Selected Topics in Java Web Application Development

Dr Gerald Loeffler, MBA

Web Engineering I
WU Wien

this presentation

- agenda
  - web applications and Java EE
    - web applications, Java EE application servers, processing paradigms, important APIs
  - availability and scalability of web applications
    - definitions, horizontal and vertical scalability, load balancers, clustering, availability through redundancy, using HttpSession
  - browser-server interaction beyond simple request-response
    - AJAX and XMLHttpRequest, JSON, Comet, Bayeux, asynchronous servlet request processing
- this is an overview presentation and will – sadly – not convey enough detail to actually start using any of the covered technologies –> see references and bibliography!

references and bibliography (1/4)

[1] Java Platform, Enterprise Edition 5 (Java EE 5) Specification, Sun Microsystems

http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html

[2] The Java EE 5 Tutorial, Sun Microsystems

http://java.sun.com/javaee/5/docs/tutorial/doc/

[3] What is an App Server?, Joseph Ottinger,

http://www.theserverside.com/tt/articles/article.tss?l=WhatIsAnAppServer

[4] Release It! Design and Deploy Production-Ready Software, Michael T. Nygard, The Pragmatic Bookshelf

[5] Auto-Complete with AJAX, Greg Murray

https://bpcatalog.dev.java.net/ajax/autocomplete/index.html

references and bibliography (2/4)

[6] ICEfaces Developer's Guide, ICEsoft Technologies

http://www.icefaces.org

[7] The XMLHttpRequest Object, W3C

http://www.w3.org/TR/XMLHttpRequest/

[8] Introducing JSON

http://json.org

[9] ECMAScript Language Specification 3rd Edition, Standard ECMA-262, ECMA

http://www.ecma.ch

[10] Bayeux Protocol 1.0draft1, The Dojo Foundation

http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html

references and bibliography (3/4)

[11] Comet: Low Latency Data for the Browser, Alex Russell

http://alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/

[12] GlassFish application server

https://glassfish.dev.java.net/

[13] Project Grizzly

https://grizzly.dev.java.net/

[14] dojo – the JavaScript toolkit

http://dojotoolkit.org/

[15] Java theory and practice: State replication in the Web tier, Brian Goetz

http://www.ibm.com/developerworks/java/library/j-jtp07294.html

references and bibliography (4/4)

