



Unit 1

Service Standards and Service Development

Lecture 1-4

REST Concept and Architecture

Dr. Yinong Chen
<https://myasucourses.asu.edu/>



Lecture Outline

- HTTP
- REST Concept
- RESTful Services
- From SOAP service to RESTful Service
- When to use SOAP services and when to use RESTful services?
- Processing URI string in RESTful service
- Mapping URI string to operations
- Output formats
- Microservice architecture

HTTP (Version 1.1)

<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

- HTTP is an application-level protocol for distributed, collaborative, and hypermedia information systems.
- HTTP messages are always **two ways**: requests from client to server and responses from server to client.

HTTP-message = Request | Response

- Request: Request-Line =

Method SP Request-URI SP HTTP-Version CRLF

- Response: Status-Line =

HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Syntax

where, *SP*: Space and *CRLF*: end of line mark

HTTP Methods

The **Method** indicates the **operation** to be performed on the resource identified by the **Request-URI**. The method is case-sensitive.

- **GET**

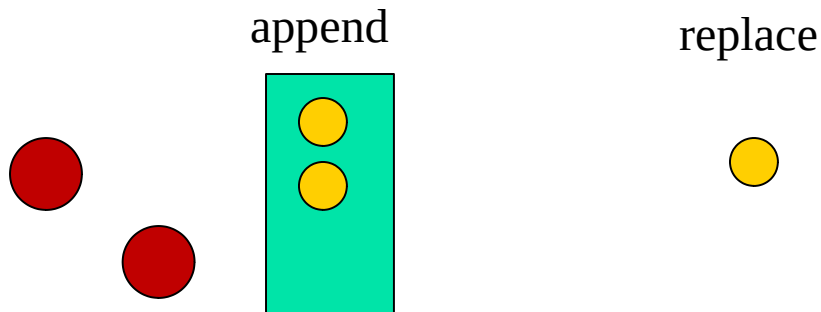
- retrieves the information (entity) identified by Request-URI.
- If the Request-URI refers to a **data-producing process** (operation), the produced data shall be returned in the response.

- **HEAD**

- The HEAD method is identical to GET, except that the server **MUST NOT** return a message-body in the response.
- Used to obtain meta-information about the entity implied by the request without transferring the entity-body itself.
- Use for making one-way call – no message body to return.

HTTP Methods (contd.)

- **PUT**: For creating and **modifying/replacing** resource. It requests the enclosed entity to be stored under the supplied Request-URI.
- **POST**: used to request the server to accept the (data) entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
- POST and PUT have some similarity, but are used in different situations:
 - POST: append;
 - PUT: replace or create



HTTP Methods (contd.)

- **DELETE**: requests that the origin server deletes the resource identified by the Request-URI.
 - The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action is **successful**. Why?
 - However, the server **SHOULD NOT** indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.
- **TRACE**: invoke a remote, application-layer loop-back of the request message.
- **CONNECT**: for use with a proxy that can dynamically switch to a tunnel (e.g. SSL tunneling)

Not to hold the client for too long



REST and RESTful Service

- **REST** (Representational State Transfer) is a style of **software architecture** for distributed hypermedia systems such as the World Wide Web.
- Proposed by Roy Thomas Fielding in his doctoral dissertation "Architectural Styles and the Design of Network-based Software Architectures" in 2000.
- Fielding is one of the principal authors of the HTTP specification versions 1.0 (1996) and 1.1 (1999).
- Services developed based on REST **concept** and conforming to the REST **constraints** are referred to as **RESTful services**.
http://en.wikipedia.org/wiki/Representational_State_Transfer

REST Concept

REST concept is about how communication is done between clients and servers using HTTP at the application level. REST concept is defined by these principles:

- A client initiates a request to a server;
- The server processes the request and returns a response;
- Communication is stateless;
- Communication is about the "representations" transfer of "resources" between a client and a server;
- Each resource is given a unique identifier, called URI (Universal Resource Identifier).

REST Architecture and REST Constraints

REST architecture is a software development style that follows these design **constraints and guidelines**:

- Client-server separation: Clients and servers are logically separated and communicate through a uniform interface conforming to HTTP.
- Stateless: Every request from a client is considered from a new client by the server.
- Client-side **cacheable**: The response delivered from the sever must contain the complete information for the client to reuse the response, if the client requests the same URI. – **Output Caching** on client side (Textbook 5.5).

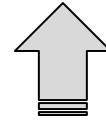
REST Architecture and REST Constraints (contd.)

