ЛАБОРАТОРНАЯ РАБОТА **№4**

**Создание Web-сервисов**

**Цель работы.** Познакомиться с технологией web-сервисов.

**Краткое теоретическое содержание**. В теоретическом материале курса представлены сведения по работе web-сервисами. Ознакомьтесь с теоретическим материалом.

Создадим web-сервис для скачивания файлов (file-upload). Прежде всего, создаем web-проект (имя WebSErviceLab), структура которого показана на рисунке 1

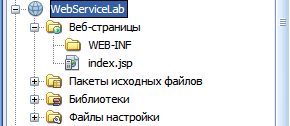


Рисунок 1. Структура проекта

Данный проект пуст. В него следует добавить web-службу. Для этого активизируем контекстное меню на имени проекта и выбираем п. Создать – Другое – Веб-Службы (рисунок 2):

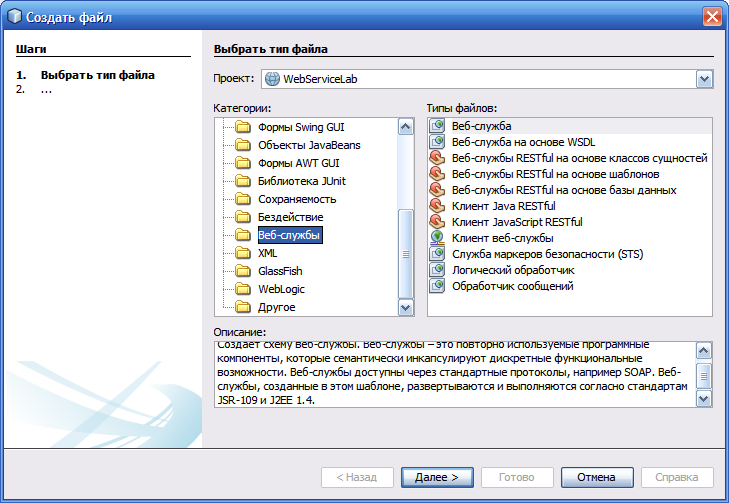


Рисунок 2. Добавление web-службы

Выбираем тип файла Веб-служба и жмем Далее. Вводим имя службы и имя пакета (рисунок 3)

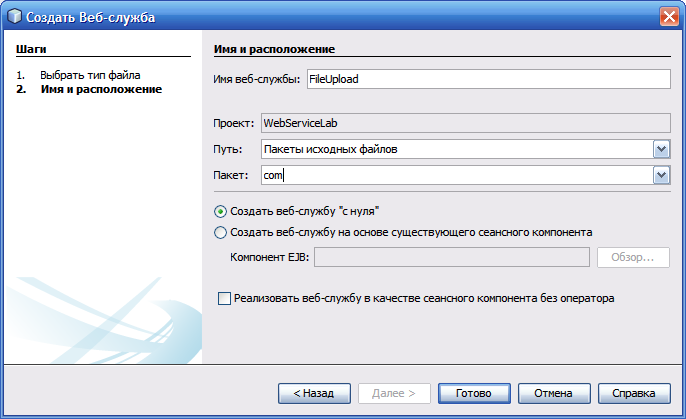


Рисунок 3. Задание имени службы и имени пакета

Жмем Готово. Система создает следующую заготовку сервиса:

package com;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "FileUpload")

public class FileUpload {

@WebMethod(operationName = "hello")

public String hello(@WebParam(name = "name") String txt) {

return "Hello " + txt + " !";

}

}

Для начала добавим в службу свой собственный метод. Щелкаем правой кнопкой мыши в окне кода и выбираем п. Вставка кода из контекстного меню (рисунок 4).

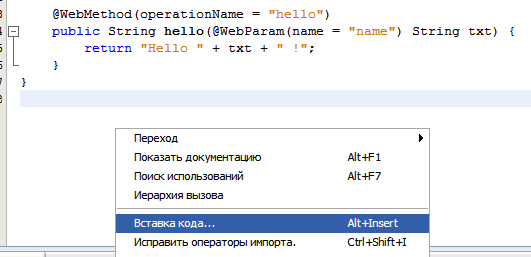


Рисунок 4. Добавление метода в web-службу

На экране появится следующее окно (рисунок 5):

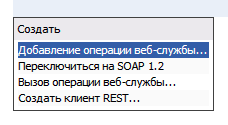


Рисунок 5. Контекстное меню для добавления метода

Выбираем п. Добавление операции веб-службы (рисунок 6).

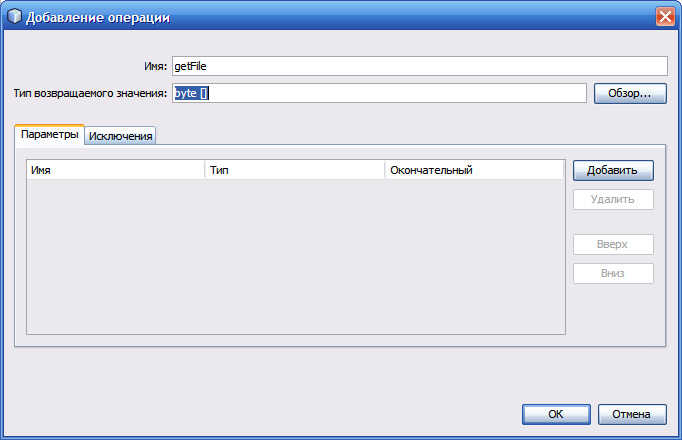


Рисунок 6. Определение имени и параметров метода

Задаем имя метода getFile и тип возвращаемого значения – массив байтов. В результате код в окне редактора принял такой вид

package com;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "FileUpload")

public class FileUpload {

@WebMethod(operationName = "hello")

public String hello(@WebParam(name = "name") String txt) {

return "Hello " + txt + " !";

