impact.

Thought experiment

Improving the performance of an inventory management system

In the following thought experiment, apply what you've learned about this objective to predict how you would refactor an existing application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are working at a company that has a custom inventory management system that has been used for years. Company personnel recently completed an overhaul of the web-based application that provides access into the system. Although they are comfortable with the look, feel, and usability of the application, they are unhappy with the performance, even after updating all their users to the most recent hardware equipment and software. They find that every interaction with the server is taking seconds. At times, this renders the application almost unusable. When you look into the application, you see that all the models in the MVC structure do calls into the database whenever they need data, even down to the level of the user's name, shift, and building, which are calls into different tables and displayed on every page the user visits.

1. As you look into the system more carefully, you see 15 calls into the database for every time a page is rendered that is strictly to provide basically static information to the page. How could you use caching to improve this?
2. After you have eliminated the redundant calls, you see that the application makes calls into the database for every request to get supporting information such as colors, product sizes, and box sizes. The application gets the complete list from the database every time and then gets the necessary size, color, and so on from that longer listing. How could caching be used to help this process?

Objective summary

- Page output caching is a shared strategy on clients and servers. Types of page output caching include full page caching and partial page caching. Donut caching and donut hole caching are types of partial page caching. Donut caching caches the majority of the page, enabling some dynamic content. Donut hole caching enables a majority of the page to be dynamic and caches some content.
- Data caching is a server-side technique that enables you to put an intermediate step between your business logic and the database. Data caching provides a way to reuse data and enhance performance by making database calls only when the cache is invalidated or expired.
- Windows AppFabric is an example of a third-party tool that enables you to create caching content on one server and share it across multiple servers in a web farm. Windows

AppFabric is a set of services built upon Windows Server that manages distributed caching. It can also be configured to manage the session in an ASP.NET MVC 4 application.

- Application caching is an HTML5 feature that enables you to create a caching manifest that describes the settings across a website or for a page.
- HTTP caching is a caching mechanism built into the HTTP protocol that handles its own version of expiration calculation and uses it to determine the response to send to the client.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You are designing a work order management application for a mid-size repair company. The application will be used by repair personnel in the field on their laptops with wireless phone connections. The connections are slow, and the laptops are several years out of date. There will never be more than 15 users at any one time, and rarely more than 2 concurrent users. What kind of caching will give the repair personnel a better user experience? (Choose all that apply.)
   - **A.** Page output caching
   - **B.** Application caching
   - **C.** Data caching
   - **D.** HTTP caching
2. You are creating a solution in which the majority of the application is dynamic, but some areas can be cached for a long time. What kind of approach would you take? (Choose all that apply.)
   - **A.** Data caching
   - **B.** Donut hole caching
   - **C.** Donut caching
   - **D.** Windows AppFabric caching
3. You are adding a reporting vertical to an enterprise application. Many of the reports will be run every morning by a set of users. Some of the reports will be identical as every member of a team will get the same report sometime in the morning. What kind of caching will provide an improvement in performance? (Choose all that apply.)
   - **A.** Data caching
   - **B.** Page output caching with a duration of two minutes
   - **C.** Page output caching with a duration of four hours
   - **D.** Windows AppFabric caching

Objective 1.6: Design and implement a WebSocket strategy

HTML5 WebSockets provide a new way to communicate with the server. Traditional communications by a webpage is request-response: the browser sends a request for information to the server, which then sends back a response. Each request and response uses a new connection, and that connection is closed after the response is returned to the client. As you can imagine, this is a poorly performing method because of the time spent creating and closing each connection. Also, such communication cannot be two way because both client and server cannot talk simultaneously, and the server does not easily maintain a connection to the client.

WebSockets uses a different approach in that it provides duplex, or two-way, communication between the server and client. Both parties can communicate at the same time, as in chatting or instant messaging clients. It also limits connection creation and disposal so that it occurs only once rather than with every message. It is essentially a TCP-based protocol that enables two-way communication to occur over a single connection.

This objective covers how to:

- Read and write string and binary data asynchronously (long-running data transfers)
- Choose a connection loss strategy
- Decide a strategy for when to use WebSockets

Reading and writing string and binary data

There are several different ways to communicate between the client and server when there are multiple potentially unnecessary calls to the server. *HTTP polling* is an ongoing conversation between a client and server in which the client appears to have a constant connection with the server based on a series of standard AJAX requests. As part of this technique, you use a JavaScript timer to send AJAX requests at regularly scheduled times. The browser creates a new request immediately after the previous response is received. This is a fault-tolerant solution, but it is very bandwidth- and server-usage intensive, especially considering most requests will return little or no data.

HTTP long polling is a server-side technique in which the client makes an AJAX request to the server to retrieve data. The server keeps the request open until it has data to return. Long polling is done to make a request in anticipation of a possible future server event. Instead of immediately returning a response, the server blocks the incoming request until the data comes up or the connection times out. This isn't a naturally occurring process in HTTP because the request-response model was not designed for it, and thus it is not a totally reliable solution. Broken connections are common, so handling them is a normal part of the implementation.

WebSockets technology is a new approach to supporting duplex communication. WebSockets acts as a replacement for HTTP in that it takes over the communications protocol between the client and the server for a particular connection. This means you should not use it as the primary means of communication between a client and server. Instead, use WebSockets to support some discrete functionality that needs two-way, long-running communication without having to support the request-response process. You will find that WebSockets work best when supporting a part of your page you designed as a partial page or are when using some kind of donut or donut hole caching.

In addition, remember that many users still use a browser that is not fully HTML5-compliant, so you have to plan in advance to manage it. *System.Web.HttpBrowserCapabilities* enables you to query a browser's version to determine whether it supports HTML5. Because the initial connection request has to come from the client, it might make more sense to put the browser check there: If the browser does not handle HTML5, the browser will have to do the work to replace the WebSocket functionality. In that case, you could include regularly timed AJAX calls, such as every 60 seconds, to substitute for the WebSocket functionality. Unless all your users are running a current browser that supports WebSockets, you need to support multiple connection paths or not offer WebSockets to users.

