Objectives and Recap

- *Recap*
  - Last week, in lab, we created a simple Java web application with Apache Tomcat
- Objectives
  - Reminders
  - Web applications
  - Regular Expression

Reminders

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

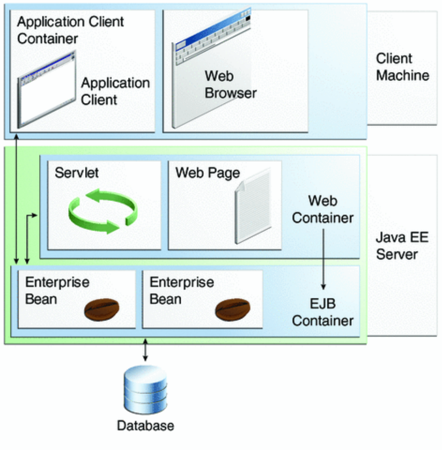
- Homework 1 due September 14
  - Don't leave it until the last minute
- Cumulative quiz on September 23
- Start thinking about forming teams of 5-6 people for the term project

Project General Requirements 361

- "Full stack" development meaning the implementation of both a browser based front-end and a Java server-side back-end
- The server-side component must be a Java enterprise/container application
- The web application should interface to a relational database management system such as MySQL or postgres. An embedded database such as SQLite is not acceptable
- The web application should enable login and user profiles and take reasonable steps to main security and data integrity
- The application should make data available through both the web front-end and through a RESTful API
- The web front-end should also be (partially at least) dynamic, meaning that some information between the client and server should be sent through AJAX requests

Architecture of a Java Web Application

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES



- This diagram[1] shows the essential components of a Java-based web application: client, server, and database

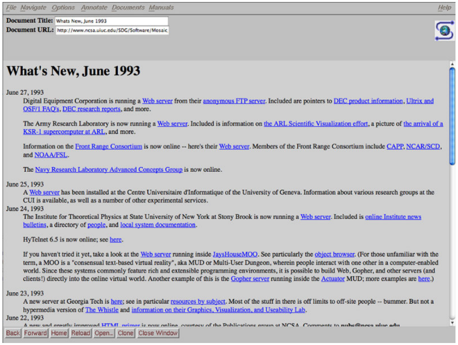
[1] Eric Jendrock et al. *The Java EE Tutorial, Release 7*. Oracle, 2014.

REST

- *Representational State Transfer* (REST) is the architecture of the world wide web[2]
- Accessing web sources with HTTP requests
- Resources identified by a URL (URI)
- Features
    - Client-Server
    - Stateless
- Most popular social media sites, search providers etc. expose a RESTful web API
- REST is based on the idea that the client sends everything needed to service the request

[2] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

Early Web



Document Title: Whats New, June 1993
Document URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic

What's New, June 1993

June 27, 1993

Digital Equipment Corporation is running a Web server from their anonymous FTP server. Included are pointers to DEC product information, Ultrix and OSF/1 FAQ's, DEC research reports, and more.

The Army Research Laboratory is now running a Web server. Included is information on the ARL Scientific Visualization effort, a picture of the arrival of a KSR-1 supercomputer at ARL, and more.

Information on the Front Range Consortium is now online -- here's their Web server. Members of the Front Range Consortium include CAPP, NCAR/SCD, and NOAA/FSL.

The Navy Research Laboratory Advanced Concepts Group is now online.

June 25, 1993

A Web server has been installed at the Centre Universitaire d'Informatique of the University of Geneva. Information about various research groups at the CUI is available, as well as a number of other experimental services.

June 24, 1993

The Institute for Theoretical Physics at State University of New York at Stony Brook is now running a Web server. Included is online Institute news bulletins, a directory of people, and local system documentation.

HyTelnet 6.5 is now online; see here.

If you haven't tried it yet, take a look at the Web server running inside JaysHouseMOO. See particularly the object browser. (For those unfamiliar with the term, a MOO is a "consensual text-based virtual reality", aka MUD or Multi-User Dungeon, wherein people interact with one other in a computer-enabled world. Since these systems commonly feature rich and extensible programming environments, it is possible to build Web, Gopher, and other servers (and clients!) directly into the online virtual world. Another example of this is the Gopher server running inside the Actuator MUD; more examples are here.)

June 23, 1993

A new server at Georgia Tech is here; see in particular resources by subject. Most of the stuff in there is off limits to off-site people -- bummer. But not a hypermedia version of The Whistle and information on their Graphics, Visualization, and Useability Lab.

