RESTFUL SERVICES

CSE416, Section 3

Will be tested in the client/server review

1

Lecture Objectives

HTTP and the Servlet API are the standards used to implement many client/server frameworks

- Understand the fundamental concepts of Web services
- Understand concepts of HTTP
- Become familiar with JAX-RS annotations
- Be able to build a simple Web service
- Understand how to pass parameters and return values with a Web services

Parts of Spring are very similar to JAX-RS

2

Reading & References

- Reading
  - Tutorials

    docs.oracle.com/javaee/7/tutorial/webservices-intro.htm#GIJTI

    (Chapters 27 and 29.1-29.3)
- Reference
  - Java EE API

    docs.oracle.com/javaee/7/api/javax/ws/rs/package-summary.html

Be careful – other JAX-RS documentation assumes knowledge of other Java EE technologies (e.g., JPA)

Session material follows Java EE 7 Tutorial text

3

Client/Server Strategies

- Generation of HTML/CSS
  - Server responds with a dynamically generated page that includes HTML, CSS, and data (inserted in the page)
  - Data insertion usually performed by a server-side scripting engine
- Web services
  - Server responds with data (no HTML and CSS)
  - Data structured based on some coordination between client and server (e.g., JSON, XML, text)

Most/all server components in your CSE416 project will likely respond to web services

4

Servlets

- Conforms to the Java Servlet API
- Normally used to implement JAX-RS (Java API for RESTful Web Services) API and other client/server frameworks
- A servlet:
  - Is a Java class that can be loaded dynamically to expand the capability of the Web server
  - Runs inside the Java Virtual Machine on the server (safe and portable)
  - Is able to access all Java APIs supported in the server
  - Does not have a main method

5

Servlet API

Acts as the controller in an MVC design

- Part of Java EE
- Low level approach to implement server handling of HTTP requests
- Servlet method signature contains a
  - request object that contains parameters, http headers, etc.
  - response object that contains typically empty objects to be populated and returned by the server
- URL requests are mapped to the servlet through annotation or an xml document
- Typically, one servlet per type of request => complicated control logic

6

Client – Servlet Model with Servlet API

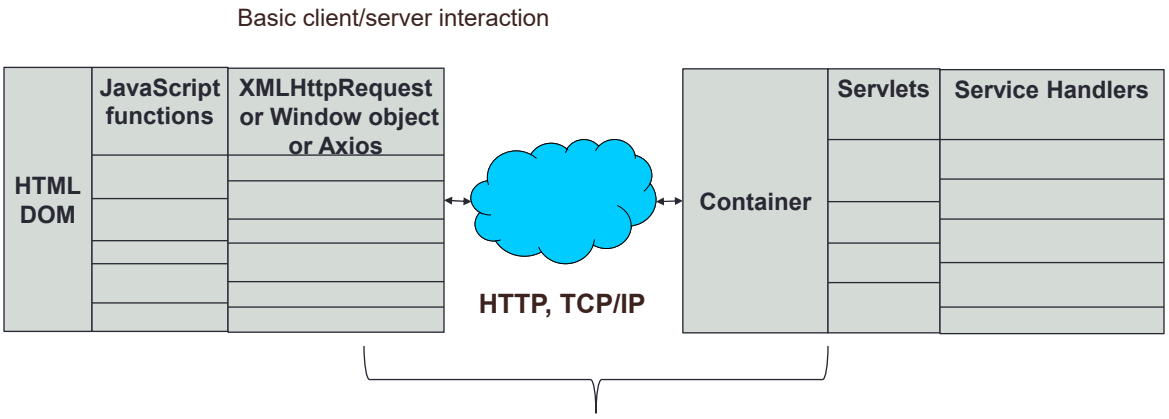
- Requires multiple servlets to handle requests
- Requires logic in servlet to route each request to a service method
- Mapping of the URL to a servlet is handled with web.xml or Java Annotation in servlet class

```
<form method="get" action=
"http://localhost:8080/CSE416/helloyou.html">
```

Servlet identified by the “helloyou.html” URL string usually acts as a controller, and routes to a service handler

7

Servlet API Client/Server Interaction



Parts of the client/server interaction are abstracted by tools/libraries

8

RESTful Web Services

- Representational State Transfer
- Architectural style for distributed systems
- Architecturally consistent with HTTP
- Provides a standard means of interoperating between software applications running on a variety of platforms and frameworks
- Use existing W3C and IETF standards (HTTP, XML, URI, MIME)