There are two parts to working with WebSockets: the client side and the server side. A WebSocket-based communication generally involves three steps:

1. Establishing the connection between both sides with a hand shake
2. Requesting that WebSocket server start to listen for communication
3. Transferring data

When a WebSocket is requested, the browser first opens an HTTP connection to the server. The browser then sends an upgrade request to convert to a WebSocket, as shown in Listing 1-6. If the upgrade is accepted and processed, and the handshake is completed, all communication occurs over a single TCP socket. Each message is also smaller because there are no extra headers after the handshake.

LISTING 1-6 Example of a WebSocket handshake upgrade request and upgrade response

WebSocket handshake upgrade request

```
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: hy6T&Ui8trDRGY5REWe4r5==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

WebSocket handshake upgrade response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: Ju6Tr4EwedOp9Uyt6jNbgFD5t6=
Sec-WebSocket-Protocol: chat
```

Listing 1-7 includes jQuery code for creating a client-side WebSocket connection.

LISTING 1-7 jQuery code for a client-side WebSocket connection

```
var socket;
$(document).ready(function () {
    socket = new   WebSocket("ws://localhost:1046/socket/handle");
    socket.addEventListener("open", function (evnt) {
      $("#display").append('connection');}, false);
    socket.addEventListener("message", function (evnt) {
      $("#display ").append(evnt.data);}, false);
    socket.addEventListener("error", function (evnt) {
      $("#display ").append('unexpected error.');}, false);
    ...

});
Or using straight method calls:

function connect(){
    try{
   var socket;
   var host = "ws://localhost:8000/socket/server/start";
   var socket = new WebSocket(host);
       message('<p class="event">Socket Status: '+socket.readyState);
       socket.onopen = function(){
            message('<p class="event">Socket Status: '+socket.readyState+' (open)');
       }
       socket.onmessage = function(msg){
            message('<p class="message">Received: '+msg.data);
       }
       socket.onclose = function(){
             message('<p class="event">Socket Status: '+socket.readyState+' (Closed)');
       }
   } catch(exception){
        message('<p>Error'+exception);
   }
}
```

Support for the WebSockets protocol was established with the release of ASP.NET 4.5 and IIS 8. The inclusion in ASP.NET 4.5 makes WebSockets available for use in your ASP.NET MVC 4 applications. ASP.NET 4.5 enables developers to manage asynchronous reading and writing of data, both binary and string, through a managed API by using a WebSockets object. This new namespace, *System.Web.WebSockets*, contains the necessary functionality to work with the WebSocket protocol.

When you are designing an application to work with WebSockets, you must determine how you will manage the connection. Typically, it should be done in either an HTTP handler or an HTTP module. You must implement the process of accepting the upgrade request on an HTTP GET and upgrading it to a WebSockets connection. This is done through implementing a method such as the following:

```
HttpContext.Current.AcceptWebSocketRequest(Func<AspNetWebSocketContext,
    Task>)
```

You need to use a delegate when implementing this acceptance because ASP.NET backs up the request that is part of the current context before it calls the delegate. You can think of this approach as being similar to managing delegates in threading. After a successful handshake between your ASP.NET MVC application and the client browser, the delegate you created will be called, and your ASP.NET MVC 4 application with WebSockets support will start. The code for managing a WebSockets connection is shown in Listing 1-8.

LISTING 1-8 C# code for managing a WebSockets connection

```
public async Task MyWebSocket(AspNetWebSocketContext context)
 {
        while (true)
        {
           ArraySegment<byte> arraySegment = new ArraySegment<byte>(new byte[1024]);

           // open the result.  This is waiting asynchronously
           WebSocketReceiveResult socketResult =
                   await context.WebSocket.ReceiveAsync(arraySegment,
                       CancellationToken.None);

           // return the message to the client if the socket is still open
           if (context.WebSocket.State == WebSocketState.Open)
           {
                   string message = Encoding.UTF8.GetString(arraySegment.Array, 0,
                           socketResult.Count);
                   userMessage = "Your message: " + message + " at " +
                           DateTime.Now.ToString();
                   arraySegment = new
                       ArraySegment<byte>(Encoding.UTF8.GetBytes(message));

                   // Asynchronously send a message to the client
                   await context.WebSocket.SendAsync(arraySegment,
                       WebSocketMessageType.Text,
                           true, CancellationToken.None);
           }
           else { break; }
        }
}
```

MORE INFO **WEBSOCKET API**

The W3C's WebSocket API specification at *http://dev.w3.org/html5/websockets/* gives you an in-depth understanding of how the WebSocket protocol works inside a browser.

Choosing a connection loss strategy

When using WebSockets, you need to determine how you are going to handle those times when you lose a connection. This functionality has to be on the client side because the server side cannot reach out to the client when there is no connection. When the connection is broken, the client might notice it when either an *onclose* or an *onerror* event is thrown, or the delegated methods are called, depending on how the connection was set up. However, it is also possible that the connection might be broken and the connection does not throw an *onerror* or *onclose*. To manage that, you need to ensure that your application can manage a connection that is no longer available. Ideally, the library will throw an *onerror* when it attempts to send a message to the server, but you need to build your application so that it is able to retain state; and if there is a disconnect in the process, it can restart, re-create a connection, and resend the message.

WebSockets can run into several types of connection issues. The entire premise is that there is a long-open socket connection for communications between the two ends. Any kind of issue that might come up in that connection, whether it is a client/server issue or any issue between the two, can cause connections to be lost. Therefore, as you design your application's use of WebSockets, you need to keep data protection and communications reset in mind.

A developer typically uses a "fire and forget" methodology, in which you send a message and assume that it is received by the listener, but that methodology might not be sufficient for WebSockets. You should architect a system that sends a message; waits for a response; and from the response, or lack thereof, determines whether the system has successfully sent the message. You also have to monitor the connection from the time you send a message until you receive a response to ensure there was no break in the connection during the transmission. If a break occurs, you should reopen the connection and resend the data. Keep in mind that the connection might have been broken after the data was received but before the sender was given the receipt; your code needs to allow for multiple receipts of information.