- Layered architecture: A server is a logical unit with a layered architecture.
 - A client identifies a server by an URI.
 - The resource may or may not reside on the server that the client is communicating with.
 - The request can be redirected to another server transparently.
 - Multiple redirections are possible.
- Code on demand: A server may process the request by executing a program on the server side or sending the code (script) to the client side to process.

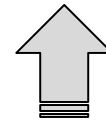
What is New?

- The concept, the principles, and the constraints all sound familiar.
- We have been applying them to design Web 1.0 (Static Web presenting data);
- What is about Web 2.0 (Web as the computing platform)? Move from data Web to computing Web.
- **SOAP/WSDL** Web services: create computing-based Web.
- RESTful services: create data-oriented computing Web
- Spiral: Data Web (no computing) \square Computing Web \square Data Web (with invisible computing behind)

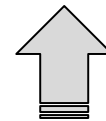
Spiral Advancement Models of Technology



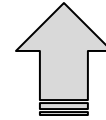
Data Web with
invisible
computing



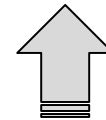
Computing
Web



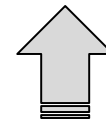
Data Web
without
computing



Centralized
Computing
(Cloud Computing)



Distributed
Computing



Centralized
Computing
(mainframe)

SOAP-Service vs. RESTful Service

- Providing a service generally involves two issues:
 - Performing (verb) a certain task for a client, and
 - the result (noun) of performing the task;
- SOAP-Services:
 - Focus on the verb “performing” the task;
 - Access a remote method/procedure;
 - Based on class concept of object-oriented language, developed by OO people.
 - Heavy-weighted services, just like classes.
- RESTful Services:
 - Focus on the noun “result” of performing the task;
 - Access a “resource” instead of a method,
 - Based on communication protocol and developed by HTTP people;
 - Can be called even in script languages.
 - Light-weighted service

RESTful Services are More Web Oriented

- Based on HTTP and Web browser;
- Can be accessed in Web browser;
- Just like a Web page, they can be viewed and bookmarked in a browser;
- The results can be cached and reused, through browser's built-in caching mechanism;
- RESTful services can be called in any programming languages, including script languages, that can make HTTP call.

Developing a Basic RESTful Service in WCF

- Windows Communication Foundation has decoupled the interface generation from the development environment. It enables
 - WSDL/SOAP interface;
 - .Net - .Net Remoting interface
 - RESTful service interface (HTTP interface).
- Using WCF:
 - Develop a WCF service
 - Make a few changes to obtain RESTful service

From a SOAP Service to a RESTful Service

Textbook Chapter 7.3 on RESTful services

- Create a SOAP-based Service in WCF;
- Use a client to test the service to make sure it works;
- Convert the SOAP-based Service into a RESTful service in follow steps:
 1. Add “`using Sytem.ServiceModel.Web`” in file IService.cs;
 2. Add the attributes [`WebGet`] for each operation contract
 3. Add the following line in the service’s markup source code
`Factory="System.ServiceModel.Activation.WebServiceHostFactory"`
 4. Remove SOAP endpoint for access, so that HTTP can be directly used for accessing the service.

From a SOAP Service to a RESTful Service

//IService.cs

using System.ServiceModel;

using System.ServiceModel.Web; // Add this namespace

[ServiceContract]

public interface IService {

[OperationContract]

[WebGet] // Add this HTTP GET attribute/directive

string GetData(int value);

[OperationContract]

[WebGet] // Add this HTTP GET attribute/directive

double PiValue();

[OperationContract]

[WebGet] // Add this HTTP GET attribute/directive

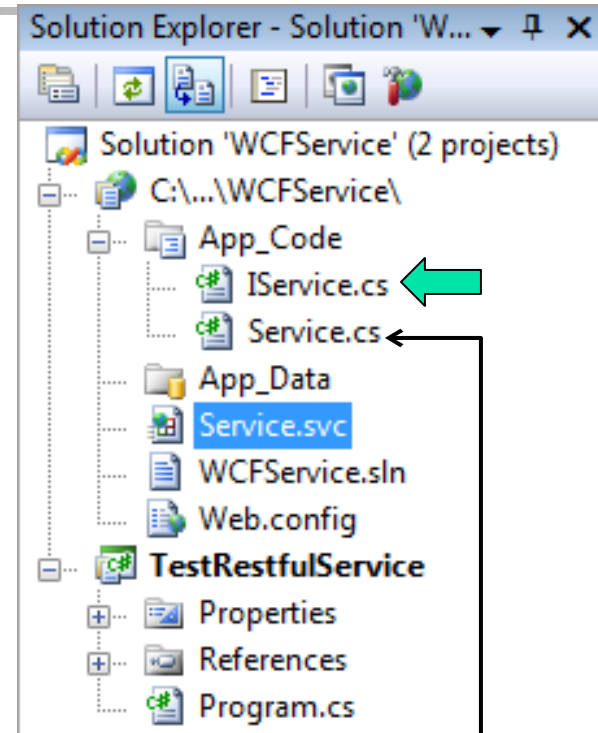
int absValue(int intValue);

