Webové služby

Obsah

Architektury orientované na služby (Service-oriented Architectures - SOA)

Motivace k SOA

- Obecný příklon k chápání IT jako poskytovatele služby - servisu - nikoli jen technologie.

- Stejně se začíná přistupovat i k vazbě mezi SW komponentami.
- Orientace na služby se ve vývoji SW stává vůdčím směrem.

Definice SOA

Architektury orientované na služby (Service-oriented Architectures - SOA) představují princip budování softwarových architektur založený na:

- komponentách umístěných v uzlech sítě
- nabízející své služby ostatním prostřednictvím dobře definovaného protokolu
- služby jsou bezestavové
- vnitřní realizace klienta obvykle nezajímá - standardy zajišťují interoperabilitu služeb

Technologie SOA

SOA v zásadě neváže na žádnou SW platformu, OS, pg. jazyk ani konkrétní standard.

V úzkém slova smyslu se za SOA dnes někdy považují architektury budované na tzv. *Web Services* (webových službách, WS) kolem množiny standardů:

- XML - data reprezentovaná značkovanými dokumenty
- HTTP jako základní webový protokol
- SOAP - jako std. protokol WS
- WSDL - popisovač webových aplikací
- UDDI - služby registrace a vyhledávání webových služeb

Charakteristiky SOA

Orientace na služby přináší:

1. Nahraditelnost služeb
2. Interoperabilitu
3. Možnost nezávislého ladění služeb
4. Usnadňuje integraci systémů třetích stran.

Vztah komponent a služeb

Oboje jsou moderní až módní záležitosti, vztah lze definovat takto:

- Komponenta je často entitou realizující určitou službu.
- Koncový uživatel nahlíží více na služby, zatímco vývojáře zajímají rovněž komponenty, které je realizují.

Odkazy

Kromě základního odkazu na Wikipedii [http://en.wikipedia.org/wiki/Service-oriented_architecture] lze navštívit/pročíst:

- What is Service-Oriented Architecture? [http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html]
- SOA Center [http://www.soacenter.com/]

Web Services

Motivace

Long long time ago

Motivace

Logn time ago

Úvod

Webové služby jsou dalším významným krokem ve vývoji distribuovaných systémů. Navazují na technologie RPC, DCOM, CORBA, RMI. Tyto technologie umožňují volat funkci na vzdáleném systému. Na rozdíl od těchto technologií je však technologie webových služeb platformě zcela nezávislá (nezávislá na operačním systému a programovacím jazyku). Platformní nezávislost je dána použitím XML jako formátu pro vzájemnou výměnu dat.

Webové služby, stejně jako webové služby s uživatelským rozhraním, umožňují jednoduchou komunikaci mezi aplikacemi na nejrůznějších platformách. Aplikace spolu komunikují prostřednictví XML zpráv dohodnutým protokolem. Klient pošle požadavek webové službě ve formě XML souboru. Tento požadavek producent vyhodnotí a odpověď pošle zpět konzumentovi, opět formou XML souboru.

Webové služby jsou postaveny na těchto technologiích:

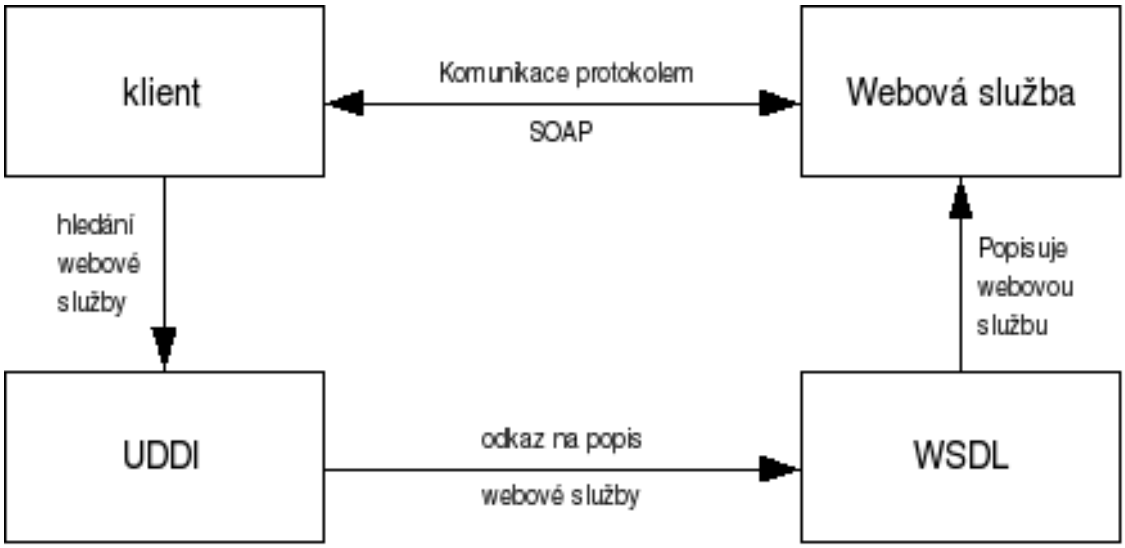
- SOAP (Simple Object Access Protocol) - Protokol používaný pro komunikaci pomocí XML zpráv.