Regarding communications reset, any interference between the client and server can break the connection, so you might end up listening to a dead connection. You need to make sure that the *onclose* and *onerror* events are managed and that you build in a recovery mechanism.

Deciding when to use WebSockets

WebSockets are an ideal solution when you need two-way communication with the server with minimal overhead. A common use of WebSockets is for an in-browser instant messaging client. A traditional dashboard solution is also a candidate for the flexibility offered by WebSockets because near-real-time updates is a value-add.

You might want to use WebSockets for any kind of communications between a server and client; however, the more traditional approach of a client timer might be a better solution in some situations. Users do not care if you are using WebSockets; they simply want reliable functionality. As you evaluate the use of WebSockets in an application, keep in mind that the WebSocket protocol requires a web browser that supports HTML5. Because the HTML5 standard is still evolving, some browsers do not completely support HTML5. Although you can check for WebSocket support on the client before initiating the request for an upgrade, you don't want to leave any users without functionality because their browser doesn't support the technology in your application. Carefully weigh the needs of your application versus the available technology.

Another strategy is to enable the controller on the server to decide whether to support WebSockets. Rather than disabling or hiding functionality on the client side, make that decision on the server side. If the server determines that a client supports WebSockets, the server can make decisions such as rendering a partial view that has the client-side functionality for the usage of WebSockets. If the server determines that WebSockets are not supported by the browser, it can instead render a partial view that uses a fallback JavaScript-based implementation using long polling or a timer. Making that decision on the server simplifies the code you need on the client side.

Another issue to consider is a reaction to one of its strengths. WebSockets do not have HTTP headers, yet they travel as if they are HTTP requests. This is a potential problem because many networks direct traffic by looking at the HTTP headers and determine how to handle messages based on values within the headers, such as CONTENT-TYPE. In those kinds of scenarios, WebSockets traffic is likely deemed malicious and the network send is cancelled. The presence of antivirus and firewall software on the client machine could have the same problem because they analyze incoming packets to determine their source and potential risk. Therefore, not only is there a client-side requirement that the browser can support the protocol but there also has to be requirements in place that your network, the user's network, and the user's machine do not stop the packet's transfer. Unfortunately, you can test whether WebSockets are supported by the browser on the client side, but the only way you can test whether the full route is allowed is to actually try to make a connection and send data. This data should be beyond the simple handshake and should mimic one of the data packets that you will use for communication. If it is received in both directions, you can assume that WebSockets are fully supported.

Thought experiment

Using WebSockets for a communications application

In the following thought experiment, apply what you've learned about this objective to predict how you would design a new application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are a consultant hired by an insurance and financial services firm, to create an online system for employees to get breaking financial news as well as to communicate with each other. The company already has a news tracking system that stores relevant news articles and sends an email with the content of the articles to some of the fund managers. The company wants to include this information as well as an instant messenger application and create a simple site for communications for all employees.

1. **Although HTML5 WebSockets appears to be a natural fit to fulfill these requirements, what kind of issues could you run into?**
2. **There appear to be two different communications processes going on. Would you use a different socket connection for each one or share the same connection? Explain your answer.**
3. **What kind of server-side services do you have to create?**

Objective summary

- HTTP polling is a JavaScript methodology of continuously polling the server to see whether there is any information that the client needs to know. Although not the most efficient method, it has the luxury of working in any browser that supports JavaScript and does not require HTML5 support.
- HTTP long polling is a way to use HTTP to mock up a way for the server to pass data back to the client, as determined by the server, by opening a long-standing connection to the server that will either time out or return data when the server determines it is necessary. Upon timeout or data return, the client can immediately open a new connection.
- WebSockets are a way to provide duplex, or two-way, communication between the client and server. Both sides can communicate at the same time to the other side. The client connects via HTTP and then sends an upgrade request to the server, which gives a WebSockets connection. You need to create both client- and server-side code to interact with the socket. After that is done, every command is basically an event that is fired when a message is received.
- WebSockets can be used in situations in which long-term, two-way communication is useful. It is not necessarily always the best solution, especially when there is a chance that the application will be viewed in older browsers that do not support HTML5 features.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. What is the technique in which the client sends a request to the server, and the server holds the response until it either times out or has information to send to the client is?
   - **A.** HTTP polling
   - **B.** HTTP long polling
   - **C.** WebSockets
   - **D.** HTTP request-response
2. You are building an application in which you want to display updated information to a website every 15 minutes. What are efficient ways to manage the update? (Choose all that apply.)
   - **A.** WebSockets
   - **B.** HTTP polling with 1-minute intervals
   - **C.** HTTP long polling
   - **D.** HTTP polling with 15-minute intervals
3. What is the first request sent to start HTTP polling?
   - **A.** HTTP DELETE
   - **B.** HTTP GET
   - **C.** HTTP CONNECT
   - **D.** Upgrade request

Objective 1.7: Design HTTP modules and handlers

HTTP modules and handlers enable an ASP.NET MVC 4 developer to interact directly with HTTP requests as they are both active participants in the request pipeline. When a request starts into the pipeline, it gets processed by multiple HTTP modules, such as the session and authentication modules, and then processed by a single HTTP handler before flowing back through the request stack to again be processed by the modules.

This objective covers how to:

- Implement synchronous and asynchronous modules and handlers
- Choose between modules and handlers in IIS

Implementing synchronous and asynchronous modules and handlers

Modules are called before and after the handler executes. They are intended to enable a developer to intercept, participate, or modify each request. Creating an HTTP module requires you to implement *System.Web.IHttpModule*, which has two methods: *void Init(HttpApplication)* and the *void* method *Dispose*. The *System.Web.HttpApplication* has 22 available events that can be subscribed to in the *Init* method that enables the module to work on the request in various stages of the process (see Table 1-3). The <httpModule> configuration section in the Web.config file is responsible for configuring the HTTP module within an application. Several tasks are performed by the *HttpApplication* class while the request is being processed. The events are useful for page developers who want to run code when key request pipeline events are raised. They are also useful if you are developing a custom module and you want the module to be invoked for all requests to the pipeline.