[16] Java Servlet Specification, Version 3.0 Early Draft, Sun Microsystems

http://jcp.org/aboutJava/communityprocess/edr/jsr315/index.html

[17] Are all stateful Web applications broken?, Brian Goetz

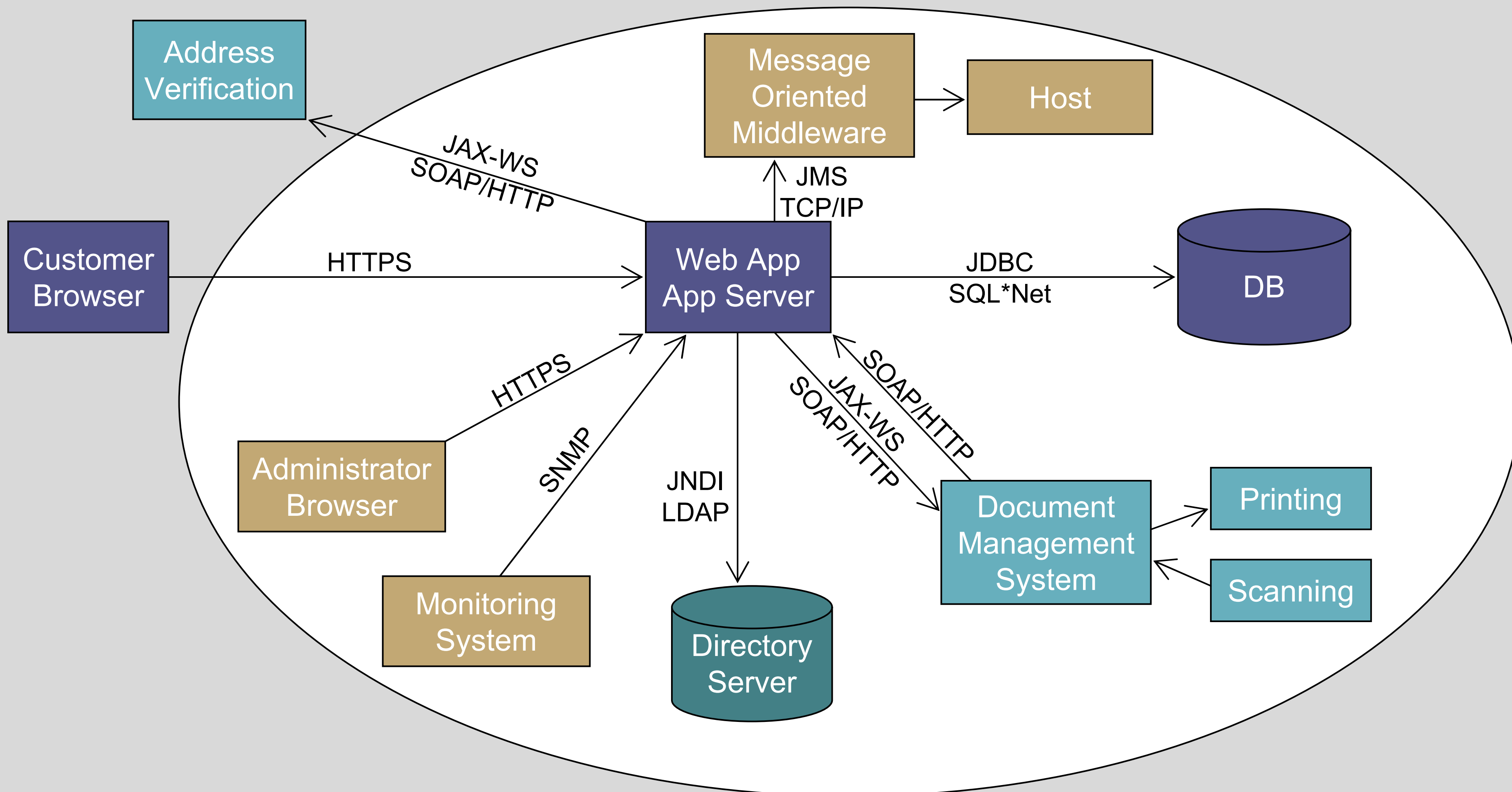
http://www.ibm.com/developerworks/java/library/j-jtp09238.html

[18] The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627, IETF, D. Crockford

http://www.ietf.org/rfc/rfc4627.txt?number=4627

web applications and Java EE

what's a web application?

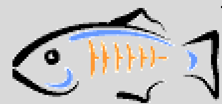


what's a Java EE application server?

- software product
  - Sun Glassfish, Oracle/BEA WebLogic, IBM WebSphere AS, ...
- runtime environment for Java EE apps/modules/comps
  - web apps, enterprise apps, connectors, ...
- exposing Java EE APIs to these applications
- providing services such as
  - listeners for HTTP/HTTPS (hence SOAP/HTTP), IIOP
  - thread pools
  - JDBC connection pools
  - transaction management
  - JMS provider (i.e., message-oriented middleware)
  - security, management, deployment, lifecycle, ...

References: [1, 3, 12]

demo GlassFish v2 admin basics



References: [12]

processing paradigms in Java EE (1/2)

- synchronous request/response
  - HTTP: Servlets, JSPs, SOAP/HTTP; IIOP; local: EJBs
  - client issues request, waits for server to respond
  - server receives request, blocks thread until response ready
  - by far the most common in typical Java EE applications
  - can be asynchronized selectively
    - asynchronous HTTP client
    - asynchronous EJBs in EJB 3.0/Java EE 6
    - async servlets in Servlet 3.0/Java EE 6

References: [2, 3, 16]

processing paradigms in Java EE (2/2)

- timer-triggered execution/batch processing
  - EJB timers
  - EJB creates timer
  - application server invokes timeout callback on EJB
- asynchronous messaging via JMS
  - message producer sends message to destination (queue, topic) hosted in JMS provider (MOM)
  - message consumer receives message from destination
    - receipt is decoupled from send
    - may be synchronous (polling) or asynchronous (callback)
- asynchronous processing of messages from connector
  - similar to asynchronous JMS message receipt via MDBs

