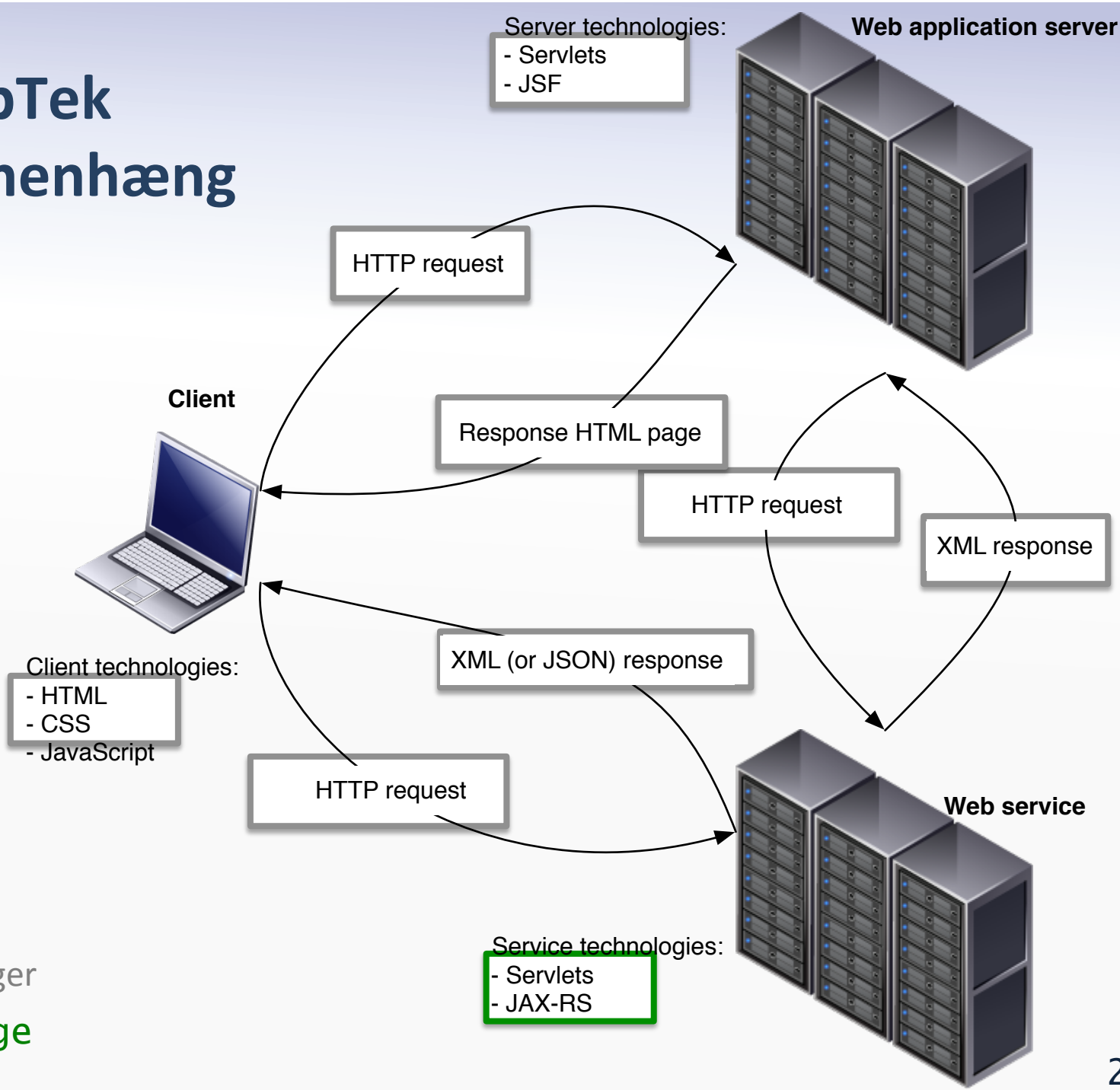
REST Web services with JAX RS

dWebTek sammenhæng



Tidligere uger

Denne uge

What is a web service?

Web service:
"software that makes services available on a network using technologies such as XML and HTTP"

(many alternative definitions exist...)