A service is a software component provided through a network-accessible endpoint

9

Types of Web Services

- JAX-WS
  - Communication using XML
  - Provides for message-oriented and RPC services
  - Uses SOAP messages
  - includes standards for security and reliability
- JAX-RS
  - Standard
  - Semantics of the data to be exchanged is understood by client and server

We only cover JAX-RS

10

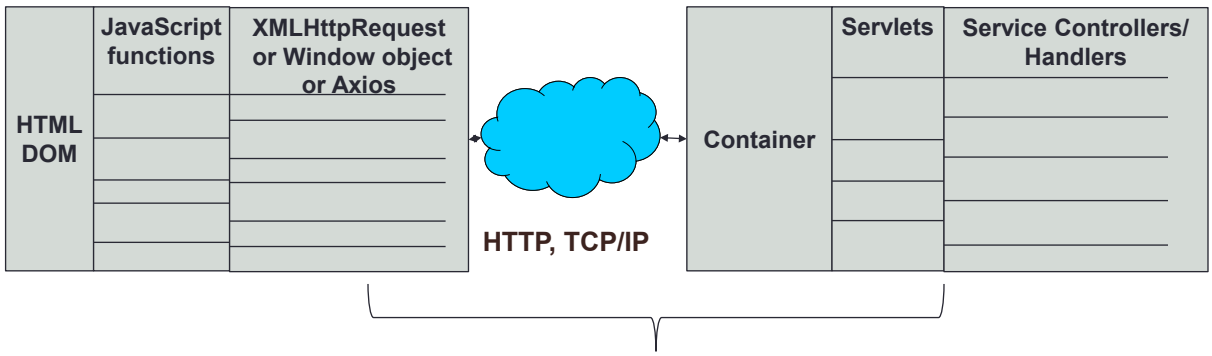
JAX-RS

- Java API for RESTful Web Services
- A standard – not a product
- Reference implementations
  - Jersey, RESTeasy, et al, along with some application servers
  - No requirement to implement on top of servlets, but many implementations do

11

Client/Server Interaction

Think of JAX-RS as extending the abstraction to the Service handlers



Parts of the client/server interaction are abstracted by tools/libraries

12

HTTP

- HyperText Transfer Protocol defines communications between a browser and a server
- Defined in specs (HTTP 1.0, HTTP 1.1, HTTP/2, HTTP/3)
- Defines:
  - Types of messages exchanged (request and response)
  - Syntax of the messages
  - Semantics of the message content
  - Rules for determining how and when a process sends and responds to a message

13

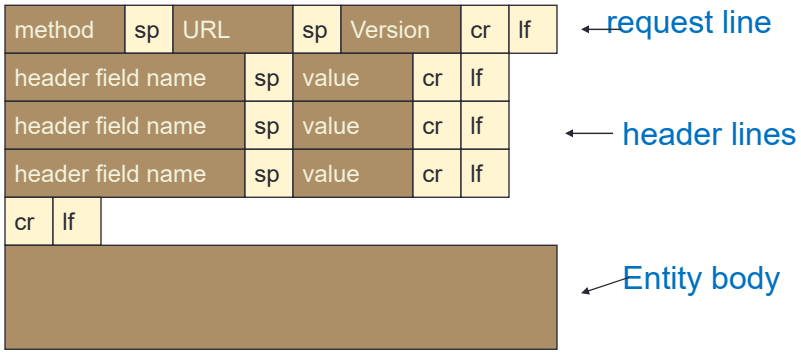
HTTP Protocol

- HTTP is a request/response (stateless) protocol
  - A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content
  - The server responds with a status line, including the message's protocol version and a success (or error) code, followed by a MIME-like message containing server information, entity meta-information, and possible entity-body content.