- WSDL (Web Services Description Language) - Standardní formát pro popis rozhraní webové služby.
- UDDI (Universal Description, Discovery and Integration) - Standardní mechanismus registrace a vyhledávání webových služeb.

Princip webové služby

Princip použití webové služby je poměrně jednoduchý. V registru webových služeb (UDDI) vybereme webovou službu, kterou chceme použít. Z popisu webové služby (WSDL), který získáme z UDDI, zjistíme bližší informace o webové službě. Pošleme této webové službě strukturovaný dotaz protokolem SOAP. Tímto protokolem také dostaneme odpověď. Strukturu dotazu i odpovědi známe z popisu služby (WSDL). Odpověď zpracujeme a zobrazíme třeba ve formě HTML stránky.

Obrázek 1. Technologie webových služeb



Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Existují systémy, v nichž lze přímo z WSDL dokumentu vygenerovat požadavek ve formátu XML nad protokolem SOAP.

SOAP

Protokol SOAP je základem webových služeb. Je to protokol umožňující volat vzdálené objekty pomocí XML zpráv. Ze všech tří standardů SOAP, WSDL a UDDI vznikl nejdříve. Jako první začala SOAP protokol vyvíjet v roce 1998 skupina firem kolem Microsoftu. Koncem roku 1999 vznikla první verze protokolu SOAP, SOAP 1.0. V průběhu následujícího roku se připojila i IBM a vznikla druhá verze protokolu SOAP, SOAP 1.1. Přestože ji konsorcium W3C oficiálně nepřijalo jako standard, je dodnes nejpoužívanější. Za standard lze považovat až verzi SOAP 1.2, která má již status W3C Recommendation.

Původní účel protokolu SOAP bylo vytvořit protokol pro vzdálené volání procedur (RPC) založený na XML. Dnes je již SOAP navržen pro obecnější uplatnění než RPC. I když se nejčastěji používá v kombi-

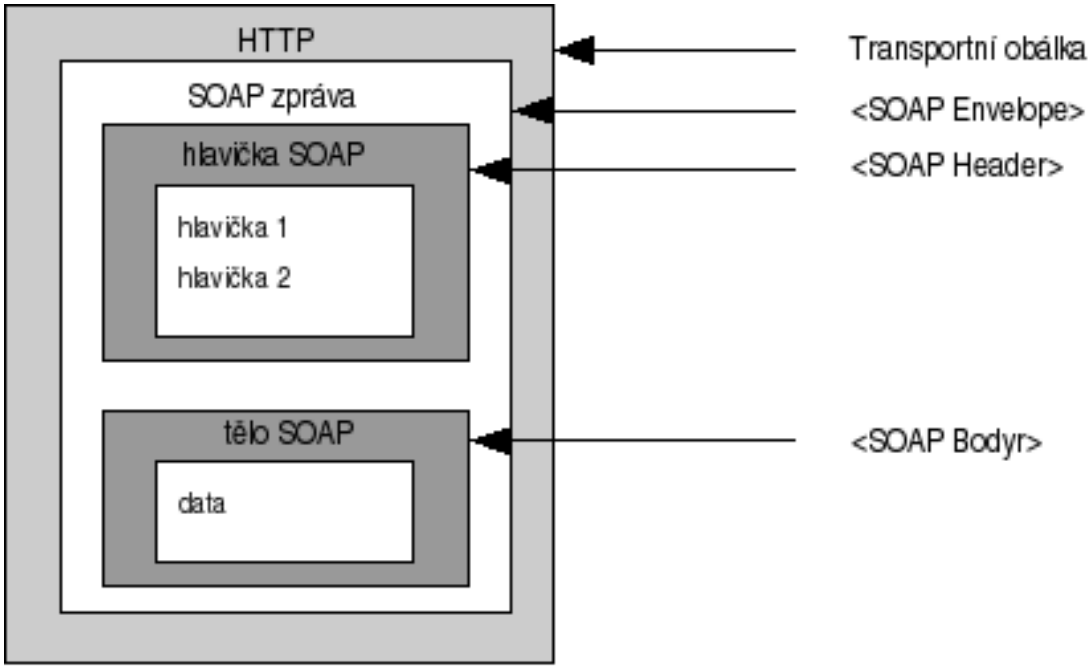
naci s protokolem HTTP, není s tímto protokolem nijak svázán. SOAP zprávy mohou být přenášeny třeba protokolem SMTP. V současné době existuje několik desítek různých implementací SOAP na nejrůznějších platformách, od C/C++, přes Javu, .NET až po Perl.