}

1

ARIZONA STATE
UNIVERSITY

[1] if not available, you need to use Add Reference and find it in framework library. Y. Chen



Service.cs
file remains
unchanged

Use WebServiceHostFactory for Hosting

Right-click the file `Service.svc` and choose “View Markup”. The markup source code of the file is open, as shown below:

- Add one line of code in the source file, which results in the code below:

```
<%@ ServiceHost Language="C#" Debug="true"  
    Service="Service" CodeBehind="~/App_Code/Service.cs"  
    Factory="System.ServiceModel.Activation.WebServiceHostFactory" %>
```

3

WebServiceHostFactory will automatically search all the methods in the service to match the requirement of the incoming URI

Remove SOAP Endpoint

4

- Open Web.config file, remove the entire `<system.serviceModel>` element, which will remove the SOAP endpoints.
- RESTful services do not support the SOAP-endpoint-based access.
- Removing this element will immediately disable the accesses from service proxies from all kinds of SOAP clients.

Remove SOAP Endpoint in Configuration

```
<system.serviceModel>  
  <services>  
    <service name="Service" behaviorConfiguration="ServiceBehavior">  
      <!-- Service Endpoints -->  
      <endpoint address="" binding="wsHttpBinding" contract="IService">  
        <identity>  
          <dns value="localhost"/>  
        </identity>  
      </endpoint>  
      <endpoint address="mex" binding="mexHttpBinding"  
contract="IMetadataExchange"/>  
    </service>  
  </services>  
</system.serviceModel>
```

Consuming the RESTful Service

- Now, if we start the service in a browser after the modification above, the browser will open a page at the address:
`http://localhost:51191/myWcfRestService/Service.svc`
- It displays an error message of “**Endpoint not found**”.
- This is because that the endpoints have been removed in the Web.config file and the service is no longer a SOAP service.
- We need to use a different protocol (HTTP) to consume (test) a RESTful service.

Consuming the RESTful Service

- After starting the service, the simplest way of consuming a RESTful service is to type the URI in a browser, with the method name and parameter value as a part of the URI.
- `http://localhost:51952/WCFService/Service.svc/PiValue`
 - It returns: 3.1415926535897931
- `http://localhost:51952/WCFService/Service.svc/absValue?intValue=-25`
 - It returns: 25

Is the Service Developed a RESTful Service?

- The developed service has removed SOAP, and it uses HTTP method GET to access the resources (operations);
- It can be accessed using in a browser;
- However, it still focuses on operations (methods), instead of the results and data: We put the **service** name, **method** name and parameters in the URI:
http://localhost:51952/WCFService/**Service.svc**/**PiValue**
http://localhost:51952/WCFService/**Service.svc**/**absValue**?intValue=-25
- We can hide the method name to make it more like a RESTful service.

We will show more development details in the next lecture. Now, we continue to explain the