References: [2, 3]

some important APIs in Java EE 5

- EJB 3.0 including Java Persistence (JPA) 1.0, JDBC 3.0
  - transactional, secure, remotable business logic components
  - persistent entitites: objects stored in a relational database
- Servlet 2.5, JSP 2.1, JSF 1.2
  - for building web user interfaces in web applications
- JMS 1.1
  - sending/receiving messages to/from queues/topics
- JTA 1.1
  - to interface with a transaction manager, e.g. to demarcate TXs
- JAX-WS 2.0, JAXB 2.0, StAX 1.0
  - expose or call web services
  - manipulate XML documents

References: [1, 2]

availability and scalability of web applications

definitions (1/3)

- workload
  - the number of transactions (requests) *requested* from a system in a given time span
  - less precise: the number of concurrent users
  - transactions per second, requests per second
  - e.g., 500 request per second
- performance
  - the time required to process a single transaction (request)
  - response time
  - varies depending on the kind of transaction and the *workload* of the system
  - e.g., 0.2s

References: [4]

definitions (2/3)

- throughput
  - the number of transactions (requests) *processed* by a system in a given time span
  - transactions per second, requests per second
  - e.g., 250 TPS
- capacity
  - the sustainable *throughput* of a system while maintaining “good enough” *performance*
  - e.g., 300 requests per second at 0.2s median response time
- scalability
  - if/how to increase the *capacity* of a system
  - typically by adding hardware

References: [4]

definitions (3/3)

- availability
  - the fraction of time the system executes transactions/requests correctly and with “expected” performance
  - 1 - downtime
  - should be qualified with type of request/functionality
  - e.g., 99.999% availability for opening new accounts, 98% availability for managing existing accounts, 90% availability for functionality accessed through admin user interface

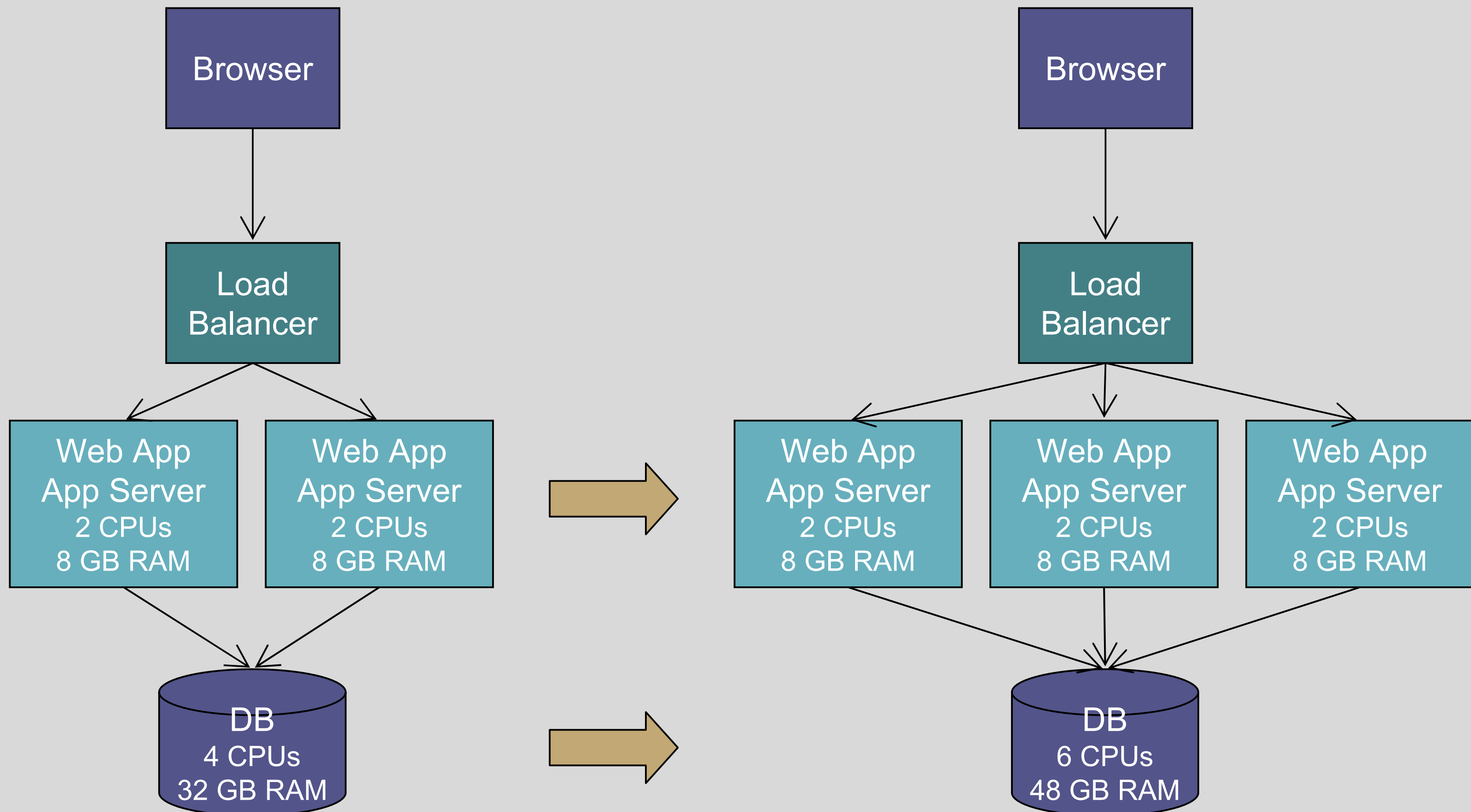
References: [4]