TABLE 1-3 ASP.NET life cycle events

Event name	Description
BeginRequest	The first event raised; always raised when processing a request
AuthenticateRequest	Raised when a security module has identified the user
PostAuthenticateRequest	Raised after the *AuthenticateRequest* event is raised
AuthorizeRequest	Raised after a security module has authorized the user
PostAuthorizeRequest	Raised after the *AuthorizeRequest* event is raised
ResolveRequestCache	Raised to let caching modules serve the requests
PostResolveRequestCache	Raised when a caching module served the request
MapRequestHandler	Raised when the appropriate *HttpHandler* is selected
PostMapRequestHandler	Raised after the *MapRequestHandler* event is raised
AcquireRequestState	Raised when the current state, such as session state, is acquired
PostAcquireRequestState	Raised after the *AcquireRequestState* event is raised
PreRequestHandlerExecute	Raised just prior to executing an event handler
PostRequestHandlerExecute	Raised when the *HttpHandler* has completed execution
ReleaseRequestState	Raised when all request event handlers are completed
PostReleaseRequestState	Raised after the *PostReleaseRequestState* event is raised
UpdateRequestCache	Raised after caching modules store the response for future use
PostUpdateRequestCache	Raised after the *UpdateRequestCache* is raised
LogRequest	Raised just prior to logging the request

Event name	Description
PostLogRequest	Raised when all *LogRequest* event handlers are completed
EndRequest	The last event raised in the HTTP pipeline
PreSendRequestHeaders	Raised just before the HTTP headers are sent to the client
PreSendRequestContent	Raised just before the content is sent to the client

The general application flow is validation, URL mapping, a set of events, the handler, and a set of events. Validation occurs when the system examines the information sent by the browser to evaluate whether it contains markup that could be malicious. The process then performs URL mapping if any URLs have been configured in the <UrlMappingsSection> section of the Web.config file. After it has completed the URL mapping process, the *HttpApplication* runs through security and caching processes until it gets to the assigned handler. After the handler completes processing the request, it goes through the recaching and logging events and sends the response back to the client. Table 1-3 lists the ASP.NET life cycle events, all of which play a strategic part in processing HTTP requests.

It is possible to do much of this work in the Global.asax file because one of the key features of this file is that it can handle application events. Implementing this functionality in a module, however, has advantages over using the Global.asax file. The Global.asax implementation is application-specific, whereas the module is much easier to use between applications. It also provides additional SoCs by enabling your ASP.NET MVC application to manage the request after it hits the handler rather than manipulating it prior to being handled by *MvcHandler*. By adding them to the global assembly cache and registering them in the Machine.config file, you can reuse them across applications running on the same machine.

An HTTP handler is used to process individual endpoint requests. Handler enables ASP.NET to process HTTP URLs within an application. Unlike modules, only one handler is used to process a request. A handler must implement the *IHttpHandler* interface. A handler is much like an Internet Server Application Programming Interface (ISAPI) extension. The *<httpHandler>* configuration section is responsible for configuring the handler by configuring the verb, path, and type that directs what requests should go to the handler. The *IHttpHandler* interface has an *IsReusable* property and a *ProcessRequest(HttpContext)* method that gives the handler full access to the request's context.

ASP.NET 4.5 enables you to write both modules and handlers so that they can handle asynchronous calls. Just as in a regular asynchronous method on the controller, the use of an asynchronous module or handler enables you to run a method so that it will not stop or affect the processing of the request. Plugging a module into the request stream is based on handling events as the process gets to a particular point. To write an asynchronous module, you need to use the await, async, and Task objects, as shown in the following example:

```
private async Task ScrapePage(object caller, EventArgs e)
{
    WebClient webClient = new WebClient();
    var downloadresult = await webClient.DownloadStringTaskAsync("http://www.msn.com");
}

public void Init(HttpApplication context)
{
    EventHandlerTaskAsyncHelper helper =
            new EventHandlerTaskAsyncHelper(ScrapePage);
    context.AddOnPostAuthorizeRequestAsync(
                    helper.BeginEventHandler, helper.EndEventHandler);
}
```

When using synchronous modules, the same thread serves the entire request, including handler and modules. That thread also cannot be used by any other request until it has completed its current request. If there is any issue in one of the modules, such as a failure to connect to a database, or an I/O problem, this thread could be paused for an extended period. If this happens, your server's (and hence your application's) throughput will be negatively affected. To avoid this potential impact, making an *HttpModule* asynchronous offers protection to your server and application as the primary thread passes the module control to another thread. Thus, if there is an issue with your module or any of its supporting systems, the primary thread is not affected. There are some potential issues with using an asynchronous module. If your application has a dependency upon work done in the module, there is a potential for a race condition between your application startup and the module completion.

Implementing an asynchronous handler is a much simpler process. By inheriting the *HttpTaskAsyncHandler*, you have a *ProcessRequestAsync* method that gives you default access to async and await for use in asynchronous method calls:

```
public class NewAsyncHandler : HttpTaskAsyncHandler
{
        public override async Task ProcessRequestAsync(HttpContext context)
        {
           WebClient webClient = new WebClient();
           var downloadresult = await
                   webClient.DownloadStringTaskAsync("http://www.msn.com");
        }
}
```

Figure 1-9 shows how an HTTP module is part of the process to and from a handler.

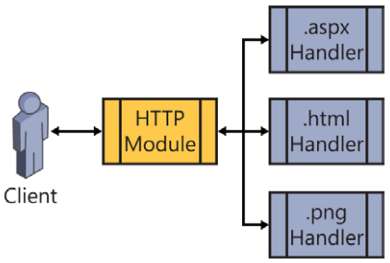


FIGURE 1-9 Process flow for an HTTP request and response

MORE INFO **WEB MODULES AND WEB HANDLERS**

For information on the ASP.NET MVC 4 default classes that implement the *IHttpModule* interface, visit *http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule(v=vs.71).aspx*.