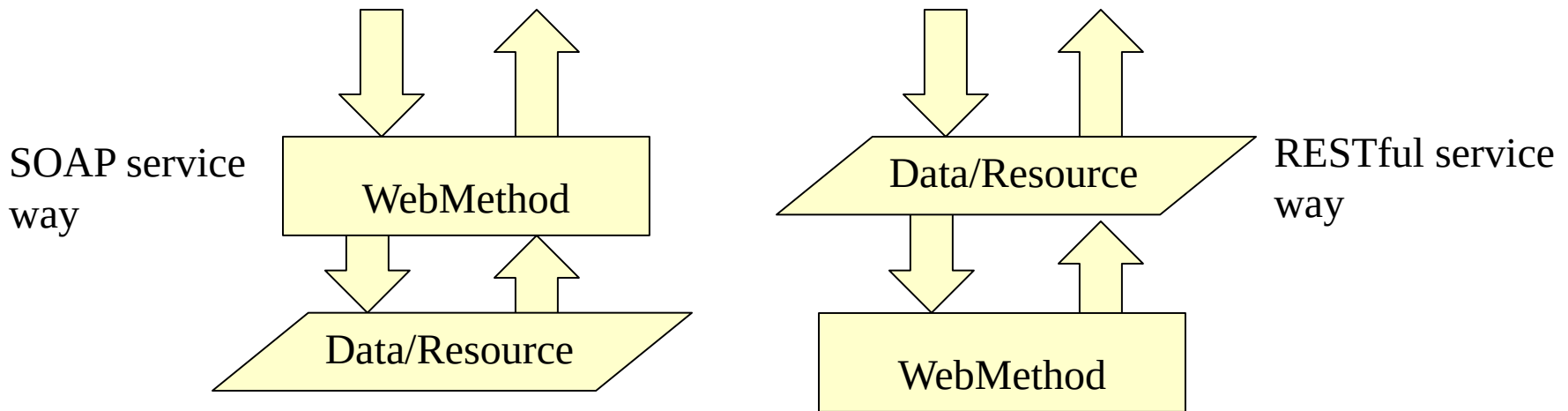
concept

The Key Ideas of RESTful Services

- Use HTTP directly without another layer of SOAP. As the focus is on exposing the resources, the HTTP verbs GET, PUT, POST, DELETE, and HEAD are sufficient for accessing the RESTful services.
- Focus on exposing data and resources instead of Web methods. Each resource is given a URI, and a set of resources can also be assigned to a URI.
- RESTful service development is about identifying resources to be exposed, and organizing them so that the resources can be exposed efficiently in different ways: individually or in sets.

Co-Exist of SOAP and RESTful Services

- We can use either (SOAP or RESTful) to do both (access methods and access data/resource);



- We should use SOAP services for computation-oriented tasks: Solving equations, find the shortest path, etc.
- We should use RESTful services for data and resource-oriented tasks: finding photos, finding books, etc.

Example: A Book Management System

We can save the information of all the books in a relational database with a schema,
(authors, title, isbn, year, publisher, keywords)

authors	title	isbn	year	publisher	keywords
Aaron	OS	123456789	2010	Kendall	Linux
John	Database	987654321	1999	Apress	relational
Chen	Programming	0757529747	2006	Kendall	Language
Chen	Protocol testing	3181460109	1993	VDI Verlag	Communication
...					

Example: a Book Management System

SOAP-Service Approach

For the given schema:

(authors, title, isbn, year, publisher, keywords)

you define a class with the following operations/methods:

- `search_by_author(string author);`
- `search_by_title(string title);`
- `search_by_isbn(string isbn);`
- `search_by_year(string year);`
- `search_by_publisher(string publisher);`
- `search_by_year_publisher(string year, string publisher);`

Define WSDL interface:

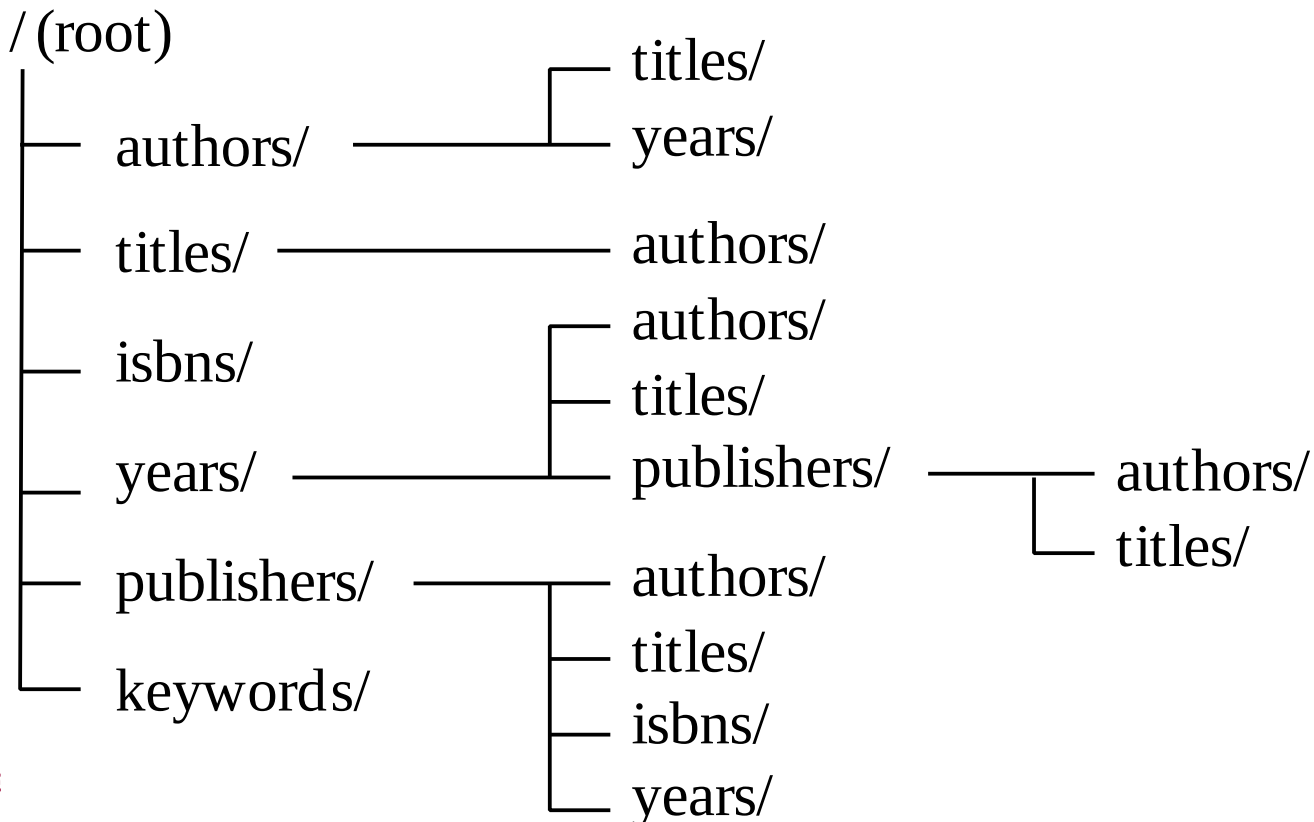
- Service
- Endpoints
- Operation
- Input message
- Output message

