

## Kapitola 9

# Vytváření služeb RESTful a využívání aplikace SilverTwit

V předcházející kapitole jste se dozvěděli, jak lze využívat výhody služeb RESTful v rozhraní API webových služeb Amazon při vytváření klientské aplikace Silverlightu 2, jež bude uživateli umožňovat vyhledávání a nakupování na webu Amazon.com. Na případové studii jsme si ukázali, jak lze požadovat data pomocí metody HTTP GET, ale existují samozřejmě i další způsoby vystavování dat pomocí HTTP přes metody POST, PUT, DELETE a další metody HTTP. V této kapitole zmíněná témata dále rozšíříme a ukážeme si, jak lze vytvářet uživatelské webové služby REST pomocí Windows Communication Foundation (WCF), podporovat akce HTTP, upravovat šablony URI a vracet data ve formátu XML nebo JSON.

Navrhujete-li webové služby za účelem vystavení funkcí a zdrojů klientským aplikacím, jsou webové služby RESTful v mnoha případech dobrou alternativou k využívání služeb založených na SOAP. Vlastní služby RESTful lze vytvářet různými způsoby; WCF však nabízí mnoho možností potřebných k vytvoření služby RESTful. V této kapitole se dozvíte, jak lze pomocí WCF vytvářet vlastní služby RESTful, jež budou odesílat a přijímat požadavky a odpovědi. Seznámíte se také s různými metodami HTTP a naučíte se, jak implementovat stavové kódy a definovat šablony URI.

Službu RESTful lze navrhovat tak, aby se data vracela v různých formátech. Běžnými datovými formáty jsou XML a JSON. V této kapitole se dozvíte, jak navrhovat služby, jež odesílají odpovědi ve formátu JSON nebo XML, a uvidíte, jak lze odpověď zpracovávat pomocí nástrojů, jako je LINQ to XML, LINQ to JSON a `DataContractJsonSerializer`. Jednotlivá témata si představíme v jednoduchých příkladech, v nichž budeme vytvářet aplikace Silverlightu 2 volající vlastní webové služby WCF RESTful.

Tyto příklady vás provedou vytvářením klienta Silverlightu Twitter, jenž čte a posílá zprávy z Twitter API RESTful pomocí HTTP s využitím akcí GET a POST. Příklad navazuje na témata, s nimiž jste se v této knize již setkali při vytváření klienta Silverlightu, vytváření služby RESTful navržené pomocí WCF, řešení problémů při komunikaci mezi doménami a LINQ. V této kapi-

tole se zaměříme na kód na straně serveru (služby RESTful) a na straně klienta (pro využívání služeb).

## Vytvoření služeb RESTful pomocí WCF

V kapitole 7 jsme probírali strukturu služby RESTful, jedinečný identifikátor URI a služby řízené zdroji. V této části si ukážeme, jak lze pomocí WCF vytvořit službu RESTful, kterou může Silverlight využívat.

Před vytvořením služby RESTful je důležité zvážit rozdíly mezi charakteristickými rysy služby RESTful a webovou službou vytvořenou na základě SOAP. Obecně se služby WCF zaměřují na provádění akcí, jako je získávání seznamu produktů (`FindProducts()`), přidávání zákazníků (`AddCustomer(Customer cust)`) nebo vyhledávání objednávek (`FindOrders(int year)`). Někteří vývojáři berou tyto typy služeb jako *slovesa*, protože jsou založeny na akcích. Webové služby založené na REST se obecně zaměřují na zdroj a ne na akci; služby založené na REST si lze představit jako služby, jež se točí kolem *podstatných jmen* (zdrojů).