Choosing between modules and handlers in IIS

Http handlers help you inject preprocessing logic based on the extension of the file name requested. When a page is requested, *HttpHandler* executes on the base of extension file names and on the base of verbs. HTTP modules are event-based and inject preprocessing logic before a resource is requested. When a client sends a request for a resource, the request pipeline emits lots of events, as listed in Table 1-3. When planning to develop an IIS feature, the first question you should ask is whether this feature is responsible for serving requests to a specific URL/extension or applies to all requests based on a set of arbitrary rules. If the key consideration is the URL, you should use an HTTP handler. If you want to work on every request regardless of URL, and you are prepared to work with an event-driven framework, you should create an HTTP module.

As you design a large application, you will find that it is an iterative process—you need to revisit previous decisions as you handle change requests or start designing new areas of the application. Perhaps you need to offer different authentication schemas for different network subnets. IIS and ASP.NET MVC handle a single authentication scheme very well, and offer other support through federation. However, that might not fit your need. Perhaps you just need to add something as simple as a network subnet to Active Directory server mapping. This affects all users, and this determination should be made before the logon process occurs. By registering an event handler for the *AuthenticateRequest* event, you can add override code that will handle your custom mapping requirements.

Each major activity we typically expect to be available for use in ASP.NET MVC code generally has its own event for adding functionality or overwriting existing procedures. You need to analyze the kinds of special needs your application has and where it makes the most sense in the process to fulfill those needs. If you need the information available to you prior to it calling your ASP.NET MVC code, it should be a module. If you want special files to be handled differently, it should be a handler.

There are some choices that are not necessarily as clear as others. For example, let's say your application has the requirement that every image to be served has to have a watermark. There are several ways to manage this. One is by creating a custom handler for all the image extensions that need to be watermarked. This would enable you to call the image, write the watermark on it, and then send it to the response. You could also do this as a module by intercepting the response after the default handler has processed it, reading in the byte stream, and making the changes at that point.

When choosing between creating a custom handler and a custom module, your major considerations are where in the process you need the custom work to occur, and what type of requests and responses it needs to support. If it needs to support every request, regardless of the item requested, you should use a module. If it needs to support requests for only a special type or URL, consider using a handler.

EXAM TIP

HTTP modules and handlers give you flexible access into the *HttpRequest* and *HttpResponse* objects. You should be familiar with the events that are raised during the process because they provide integration points for HTTP modules. You should also consider the impact of creating custom HTTP handlers and the effect a custom handler might have on your typical ASP.NET MVC site. Becoming familiar with the default modules and handlers that support ASP.NET MVC will also be useful.

Thought experiment

Using HTTP handlers and HTTP modules as services

In the following thought experiment, apply what you've learned about this objective to predict how you would design a new application. You can find answers to these questions in the "Answers" section at the end of this chapter.

You have been asked to create a set of web services. They will not be standard web services; they will be based entirely on HTTP modules and HTTP handlers. These services will be REST-based and need to support authentication.

1. **What would be the most standard way to use HTTP modules and HTTP handlers to fill this need?**
2. **If you needed to add custom authentication, where would be the best place to put that functionality?**
3. **Do you think that creating web services in handlers and modules would result in a responsive application, or do you think performance would suffer? Why?**

Objective summary

- HTTP modules and handlers insert into the request processing path in IIS.
- Modules fit into the process on the way down to the handler, and on the way back out from the handler. A synchronous module has an *Init* method that enables you to set a handler for one of the events attached to the request process.

- An asynchronous module is more complicated to work with, but with *async*, *await,* and *Task* you can create an HTTP module that can handle long-running tasks without stopping the process.
- Handlers are the destination of the request process and serve requests for a particular URL/extension. A handler can be synchronous or asynchronous, depending on the base class they extend.
- Choosing which one to create is a matter of determining where in the request process you need to add your functionality. If your requirements expect you to be able to handle a specific URL or extension differently from others, a handler is probably what you need to create. If you instead want to act when something happens during the process, you should use a module.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. In an HTTP module, can you redirect the request to a different handler than is in the routing table? If so, what event would you handle?
   - **A.** Yes, and you handle the *PostAuthorizeRequest* event.
   - **B.** No, after the request starts into the process, it either continues through to the mapped handler or throws an error.
   - **C.** Yes, and you handle the *MapRequestHandler* event.
   - **D.** Yes, and you handle the *ReleaseRequestState* event.
2. When you are creating a custom handler, what is the parameter being passed into the *ProcessRequest* method?
   - **A.** *object, EventArgs*
   - **B.** *HttpApplication*
   - **C.** *HttpContext*
   - **D.** *Object*
3. What is the best way to intercept every request for an image on your site and ensure that a watermark is added to the image?
   - **A.** An HTTP module handling the *AuthorizeRequest* event
   - **B.** A custom HTTP handler set to handle .htm and .html pages
   - **C.** A custom HTTP handler configured to serve .png and .jpg files
   - **D.** An HTTP module handling the *PostRequestHandlerExecute* event

Chapter summary

- A properly architected ASP.NET MVC 4 application likely has many layers, or logical groupings of code. The client layer contains the JavaScript/jQuery code that is run in the browser. As you move in deeper, the application layer is connected to the client layer through HTTP requests, and contains the models, views, and controllers. The models can call into another potential layer below that where business logic is managed. This business layer can then call into a cache layer that manages an in-memory snapshot of recent data to enhance performance. This layer can call into the data access layer to select and save the data.
- Session management is a state management mechanism built into Microsoft Internet Information Services (IIS). Session management is highly configurable; you can set session management at the IIS level across all applications or just a single website. You can also manage this configuration in all levels of the .config file structure up to and including the Web.config file. You can set sessions to be managed InProc, which is the best-performing method as the server calls into its internal memory; or OutProc, which is where the server uses an external source to manage session. This external source can be a state server or a SQL server, or you can even create a custom session manager. Sessions are identified on the server by a unique ID value. IIS enables you to set this to be a value in a cookie or to put the value in a query string. This value must be included somehow if you want the server to be able to find any state information.
- Scalability and reliability should be taken into account whenever you are planning an application because it might affect design considerations. A typical deployment strategy for a website would be at least two web application servers so that there would be some redundancy in case one of the machines fails. You need to plan for this if you want the user transition to be smooth. Websites can also use a web farm, or group of web application servers, to run a site. These are smaller commodity physical or virtual servers in which traffic is distributed to each by a load balancer.