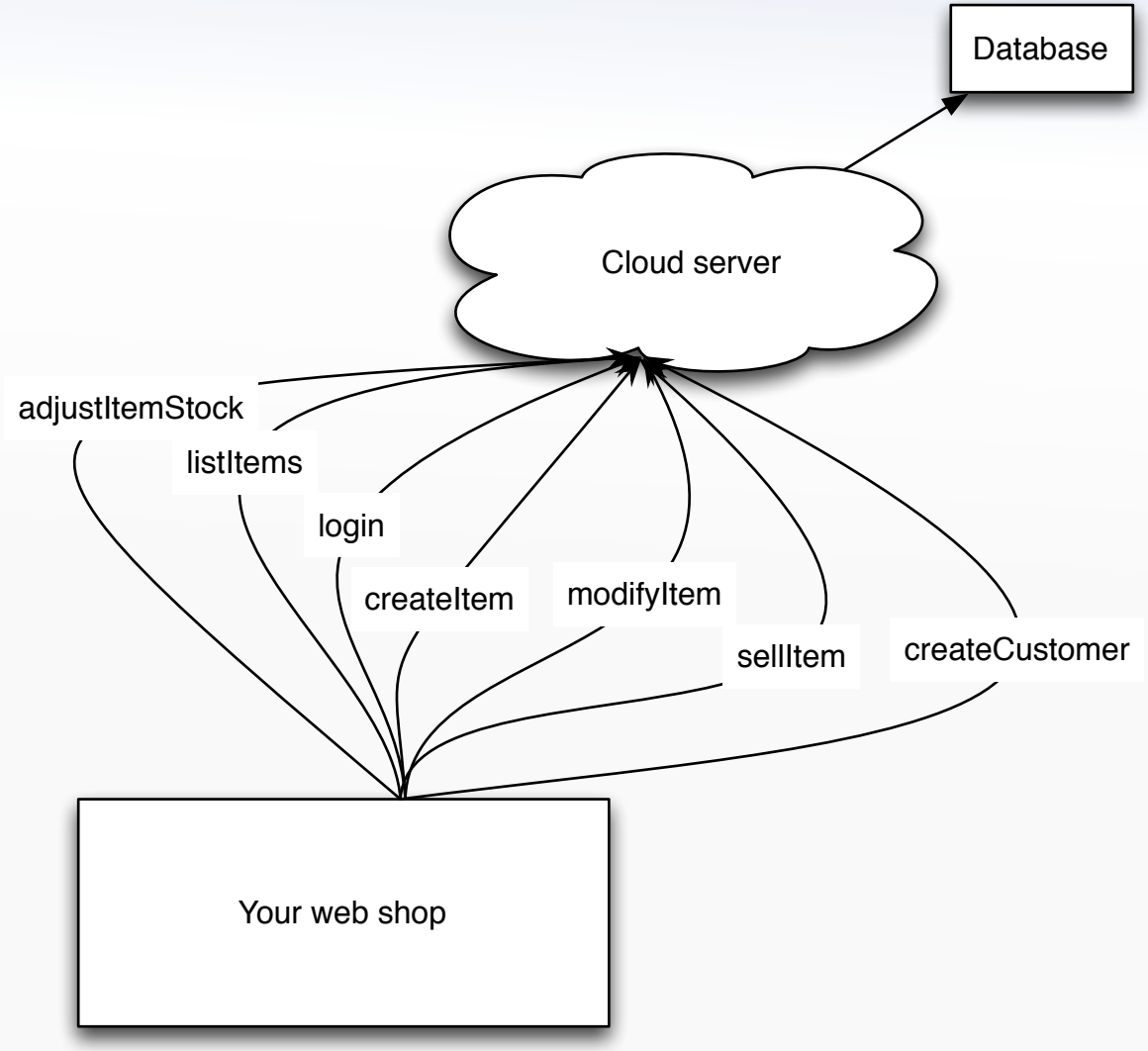
Service oriented architecture (SOA)

Service-Oriented Architecture (SOA):
"development of applications from distributed collections of smaller loosely coupled service providers"

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- Responses
- Data storage
- URL mapping
- Example: Business cards

The cloud server is a web service!