scalability

- vertical scalability
  - add hardware (CPUs, RAM, disks, …) to existing servers
  - limited by type/model of server
  - typical for scaling database servers
- horizontal scalability
  - add new servers
  - also adds redundancy
  - typical for scaling web/application servers
- linear scalability
  - increasing available hardware (horizontally or vertically) by a factor of F increases *capacity* by the same factor
    - 300 rps with 2 servers, 600 rps with 4 servers, @ same resp time

References: [4]

horizontal and vertical scalability



References: [4]

load balancer

- client directs request (HTTP or any TCP/IP-based protocol) at load balancer (through its virtual IP) which forwards it to one of several back-end servers
    - web app on back-end servers gens URLs with lb's DNS name
- configurable distribution algorithm
    - round-robin, weighted/unweighted, request/load-based, ...
    - session affinity: lb routes all requests for the same HTTP session to the same server (inspects cookies/request URLs)
- detects health of back-end servers
    - periodically probes servers with health-check requests
- hardware (F5, Altheon) or (with limitations) software (Apache mod_proxy_balancer)

References: [4]

application server clustering (1/3)

- all serious Java EE application servers are clusterable
- keeping *state* in application server nodes makes the nodes distinguishable (non-interchangeable)
  - here: HTTP session state
  - but also: stateful session bean instances, in-memory caches, ...
- the primary feature of clustering is *state replication*, i.e., the copying of state from the app server node in the cluster that has the "authoritative copy" of that piece of state (typically because that's where it has last been changed)
  - so that the heap (RAM) of that app server node is not the only place where that state is kept
  - and so that that state can be accessed from other nodes

References: [4, 15]

application server clustering (2/3)

- where can (session) state be copied to?:
  - to other app server nodes (backup servers) in the cluster
    - must decide to how many (at least 1) and which one(s)
    - load-balancer must be able to identify the backup servers
  - to a database shared amongst all app server nodes
    - slow, database quickly becomes bottleneck
  - to a filesystem shared amongst all app server nodes
  - (to the browser, via cookies or Bayeux)
- when to replicate state (scalability vs. correctness):
  - when it has changed (before sending HTTP response)
    - complete state or only changes; synchronous or asynchronous
  - at regular intervals