- Web services are an increasingly common way for applications to access information. In a service-oriented architecture (SOA), web services stand as the gateway to information. An ASP.NET MVC 4 developer or designer needs to be able to both create and consume web services. The Web API enables developers to use an ASP.NET MVC approach to providing REST services. You can also create REST services by simply using a controller that returns JSON- or XML-formatted data. Consuming web services is equally important because many companies now wrap their data access layer in a web service, which means the models communicate with web services rather than directly to a database.
- Windows Azure provides off-premise capabilities for running websites, data storage, and other application features such as a service bus. These services are highly customizable and support many different hosting and management needs. You have access into the startup, run, and shutdown processes of a web role, and can deploy only parts of your application to the cloud in a hybrid solution.
- HTTP is a request-response communications method in which the client sends a request to the server and the server responds with the information. These requests can be of various types, including PUT, GET, and DELETE. WebSockets changes that paradigm by enabling the developer to add client-side code that will set up a two-way, long-running connection between the client and the server. It allows information to path from the server to the client with anything from the client side other than the initial setup of the connection. The messages passed are smaller because there is minimal header information, and both client and server can send information simultaneously.
- Because ASP.NET MVC 4 is a layer upon ASP.NET, the stack provides an entire framework for managing HTTP requests and responses. Developers can intercept requests and responses, as well as provide a customized handler that creates the response HTTP modules that enable you to intercept requests as they pass through the various stages on their way to the handler. These modules also enable you to intercept the response on its way back out from the handler. It is a highly customizable way to create unique workflows for different needs.

Answers

This section contains the solutions to the thought experiments and answers to the lesson review questions in this chapter.

Objective 1.1: Thought experiment

1. By default, most of the HTML design that went into the original site would be put into the views. Common areas such as navigation or boilerplate text could be put into a layout page if there is a lot of reuse. Also, any areas that might need independent functionality could be put into a partial view.
2. There are several ways that this could be done. The first is to simply put the form online and email the form results to the department. This gives them minimal advantage over their current process and is not in real time. The next is to use a data storage mechanism such as SQL Server to store the information in a database. This enables them to have reports built as needed as well as giving them real-time access into the data.
3. A typical breakdown of models for this situation could be a model for the user, a model for the pet, and a model for the license year. The user might have multiple pets and each pet might have multiple licenses, one for each year. The models would handle the access into and out of the database. This application is a candidate for using the Repository pattern because the system used to manage the backend might be replaced as more departments go online and the municipality might standardize in a different direction.

Objective 1.1: Review

1. **Correct answer:** A
   - **A. Correct:** Because you do not have control over the responsiveness of the third-party provider and you do not know how much data might be returned from each call, you should wrap the call in the asynchronous framework. Providing the data in a strongly-typed model gives it more flexibility than working with the raw XML on the client side.
   - **B. Incorrect:** You do not know how long the call to the third party will take, and putting a synchronous call into the main page will not give any response until the call is completed.
   - **C. Incorrect:** You do not want to use a synchronous call in this case due to the unknown response time.
   - **D. Incorrect:** Although you can take this approach, it infers that you will manipulate the third-party response data in either the controller or the view. SoC recommends that this manipulation occur in a model.

2. **Correct answer:** B
   - **A. Incorrect:** You should not perform any data manipulation in the controller.
   - **B. Correct:** You will have a better chance of code reuse if you break down the separate calls into their own models and then create another model to pull them together and compile them.
   - **C. Incorrect:** Although this would be a plausible way to implement the solution, it is not the best. If any other work came up that uses any of the calls within this model, you will either have to refactor the code to extract it at that point or have duplicate code.
   - **D. Incorrect:** The fact that this data can be merged into a single table display shows there is some intrinsic business worth to the information in this format. Merging on the client side goes against SoC considerations.
3. **Correct answers:** C, D
   - **A. Incorrect:** You should incorporate your team into the project as soon as possible.
   - **B. Incorrect:** Because the team has no experience with object-oriented programming, the Code First approach is unlikely to be the most efficient way to create the new schema.
   - **C. Correct:** The use of the Entity Designer as an integral component in the Model First approach will help unfamiliar users to walk through the process.
   - **D. Correct:** There is already a working relational database for the application, although it is in a system that will be replaced. A port of the design should be considered.
4. **Correct answer:** C
   - **A. Incorrect:** It is the designer's job to ensure that any known enhancements or future changes are accounted for. Although this approach follows the requirements, it is not the best long-term solution.
   - **B. Incorrect:** This solution does not provide the proper level of abstraction; it requires either a "one or the other" approach to supporting the HR system, or an approach in which you have to manage which database you are calling from within each data call.
   - **C. Correct:** Using the Repository pattern will give you a level of abstraction into the data layer. When you create the second data access component for the new HR system, you can then differentiate on a user or departmental level which implementation to use.
   - **D. Incorrect:** This solution does not provide the proper level of abstraction; it requires either a "one or the other approach" to supporting the HR system, or an approach in which you have to manage which database you are calling from within each data call.

Objective 1.2: Thought experiment

1. Yes, because you cannot query the databases directly, you need to deploy some kind of solution that gives you access to the data. This could be a web services wrapper to enable you to call the data remotely or an application that will manage aggregating the data.
2. The primary concerns would be the inability to guarantee responsiveness and the need to manage connection issues.
3. AppFabric acts as a service bus, so it provides a single point of contact/service connector that would manage the calls out to the remote systems by routing the requests to the appropriate server.