V podstatě se pomoci HTTP requestu metodou POST místo obvyklých HTTP parametrů přenese speciální XML soubor s definovanou strukturou, který definuje jaka funkce a s jakýma parametrama se má na volaném serveru vyvolat. Výsledek volaní je opět XML soubor.

Struktura SOAP zprávy

Každá SOAP zpráva je jednoduchý XML dokument. Tento XML dokument má kořenový element obálku (envelope). Obálka obsahuje další dva elementy, nepovinnou hlavičku (header) a tělo (body).

Obrázek 2. Struktura SOAP zprávy



Význam těchto elementů je následující:

- Obálka - Kořenový povinný element zprávy.
- Hlavička - Nepovinný element, obsahuje pomocné informace pro zpracování zprávy (například informace o autentizaci).
- Tělo - Obsahuje vlastní zprávu.

Použití SOAPu

Nyní si ukážeme příklad klasické SOAP zprávy zaslané webové službě. Ptáme se na hodnotu akcií s kódem DIS. V těle zprávy je požadavek na volání vzdálené funkce `GetLastTradePrice`, která zjistí poslední známou cenu akcií s kódem DIS.

Příklad 1. Volání SOAP

```
<soapenv:Envelope
  xmlns:soapenv=
        "http://schemas.xmlsoap.org/soap/envelope/">
  soapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/>
    <soapenv:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <m:tickerSymbol>DIS</m:tickerSymbol>
        </m:GetLastTradePrice>
    </soapenv:Body>
</soapenv:Envelope>
```

Jak by mohla vypadat odpověď webové služby na tuto SOAP zprávu vidíme níže.

```
<soapenv:Envelope
  xmlns:soapenv=
         "http://schemas.xmlsoap.org/soap/envelope/">
  soapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/>
    <soapenv:Body>
        <m:GetLastTradePriceResponse xmlns:m="Some-URI">
            <m:price>34.5</m:price>
        </m:GetLastTradePriceResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Element price obsahuje hodnotu akcií s kódem DIS. Jméno elementu, GetLastTradePriceResponse, není důležité a specifikace SOAP jej nedefinuje. Je však konvencí uvádět jej jako jméno metody s připojeným řetězcem „Response“ na konci.

SOAP hlavičky

V předchozím příkladu SOAP zpráv nebyla pro jednoduchost zanesena žádná SOAP hlavička. Pokud ovšem přítomna je, musí se objevit jako první element elementu Envelope. Hlavička může obsahovat několik tzv. bloků hlaviček. Každý blok hlavičky musí patřit do nějakého jmenného prostoru (namespace). Každý blok může obsahovat atribut mustUnderstand. Pokud příjemce zprávy nerozumí některému bloku, kde atribut mustUnderstand má hodnotu true, musí celou zprávu odmítnout.

V následujícím příkladě hlavička s atributem mustUnderstand nastaveným na hodnotu true říká, že po-

kud příjemce zprávy nerozumí transakcím, musí zprávu odmítnout.