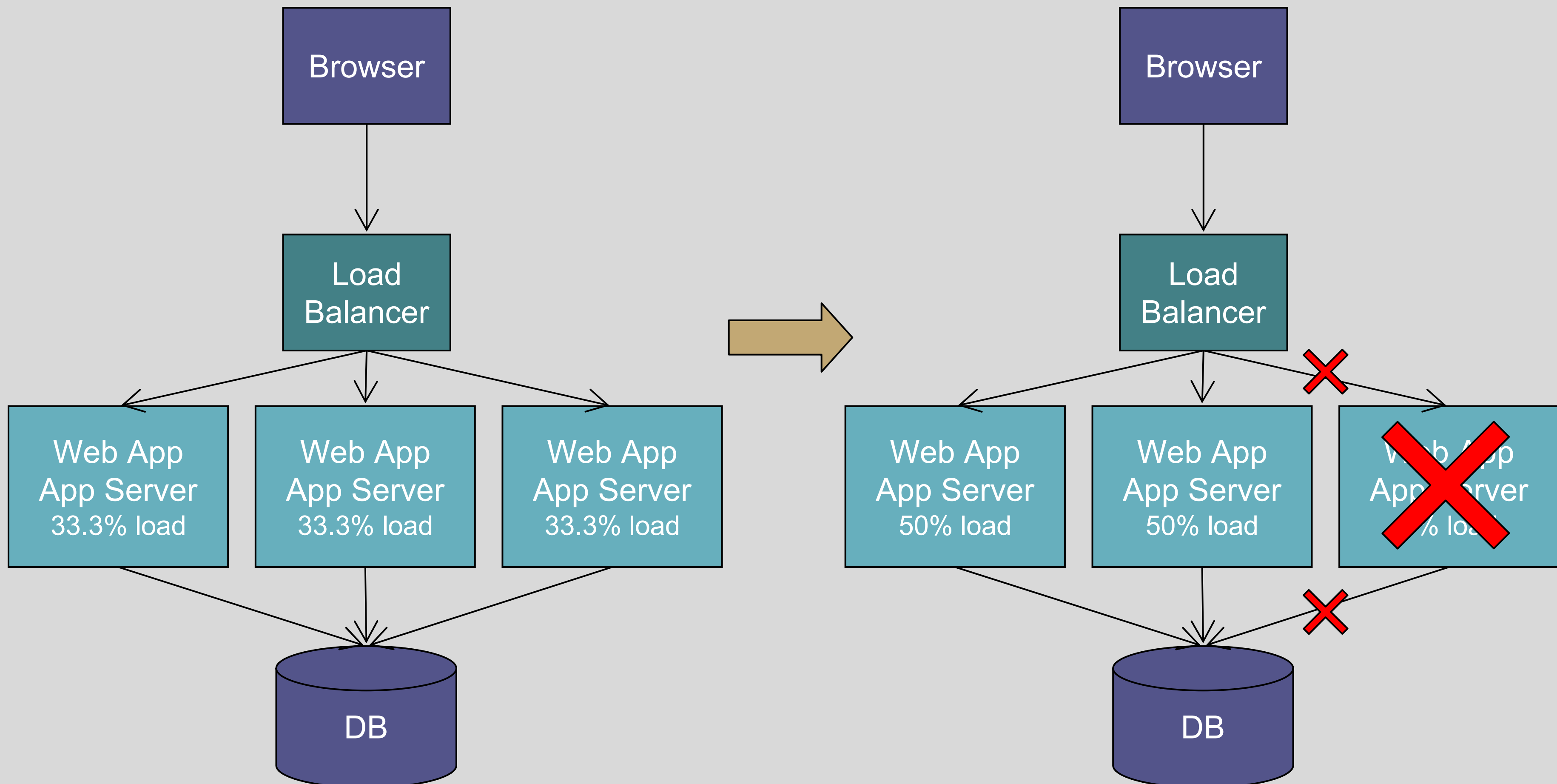
References: [4, 10, 15]

application server clustering (3/3)

- cluster management (including state replication) inevitably adds overhead, resulting in less than ideal scalability
  - state replication requires resources
    - CPU for serialising/de-serialising state
    - RAM and/or disk space for the state itself and its transport form
    - network bandwidth for replicating state
  - only a "shared-nothing" architecture can scale linearly
  - if scalability is of the essence, and (session) state has to be provided (as it usually must) then the only option is to sacrifice availability by not replicating session state
    - removing the need for the most important feature of an application server cluster