Zatímco ve WCF může být metoda služby `FindProducts()`, v operaci webové služby REST by identifikátor URI mohl být voláním metody HTTP GET na `http://silverlight-data.com/MyService.svc/Product`. Ačkoli metoda služby WCF může být `AddCustomer(Customer cust)`, služba RESTful může využívat metodu HTTP POST na URI `http://silverlight-data.com/MyService.svc/Custom`. Styly těchto služeb se jasně liší jak v implementaci, tak v návrhu. Při využití druhého stylu musí vývojář více přemýšlet o práci se zdroji, zatímco u prvního stylu jde o akci, jež má být provedena.

## Vytváření služby RESTful

Chcete-li vytvořit webovou službu RESTful pomocí WCF při využití IIS nebo Windows Process Activation Service (WPAS), proveďte následující kroky:

1. Vytvořte soubor `RESTfulService.svc` v kořeni projektu.
2. Vytvořte soubor s kódem `RESTfulService.cs`.
3. Nakonfigurujte službu RESTful v souboru `*.config` nebo ve vlastním souboru `.svc`. Volitelně lze vytvořit rozhraní `IRESTfulService.cs`.
4. Definujte vlastnosti jednotlivých operací.

Prvním krokem při vytváření služby RESTful pomocí WCF je přidání nového souboru služby WCF do projektu. Ve zkušebním kódu je využit projekt webové aplikace hostící webové služby RESTful WCF. Službu WCF lze vytvořit pomocí šablony služby WCF nebo prostřednictvím šablony Silverlight-enabled WCF Service. Bez ohledu na zvolenou šablonu je nutné upravit konfiguraci služby tak, aby podporovala službu RESTful.

## Vytváření rozhraní služby

Kód ve zkušební aplikaci vytvořil službu WCF pomocí šablony služby WCF. Tato služba s názvem *RESTfulService.svc* implicitně definuje své kontrakty služby a operace v souboru *RESTfulService.cs*. Službu a kontrakty operací lze případně přesunout do rozhraní.

Definování rozhraní je volitelný úkon, jenž umožňuje veřejně vystavovat metody webové služby, přičemž služby jsou definovány stručně a jasně na jednom místě. Metody webové služby (rovněž označované jako *operace*) definujete v rozhraní spolu s atributy, jež určují charakteristické rysy (například šablona URI a akce HTTP) každé operace. Ve zkušebním kódu je služba implementována ve třídě *RESTfulService*, jež implementuje rozhraní s názvem *IRESTfulService*. Rozhraní *IRESTfulService* definuje kontrakt služby a operace.

## Konfigurace služby RESTful

Konfigurace webové služby WCF podporující REST vyžaduje jiné aspekty, než webové služby WCF podporující SOAP. Služba RESTful nemusí zveřejňovat svá metadata, a proto není nutné vytvářet specifické chování služby. To znamená, že lze v konfiguračním souboru vynechat podřízený prvek *serviceBehaviors* v *system.serviceModel* služby REST.

Služby založené na SOAP, jež jsou využívány aplikacemi Silverlightu, musí využívat vazbu *basicHttpBinding*. Naproti tomu webové služby WCF REST využívají vazbu *webHttpBinding*. Pro služby vystavované pomocí *webHttpBinding* nejsou zveřejňována žádná metadata, a proto nepotřebujete část *serviceBehavior* pro služby REST. Ačkoli nepotřebujete část *serviceBehavior*, webová služba REST vyžaduje režim práce koncového bodu. V příkladu 9.1 vidíte správné uspořádání webové služby WCF RESTful, včetně nastavení *webHttp* v chování koncového bodu s názvem *webBehavior*.



Celou sadu kódů pro všechny příklady v této kapitole naleznete v řešení *RESTfulClient*, jež se nachází ve složce s kódem pro tuto kapitolu.