Objective 1.2: Review

1. **Correct answers:** B, C
   - **A. Incorrect:** Although you would eventually be able to get a WCF REST services, it would not be efficient.
   - **B. Correct:** Using the Web API is a straightforward way to present REST services.
   - **C. Correct:** Using ASP.NET MVC is another way to create a controller that will return XML.
   - **D. Incorrect:** An ASMX web service is SOAP-based, not REST-based.
2. **Correct answers:** B
   - **A. Incorrect:** Selecting Add Reference does not enable you to create a proxy.
   - **B. Correct:** *HttpService.Get* gets the output of a REST service.
   - **C. Incorrect:** Visual Studio creates a proxy for you from the WSDL at the site you select.
   - **D. Incorrect:** A WCF proxy class needs endpoints and bindings. REST services do not use, nor understand, WCF endpoints and bindings.
3. **Correct answers:** C, D
   - **A. Incorrect:** Both the web application and the database storage are being run from within the company network. Although this is a distributed design, it is not a hybrid application.
   - **B. Incorrect:** Both the web instance and the data repository are using Windows Azure technology. This is not a hybrid app; it is a fully deployed Windows Azure application.
   - **C. Correct:** Part of the application is being run in the Windows Azure environment; the other part is being run in the corporate network environment.
   - **D. Correct:** Part of the application is being run in the Windows Azure environment; the other part is being run in the corporate network environment.

Objective 1.3: Thought experiment

1. When you need something to run the lifetime of the application, putting it into the override of the *Run method* is the best solution. It enables you to create a timer that fires an event every x minutes that you want to run the check.
2. They both would work. The advantage to putting them in a worker process is that they can continue to function if the Web role has stopped for some reason. This is particularly useful if there are other non-web ways of getting information into the database, and you are still at risk for orphaned data.
3. Provided that the processes are console applications, it should be relatively straightforward to move them into methods that can be called from within startup process.

Objective 1.3: Review

1. **Correct answer:** B
    - **A. Incorrect:** If the startup task fires an unhandled error, the role startup stops in a failure. The task will not complete successfully.
    - **B. Correct:** The task will stop processing and return a non-zero value.
    - **C. Incorrect:** The task will stop in error. The *OnStop* process will not run because the role will not get that far.
    - **D. Incorrect:** The task will stop processing. It will not try to continue to run on a lower security setting.
2. **Correct answers:** A, C, D
    - **A. Correct:** AppCmd.exe enables the configuration of virtual sites.
    - **B. Incorrect:** There is no capability to manage users in AppCmd.exe.
    - **C. Correct:** AppCmd.exe supports the administration of general configuration sections.
    - **D. Correct:** AppCmd.exe manages the support of applications.
3. **Correct answers:** A, B, D
    - **A. Correct:** Creating and running an application in parallel is what the *Run* method was designed to allow.
    - **B. Correct:** The polling service is a good example of an activity in which the *Run* method enables a process to work independently of the main role.
    - **C. Incorrect:** The error handling will be managed in the *OnError* event and will not involve the overridden *Run* method.
    - **D. Correct:** Creating and running an application in parallel is what the *Run* method was designed to allow.

Objective 1.4: Thought experiment

1. There are many different ways that you could provision the servers. A typical approach would be to use two servers for SQL Server, with the data replicated between the servers. One of the servers would be the primary SQL server while the other would be the secondary, redundant fallback SQL server. Two other machines could be set up as a web farm to handle the web requests. A fifth server could be added to the web farm, or kept in reserve in case of a failure in either of the server blocks.

2. There does not appear to be any real special cases for state management, so an Out-Proc solution in which IIS is configured to use SQL Server to manage sessions should be acceptable. This would enable the application to send requests to any server in the farm without a loss of state data. Typically, it is best to use the IIS built-in state management systems where available because it frees your team from having to write code that might be redundant.

3. It depends on the two servers that were lost. Using at least two servers for the data tier and the web tier should give you some contingency for hardware failures because it is rare that more than one server goes out at a time. However, if two servers are lost at the same time, the only real risk would be some downtime as you roll the fifth server in to replace one of the ones that was lost. The only real data loss might be if both servers in the database tier were lost, in which case it is likely that there will be some data loss. If you lose one in each tier, or even both web servers in the farm, you can provision the fifth server as a web server without any loss of data other than in those requests that the server was processing as it went down.

Objective 1.4: Review

1. **Correct answers:** C, D

   A. **Incorrect:** InProc does not support web farms as session items are stored only in the individual server's memory.

   B. **Incorrect:** SQLServer is not available in the application stack. This means that using the default SQLServer state is not possible.

   C. **Correct:** Using a shared state server across the web farm is an available option. Using a state server designates one server to maintain state for all the servers that connect to it.

   D. **Correct:** A custom session provider enables you to maintain state as necessary by doing the work in your custom code. It is generally used when you try to use a different RDBMS system or when you do not want to use the default session database design.

2. **Correct answers:** B, D
    - **A. Incorrect:** *localStorage* is HTML5 and is not available in all browsers.
    - **B. Correct:** *Query string* information is available across all browsers and is usable on both the client and server.
    - **C. Incorrect:** Although *ViewState* is available in a form field on the page, it is encrypted and cannot be used on the client side. It is also not used by many ASP.NET MVC 4 constructs.
    - **D. Correct:** Cookies can be stored for a period of time on the client and be read from either client- or server-side operations.
3. **Correct answers:** A, D
    - **A. Correct:** Your application needs to manage whatever information might be required to access the state management system.
    - **B. Incorrect:** The HTTP headers are usually not used as part of state management.
    - **C. Incorrect:** Because your application is sessionless, there is no need to manage session in the Web.config file.
    - **D. Correct:** Your application needs to manage the passing of the identifier between requests.

Objective 1.5: Thought experiment

1. There are several ways that caching could help this process. The first is to get the information from the database and store it in *localStorage*. That way you never need to call the server again unless the client realizes it does not have the information any more. You could also use donut caching or donut hole caching, whichever is more appropriate, to cache that area of the page where the information doesn't change. If you assign a duration of 30 minutes, you decrease a lot of redundant database calls.
2. Because the list of colors, sizes, and so on is the same for all users, you could store this information in a data cache layer in which the system will make only one call into the database every x minutes and will make that same set of returned information available to all users of the system. This gives you an immediate performance gain across all users.

Objective 1.5: Review

1. **Correct answers:** A, D
    - **A. Correct:** Page output caching will cache content at the client side to eliminate some of the required downloads. It is useful in a limited bandwidth environment. It can also be used in donut hole and donut caching scenarios for partial client-side caching.