References: [4, 15]

availability through horizontal scalability



References: [4, 15]

using HttpSession correctly (1/2)

- all session attributes must be Serializable
- call HttpSession. setAttribute() after changing an attribute
  - this allows you to rely on replication of changed state only
- keep overall session small
- keep each attribute small
- use HttpSession invalidate() and removeAttribute()
- expect concurrent (multi-threaded) access to HttpSession and each session attribute!
- if using a web framework try to set “synchronize on session”
- mark web app as distributable to enable clustering
  - web.xml: <distributable />

References: [16, 17]

using HttpSession correctly (2/2)

```
    final HttpSession session = req.getSession();

    final Cart cart = getCart(session);
    cart.addItem("#123", 2); // atomic/synchronized
    setCart(session, cart); // make change to cart explicit

  private Cart getCart(HttpSession session) {
    Cart cart;
    synchronized (session) {
      cart = (Cart) session.getAttribute("cart");
      if (cart == null) {
        cart = new Cart();
        setCart(session, cart);
      }
    }
    return cart;
  }

  private void setCart(HttpSession session, Cart cart) {
    session.setAttribute("cart", cart); // atomic/synchronized
  }
```

References: [16, 17]

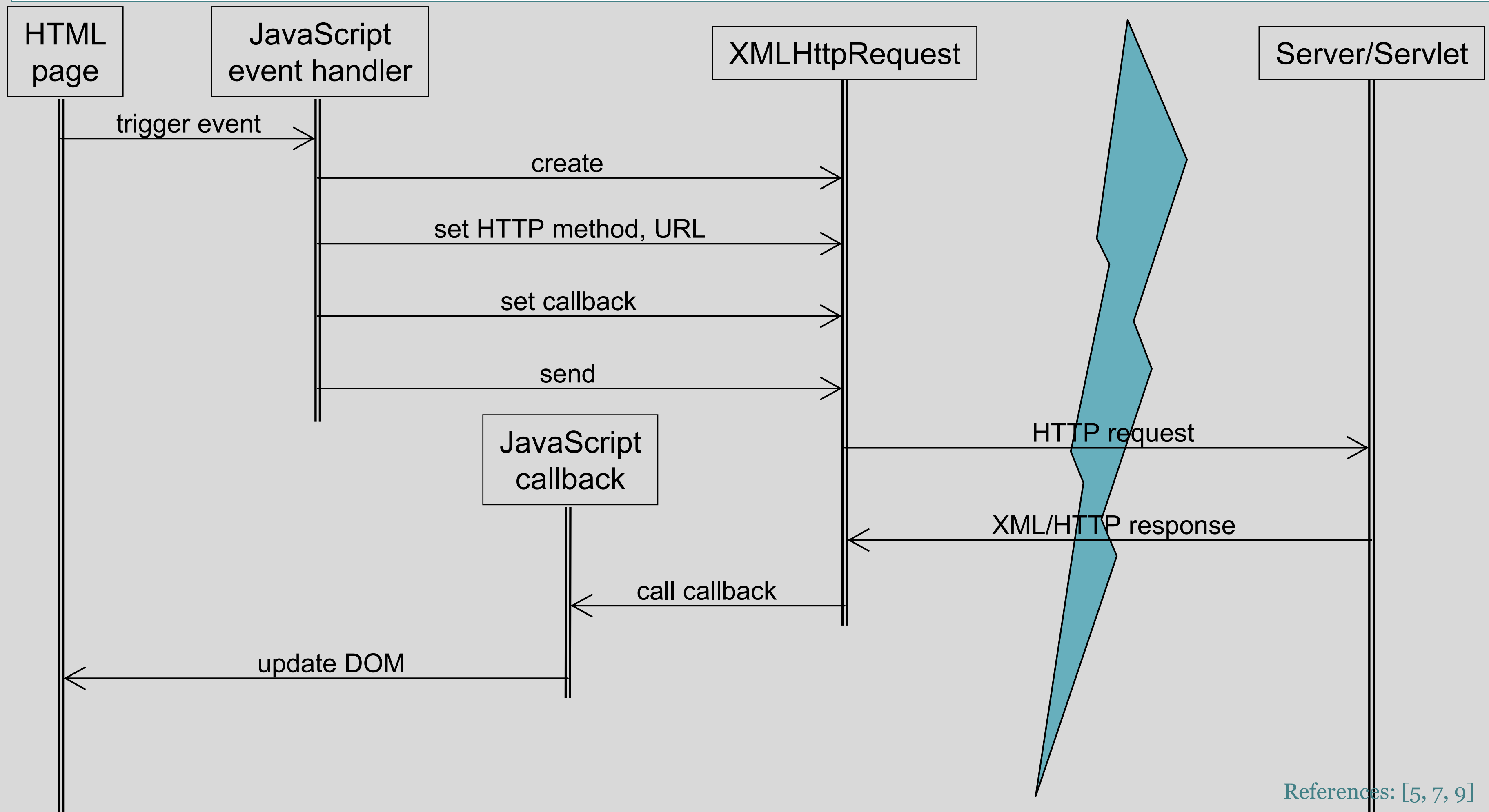
browser-server interaction beyond simple request-response