### Příklad 9.1 Konfigurace RESTful

```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="webBehavior">
        <webHttp/>
      </behavior>
    </endpointBehaviors>
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
  <services>
    <service name="RESTfulService.RESTfulService">
      <endpoint address="" behaviorConfiguration="webBehavior">
```

```

        bi ndi ng="webHt tpBi ndi ng"
        bi ndi ngConfi gurati on=""
        contract="RESTful Servi ces. I RESTful Servi ce" />
    </servi ce>
</servi ces>
</system. servi ceModel >

```

Část `service` v konfiguraci definuje název služby a její podřízený prvek `endpoint` definuje podrobné informace o koncovém bodu webové služby RESTful. Atribut `name` prvku `service` představuje úplný název implementované služby. Atribut `address` je v příkladu 9.1 ponechán prázdný. To znamená, že koncový bod bude použit v případě, že identifikátor URI odkazuje do kořene služby. Definujete-li více koncových bodů – například jeden pro službu RESTful a další pro službu SOAP – lze je rozlišit pomocí atributu `address`. Pro koncový bod RESTful lze například nastavit adresu `rest` a koncový bod SOAP může mít nastavenou adresu `soap`. Díky tomu byste mohli volat oba typy služeb jednoduše přidáním jedné z těchto adres do identifikátoru URI. Například následující URI odkazuje na webovou službu REST:

*<http://silverlight-data.com/MyService.svc/rest/>*

Tento odkaz sleduje webovou službu SOAP:

*<http://silverlight-data.com/MyService.svc/soap/>*

Atribut `behaviorConfiguration` odkazuje na režim koncového bodu `webBehavior`. V tomto případě to umožňuje službě podporovat charakteristické rysy REST. Pro tento koncový bod REST není nutné provádět žádnou konfiguraci vazby.

Atribut `contract` odkazuje na název třídy nebo rozhraní definující kontrakt služby. V tomto příkladu je kontraktem rozhraní s názvem `IRESTfulService`. Využijete-li k vytvoření služby šablonu Silverlight-enabled WCF Service, nevytvoří tato šablona rozhraní. Pokud rozhraní přidáte, případně upravíte kód (jako v tomto příkladu), je nezbytné změnit název kontraktu v konfiguraci tak, aby odkazoval na rozhraní a nikoli na třídu. Ve své podstatě musí být kontrakt názvem objektu, jenž má atribut `ServiceContract`.

Posledním atributem koncového bodu je vazba. Služba WCF implicitně využívá režim vazeb `wsHttpBinding`, který nemohou využívat klienti Silverlightu. Využijete-li šablonu Silverlight-enabled WCF Service, implicitně se nastaví režim vazeb `basicHttpBinding`. Při vytváření a využívání služby WCF založené na SOAP (v tomto případě aplikaci Silverlightu) nastavíte režim vazeb na `basicHttpBinding`. Žádné z těchto nastavení vazeb nepodporuje služba RESTful, proto je nastaven režim vazeb `webHttpBinding`.

## Definování kontraktu

Poté, co jste vytvořili a nakonfigurovali webovou službu WCF RESTful, jež je připravena k využívání, je nutné vystavit jednotlivé operace, které bude služba vykonávat. Tyto operace mají

atribut `OperationContract` ve jmenném prostoru `System.ServiceModel` a nacházejí se v rozhraní `IService`.

Ke každé operaci REST lze přidat další atribut. Jmenný prostor `System.ServiceModel.Web` obsahuje atributy `WebGet` a `WebInvoke`. Tyto atributy deklarují, že operace webové služby podporuje jednu z metod HTTP, jako GET, POST, PUT nebo DELETE. Nemá-li operace kontraktu služby atribut `WebGet` nebo `WebInvoke`, využije se implicitně metoda POST, přičemž `UriTemplate` představuje název operace. V praxi se však osvědčilo přidávat ke všem operacím atribut `WebGet` nebo `WebInvoke`.



Před přidáním atributů `WebGet` a `WebInvoke` musí projekt odkazovat na sestavení `System.ServiceModel.Web.dll`.