14

Request Message Format

- The http request is specified by the request line, a variable number of header fields, and the entity body



15

HTTP Methods

- OPTIONS – request for information concerning communications options (e.g., support of http 1.1)
- GET – retrieve information
- HEAD – identical to GET, except the server does not return a message body
- POST – modify a server resource
- PUT – store the enclosed entity
- DELETE – request that the resource be deleted
- TRACE – response contains the entire message request in the response body
- CONNECT – used in SSL tunneling

16

HTTP Request/Response

HTTP-TRACKER Chrome Plugin

Request Details	
Headers	
documentLifecycle	active
frameType	outermost_frame
method	GET
url	https://www3.cs.stonybrook.edu/~cse416/Section01/
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.9
sec-ch-ua	"Google Chrome";v="105", "Not)A;Brand";v="8", "Chromium";v="105"
sec-ch-ua-mobile	?0
sec-ch-ua-platform	"Windows"
Sec-Fetch-Dest	document
Sec-Fetch-Mode	navigate
Sec-Fetch-Site	none
Sec-Fetch-User	?1
Upgrade-Insecure-Requests	1
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x6

Response Details	
Headers	
documentLifecycle	active
frameType	outermost_frame
fromCache	false
ip	130.245.27.3
method	GET
statusCode	200
statusLine	HTTP/1.1 200 OK
url	https://www3.cs.stonybrook.edu/~cse416/Section01/
Accept-Ranges	bytes
Connection	Keep-Alive
Content-Encoding	gzip
Content-Length	7608
Content-Type	text/html
Date	Tue, 13 Sep 2022 17:56:28 GMT
ETag	"5923-5e830a268ed4f-gzip"
Keep-Alive	timeout=5, max=100
Last-Modified	Thu, 08 Sep 2022 20:54:24 GMT
Server	Apache/2.4.18 (Ubuntu)
Vary	Accept-Encoding

17

HTTP Request Headers

- Accept
- Accept-charset
- Accept-encoding
- Accept-language
- Authorization
- Cache-control
- Connection
- Content-length
- Content-type
- Cookie
- Expect
- From
- Host
- If-match
- If-modified-since
- If-none-match
- If-range
- If-unmodified-since
- Pragma
- Proxy-authorization
- Range
- Referer
- Upgrade
- User-agent
- Via

18

HTTP Response Message

- HTTP response messages consist of:
  - Status line (protocol version, status code, status message)
  - Header lines (date, server, last-modified, content-length, content-type)
  - Entity body

19

HTTP Status Codes

- Examples:
  - 200 – OK
  - 100 – Continue
  - 404 – Not found

There are other codes in http that might allow you to distinguish categories of errors

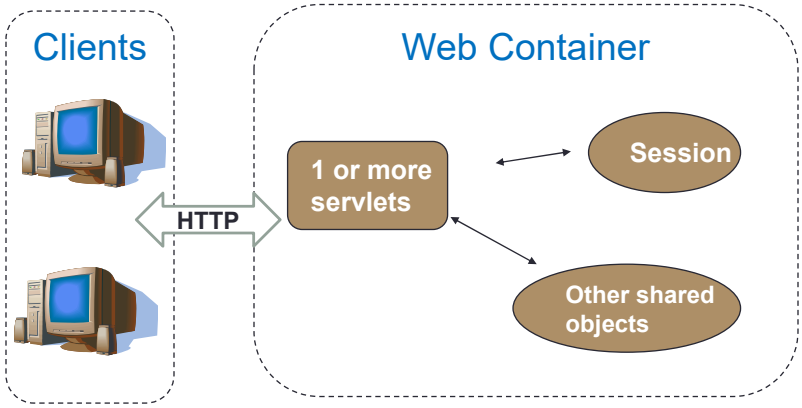
20

HTTP Response Headers

- Accept-Ranges
- Age
- Allow
- Cache-Control
- Connection
- Content-Encoding
- Content-Language
- Content-Length
- Content-MD5
- Content-Type
- Date
- Etag
- Expires
- Last-Modified
- Location
- Refresh
- Server
- Set-Cookie
- Via
- Warning

21

Typical Client/Server Interaction



Multiple client requests can be simultaneously executing the same servlet

22

Principles of REST Architectural Style

- Resource identification through URI
- Uniform interface – CRUD access defined in HTTP methods (PUT, GET, POST, and DELETE)
- Self-descriptive messages – content can be accessed in a variety of formats (e.g., HTML, XML, plain text, PDF, JPEG, JSON, etc.)
- Metadata about the resource is available
- Stateful interactions through links – Interactions are stateless (request messages contain state info)