AJAX

- = **A**synchronous **Ja**vaScript and **X**ML
- a client-side web programming technique that makes web applications more dynamic
- JavaScript embedded in an HTML page (typically dynamically generated by a JSP) issues asynchronous HTTP/HTTPS requests (to the server from which it was loaded) which return XML (or JSON or similar). The response content is used to dynamically update the DOM of that HTML page (rather than load a new page).
    - made possible by XMLHttpRequest object in JavaScript
- applications: auto-completion of entered text, on-the-fly form data validation, refreshing data; think Google

References: [5, 7, 9, 14]

AJAX browser-server interaction



References: [5, 7, 9]

working with XMLHttpRequest in JavaScript

```
var xhr;

function sendRequest() {
  // extract request data from HTML page and (e.g.) pack into URL
  var field = document.getElementById("dataField")
  var url = "/autocomplete?data=" + encodeURIComponent(field.value)
  xhr = new XMLHttpRequest() // browser-dependent!
  xhr.onreadystatechange = callback
  xhr.open("GET", url, true) // async GET request
  xhr.send(null) // no body in GET request, request data is in URL
}
function callback() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    // update DOM based on xhr.responseText or xhr.responseXML
  }
}
```

References: [5, 7, 9, 14]

using AJAX-enabled JSF components

```
<ice:selectInputText width="300"
   rows="#{bean.numCities}"
   value="#{bean.selectedCity}"
   valueChangeListener="#{bean.valueChanged}">
     <f:selectItems
       value="#{bean.matchingCities}"/>
</ice:selectInputText>
```

- JavaServer Faces GUI components can be AJAX-enabled (-> ICEfaces)
- capitalizes on JSF technology base
- no JavaScript necessary
- #{} refers to server-side Java code in backing beans

SelectInputText using selectItem as child for plain text.

new y

New York
New York Mills
New Zion
Newalla
Newark
Newark Valley
Newaygo
Newberg
Newbern
Newberry
Newberry Springs
Newborn
Newburg
Newburgh
Newbury

References: [2, 6]

AJAX web application characteristics

- rich internet applications (RIA), “application on a page”
- depart from request-response nature of traditional (“web 1.0”) web applications
  - many more requests, at higher frequency (1 per second?)
    - roundtrip latency of proportionally higher importance
    - high likelihood of concurrent requests for each user session
  - typically smaller requests and responses
    - especially if using JSON
  - may make browser back button meaningless (yet again)
- AJAX requests must contain the user session identifier just like any other HTTP requests
- in general, AJAX requires more server-side resources

References: [4]

JSON

- = **J**ava**S**cript **O**bject **N**otation
- a data-interchange format encoding unordered sets of name-value pairs and ordered lists of values, possibly nested, as JavaScript source code (object and array definitions)
  - more concise than XML
  - could (ignoring security!) be eval()-ed from JavaScript
  - can be parsed and generated from any language

