Servlets

Devdatta Kulkarni

What about server side?

- What concerns need to be handled on the HTTP server?
  - 3 minute exercise

Server side concerns

- Being available to accept connections
- Responding to legitimate requests in timely manner
- Handle concurrent client requests
- Handle large content
- Authentication and authorization
- Caching
  - Handling different kinds of headers

Server side concerns

- Track resources and permissions
- Correctly respond to HTTP methods
  - GET, HEAD, POST, PUT
- Correctly create the required response headers for each request
- Manage and maintain resources
- Request logging

Approaches

- Approach 1: Build as part of your application
  - Example:
    - java TestServerSocket
    - java TestHttPClientSocket localhost /
  - Each web application implements the server functionality
    - Lot of work
    - May get repetitive after implementing few applications

Approaches

- Approach 2:
  - Build a framework that handles common things:
    - Request handling
    - Logging
    - Authentication
  - Let the application handle:
    - Defining resources
    - Defining access control policies
    - Adding header values
  - *Servlets*

Servlets

- A servlet is a Java-based *Web component*, managed by a *container*, that generates *dynamic content*
  - Web component
    - Understands HTTP
  - Container
    - Software entity which manages Servlets through their life-cycle
  - Dynamic content
    - Content that depends on input data
- Specification:
  - http://download.oracle.com/otndocs/jcp/servlet-3_1-fr-eval-spec/index.html

Servlet Container

- Servlet container is a part of a Web server or application server that
  - contains and manages servlets through their lifecycle
  - provides network services over which requests and responses are sent

Servlet Container Functions

- *Must* implement HTTP/1.0 and HTTP/1.1
- *May* support additional request/response-based protocols such as HTTPS
- Container may perform caching subject to RFC 2616:
  - May respond to requests without delivering them to the servlet
    - E.g.: Send static resources from Servlet container cache
    - Tomcat has the “cachingAllowed” attribute which controls this
      - http://tomcat.apache.org/tomcat-7.0-doc/config/context.html
  - May modify requests from the clients before delivering them to the servlet
    - E.g.: Decoding URL encoded values in the request
      - Example: query-parameters
  - May modify responses from the servlets before sending them to the clients
    - E.g.: Adding the ‘Date’ header

Running Servlet examples

- Setting up Eclipse Neon
- Setting up Tomcat
  - Tomcat is a servlet container that we will use
- Creating a Servlet project
  - Dynamic Web project
    - Select “Target runtime”
      - Apache Tomcat v8.0
- hello-world
  - HelloServlet

Setting up Eclipse and Tomcat

- Eclipse Java EE IDE for Web Developers.
  - Version: Neon Release (4.6.0)
  - Build id: 20160613-1800
- Tomcat
  - Download Apache Tomcat 9.0
- Setup Eclipse to use Tomcat
  - Eclipse -> Preferences -> Server -> Runtime Environments -> Add
  - Check Page 35 of ``Java for Web Applications'' book

Creating Servlet Project

- Option 1: Create project from scratch
- Option 2: Import existing project

Creating project from scratch

- In Eclipse, Servlet-based projects are called ‘Dynamic Web Project’
  - File -> New -> Other -> Web -> Dynamic Web Project
    - Give ‘Project Name’ -> ‘Finish’
    - Select “Target runtime”
      - Apache Tomcat v9.0

- Dynamic Web Project structure

  /WebContent

  /META-INF

  /WEB-INF

- Setting up the Dynamic Web Project
  - Add web.xml to WEB-INF
    - See web.xml in ‘hello-world’ for example
    - Add HelloServlet class
      - Eclipse may add the class inside a package; update ‘servlet-class’ attribute in web.xml to include <packagename>.<classname>

Running Servlet

- Right click on the project
- Select ‘Run As’ -> ‘Run on Server’
- Choose the Tomcat v8.0 Server that we configured -> ‘Next’
- Servlet will be now accessible at:

  http://localhost:8080/<context_root>/<url-pattern>

  - Context Root
    - Right click on the project
    - Web Project settings
  - url-pattern
    - Specified in web.xml

Importing existing project

- File -> Import -> Maven -> Existing Maven Projects
- Browse to the directory where you have cloned the repo and select the hello-world project
- In Project Explorer / Package Explorer menu, right click the project and then Maven -> Update Project
- Once Maven has finished downloading the dependencies, right click on the project and choose Run As -> Run on Server
- Run on the configured Tomcat instance