Two parameters

Example: Book Management System

RESTful Service Approach

Represent the resources in a resource tree, which is a rooted tree with the root representing all resources, and each tree node representing a subset of resources of its parent node:





Using URI to Access Resources

- Method-Oriented Resource Access in URI:

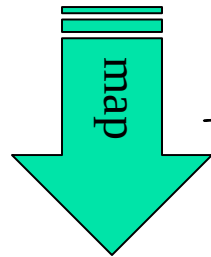
<http://localhost:51952/WCFService/Service.svc/absValue?intValue=-25>

http://mylib.asu.edu/bookService.svc/search_by_author?author=Aaron

- Result-Oriented Resource Access in URI:

<http://mylib.asu.edu/authors/{author=Aaron}>

How is this URI mapped to resource: all books written by Aaron?



How to map?

`search_by_author(string author);`

How to map URI to a Method behind?

1. **String-process** the incoming URI and obtains words in the string;
2. **Match** URI to the service
3. **Identify** the method that can process the request.
4. **Find** any variables and variable values in the URI and map them to the method parameters and parameter values.
5. **Determine** the HTTP method (GET, POST, PUT, etc.) used in the request and whether it's allowed for the resource.
6. **Read** the resource representation found in the entity body (if any).
7. **Combine** all of this information to perform the underlying service logic (or call the corresponding method).
8. **Generate** an appropriate HTTP response, including the proper status code, description, and outgoing resource representation in the response entity body (if any).

WCF Support to URI Process

<http://msdn.microsoft.com/en-us/library/bb675245.aspx>

- We could implement RESTful service concept without a programming environment support, e.g., string processing. The program will be extreme long & tedious.
- WCF provides mechanisms and classes to facilitate the implementation of RESTful services, for example
 - **Uri**: convert a string into a URI object
 - **UriTemplate**: Define the template of input URI
 - **UriTemplateMatch**: Match two URIs
- Example: to match the **built-in** and **incoming** URIs:
<http://mylib.asu.edu/years/{year=2010}/authors/{author=Aaron}>
<http://mylib.asu.edu/years/publishers/authors>,

URI Processing

UriTemplate Class

- The built-in URI template:

`http://mylib.asu.edu/years/{year=2010}/authors/{author=Aaron}`

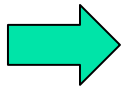
UriTemplateMatch

`http://mylib.asu.edu/years/publishers/authors`

(Client entered this URI in a browser)

Example: Analyze the Incoming URI String

```
using System;
using System.ServiceModel.Web;
class ConsoleAppUriMatch {
    static void Main(string[] args) {
        Uri baseUri = new Uri("http://mylib.asu.edu");
        UriTemplate myTemplate = new
UriTemplate("/{years}/{publishers}/{authors}");
        Console.WriteLine("Built-in URI path segments are");
        foreach (var segName in myTemplate.PathSegmentVariableNames) {
            Console.WriteLine(segName);
        }
        Console.WriteLine("Please enter the Incoming URI with path segments ");
        string myUri = Console.ReadLine();
        // enter this URI: http://mylib.asu.edu/years/publishers/authors
    }
}
```