References: [8, 9, 18]

JSON vs XML: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5">
  <display-name>My Web App</display-name>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
</web-app>
```

439 bytes

References: [8]

JSON vs XML: JSON


```
{
"web-app": {
  "version": "2.5",
  "display-name": "My Web App",
  "servlet": {
    "servlet-name": "Faces Servlet",
    "servlet-class": "javax.faces.webapp.FacesServlet"
  },
  "servlet-mapping": {
    "servlet-name": "Faces Servlet",
    "url-pattern": "/faces/*"
  }
}
}
```


291 bytes

References: [8, 9, 18]

Comet

- aka AJAX push, server push
- a technique to push data from the server to the browser
  - literally opening an HTTP request from the server to the browser would require a HTTP server running in the browser and fails because of numerous security restrictions
- typical implementation: AJAX with long polling
  1. browser uses XMLHttpRequest to send request to server
  2. server creates response only when there is data to be sent to the browser
     - request may time out if this takes too long
  3. after receiving response (or timeout) browser immediately opens next request...goto 1.

References: [11, 13, 14]

Bayeux

- a nascent web-centric (asynchronous) messaging protocol
  - designed primarily for HTTP/HTTPS
  - messages are JSON objects
  - messages sent to/received from named channels: publish-subscribe messaging
  - message delivery client->server, server->client, client->browser
    - client: any Bayeux-enabled HTTP client, e.g. JavaScript executing in a browser using e.g. dojo library
- defines a concrete, interoperable approach to Comet and more general messaging architectures
- implemented e.g. in Grizzly which is used in GlassFish v3

References: [10, 12, 13, 14]

asynchronous servlets (Servlet 3.0/Java EE 6)

- API to allow a servlet to suspend/resume request processing
  - ServletRequest suspend(), resume(), complete()
- allows the app server to free the request processing thread until a response can be produced
  - serve more web clients without increasing thread pool size
- should be used whenever a servlet can not produce a response immediately
  - such as when waiting for the response to a SOAP request, a JMS response message or even a JDBC connection (to become available from a pool)
  - essential for Comet (long polling)

References: [16]

asynchronous servlet request processing (1/2)

```
public class AsyncServlet extends HttpServlet {

  private MyResult getResult(HttpServletRequest req) {
    return (MyResult) req.getAttribute("result");
  }
  private void setResult(HttpServletRequest req, MyResult result) {
    req.setAttribute("result", result);
  }

  @Override
  protected void doGet(final HttpServletRequest req, HttpServletResponse resp)
   throws IOException {
    if (req.isInitial() || getResult(req) == null) {
      processRequestAsync(req);
    } else {
      respond(req, resp);
    }
  }
```

References: [16]

asynchronous servlet request processing (2/2)

```
  private void processRequestAsync(final HttpServletRequest req) {
    req.suspend();
    final ExecutorService executor = Executors.newSingleThreadExecutor();
    executor.execute(new Runnable() {
      public void run() {
        final MyResult result = longRunningRequestToProduceResult();
        setResult(req, result);
        req.resume();
        executor.shutdown();
      }
    });
  }
  private void respond(HttpServletRequest req, HttpServletResponse resp) throws
   IOException {
    final MyResult result = getResult(req);
    resp.setContentType("text/html");
    final PrintWriter pw = resp.getWriter();
    pw.println("<html><body>The result is " + result.getResult() + "</body></html>");
    pw.close();
  }
}
```

References: [16]

summary

- we have covered
  - web applications and Java EE
    - web applications, Java EE application servers, processing paradigms, important APIs
  - availability and scalability of web applications
    - definitions, horizontal and vertical scalability, load balancers, clustering, availability through redundancy, using HttpSession
  - browser-server interaction beyond simple request-response
    - AJAX and XMLHttpRequest, JSON, Comet, Bayeux, asynchronous servlet request processing
- for more detail see the references and bibliography!
  - or attend Web Engineering II

Selected Topics in Java Web Application Development

Dr Gerald Loeffler, MBA

Web Engineering I
WU Wien