Web Services

- Používají HTTP
- Existují dvě varianty:
  - Služby postavené na protokolu SOAP
    - Java standard pro vytváření : JAX-WS
  - RESTfull služby
    - Java standard pro vytváření : JAX-RS

Web Services na SOAP

- Žádost i odpověď se posílá v XML
  - Definováno SOAP protokolem, uživatelsky obtížně čitelný
- Služba je popsána souborem WSDL, který definuje všechny publikované metody a jejich parametry a návratové hodnoty
  - Např. v příkladu dostupný na http://localhost:8080/webservices/wsbook?wsdl
- Referenční implementace v Javě: JAX-WS RI od Oracle

Web Services na SOAP
JAX-WS anotace

- Velmi jednoduché vytvořit – stačí několik málo anotací
- Konfigurace může být obtížnější
  - V plném Java EE (Glassfish) prostředí k dispozici zadarmo
  - Jinak (např. Tomcat) potřeba nakonfigurovat
- Pro zveřejnění třídy se službami anotace @javax.jws.WebService
- Metody služeb anotovat @javax.jws.WebMethod
- Jako parametry i návratové hodnoty lze bez problémů používat základní typy (array, List...) a beany s gettery a settery

RESTfull

- Po HTTP se může posílat XML, HTML, JSON, atp., ale to obsahuje pouze vlastní data
- Požadavek, co provést a s čím, je obsažen pouze v URL a metodě HTTP:
  - S čím – je dané URL, které by mělo jednoznačně reprezentovat nějaký objekt aplikace, např. http://localhost:8080/webservices/jersey/books/16 identifikuje knihu s ID 16
  - Co provést – je dané metodou HTTP. Určitě znáte GET a POST, zde použité pro získání objektu a pro uložení nového objektu. Dále existuje PUT a DELETE, zde pro update objektu a pros smazání objektu.
- Referenční implementace v Javě: Jersey od Oracle

RESTfull – JAX-RS anotace

- Anotace třídy @javax.ws.rs.Path(“/cesta”), pak služba poslouchá na cestě /aplikace/cesta
- Metody se také mohou anotovat @Path(“dalsicesta”), pak je tato metoda dostupná na /aplikace/cesta/dalsicesta
- U metody mohou být v @Path proměnné, např. “/cesta/{id}”, pak je id dostupné v parametru metody pomocí @PathParam("id") long id
- U publikované metody musí být jedno z @GET, @POST, @PUT, @DELETE pro výběr, na kterou HTTP akci má metoda reagovat
- U metody můžou být anotace @Produces a @Consumes podle toho, jaká data přijímají/odesílají v tělě HTTP (typicky vlastní data objektu) @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
  - Opět můžete přijímat/vracet s beanami, tímto řeknete, jaká serializace je podporována
  - Je třeba ještě beany příslušně anotovat pro externí knihovnu zajišťující patřičnou serializaci (viz příklad)
- Pokud použijete @Consumes(MediaType.APPLICATION_FORM_URLENCODED), lze číst parametry odeslaného formuláře (HTML form, který posílá parametry URL encoded) pomocí anotace @FormParam("form_param") na parametru metody
- Můžete také číst parametry v URL (za ?, vhodné pro nějaké vedlejší upřesnění požadavku) pomocí anotace @QueryParam("url_param")

RESTfull pomocí Spring MVC

- RESTfull lze implementovat i ve Spring MVC, které ale nevyužívá standard JAX-RS
- Anotace s podobnými funkcemi:
  - @RequestMapping(value=..., method=..., consumes=..., produces=...)
  - @ResponseBody
  - @RequestParam, @PathVariable

Klienti

- Programoví klienti
  - SOAP – lze vytvořit např. pomocí NetBeans odkazem na WSDL
    - Z něj se replikuje model a vznikne třída, která služby zprostředkuje
  - RESTfull – opět pomocí NetBeans
    - Implementováno Jersey klientem
    - Vznikne třída, která služby zprostředkuje
    - Obsaženo v příkladu
- Javascript klienti
  - RESTfull služby lze volat z webové stránky Javascriptem prostřednictvím AJAX, jednoduše např. pomocí jQuery
  - V příkladu v index.jsp

Příklad na Web Services

- Na stránkách cvičení wpa-webservices-v2.zip (doplněná verze)
- Obsahuje serverovou část na SOAPové a RESTfull (různé implementace) služby a klienta na RESTfull
- Data pro ukázku pouze statická v paměti
- Springová service BookService pro základní operace, klasicky s interface, přes který je využívána z jednotlivých webových služeb
- Pro implementaci služeb vizte příklad, dále zmíníme potřebnou konfiguraci

Příklad na Web Services – JAX-WS

- Je třeba zaregistrovat obslužný Servlet s podporou Springu, tedy ve web.xml např.:

```
<servlet>
     <servlet-name>WsBookService</servlet-name>
     <servlet-class>com.sun.xml.ws.transport.http.servlet.WSSpringServlet</servlet-class>
     <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
     <servlet-name>WsBookService</servlet-name>
     <url-pattern>/wsbook</url-pattern>
 </servlet-mapping>
```

- A poté pomocí Springu nakonfigurovat službu (pro bean @Component("wsBookService") @WebService public class WsBookService):

```
<wss:binding url="/wsbook">
    <wss:service>
        <ws:service bean="#wsBookService"/>
    </wss:service>
</wss:binding>
```

Příklad na Web Services – JAX-RS

- Je opět třeba zaregistrovat obslužný Servlet s podporou Springu, tedy ve web.xml např.:

```
<servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.spring.container.servlet.SpringServlet</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>cz.cvut.kbss.wpa.webservices.jersey</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/jersey/*</url-pattern>
</servlet-mapping>
```

- A poté pomocí Springu nakonfigurovat podporu pro serializaci:

```
<mvc:annotation-driven>
    <mvc:message-converters>
        <bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"/>
        <bean class="org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter"/>
    </mvc:message-converters>
</mvc:annotation-driven>

<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
    <property name="defaultContentType" value="application/json"/>
    <property name="mediaTypes">
        <map>
            <entry key="html" value="text/html"/>
            <entry key="xml" value="application/xml"/>
            <entry key="json" value="application/json"/>
        </map>
    </property>
</bean>
```

Příklad na Web Services – Spring REST

- Je opět třeba zaregistrovat obslužný Servlet, tentokrát se jedná přímo o Springový, ve web.xml tedy např.:

```
<servlet>
    <servlet-name>spring-rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/context/restContext.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring-rest</servlet-name>
    <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Příklad na Web Services – závislosti

- Pro RESTfull se Springem – Spring MVC
  - `org.springframework:spring-web`
  - `org.springframework:spring-webmvc`
- Pro podporu RESTfull serializace do JSON
  - `org.codehaus.jackson:jackson-core-asl`
  - `org.codehaus.jackson:jackson-mapper-asl`
- Pro Jersey RESTfull
  - `com.sun.jersey:jersey-server`
  - `com.sun.jersey:jersey-servlet`
  - `com.sun.jersey:jersey-json`
  - `com.sun.jersey.contribs:jersey-spring`
- Pro JAX-WS s JAX-WS RI
  - `com.sun.xml.ws:jaxws-rt`
  - `org.jvnet.jax-ws-commons.spring:jaxws-spring`
  - `org.apache.xbean:xbean-spring`
- Pro RESTfull klienta s Jersey
  - `com.sun.jersey:jersey-client`
  - `com.sun.jersey.contribs:jersey-multipart`
- Vše jsou Maven artifacts, pro verze atp. vizte příklad