June 22, 1993

A new and greatly improved HTML primer is now online, courtesy of the Publications group at NCSA. Comments to **pubs@ncsa.uiuc.edu**

- A typical web page of the time, early Netscape browser
- Dial-up modems had a practical rate limitation of around 56 kbps (56,000 bits in 1 second)
  - 1 Floppy Disk: $(1.44)(10^6)(8)(1/56,000)(1/60) \approx 3.43$ minutes
  - 1 CD: $(650)(10^6)(8)(1/56,000)(1/60^2) \approx 26.8$ hours
- Note that there is just text and links

HTTP Methods

- `GET`: requests data from a url
- `POST`: submit data to be processed at a url
- These methods behave differently in terms of caching, interaction with back button, etc.
- `PUT`: place some data/document at a url
- `GET` and `PUT` must be *idempotent* operations

Front-End Development

- *Front-end web development*: Design and coding of client-side HTML, CSS, and Javascript
- Although web browsers are extremely common there is still lots of fragmentation
    - Screen size
    - PC and mobile
    - Operating System
    - Browser
- Dynamic behavior of websites is achieved with Javascript

Notes on Javascript

- The actual standard describing the Javascript language is called “ECMAScript” (European Computer Manufacturers Association)
- Original prototype designed by Brendan Eich in 10 days in 1995
- Javascript is not really related to the Java programming language (Java was simply popular at the time)

Browsers

- Browsers render HTML files
- Utilize a Document Object Model (DOM)
    - Tree-like structure describing the page
    - Root node for entire page
    - `head` and `body` nodes
    - Anchors, paragraphs, tables, lists, list entries, ...
- Browser *renders* a page to its viewport
- Scripts are included within a page via `<script> </script>` tags

Notes on jQuery

- jQuery is the most popular Javascript library today
- Free, open source software (MIT License)
- Tools for making scripting easier: selecting elements, adding handlers, ...
- Concise syntax
- Handles differences in browsers

Scripting with and without jQuery

```
// select all first paragraph
document.getElementsByTagName("p")[0].innerHTML = "New text.";

// same query in jQuery
$("p").html("New text.");
```

- These lines of code will each replace the contents of the "first" <p> with "New text."
- The dollar sign is a shorthand for `jQuery()`
- The `jQuery()` function is heavily overloaded, we can pass many different things to it depending on what we want to achieve

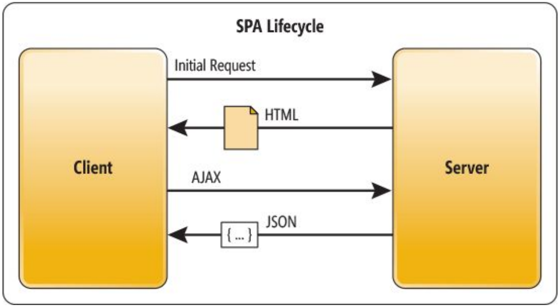
Java Server

- Java EE is a set of specifications and APIs
- There are many specific implementations of these specifications, some more complete than others
- Apache Tomcat is a Servlet Container which does not implement the complete JavaEE Spec
    - *TomEE* is a more complete EE server built on top of Tomcat
- For your projects the server has a few important functions
    - Serve up HTML content to the user
    - Serve machine readable content to the browser (enable dynamic pages)
    - Connect to the database when needed

Serving up HTML

- You might have some static HTML/JS pages that you simply want to make available through your server
- Recall that Java EE Web apps rely on a certain directory structure
- Static HTML Content can be placed in `src/main/webapp/` (we can also put subdirectories under `webapp`)
- JSP pages can be served in the same way
  - Javaserver Pages (JSP) are HTML documents containing special tags <%...%> that allow Java code to be embedded
  - The EE server will dynamically convert these pages into servlets

Single Page Application Sequence Diagram



- This sequence diagram shows the basic interaction we would like to achieve
- The user makes an initial request, and subsequent communications with the server happen through *ajax*

AJAX and jQuery

- AJAX requests can also be made through the `jQuery` function
- Syntax is a bit simpler than using xhr directly
- `$.ajax()`
  - Asynchronous Ajax request
  - Pass in the parameters of the request as a Javascript object: url, callback function, type of request, etc.
- Also have convenience methods for Ajax requests: .load(), .get(), .post()

Things to Consider

- Browsers enforce the *Same-Origin Policy* for security reasons
- HTML/Javascript is not allowed to request data from anywhere on the internet except in certain cases (e.g. loading images)
- Data between the client and server should be kept consistent, this is also called *data-binding*
- For example consider the following mapping between RESTful API calls and `backbonejs` functions
  - GET /books/ .... collection.fetch();
  - POST /books/ .... collection.create();
  - GET /books/1 ... model.fetch();
  - PUT /books/1 ... model.save();
  - DEL /books/1 ... model.destroy();

Configuration for the DB

```
<Resource name="jdbc/TestDB"
          auth="Container"
          type="javax.sql.DataSource"
          maxActive="100"
          maxIdle="30"
          maxWait="10000"
          username="username"
          password="password"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/publications"/>
```

- Need to inform our server or servlet container where our database is
- In Tomcat we can specify this information as a JNDI resource in the `context.xml` file

Getting DB Connection from Java

```
private static Connection getDatabaseConnection() {
    Connection conn = null;

    try {
      Context initCtx = new InitialContext();
      Context envCtx = (Context) initCtx.lookup("java:comp/env");
      DataSource ds = (DataSource) envCtx.lookup("jdbc/TestDB");
        conn = ds.getConnection();
      } catch (Exception e) {
        e.printStackTrace();
      }

    return conn;
}
```

- Assuming DB is running and has been defined in the context, connection can be accessed as above
- Obviously in a production environment have be careful about null-checking etc.