Hello World

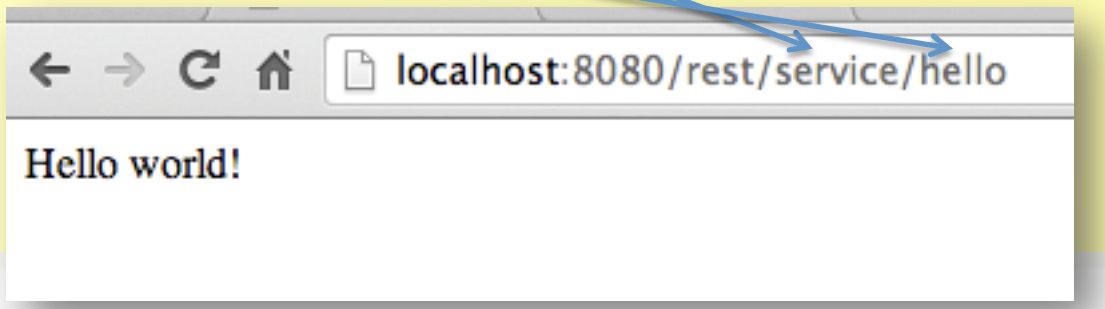
HelloResource.java:

```
package dk.cs.dwebtek;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("service")
public class HelloResource {

    @GET
    @Path("hello")
    public String hello() {
        return "Hello world!";
    }
}
```



REST

Representational **S**tate **T**ransfer

- The *architectural style* used in HTTP
  - Inspired by, but not limited to HTTP!
  - Simple client-server model (no intermediaries)

- Centered around resources:
  - A resources is identified by its URL (one URL for one resource)
  - Resources are the 'objects' or **nouns** of REST

- Requires communication to be stateless

REST APIs

- Manipulation of resources happens through representations of the resource:
  - Resources are typically represented as XML
- Four universal **verbs**. For each URL:
  - GET: Returns a representation of the resource (never changes it)
    - GET must be **idempotent** and the result **cacheable**
  - POST: Updates the resource
  - PUT: Adds a resource
  - DELETE: Deletes a resource
- Response indicates the outcome
  - Might use HTTP response codes

REST URLs

- As we saw, URLs are central to REST APIs
- REST encourages readable URLs
  - /news?showID=1337
    -> /news/Mathias_Schwarz_Wins_Nobel_Prize
- Contrary to common belief pretty URLs are not central to REST

JAX RS

- Framework for implementing REST web services
- Builds on top of the Servlet framework
  - Easier dispatching (getting from a URL to some code)
  - Less code to handle requests
    - One URL per resource. REST!
    - Highly dependent on Java reflection
  - Support for custom datatypes
- Like Servlets, not intended for HTML generation!
  - Combines well with JSF and Servlets

Overview

- REST basics
- JAX RS
- Example: Hello World
- **Requests**
- Responses
- Data storage
- URL mapping
- Example: Business cards

Requests

- Request data is indirectly represented
  - Data is *injected* as method parameters
  - Annotations map HTTP parameters to method parameters
- Underlying framework is still Servlets
  - We access the Servlet framework through injection (later)

Request types

- All HTTP request types are supported:

```
@Path("service")
public class SayHelloResource {

    @GET
    @Path("hello")
    public String hello() {
        return "Hello world!";
    }

    @DELETE
    @Path("hello")
    public String test() {
        return "Delete world!";
    }
...
}
```

Parameters 1/2

- `@QueryParam:` parameter on the form: `url?p=foo`
- `@PathParam:` parameter that is part of a URL: `/url/`*`foo`*`/`

```
@GET
@Path("user/{user}")
public String getUser(@PathParam("user") String user) {
        ....
}
```

- `@FormParam:` parameter that originates from a HTML form
  - Stored in request body
- `@HeaderParam`: The value of a header
  - HTTP headers
- `@CookieParam`: The value of a cookie
- `@BeanParam`: Inject values into bean (similar to JSF)

Parameters 2/2

- Parameters can be of type
  - String
  - Primitive types: int, boolean, ...
  - Types that declare a contructor with a single String argument
  - Types that declare a static `valueOf(String s)` method
  - java.util.List/java.util.Set of the above
  - ...