```
<env:Header>
<t:transaction
    xmlns:t=http://thirdparty.example.org/transaction
    env:encodingStyle=http://example.com/encoding
    env:mustUnderstand="true" >5</t:transaction>
</env:Header>
```

Pokud by byla hodnota atributu mustUnderstand false, znamenalo by to, že příjemce nemusí tomuto bloku hlavičky rozumět, přesto by měl být schopen zprávu úspěšně zpracovat.

Transportní mechanismy

Přestože SOAP nijak nedefinuje, který protokol má být použit pro přenos SOAP zpráv, nejčastěji se používá protokol HTTP. Důvodem je jeho rozšířenost. Také většina firewallů umožňuje většinou neomezenou komunikaci na portu rezervovaném pro HTTP protokol, takže lze tento protokol použít jako transportní protokol pro SOAP komunikaci. Právě nutnost povolovat další porty pro komunikaci byla hlavní nevýhodou technologií DCOM a CORBA. Při použití protokolu HTTP jako transportního protokolu pro přenos SOAP zpráv jsou SOAP zprávy obsaženy v těle HTTP požadavku či odpovědi. HTTP požadavek pak musí obsahovat hlavičku SOAPAction, která danou SOAP zprávu identifikuje. Protokol HTTP je také jediný transportní protokol, který je zanesen přímo ve specifikaci protokolu SOAP. Následuje příklad SOAP zprávy z příkladu z kapitoly „Struktura SOAP zprávy“, přenášené protokolem HTTP. HTTP požadavek vypadá takto:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 555
SOAPAction: "urn:StockQuote#GetLastTradePrice"

<soapenv:Envelope
  xmlns:soapenv=
        "http://schemas.xmlsoap.org/soap/envelope/">
  soapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/>
    <soapenv:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <m:tickerSymbol>DIS</m:tickerSymbol>
        </m:GetLastTradePrice>
    </soapenv:Body>
</soapenv:Envelope>
```

Obsah odpovědi musí být stejně jako obsah požadavku identifikován jako XML dokument pomocí příslušného MIME typu *text/xml* v hlavičce *Content-Type*. Odpověď na tento požadavek vypadá takto:

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset="utf-8"
Content-Length: 450

<soapenv:Envelope
  xmlns:soapenv=
        "http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/>
    <soapenv:Body>
        <m:GetLastTradePriceResponse xmlns:m="Some-URI">
            <m:price>34.5</m:price>
        </m:GetLastTradePriceResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Při použití HTTP jako přenosového protokolu SOAP zpráv jsou používány standardní návratové kódy protokolu HTTP. Pokud se při zpracování SOAP zprávy nevyskytla chyba, vrátí protokol HTTP návratový kód 200, jak vidíme na příkladu HTTP odpovědi výše. Pokud se naopak vyskytla při vyřizování SOAP zprávy jakákoli chyba, vrátí webová služba chybový kód 500, Internal server error. V případě chyby je v těle SOAP zprávy obsažen element *Fault*, který popisuje danou chybu přesně podle SOAP specifikace.

WSDL

Pokud chceme používat nějakou webovou službu, musíme o ní znát některé základní informace. Především potřebujeme vědět, kde se webová služba nachází, jakými protokoly s ní můžeme komunikovat a jaké operace webová služba implementuje. K tomuto účelu slouží protokol WSDL, který popisuje rozhraní webové služby. Známe-li popis webové služby, můžeme vytvořit smysluplnou SOAP zprávu. SOAP zprávu můžeme vytvořit ručně nebo automatizovanými nástroji, které z popisu webové služby SOAP zprávu vytvoří.

Protokol WSDL vznikl z potřeby strukturovaně popsat rozhraní webových služeb. WSDL je založen na XML, je tedy platformně nezávislý. O jeho vznik se zasloužili především firmy Microsoft a IBM. Protokol WSDL vznikl ze tří jazyků: NASSL, SCL a SDL. V současnosti se používá verze WSDL 1.1. Konsorcium W3C vytvořilo pracovní skupinu Web Services Description Working Group, která nyní pracuje na vývoji verze WSDL 1.2.

Vztah mezi SOAP službou a WSDL je asi jako mezi zkompilovanou C knihovnou a header souborem se seznamem funkcí v ní obsažených.