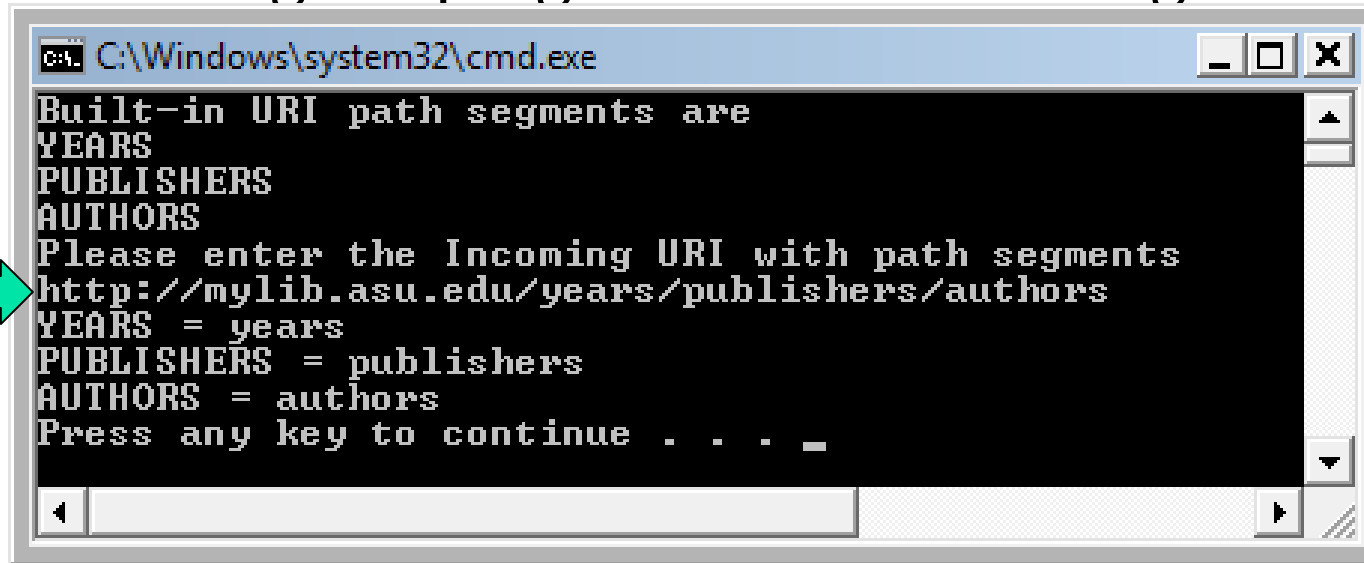


Example: Analyze the Incoming URI String

```
// CONTINUED FROM LAST PAGE
Uri incomeUri = new Uri(myUri); // convert string into URI format
UriTemplateMatch myMatch = myTemplate.Match(baseUri,
incomeUri);
if (myMatch != null) {
    var matched = myMatch.BoundVariables;
    string myVars;
    foreach (var k in matched.Keys) {
        myVars = k.ToString();
        Console.WriteLine("{0} = {1}", myVars, matched[myVars]);
    }
}
else
    Console.WriteLine("The incoming URI does not match the built-in
URI");
```

Output of the Program

- Executing the program with a matching incoming URI:

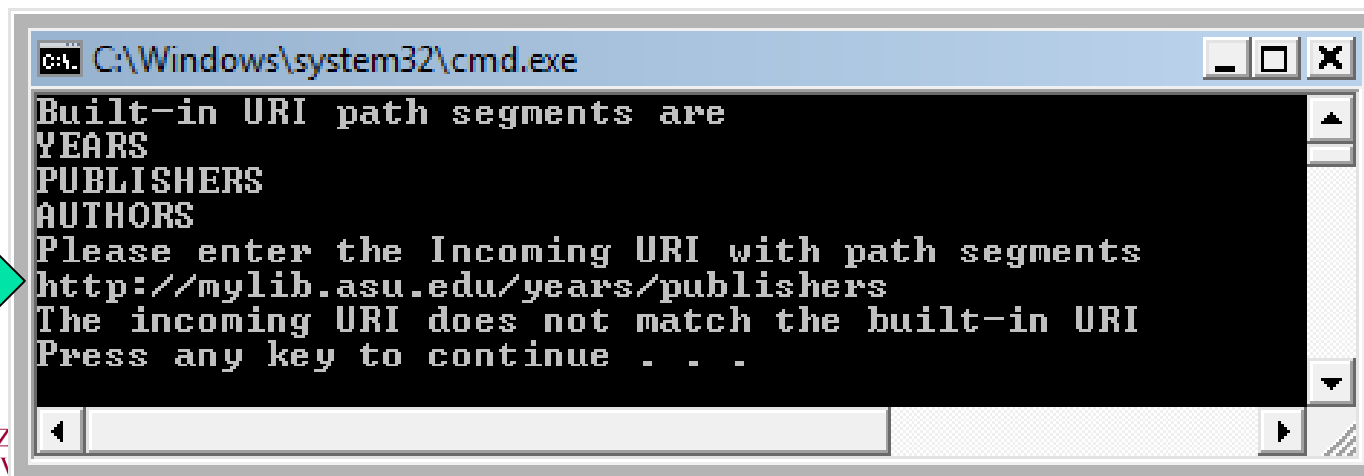


A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text displayed is as follows:

```
Built-in URI path segments are  
YEARS  
PUBLISHERS  
AUTHORS  
Please enter the Incoming URI with path segments  
http://mylib.asu.edu/years/publishers/authors  
YEARS = years  
PUBLISHERS = publishers  
AUTHORS = authors  
Press any key to continue . . . _
```

A green arrow points to the input line "http://mylib.asu.edu/years/publishers/authors".

- For other inputs



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text displayed is as follows:

```
Built-in URI path segments are  
YEARS  
PUBLISHERS  
AUTHORS  
Please enter the Incoming URI with path segments  
http://mylib.asu.edu/years/publishers  
The incoming URI does not match the built-in URI  
Press any key to continue . . .
```

A green arrow points to the input line "http://mylib.asu.edu/years/publishers".

Composing URI Using WCF Built-in Classes

// create the base address

```
Uri baseUri = new Uri("http://mylib.asu.edu");
```

// create the path from tree root to the child node

```
UriTemplate myTemplate = new
```

```
    UriTemplate("years/{year}/publishers/{publisher}");
```

// Assign values to variable to complete URI

```
Uri newBooksUri =
```

```
    myTemplate.BindByPosition(baseUri, "2010", "Kendallhunt");
```

// create the path from tree root to the child node

```
myTemplate = new UriTemplate("authors/{author}");
```

// Assign values to variable to complete URI

```
newBooksUri = myTemplate.BindByPosition(baseUri, "Aaron");
```

Define the Map from URI to Methods

[ServiceContract]

public partial class BookService {

[WebGet(UriTemplate = "{author}?tag={tag}")] //search by author

[OperationContract]

Books search_by_author(string username, string tag) {...}

[WebGet(UriTemplate = "{title}?tag={tag}")] //search by title

[OperationContract]

Books search_by_title(string title, string tag) {...}

//search by year and by publisher

[WebGet(UriTemplate = "years/{year}/publishers/{publisher}")]

[OperationContract]

Books search_by_year_publisher(string year, string publisher) {...}

...

Resource Representations

- In many cases, a URI request will return a set of items.
- For example, the URI `http://mylib.asu.edu/years/{year=1999}` will return all books in 1999 in the library.
- How do we present the data to the user who made the request?
 - XML: Requires XML schema
 - POX (Plain Old XML): A simple XML with given schema
 - RSS, Atom Feed, and AtomPub
 - JSON (JavaScript Object Notation)
- To specify the output form, add format requirement, for example:
`http://mylib.asu.edu/years/{year=2010}&{format=rss}`

Microservices vs Web Services

40

- **Microservices** and **microservice application architecture** are a software development technique, a variant of SOA style.
- A Web application is a collection of loosely coupled (smaller) **service instances**, created from **microservice templates**, which are fine-grained and the protocols are lightweight.
- **Service mesh** is a configurable infrastructure layer for microservices. It makes communication between **service instances** flexible, reliable, and fast.
- Smaller SOAP and RESTful services can be microservices.
- Since RESTful services are typically lighter weighted services and are more often used as microservices.
- Small APIs and library functions that do not belong to SOAP and RESTful (HTTP) services can be considered microservices.