Na obrázku 9.1 vidíte několik charakteristických rysů služby webové operace REST, jež lze nastavit, včetně atributů `WebGet` či `WebInvoke`. Atribut `WebGet` indikuje, že bude proveden požadavek HTTP GET a lze požadovat odpověď. Atribut `WebInvoke` určuje, že bude provedena metoda HTTP GET, POST, PUT nebo DELETE. Ačkoli služby RESTful vytvořené pomocí WCF podporují metody PUT a DELETE, Silverlight může využívat pouze metody HTTP GET nebo HTTP POST pomocí třídy `WebClient` nebo `HttpWebRequest`. Silverlight však může využívat metody PUT nebo DELETE prostřednictvím rozhraní Javascript API a voláním přes objekt `XmlHttpRequest`.

<b>Metoda HTTP</b>	GET (WebGet)
	POST (WebInvoke)
<b>UriTemplate</b>	Jedinečný identifikátor URI
<b>Styl parametrů</b>	Samostatný segment
	Výchozí hodnota
<b>Formát odpovědi</b>	JSON
	XML
<b>Styl textu</b>	Prostý
	Zabalený
<b>Stavové kódy HTTP</b>	<input type="text"/> <input type="text"/> <input type="text"/>

Obrázek 9.1 Návrh služby RESTful

Příklad 9.2 ukazuje kontrakt služby a první operaci pro rozhraní `IRESTfulService`. `UriTemplate` v příkladu 9.2 indikuje, že kombinace cesty služby a `UriTemplate` vytvoří jedinečnou cestu, jež vyvolá operaci `FindProduct1`. Služba je hostována na adrese `http://localhost/RESTfulServices/RESTfulService.svc`. Úplná cesta pro vyvolání této operace a předání hodnoty pro parametr `productIdString` zní `http://localhost/RESTfulServices/RESTfulService.svc/Product/1001`.

Atributy `WebGet` indikují, že bude proveden požadavek HTTP GET. `UriTemplate` definuje, že ve zdroji `Product` bude dotazován určitý identifikátor produktu. `ResponseFormat` určuje, že odpověď bude odeslána zpět klientovi ve formátu XML a `BodyStyle` definuje, že požadavek a odpověď budou v prostém textu a nebudou zabaleny. V případě potřeby lze požadavky a odpovědi zabalit do kontejnerového prvku. V následujících příkladech není nutné požadavky a odpovědi zabalit, proto jsou označeny jako `Bare`. Zabalené požadavky a odpovědi jsou ideální v případě, kdy obsah zprávy vyžaduje kontejnerový prvek. Například zpráva odpovědi může být částí XML bez kořenového prvku. Použijete-li `WebMessageBodyStyle.Wrapped`, může být část XML automaticky zabalena s vnějším prvkem XML.

### Příklad 9.2 Kontrakt služby a operace

**C#** `[ServiceContract(Namespace = "http://www.silverlight-data.com")]`

```
public interface IRESTfulService
{
    // #1
    [OperationContract]
    [WebGet(UriTemplate = "Product/{productIdString}",
        ResponseFormat = WebMessageFormat.Xml,
        BodyStyle = WebMessageBodyStyle.Bare)]
    Product FindProduct1(string productIdString);
    // Zde jsou definované další kontrakty ...
}
```

**VB** `<ServiceContract(Namespace := "http://www.silverlight-data.com")> _`

```
Public Interface IRESTfulService
    ' #1
    <OperationContract, WebGet(UriTemplate := "Product/{productIdString}", _
        ResponseFormat := WebMessageFormat.Xml, _
        BodyStyle := WebMessageBodyStyle.Bare)> _
    Function FindProduct1(ByVal productIdString As String) As Product

    ' Zde jsou definované další kontrakty ...
```

`End Interface`

Implementace metody `FindProduct1` v příkladu 9.3 ukazuje, že parametr v `UriTemplate` se shoduje s názvem v metodě `FindProduct1`. Každý parametr definovaný v `UriTemplate` je nutné

uzavřít do složených závorek a parametr se musí shodovat s názvem parametru v metodě. Je-li parametr součástí cesty URI, musí se jednat o řetězec `string`. Je-li parametr součástí dotazovacího řetězce, je podporován základní typ konverze. Protože vlastnost `ProductId` třídy `Product` je celé číslo, je nejprve nutné převést parametr `productIdString` na celé číslo tak, jak to vidíte v příkladu 9.3.