CRUD=Create, Read, Update, and Delete

23

Implications of REST Style

- Interactions are predominantly computer-computer, not human-computer
- Resource based URI

URI requests are usually nouns, not verbs

- Typically published as an API, so design and URI naming important
- Expanded and more precise use of HTTP methods
- Expanded use of HTTP status codes
- Content negotiation between client and server

24

Example

- We start by building a very simple RESTful service
- Later we will extend this by
  - Passing parameters to the server
  - Negotiating content
  - Returning content

For all the examples, think of accessing the resources from your html/JavaScript running in your browser

25

Creating a RESTful Root Resource Class

- Root resource classes are POJOs (plain old Java objects)
- Annotated with @Path or a request method designator (@GET, @PUT, @POST, or @DELETE)

JAX-RS uses Java Annotations

26

JAX-RS Annotation Summary …

Annotation	Description
@PATH	Relative URI indicating where the class will be hosted. Can also embed variables (e.g., /helloworld/{username})
@GET	Corresponds to the HTTP GET method. A Java method annotated with @GET will handle GET requests
@POST	Corresponds to the HTTP POST method
@PUT	Corresponds to HTTP PUT method. Intended for resource updates
@DELETE	Corresponds to HTTP DELETE method

27

… JAX-RS Annotation Summary

Annotation	Description
@HEAD	Corresponds to HTTP Method.
@PathParam	Parameter extracted from the request URI. Parameter names correspond to the URI path template variable names specified in the @PATH annotation
@QueryParam	Extracted from the query string
@Consumes	Specifies the MIME type sent by client
@Produces	Specifies the MIME type produced (e.g., “text/plain”)
@ApplicationPath	Defines the URL mapping. Base URI for all resource URIs specified by @Path

28

IDE Support

- Your IDE will likely feature support for RESTful service
  - Java Web project
  - Reference implementation of JAX-RS (e.g., Glassfish)
  - Correct application directory
  - Some starter code

29

HelloWorld.java

```
@Path("helloworld")
public class HelloWorld {
    @Context
    private UriInfo context;

    public HelloWorld() {
    }
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String getHtml() {
      return "<html><body><h1>Hello,World!!</h1></body></html>";
       }
    @PUT
    @Consumes(MediaType.TEXT_HTML)
    public void putHtml(String content) {
    } }
```

This http GET request will return html

Identifies the MIME type of the response

30

MediaType Class

- javax.ws.rs.core.MediaType
- An abstraction for JAX-RS media types
- Contains String constants
- Examples
  - TEXT_HTML – “text/html”
  - TEXT_PLAIN – “text/plain”
  - APPLICATION_JSON – “application/json”

Best to use the MediaType constant strings to avoid typing errors

31

Passing Parameters to a RESTful Service?

- Without using the servlet parameters (request and response) directly, we need a different way to pass parameters from client to server
  - Use the url query string (form data set)
    - Parameters are mapped to arguments in the method signature
  - Use the URL
    - URI components become an argument to the method responding to the request

```
http://example.com/users/Kelly
```

Acts as a parameter

32

Extracting Query Parameters – URL Query String

- Your web service can extract parameters in form dataset

```
@Path("smooth")
@GET
public Response smooth(
@DefaultValue("2")      @QueryParam("step") int step,
@DefaultValue("true")   @QueryParam("min-m") boolean hasMin,
@DefaultValue("true")   @QueryParam("max-m") boolean hasMax,
@DefaultValue("true")   @QueryParam("last-m") boolean hasLast,
@DefaultValue("blue")   @QueryParam("min-color") ColorParam minColor,
@DefaultValue("green")  @QueryParam("max-color") ColorParam maxColor,
@DefaultValue("red")    @QueryParam("last-color") ColorParam lastColor
) { ... }
```

The form dataset is contained in the URL for a GET

Instantiated with the user-defined class constructor

Notice that parameters are parsed into Java types

Missing parameters assume default value

A 400 error code is returned if parameter cannot be parsed

33

Extracting Form Parameters from POST Request

- Recall that form parameters in a POST request are not contained in the URL (they are in the HTTP body)