- We can extend the list of supported types through `MessageBodyReader`

MessageBodyReader

```
@Provider
public class JDOMReader implements MessageBodyReader<Document> {
    @Override
    public boolean isReadable(Class<?> type,
                              Type genericType,
                              Annotation[] annotations,
                              MediaType mediaType) {
        return type == Document.class;
    }

    @Override
    public Document readFrom(Class<Document> type,
                             Type genericType,
                             Annotation[] annotations,
                             MediaType mediaType,
                             MultivaluedMap<String, String> httpHeaders,
                             InputStream entityStream)
            throws IOException, WebApplicationException {
        try {
            SAXBuilder saxBuilder = new SAXBuilder();
            return saxBuilder.build(entityStream);
        } catch (JDOMException e) {
            throw new BadRequestException(e.getMessage());
        }
    }
}
```

Type to convert to

Request body

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- **Responses**
- Data storage
- URL mapping
- Example: Business cards

Responses

- Multiple ways to generate responses:
  1. An easy, direct way: return the value from the method:

```
@GET
@Path("hello")
public String hello() {
    return "Hello world!";
}
```

     - Extensible through message body writers
  2. Response object: fine-grained control over:
     - Headers
     - HTTP status code
     - etc.

Response objects and builders

- Objects of type Response allow fine-grained control:
  - HTTP status code
  - MIME types
  - Headers
  - ...

```
@GET
@Path("getHello")
public Response getHelloResponse() {
    String message = (String) servletContext.getAttribute("message");
    Response.ResponseBuilder builder = Response.status(200);
    builder.header("Cache-Control", "max-age=3600");
    builder.entity(message);
    builder.type("text/plain");
    return builder.build();

}
```

MessageBodyWriter (1/2)

- Support for writing Java objects to responses.

```
@Produces("text/xml")
@Provider
public class JDOMWriter implements MessageBodyWriter<Document> {
    @Override
    public boolean isWriteable(Class<?> type,
                               Type genericType,
                               Annotation[] annotations,
                               MediaType mediaType) {
        return type == Document.class;
    }

    @Override
    public long getSize(Document document,
                        Class<?> type,
                        Type genericType,
                        Annotation[] annotations,
                        MediaType mediaType) {
        return -1; //Not used by JAX-RS 0
    }
```

Not used, but must be implemented :-/

MessageBodyWriter (2/2)

```
    @Override
    public void writeTo(Document document,
                        Class<?> type,
                        Type genericType,
                        Annotation[] annotations,
                        MediaType mediaType,
                        MultivaluedMap<String, Object> httpHeaders,
                        OutputStream entityStream)
            throws IOException, WebApplicationException {
        XMLOutputter outputter = new XMLOutputter();
        outputter.output(document, entityStream);
    }
}
```

- You can do this for any Java type

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- Responses
- **Data storage**
- URL mapping
- Example: Business cards

Data storage

- Servlet framework for access to data storage scopes:
  - Shared state: ServletContext
  - Session state: HttpSession
  - Transient state: local variables and HttpServletRequest
- Objects are injected as method parameters:
  - Can also inject in constructor!

Data storage through injection 1/2

- Access to server data is done through **injection:**
  - Declare and `@Context` annotate fields, method parameters
  - JAX RS **injects** objects into these fields and parameters!

```
@Path("message_service")
public class MessageResource {
   @Context ServletContext servletContext;

   @POST
   @Path("setHello")
   public String setHello(@QueryParam("message") String message) {
      servletContext.setAttribute("message", message);
      return "Message set!";
   }

   @GET
   @Path("getHello")
   public String getHello() {
      return (String) servletContext.getAttribute("message");
   }
}
```

Data storage through injection 2/2

- All servlet lifecycle objects can be injected
  - HttpSession, HttpServletRequest, HttpServletResponse, ...
  - One resource object per request, so no race conditions

```
@Path("message_service")
public class MessageResource {
   HttpSession session;

   public ShopService(@Context HttpServletRequest servletRequest) {
      session = servletRequest.getSession();
   }

   @POST @Path("setHello")
   public String setHello(@QueryParam("message") String message) {
      session.setAttribute("message", message);
      return "Message set!";
   }

   @GET @Path("getHello")
   public String getHello() {
      return (String) session.getAttribute("message");
   }
}
```

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- Responses
- Data storage
- **URL mapping**
- Example: Business cards