Parametr lze rovněž definovat jako součást složeného segmentu; ve složeném segmentu existuje cesta a parametr ve stejném segmentu URI. Například `UriTemplate` z `Product` (`{productIdString}`) obsahuje složený segment, přičemž parametr (uvnitř závorek) je předáván ve stejném segmentu, jako `Product`. Příklad by mohl vypadat takto:

```
/somepath/some.svc/Product(123)
```

Po získání příslušného produktu pomocí LINQ to Object do metody `FindProduct1`, vrátí se `Product` do klienta prostřednictvím `Response`. Protože `ResponseFormat` je nastaven na `WebMessageFormat.Xml`, je `Product` před odesláním zpět v rámci `Response` převeden do XML. Před příchodem verze .NET SP1 bylo nutné přidávat do třídy a jejích vlastností atributy `DataMember` a `DataContract`. Chcete-li do procesu serializace přidat nebo z něj vyloučit určité vlastnosti, lze počínaje verzí .NET SP1 přidávat ke třídě `Product` atribut `DataContract` a k jejím vlastnostem atribut `DataMember`.

### Příklad 9.3 Jednoduchý požadavek GET na produkt

```
C# public Product FindProduct1(string productIdString)
{
    int productId = int.Parse(productIdString);
    List<Product> products = GetProductList();
    var query = from p in products
                where p.ProductId == productId
                select p;
    Product product = query.FirstOrDefault() as Product;
    return product;
}
```

```
VB Public Function FindProduct1(ByVal productIdString As String) As Product
    Dim productId As Integer = Integer.Parse(productIdString)
    Dim products As List(Of Product) = GetProductList()
    Dim query = _
        From p In products _
        Where p.ProductId = productId _
        Select p
    Dim product As Product = TryCast(query.FirstOrDefault(), Product)
    Return product End Function
```

## Využívání služeb REST

Poté, co definujete službu, lze nastavit klienta, jenž bude tuto službu využívat. Klientská aplikace Silverlightu s názvem `RESTfulClient` (viz obrázek 9.2) ve zkušebním kódu volá službu `RESTful` (viz obrázek 9.3) pomocí třídy `WebClient` a jedinečného identifikátoru URI, jenž odpovídá operaci `FindProduct1`. Aplikace zobrazí seznam operací, jež volají různé metody webové služby z projektu `RESTfulService`. Příklad 9.4 ukazuje, jak klientská aplikace Silverlightu využívá metodu webové služby `FindProduct1`.



Každá metoda volání má v komentářích stejné číslo, jako odpovídající metoda webové služby. Díky tomu lze lépe rozpoznat, které metody volají jednotlivé metody webové služby.

### Příklad 9.4 Volání služby RESTful z aplikace Silverlightu

```
C# private readonly string _domain = "localhost:9726";

public string BaseUri
{
    get { return string.Format("http://{0}/RESTfulService.svc", _domain); }
}

private void GetProductAsync_Async_IIDInSegment()
{
    WebClient wc = new WebClient();
    wc.DownloadStringCompleted += ParseProducts_Async;
    int productId = 1001;
    string urlString = string.Format("{0}/Product/{1}", BaseUri, productId);
    Uri uri = new Uri(urlString);
    wc.DownloadStringAsync(uri);
}

VB Private ReadOnly _domain As String = "localhost:9726"

Public ReadOnly Property BaseUri() As String
    Get
        Return String.Format("http://{0}/RESTfulService.svc", _domain)
    End Get
End Property

Private Sub GetProductAsync_Async_IIDInSegment()
    Dim wc As New WebClient()
    wc.DownloadStringCompleted += ParseProducts_Async
    Dim productId As Integer = 1001
```