- http://localhost:8080/<context-root>/<url-pattern>

Maven

- Tool that allows us to specify application's dependencies in a *declarative manner*

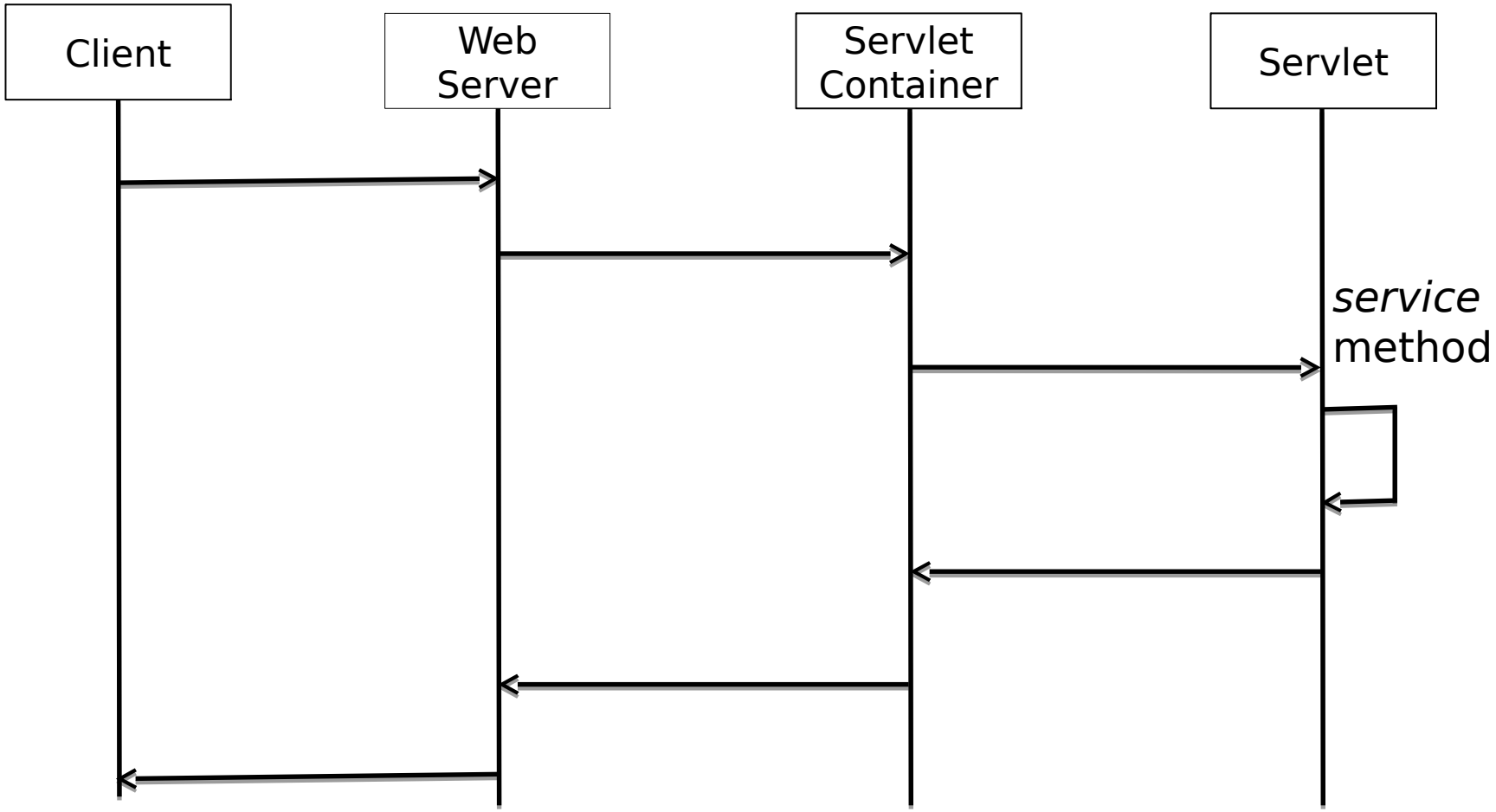
- Dependencies are specified in *pom.xml*

Context Root

- A context root identifies a web application in a Java EE server
  - Think of this as a pointer/reference that allows the web container to identify a web application