From Microservices to Service

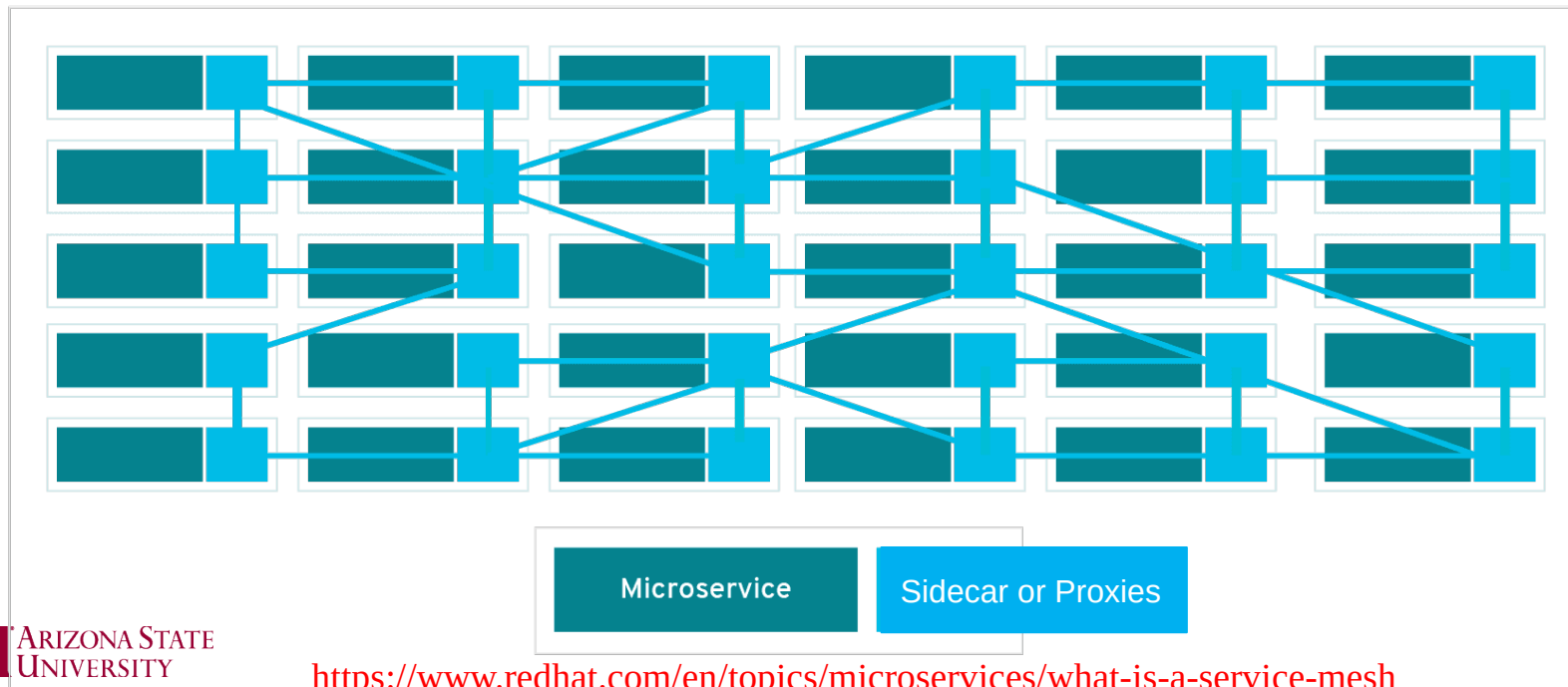
- **Microservices** platforms define and organize the microservices in such a way that they become standard components for composition.
- Computer organization have a few micro instructions:
IF, ID, EX, MEM, WB

Analogy:
microinstruction
vs. instruction



A program consists of microinstructions

- Service Mesh: You select micro services combination to define your application



Microservices Platforms

- **Istio** is an open source service mesh platform that provides a way to control how microservices share data with one another.

<https://www.redhat.com/en/topics/microservices/what-is-istio>. It includes

- APIs that let Istio integrate into any logging platform.
 - Its architecture is divided into the data plane and the control plane to implement the sidecar (proxy) and service container concepts.
 - Changing an application is about changing the mesh network
- **Spring Framework / Spring Cloud**: A Java-based microservice architecture that support Servlet API, WebSocket API, JSON Binding API, RESTful service, etc., integration; (<https://spring.io/projects/spring-cloud>)
 - It provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state).
 - Coordination of distributed systems leads to **boilerplate** patterns (frequently used code), and can quickly stand up services and applications that implement those patterns.

Lecture Summary

- HTTP and REST Concept
- RESTful Services
- From SOAP service to RESTful Service
 - With full implementation detail
- When to use SOAP services and when to use RESTful services
 - Focus on computing/method
 - Focus on results
- Processing URI string in RESTful service
- Mapping URI string to operations
- RESTful service's Output formats
- Microservices Architecture and Service Mesh

- **Developing REST Services:**
 - Developing RESTful Service
 - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
 - RESTful Service of a Random String Generator
 - RESTful Service of an Image Verifier
 - Synchronous RESTful Service Calls
- **Consuming Services**
 - Asynchronous SOAP Calls
 - Asynchronous RESTful Calls