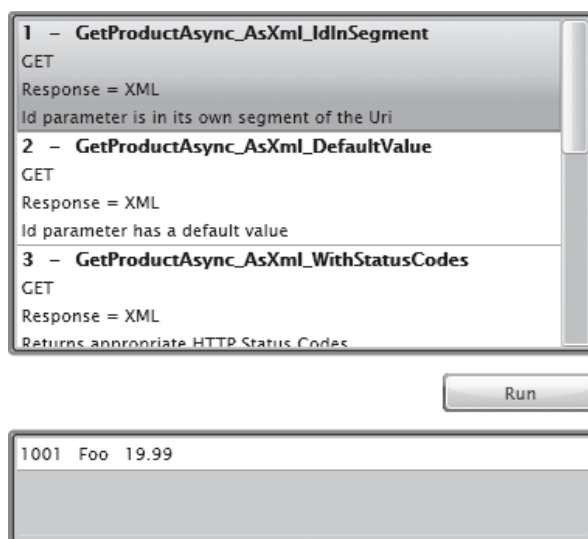


```

Dim urlString As String = String.Format("{0}/Product/{1}", BaseUri, productId)
Dim uri As New Uri(urlString)
wc.DownloadStringAsync(uri)
End Sub

```

Metoda `ParseProducts_AsXml` obsluhuje událost `DownloadStringCompleted` pro metody ve zkušebním kódu, jež získávají jeden či více produktů ve formátu XML. Parsuje kód XML tak, aby byly nalezeny hodnoty vlastností `ProductId`, `UnitPrice` a `ProductName` a pro každou z nich vytváří instanci třídy `Product`. Metody `ToInt` a `ToDecimal` jsou rozšiřující metody třídy `string`, jež jednoduše parsuje hodnotu řetězce na celé nebo desetinné číslo. Jsou definovány ve třídě `ConversionExtensions` v projektu `RESTfulClient`. Tato definice dotazu LINQ to Query funguje, ať již webová služba vrátí jeden či více produktů.



Obrázek 9.2 Získání produktu z webové služby RESTful

Klientská aplikace Silverlightu se nachází v jiné doméně než webová služba (jako u skutečných aplikací). Protože se jedná o mezidoménový požadavek, je nutné umístit do kořene webu soubor s mezidoménovými zásadami, jenž umožňuje přístup ke službám. Zkušební kód pro tuto kapitolu obsahuje kopii souboru *clientaccesspolicy.xml* v kořeni domény. V příkladu 9.5 vidíte, jak klientská aplikace Silverlightu parsuje XML odpovědi pomocí LINQ to XML.

### Příklad 9.5 Parsování odpovědi pomocí LINQ to XML

```

private void ParseProducts_AsXml(object sender, DownloadStringCompletedEventArgs e)
{
    string ns = "";
    string rawXml = e.Result;

```

```

XDocument xdoc = XDocument.Parse(rawXml);
var query = from product in xdoc.Descendants(ns + "Product")
select new Product
{
    ProductId = product.Element("ProductId").Value.ToInt(),
    ProductName = product.Element("ProductName").Value,
    UnitPrice = product.Element("UnitPrice").Value.ToDecimal();
};
List<Product> products = query.ToList() as List<Product>;
IstProducts.DataContext = products;
}

```

```

VB Private Sub ParseProducts_AsXml (ByVal sender As Object, _
    ByVal e As DownloadStringCompletedEventArgs)
    Dim ns As String = ""
    Dim rawXml As String = e.Result
    Dim xdoc As XDocument = XDocument.Parse(rawXml)
    Dim query = _
        From product In xdoc.Descendants(ns & "Product") _
        Select New Product With _
        { _
            .ProductId = product.Element("ProductId").Value.ToInt(), _
            .ProductName = product.Element("ProductName").Value, _
            .UnitPrice = product.Element("UnitPrice").Value.ToDecimal() _
        }
    Dim products As List(Of Product) = TryCast(query.ToList(), List(Of Product))
    IstProducts.DataContext = products End Sub