WSDL dokument definuje webovou službu (service) jako množinu síťových uzlů, neboli bran (port). Brána je definována znovupoužitelnou vazbou (binding) a síťovou adresou. Znovupoužitelná vazba říká, jakým způsobem se s danou webovou službou spojím. Síťová adresa určuje, kde se webová služba nachází. V každé vazbě jsou definovány operace (operation), které daná webová služba implementuje. Tyto operace jsou však nejprve popsány abstraktně a až potom jsou svázány s konkrétní vazbou.

WSDL tedy nepopisuje sémantiku operací, pouze syntaxi jejich volání. Sémantiku je možné nanejvýš popsat lidským jazykem v tagu <documentation>, ale to není strojově zpracovatelný popis.

WSDL

WSDL je jazyk založený na XML, obsahující tyto hlavní elementy:

- <types> definice datových typů
- <message> komunikační zpráva - odpovídá zavolání nebo návratu z funkce
- <portType> souhrn operací - odpovídá rozhraní (Java) nebo hlavičkovému souboru (C)
- <operation> odpovídá metodě (Java) nebo funkci (C)
- <binding> definuje možný způsob přístupu různými protokoly
- <service> a <port> pro každý <binding> definují adresu na kterou se má příslušný protokol spojit

Apache Axis

Apache Axis je implementaci SOAP protokolu. Obsahuje celý http://ws.apache.org/axis [???]

Kromě SOAP enginu obhajuje Apache Axis také:

- jednoduchý stand-alone server
- plugin server do Tomcatu,
- podporu Web Service Description Language (WSDL),
- nástroj pro generování Java tříd z WSDL.
- příklady
- nástroj pro monitorování TCP/IP paketů.

WS Example

1. vytvoříme si jednoduchou metodu, kterou budeme chtít zpřístupnit jako webovou službu

```
package mypackage;

public class Calculator {
  public int add(int i1, int i2) {
    return i1 + i2;
  }
}
```

2. zkompilujeme
3. pomocí utility Java2WSDL vytvoříme WSDL soubor

```
java org.apache.axis.wsdl.Java2WSDL -o Calculator.wsdl
  -l"http://localhost:8080/axis/services/Calculator"
  -n"urn:Mypackage" -p"mypackage" "urn:Mypackage"
  mypackage.Calculator
```

4. pomocí utility WSDL2Java vytvoříme potřebná napojení

```
java org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -S true -Nurn:Mypackage
  mypackage Calculator.wsdl
```

Axis nám tímto vygeneroval všechny potřebné soubory:

- CalculatorSoapBindingImpl.java : Java soubor obsahující defaultní serverovou implementaci webové služby Calculator. Je potřeba implementaci změnit podle vlastních potřeb.
- Calculator.java: Nové rozhraní obsahující odpovídající java.rmi.Remote použití.
- CalculatorService.java: Java soubor obsahující klient service rozhraní.
- CalculatorServiceLocator.java: Java soubor obsahující klient service implementaci.
- CalculatorSoapBindingSkeleton.java: Server skeleton.
- CalculatorSoapBindingStub.java: Klient stub.
- deploy.wsdd: Deployment descriptor.
- undeploy.wsdd: Undeployment descriptor.

Pomocí Admin klienta deploydneme serverovou část:

Zkompilujeme a nahrajeme serverovou část do tomcatu do adresáře: axis/WEB-INF/classes.

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

A nyní už můžeme naši webovou službu vyzkoušet. Nejprve se přesvědčíme zdali opravdu na příslušné adrese služba běží.

```
http://localhost:8080/axis/services/Calculator?wsdl
```

Dále potřebujeme vytvořit nějaký příklad volání:

```
package mypackage;

public class Main {

    public static void main (String[] args) throws Exception {
      int first = Integer.parseInt(args[0]);
      int second = Integer.parseInt(args[1]);
      Calculator binding = new CalculatorServiceLocator().getCalculator();
      int result = ((CalculatorSoapBindingStub)binding).add(first, second);
      System.out.println(first + " + " + second + " = " + result);
   }
}
```

Poté už stačí jen spustit:

```
java mypackage.Main 1 3
```

A dostaneme jako výsledek webové služby:

1 + 3 = 4

WS a NetBeans

Vývojové prostředí NetBeans nám umožňuje poměrně komfortní práci s webovými službami. A to jak generování WSDL ze zdrojového kódu, tak i generování Java skeletu z WSDL. Pro vyzkoušení WS potřebujeme také J2EE aplikační server. Nejjednodušší je proto si stáhnou z http://www.netbeans.org/ [???]verzi s přibaleným Sun One Application serverem 8.1.