- **B. Incorrect:** Application caching is an HTML5 feature, and it is unlikely that the older laptops will be able to support the feature.
- **C. Incorrect:** Data caching might decrease some server time, but with the limited number of users, it is unlikely that the data access would be an issue.
- **D. Correct:** HTTP caching will help response time even though there is not much a developer needs to do to implement the caching.

2. **Correct answers:** B, C
   - **A. Incorrect:** Although data caching can add some support in a highly dynamic situation, it does not support the capability to have long-term caching.
   - **B. Correct:** Donut hole caching provides the ability to cache parts of each page.
   - **C. Correct:** Donut caching is another approach that gives the ability to cache parts of the application.
   - **D. Incorrect:** AppFabric caching would provide some support in a highly dynamic situation, but it does not suit the need to store some of the page output.
3. **Correct answers:** A, C, D
   - **A. Correct:** Data caching with the appropriate timeout will enable the data needed for the reports to be stored so that the call to the database is not necessary.
   - **B. Incorrect:** Although a page output caching would be useful, the short time frame of two minutes means that the cache will likely expire before the next user requests the page.
   - **C. Correct:** A page output caching of four hours caches the output of the report for the whole morning and should eliminate the need for the report to be run a second time.
   - **D. Correct:** AppFabric caching acts much like data caching to eliminate the need for additional calls to the database to generate the reports.

Objective 1.6: Thought experiment

1. The most common set of issues you would encounter when creating a solution that includes WebSockets is the nonuniversal support for HTML5. It is possible aspects of the company's business still run on non-HTML5-compliant browsers. Other issues you could encounter include proxy servers, firewall filters, and other security systems that might look at nontraditional HTML communications as a threat.
2. When following a traditional SoC route, the design should manage each different type of communication separately, even though it might be on the same page. This would give them the opportunity to change independently of each other, perhaps by moving to a different server or even starting to take the news feed from a third-party service directly.

3. You need to create the server-side application that will be notified of news articles and send the information to the users. You also need to create a server-side application that will manage the instant messaging part of the application. Theoretically they could be the same application, but it would be prudent to design them in such a way that they could scale separately and independently.

Objective 1.6: Review

1. **Correct answer:** B
   - **A. Incorrect:** In HTTP polling, the client sends a request to the server, and as soon as the response is returned, it sends a new request.
   - **B. Correct:** In HTTP long polling, the client sends a request to the server, and the server holds it open until it either has something to return to the client or the connection times out.
   - **C. Incorrect:** WebSockets are a way for two-way communication between the client and the server. The server does not hold onto the response.
   - **D. Incorrect:** The request-response path is a traditional HTTP connection.
2. **Correct answers:** A, D
   - **A. Correct:** WebSockets can be used to pass information between the client and server.
   - **B. Incorrect:** HTTP polling can provide the need, but the 1-minute refresh interval would not be efficient.
   - **C. Incorrect:** HTTP long polling is not a valid strategy. The typical timeout on a single request is less than 15 minutes, and chaining multiple requests to get the 15-minute timespan is resource intensive.
   - **D. Correct:** HTTP polling with 15-minute intervals is a valid way to get the information within the required time frame.
3. **Correct answer:** B
   - **A. Incorrect:** HTTP DELETE is not used to start the WebSocket connection; it is instead used to perform a delete on a discrete item.
   - **B. Correct:** The first request to open a WebSocket connection is a standard HTTP GET. After the request is received, the browser sends a separate upgrade request.
   - **C. Incorrect:** HTTP CONNECT converts the request connection to a transparent TCP/IP tunnel.
   - **D. Incorrect:** The upgrade request is sent after the server has responded to an HTTP GET request.

Objective 1.7: Thought experiment

1. Creating an HTTP handler is a relatively simple way to create a customized process to return XML or JSON return objects. Using it in a RESTful scenario is more complicated because there is no extension to also map the handler. You have to manage all requests without an extension and then filter the URL request to see what the appropriate response would be.
2. The *AuthenticateRequest* and *AuthorizeRequest* events are the traditional access points for authorization and authentication. You add the event handlers in the *Init* method and you have access to the entire HTTP Request in the module as it moves through the application stack.
3. It would be a relatively responsive application, especially when comparing it to traditional ASP.NET MVC applications. Because it would use its own custom handler, a lot of the overhead of MVC would be left out of the process.

Objective 1.7: Review

1. **Correct answer:** C
   - **A. Incorrect:** The *PostAuthorizeRequest* event is thrown before the handler is mapped.
   - **B. Incorrect:** You can handle the mapping of the request in the *MapRequestHandler*.
   - **C. Correct:** You handle the mapping of the request in the *MapRequestHandler*.
   - **D. Incorrect:** The *ReleaseRequestState* is thrown after the handler has completed.
2. **Correct answer:** C
   - **A. Incorrect:** *object, EventArgs* are the parameters used for the event handlers thrown during the startup process. The event handlers are assigned in the *Init* method.
   - **B. Incorrect:** *HttpApplication* is the parameter used in the *Init* method.
   - **C. Correct:** The *ProcessRequest* method takes *the HttpContext* parameter.
   - **D. Incorrect:** There are no default methods that just accept an object parameter.
3. **Correct answer:** C
   - **A. Incorrect:** A module is not the best way to handle the request because it would have to deal with every HTTP request rather than just the image calls.
   - **B. Incorrect:** Serving .htm and .html pages will not create watermarks on image files.
   - **C. Correct:** Intercepting every request for .jpg and .png files is the easiest way to consistently add watermarks to the images.
   - **D. Incorrect:** A module is not the best way to handle the request because it would have to deal with every HTTP request rather than just the image calls.

Index

About the Author

WILLIAM PENBERTHY is a software developer and educator living in Denver, Colorado. William has been working in various aspects of the software development life cycle for more than 25 years, focusing on Microsoft technology-specific development since 2005. He has been part of the development of more than 125 different applications, ranging from client applications to web services to websites, and has taught software development classes and in-services since 1998.

William is an application development consultant for RBA (*http://www.rbaconsulting.com*). RBA was named a Microsoft 2013 Partner of the Year and specializes in offering custom application development, infrastructure, portals, data management, and digital strategy solutions for clients.