Maintaining State

- HTTP protocol is fundamentally stateless
- The client and server side applications must cooperate to provide the user with a seamless experience
- Mechanisms for doing this are *url rewriting* and *cookies*
- *Example*: We want to maintain some information as the user navigates through multiple pages, e.g. handling a login process

Interaction between state and servlets

```
HttpSession session = request.getSession();

String name = request.getParameter("Name");
String value = request.getParameter("Value");

if ( name != null && value != null ) {
    session.setAttribute( name, value );
}
```

- From the point of view of a servlet programmer the session information is accessed through a special class called `HttpSession`
- Basically a key-value store that we manipulate with getter and setter methods

URL Parameters

```
..../HelloWorldServlet?name=mark&word=test
```

- Parameters can be passed to the server using the notation above
- question mark for first parameter and ampersand for other parameters
- Alternative way to pass data to the server is with the POST method

Properties of sessions

- The session object is like a hash-table that can be accessed with getter and setter methods
- If we want to clear a session we can call the `invalidate()` method on the session object
- Sessions can also time-out, behavior can be configured in the `web.xml` configuration file

Controlling Access with Sessions and Filters

- Oftentimes, we will like to restrict certain parts of our web application only to users who have authenticated
- Java EE provides a simple solution this with *Filters*

Filters in Java EE

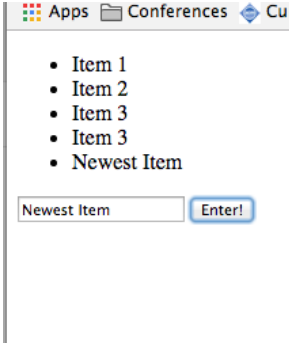
```
@WebFilter("/p/*")
public class ContestFilter implements Filter {
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
                         throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpSession session = req.getSession();
        String test = (String) session.getAttribute("user");
        HttpServletResponse res;
        if (test == null) {
            res = (HttpServletResponse) response;

            res.sendRedirect("/mywebapp/login.html");
        } else {
            chain.doFilter(request, response);
        }
    }
}
```

- In this case we created a protected folder that will that is always routed through this filter, no login means the user is redirected

The Simple Todo List

- In a todo list, the user enters some text that gets placed into an unordered list
- Such an application would be easy to implement in the client (browser) but the data would be lost every time the page was closed or refreshed
- Data should persist and be consistent some external data store



- Item 1
- Item 2
- Item 3
- Item 3
- Newest Item

Newest Item

Presentation and Data Layers

- Logically, the list items should reside on the server and the client-side should only act as an interface
- Beginning to think in terms of components and architecture
- *Endpoint Servlet*: Act as the controller and encapsulate the model (data) in the form of a simple list data structure
- *HTML*: Javascript to make an Ajax request when the page loads and automatically populate the html list with values from the request (client-side rendering)

Endpoint Servlet

```
public EndpointServlet() {
        super();
        listItems = new ArrayList<String>();
    }
```

- `POST`: reads the data submitted by the user and add it to the internal representation of the todo list
- `GET`: return JSON string of the current state of the list as it resides on the server
- *Model*: A List of Strings

GET Behavior

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) throws
                     ServletException, IOException {
        Gson gson = new Gson();
        String json = gson.toJson(listItems);

        response.getWriter().append(json);
}
```

- GET request simply returns the data stored in our model
- Gson is a library for mapping back and forth between Java classes and JSON data

POST Behavior

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws
                      ServletException, IOException {
            Map<String, String[]> m = request.getParameterMap();

    if (m.containsKey("newItem")) {
            String[] s = m.get("newItem");
            response.getWriter().append(s[0]);
            listItems.add(s[0]);
        }
}
```

- When the Endpoint receives a POST request it adds the value in the parameter to the model

Dealing with Requests

- When writing a web application, you may want to branch based on the request URL
  - Using the `@WebServlet` servlet annotation, one servlet can process requests for many different URLs
- RESTful apis are about accessing resources, we want to provide a sensible interface to those resources
- For example, for the to-do list we want our api to expose both the collection and the individual items
- One way to process a lot of string data is to use *regular expressions*