- http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html

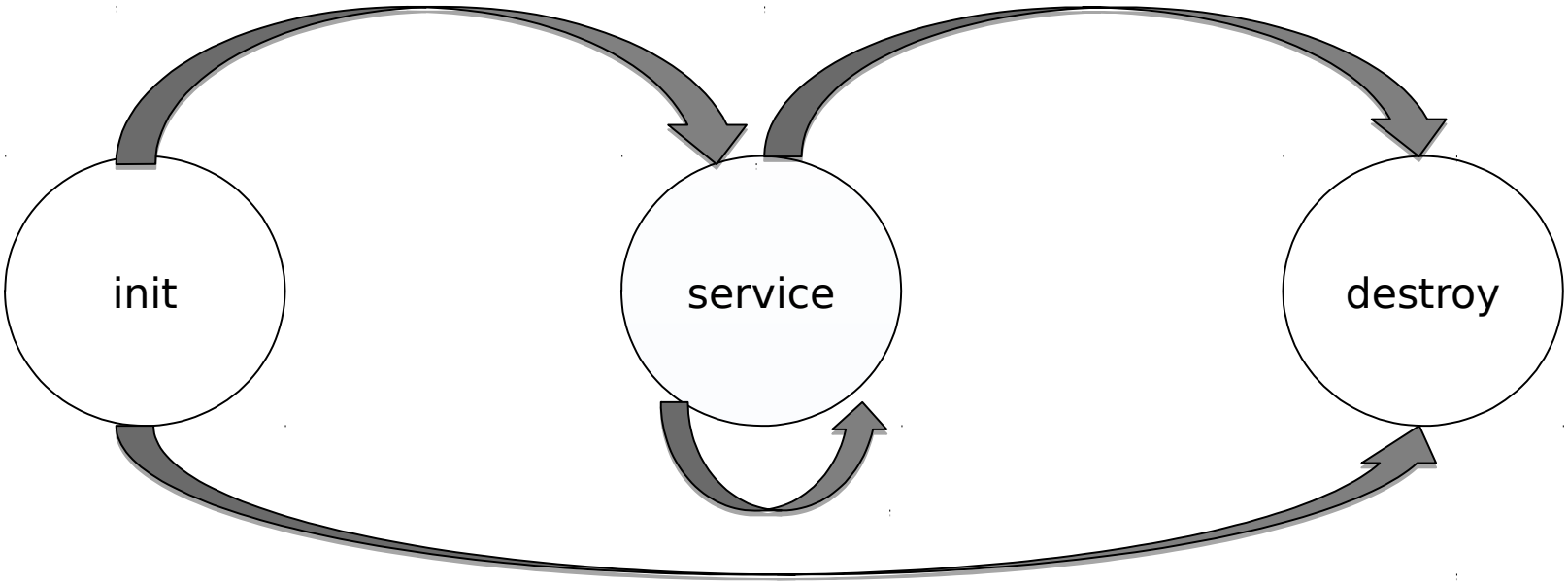
Interaction



Servlet Life-cycle

- Loading and Initialization
  - Servlet container is responsible for loading and instantiating servlets
    - When the container is started or when the servlet is needed
  - Servlet container calls servlet's 'init' method
  - Servlet container calls servlet's 'service' method
- Destroy
  - Servlet container calls servlet's 'destroy' method

Servlet Life-cycle



Web.xml

- How does the Servlet container know where to route incoming requests?
  - Solution: web.xml
- web.xml is the ‘Deployment descriptor’
- Application developer should use it to define
  - Servlet’s name
  - Servlet mapping
  - Servlet’s parameters
  - When to start a Servlet
    - <load-on-startup>
      - Example: Use test-load-on-startup

Servlet Interface

javax.servlet

```
public interface Servlet {
    void init(ServletConfig config)
    void service(ServletRequest req,   ServletResponse res)
    void destroy()
}
```

http://docs.oracle.com/javaee/6/api/javax/servlet/Servlet.html

HTTPServlet

public abstract class HttpServlet {

protected void doGet(HttpServletRequest req, HttpServletResponse res)

protected void doPost(HttpServletRequest req, HttpServletResponse res)

}

http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html

Number of Instances

- In a non-distributed environment
  - the servlet container uses *only one instance per servlet declaration*
- In a distributed environment
  - the servlet container creates *one instance per servlet* declaration *per Java Virtual Machine (JVM)*

Servlets and threading

- Servlet container creates a single instance of a Servlet
- Multiple threads can be active in a Servlet's 'service' method
- Servlet container *does not* synchronize access
- It is up to the application developer to synchronize the access

- Example:
  - Run test-thread-servlet
  - Run test-servlet.sh
    - ./test-servlet.sh

Reading

- Read chapters 1, 2, and 3 from the ``Java for Web Applications'' book