Webové služby typu REST

Motivace

"Klasické" webové služby jsou v poslední době kritizovány z několika důvodů:

- režie služeb je velká
- flexibilita často zbytečně velká, protokoly mají více vrstev, než je prakticky nutné (SOAP)
- psaní služeb zahrnuje vytváření řady složitých (XML) popisovačů - nelze psát bez spec. nástrojů

- nasazení služeb má poměrně velkou vstupní bariéru: vyžaduje aplikační server a speciální klientský SW
- interoperabilita mezi platformami (a jazyky) není vzhledem ke složitosti standardů stoprocentní

Webové služby typu REST (Representational State Transfer) jsou "lehkou" (lightweight) alternativou k SOAP službám.

Co je REST

REST je tzv. *architekturní styl* (architectural style), tj. soustava architekturních omezení (architectural constraints), definujích, jak budovat webové služby (aplikace). Není to tedy standard, technologie, nástroj, ani API...

REST vychází z úspěšných postupů používaných na webu od samého počátku.

Autorem REST je Roy Fielding, jenž principy poprvé představil ve své disertaci [http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm] *Architectural Styles and the Design of Network-based Software Architectures*. Jeho definice říká: „Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.“

Podrobnější objasnění principů najdeme např. na serveru O'Reilly webservices.xml.com [http://webservices.xml.com].

Principy

REST je metodika (architekturní styl), jak psát webové služby, či spíše, jak webové služby vystupují vůči klientovi.

REST používá některé pojmy:

zdroj (resource)	typicky odpovídá nějaké (datové) entitě, předmětu zájmu; něčemu, co lze číst, ale obvykle i vytvářet a měnit.
identifikace zdroje	zdroj je identifikován svým URI (např. URL)
reprezentace zdroje	to, co je na požadavek posláno klientovi, tj. např. XML soubor, HTML stránka, obrázek...

Co REST aplikace používají

REST sám o sobě není standardem, REST aplikace jsou ale na standardech přísně založeny - a to na běžných a zavedených jednoduchých standardech:

- HTTP (přístup ke zdrojům)
- URL (identifikace zdrojů)
- XML/HTML/GIF/JPEG/atd. (reprezentace zdrojů)
- text/xml, text/html, image/gif, image/jpeg, etc (MIME typy zdrojů)

Charakteristika REST služeb

Client-Server	klient "si řekne" o zdroj (princip "pull")
Stateless	(bezestavový) každý klientský dotaz obsahuje veškeré informace potřebné pro vyřízení dotazu na serveru (server neuchovává stav)
Cache	zdroje musejí být označeny, zda podporují kešování
Uniform interface	rozhraní přístupu ke zdrojům (čtení, vytvoření, smazání, modifikace) je jednotné (HTTP GET, PUT, DELETE, POST)
Named resources	zdroje jsou identifikovány pomocí URI (typicky URL)
Interconnected resource representations	reprezentace zdrojů jsou propojeny pomocí URI (URL) a klient tedy může přecházet mezi stavy
Layered components	mezi klienta a server se službou lze umisťovat proxy, keše, brány...

Co rozmyslet před návrhem REST služby

1. Co jsou URI (tj. zdroje)?
2. Jaký formát?
3. Jaké metody každé URI podporuje?
4. Jaké návratové kódy jsou vraceny?

Postup návrhu

1. Identifikovat zdroje.
2. Vytvořit pro ně systém pojmenování (URI).
3. Určit, jaké metody bude každý zdroj (URI) podporovat (GET, POST, PUT, DELETE?).
4. Čtení zdroje (GET) nesmí mít vedlejší efekt (nesmí nic modifikovat).

5. Reprezentace (např. vrácená stránka) by měla umožnit klientovi pokračovat dále, mít odkazy na další zdroje.

6. Zdroje nemusí (neměly by být) obrovské - lze vždy zpřístupnit jen část a přes odkazy nechat klienta dotaz upřesnit.

7. Specifikovat schéma pro reprezentaci zdroje (DTD, XML Schema, RelaxNG, Schematron). Rovněž pro vytvoření a modifikace zdroje je dobré schéma zveřejnit.

8. Popsat službu pomocí WSDL nebo aspoň HTML.