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("name") String name) {
// Store the message
}
```

Other annotation exists to extract a Map of name-value pairs

34

Data Exchange

- We define the data exchanged by using annotation for Produces and Consumes
- Content format is negotiated by the client and server based on the annotation and the ability of each to handle various formats

35

@Consumes

- @javax.ws.rs.Consumes
- Defines the MIME type the server methods can accept
- Defined at either the class level or the method level
- Selected values
  - application/json
  - application/octet-stream
  - text/html
  - text/plain
  - multipart/form-data
  - application/x-www-form-urlencoded

Strings defined in javax.ws.rs.MediaType

Example

```
@Consumes({MediaType.TEXT_PLAIN,
MediaType.TEXT_HTML})
```

Typical browser encoding of the form data set

36

@Produces

- @javax.ws.rs.Produces
- Defines the MIME type that a REST resource class method can return to the client
- Defined at either the class level (defaults for all methods) or method level
- Selected values
  - application/json
  - application/octet-stream
  - text/html
  - text/plain

Example

```
@Produces({"image/jpeg,image/png"})
```

37

Path Templates

- A path can be defined with a path template – placeholder for a value to be defined by the user
  Example: `http://example.com/users/Kelly`
- Parameter is obtained in the following example

```
@Path("/users/{username}")
public class UserResource {
  @GET
  @Produces("text/html")
  public String getUser(@PathParam("username") String
     userName) {
...
  }
}
```

38

Web Resources Style

- The PathParameter annotation provides a different style in requesting Web resources
- Example

  `localhost:8080/CSE336-Services/library/librarycards/124`

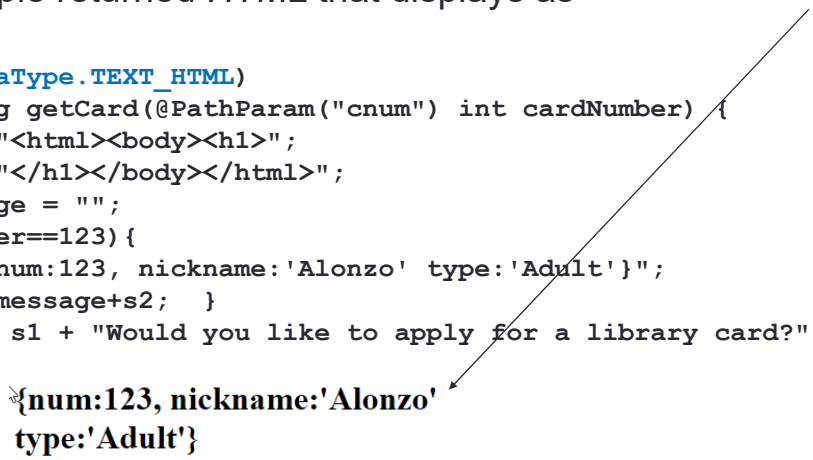
- Made to appear as a data retrieval where the path (e.g., librarycards) appears as a plural data resource, and the path parameter (e.g., 124) appears as if it were an index in the repository for the data resource

39

@Produces Annotation

- The above example returned HTML that displays as

```
@GET
@Produces(MediaType.TEXT_HTML)
  public String getCard(@PathParam("cnum") int cardNumber) {
  String s1 = "<html><body><h1>";
  String s2 = "</h1></body></html>";
  String message = "";
  if (cardNumber==123){
    message="{num:123, nickname:'Alonzo' type:'Adult'}";
    return s1+message+s2;  }
  else {return s1 + "Would you like to apply for a library card?" +
s2; } }
```



40

@Produces Example

- If we change the @Produces annotation, the response is not evaluated as HTML, and only appears as plain text

```
@GET
 @Produces(MediaType.TEXT_Plain)
   public String getCard(@PathParam("cnum") int cardNumber)
{
```

```
<html><body><h1>{num:123, nickname:'Alonzo', type:'Adult'}</h1></body></html>
```

41

@Produces Example

- If we again change the @Produces annotation, when called with localhost:8080/CSE416-Services/library/librarycards/125 it returns the JSON string

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getCard(@PathParam("cnum") int cardNumber) {
String message="{num:125, nickname:'Alonzo' type:'Adult'}";
  if (cardNumber==125){
    return message;
  }
}
```

42

Have You Achieved the Lecture Objectives?

- Understand the fundamental concepts of Web services
- Understand concepts of HTTP
- Become familiar with JAX-RS annotations
- Be able to build a simple Web service
- Understand how to pass parameters and return values with a Web services

43