```

## Výchozí hodnoty

V příkladu 9.6 vidíte, jak lze upravit identifikátor URI, aby parametr `productIdString` mohl mít výchozí hodnotu. Je-li vyvolán URI shodující se s `UriTemplate`, jenž vynechává parametr `productIdString`, bude tomuto požadavku odpovídat následující operace `UriTemplate`.

### Příklad 9.6 Upravená šablona URI ve službě

```

C# // #2
[OperationContract]
[WebGet(UriTemplate = "Product2/{productIdString=1011}",
    ResponseFormat = WebMessageFormat.Xml,
    BodyStyle = WebMessageBodyStyle.Bare)]
Product FindProduct2(string productIdString);

```

```

VB ' #2

```

```

<OperationContract, _
    WebGet(UriTemplate := "Product2/{productIdString=1011}", _
    ResponseFormat := WebMessageFormat.Xml, _
    BodyStyle := WebMessageBodyStyle.Bare)> _
Product FindProduct2(String productIdString)

```

Kód pro metodu `FindProduct2` je stejný jako je kód pro `FindProduct1` (viz příklad 9.3). Instance `Product` je převedena do XML a poslána zpět klientovi Silverlightu v `Response`, kde je parsována pomocí LINQ to XML v metodě `ParseProducts_AsXml` (viz příklad 9.5). Výsledky jsou pak svázány do `ListBox` a zobrazí se uživateli.

## Stavové kódy HTTP

Stavové kódy umožňují informovat o stavu operace, jež je vyvolána aplikací přes HTTP.

Při vývoji webových služeb RESTful je vhodné vracet po dokončení operace stavové kódy HTTP. Stavové kódy HTTP mohou signalizovat úspěšný průběh i hlásit chyby operací webové služby. Obsahuje-li například parametr neočekávanou hodnotu, lze nastavit stavový kód HTTP na hodnotu enumerátoru `HttpStatusCode.BadRequest`, jež je převedena na stav HTTP 400. Pokud metoda nalezne produkt a úspěšně jej vrátí, lze nastavit stavový kód HTTP na `HttpStatusCode.OK`, což vrátí stav HTTP 200.

K výchozím stavovým kódům patří 200 (OK) a 500 (Vnitřní chyba serveru). Nedojde-li v metodě k žádné výjimce a v metodě není explicitně nastaven žádný stavový kód, vrátí se stavový kód 200. Pokud dojde k výjimce a není nastaven žádný stavový kód, vrátí se stavový kód 500.

Výčet `HttpStatusCode` naleznete ve jmenném prostoru `System.Net`. Kódy mapují přímo do standardních stavových kódů HTTP, jako je 400 pro `bad request`, 404 pro `not found` a 201 pro `created`. Kód v příkladu 9.7 vyhledá produkt a nastaví stavový kód HTTP na příslušnou hodnotu enumerátoru. Tyto hodnoty lze rozpoznat, spustíte-li službu pomocí nástroje, jako je například Fiddler2 (viz obrázek 9.3).

#	Result	Pro...	Host	URL	Body	Caching	Content-Type	Process
27	200	HTTP	localhost	/RESTfulServices/RESTfulService.svc/Product3/1001	354	private	application/xml; ...	

Obrázek 9.3 Stav HTTP 200

### Příklad 9.7 Přiřazování stavových kódů HTTP

```

4 // # 3
public Product FindProduct3(string productIdString)
{
    if (productIdString.Length == 0)
    {
        WebOperationContext.Current.OutgoingResponse.StatusCode =
            HttpStatusCode.BadRequest;
    }
}

```