Regular Expressions (Java Classes)

- Regular Expressions (or *regex*) are a type of syntax for defining pattern matchers
  - Almost all languages have some support for regular expressions
  - Notation is generally consistent
  - A regular expression defines a (potentially infinite) set of strings
- In Java we use the `java.util.regex.Pattern` and `java.util.regex.Matcher` classes to work with regular expressions

```
Pattern pattern = Pattern.compile("[abc]+");
Matcher m = pattern.matcher("aabba");

m.matches(); // returns true for match
```

Regular Expressions (Basic)

NAZARBAYEV UNIVERSITY SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

- A simple string is the simplest regular expression, searching for a fixed pattern
- For example, if we want to find the occurrences of the word ‘hello‘ in a sentence then we define a regex `"hello"`
- There are two important methods on the matcher object
  - `.find()`: search the input for a pattern
  - `.matches()`: attempt to match the entire input against the pattern

```
pattern = Pattern.compile("hello");
Matcher m = pattern.matcher("Can you say hello?");

// prints From 12 to 17
if (m.find()) {
    System.out.println("From " +
                      m.start() +
                         " to " +
                         m.end());
}
```

Regular Expressions (Syntax)

- There is more to regex than finding a fixed pattern in a string
- Regex is almost an entire language unto itself (fairly consistent across implementations)
- Some of the most important regex features are the following
  - Alternatives, one thing *or* something else, indicated with vertical bar | or square brackets
  - Character classes, express common ranges of characters (lowercase letters, digits)
  - Quantification, express how many times a pattern should occur
  - Grouping, extract sub-patterns in a match

```
Pattern.compile("b[ioa]t"); // matches bit, bat, and bot
Pattern.compile("b(i|o|a)t"); // same as above
```

Regular Expressions (Character Classes)

- Set of characters enclosed in square brackets, match single character of input
- Some predefined ranges in Java
  - [0-9]: digits
  - [a-z]: lowercase letters
  - [A-Z]: uppercase letters
  - [a-zA-Z]: upper and lower case letters
  - .: matches any single character
- Character classes can be negated with the caret symbol ˆ
- Java also supports some POSIX character classes, see example below

```
// match any single character except a,b, or c
Pattern.compile("[^abc]");

// pattern for posix punctuation class
pattern = Pattern.compile("\\p{Punct}");
```

Regular Expressions (Quantification)

- We can also specify how many times something should occur using the *quantification* meta-characters: ?,*,+
  - `A*`: Kleene star, ‘A’ occurs zero or more times
  - `A+`: ‘A’ occurs one or more times
  - `A?`: ‘A’ occurs zero or one times
  - `A{n}`: ‘A’ occurs *n* times

```
// match any character any number of times
pattern = Pattern.compile(".*");

// match one or more digits followed by three letters
// e.g. 01234tuv
pattern = Pattern.compile("[0-9]+[a-zA-Z]{3}");

// two sequences of digits separated by a lowercase letter
// 11111a2222, a22222, 11111a all match
pattern = Pattern.compile("[0-9]*[a-z]?[0-9]*");
```

Regular Expressions (Grouping)

- *Capture groups* allow us to capture part of a match
  - For example, we are processing names and want to quickly separate the first and last names
  - Use parentheses to indicate a group and retrieve group with the `group(int a)` method on the matcher

```
pattern = Pattern.compile("/([a-z]+)/([0-9]+)");

Matcher m = pattern.matcher("/abc/001");
m.matches();

// prints abc
System.out.println(m.group(1));

// prints 001
System.out.println(m.group(2));
```

Regular Expressions (Caveats)

- Regular expressions are a powerful notation for expressing patterns, but it is important not to overdo them
- Can make your code more difficult for others to interpret
- Easy to write a regex that you think is doing one thing but that is actually doing something else
- Consider the date validator below
- Strictly speaking, regular expressions are equivalent to the least expressive models of computation
    - The set of strings specified by a regex is a formal language
    - Most restricted grammar in the Chomsky hierarchy (Type-3 Grammars)

```
pattern = Pattern.compile("[0-9]{2}/[0-9]{2}/[0-9]{2}");
Matcher m = pattern.matcher("10/11/12");

System.out.println(m.matches());
```

lodash and Templating

NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

```
// Create a template function
var listTemplate = _.template('<p><%= name %></p>');

// ...
// Use the template function
$('#maindiv').append(listTemplate({name : e}));
```

- A *template* is a special function that performs string interpolation
  - String interpolation: create a string by substituting for placeholder values
- In the example, we use `lodash`[3] (a utility library for javascript) to create a template function
- The template function takes a javascript object as an argument

[3] `https://lodash.com/docs/#template`

Next Lecture

- Designing RESTful APIs with JAX-RS