URL mapping

- Like in JSF an Servlets, we must map URLs to code:
  - @Path annotation on class and method controls URLs
  - @GET, @POST etc. determine supported request methods
- URLs and parameters can overlap in JAX-RS:
  - **@Path("{id}")** matches *anything* up to a /
  - **@PathParam** can accept the value of *id* as parameter
- No extra wiring in web.xml needed

JAX RS resources

- A resource is a collection of methods (a class)

```
@Path("service")
public class SayHelloResource {

   @GET
   @Path("hello/{name}")
   public String hello(
      @PathParam("name") String name) {
      return "Hello "+ name + "!";
   }

   @GET
   @Path("goodbye")
   public Object test() {
      return "Goodbye world!";
   }
...
}
```

- Default: new instance per request
  - *Unlike* servlets
- Paths are (relative) URLs of methods
  - Cannot overlap
- URLs become:
  web.xmlURL/*resourceURL*/**methodURL**

Consumes and produces

- Two annotations can limit the scope of methods:
  - `@Consumes`: Content type of input
  - `@Produces`: Content type of output

- Method will only be used if the `@Produces` annotation matches the type that the client requests

```
@GET
@Path("hello")
@Produces("text/plain")
public String hello() {
    return "Hello world!";
}
```

Web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">
    <display-name>Restful Web Application</display-name>

    <servlet>
        <servlet-name>jersey-servlet</servlet-name>
       <servlet-class>
           org.glassfish.jersey.servlet.ServletContainer
       </servlet-class>
       <init-param>
             <param-name>jersey.config.server.provider.packages</param-name>
             <param-value>dk.cs.dwebtek</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>jersey-servlet</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Clicker question

- How are parameters received in JAX RS?
  - As method parameters 
  - Through the JAXRequest object
  - Through Java beans

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- Responses
- Data storage
- URL mapping
- **Example: Business cards**

Business Cards

```
<cardlist xmlns="http://businesscard.org"
          xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <title>
    <xhtml:h1>My Collection of Business Cards</xhtml:h1>
    containing people from <xhtml:em>Widget Inc.</xhtml:em>
  </title>
  <card>
    <name>John Doe</name>
    <title>CEO, Widget Inc.</title>
    <email>john.doe@widget.com</email>
    <phone>(202) 555-1414</phone>
  </card>
  <card>
    <name>Joe Smith</name>
    <title>Assistant</title>
    <email>thrall@widget.com</email>
  </card>
</cardlist>
```

Business card service 1/2

Store, list, and retrieve business cards!

```
@Path("businesscards")
public class BusinessCardService {
    private @Context ServletContext sc;
    private Namespace ns =
                Namespace.getNamespace("http://businesscard.org");

    @GET
    @Produces("text/xml")
    @Path("card")
    public Document getBusinessCard(String name) {
        return getDocumentMap().get(name);
    }

    @POST
    @Path("card")
    public void storeBusinessCard(Document d) {
        String name = d.getRootElement().getChildText("name", ns);
        getDocumentMap().put(name, d);
    }
```

Business card service 2/2

```
    @GET
    @Produces("text/xml")
    @Path("cards")
    public Document getAllCards() {
        Map<String, Document> map = getDocumentMap();
        Document d = new Document();
        Element cardlist = new Element("cardlist", ns);
        d.setRootElement(cardlist);
        for (Document card : map.values()) {
            cardlist.addContent(card.getRootElement().clone());
        }
        return d;
    }

    private Map<String, Document> getDocumentMap() {
        Map<String, Document> map =
                (Map<String, Document>) sc.getAttribute("document_map");
        if (map == null) {
            map = new HashMap<>();
            sc.setAttribute("document_map", map);
        }
        return map;
    }
}
```

Clicker spørgsmål

- Which is not a feature of JAX RS?
  - Dispatching
  - HTML templates 
  - Parameter parsing
  - REST style web services

Overview

- REST basics
- JAX RS
- Example: Hello World
- Requests
- Responses
- Data storage
- URL mapping
- Example: Business cards

Online resources

- JAX RS tutorial: http://docs.oracle.com/javaee/7/tutorial/doc/jaxrs.htm

- JAX RS Specification: https://jax-rs-spec.java.net/