}

@WebMethod(operationName = "getFile")

public byte [] getFile() {

//TODO write your implementation code here:

return null;

}

}

Наша задача – реализовать метод getFile. Мы намереваемся пока просто передать текстовый файл, конвертированный в массив байтов, клиенту. Причем отсылаем массив байтов, в который этот файл преобразуем. Текстовый файл заранее определен. Код нашего метода примет такой вид

package com;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.nio.file.Path;

@WebService(serviceName = "FileUpload")

public class FileUpload {

@WebMethod(operationName = "hello")

public String hello(@WebParam(name = "name") String txt) {

return "Hello " + txt + " !";

}

@WebMethod(operationName = "getFile")

public byte [] getFile() {

byte [] buf=null;

java.awt.FileDialog fd=null;

java.awt.Dialog di=null;

try

{

String s1="c:/Users/OLEG/work/my.txt";

Path path = Paths.get(s1);

buf = Files.readAllBytes(path);

}

catch(Exception ex)

{

}

return buf;

}}

Выполним сервис и разместим его (deploy) используя опции меню. Сначала из контекстного меню выберем Очистить и построить, затем – Развернуть.

Серверная часть готова.

Создаем клиента как обычное Java приложение (Java Application с именем WebServiceClient). Вставляем в проект web-клиента (приводимая ниже последовательность скриншотов показывает, как это делается – рисунки 7-9).

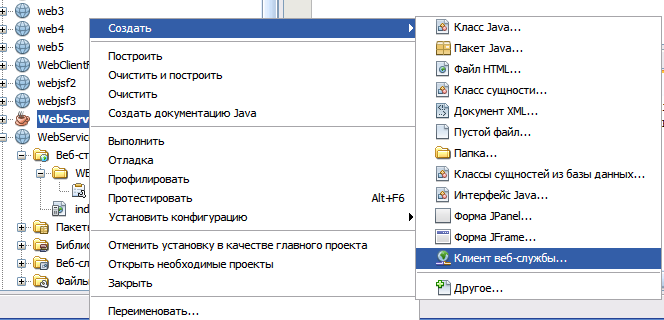


Рисунок 7. Создание клиента web-службы

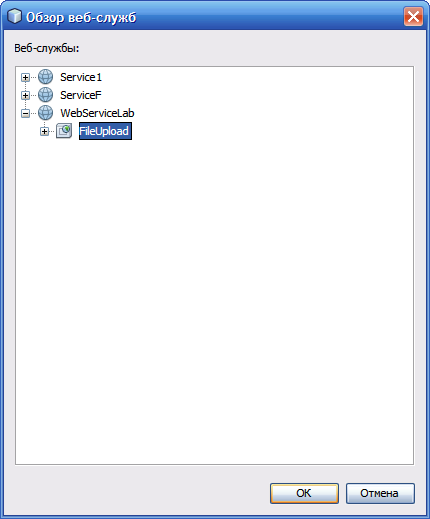


Рисунок 8. Добавление класса web-сервиса

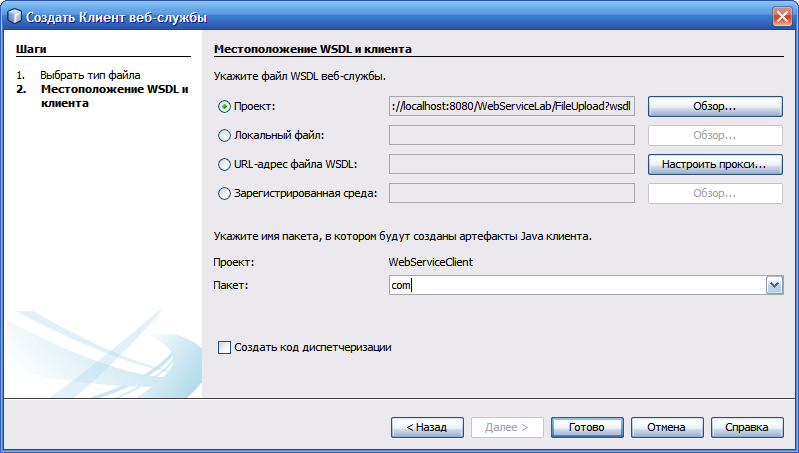


Рисунок 9. Ссылка на проект web-сервиса

Структура проекта (с серверной и клиентской частями) показана на рисунке 10:

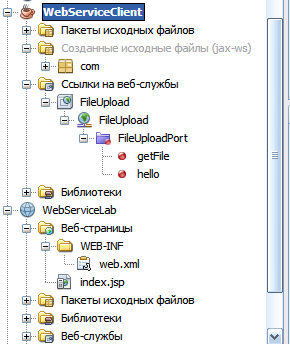


Рисунок 10. Структура проекта

Программируем клиентскую часть таким образом

package webserviceclient;

import com.FileUpload;

import com.FileUpload\_Service;

public class WebServiceClient {

public static void main(String[] args) {

FileUpload\_Service service =

new FileUpload\_Service(); // создаем объект сервиса

System.out.println(service.getFileUploadPort().hello

("OLEG GERMAN")); // сервис имеет порты, вызывает метод hello, рез-т возвращается клиенту (Олег герман)

try

{

byte [] fromserv

= service.getFileUploadPort().getFile();

if((fromserv==null)||(fromserv.length<=0))

{

System.out.println("File was not sent");

}

else

{

String str = new String(fromserv); // массив байтов преобразуем в string

System.out.println(str);

}

}

catch(Exception ex)

{

System.out.println

("Cannot get file from service:"+ex.getMessage());

}

}

}

Файл считывается в следующем фрагменте

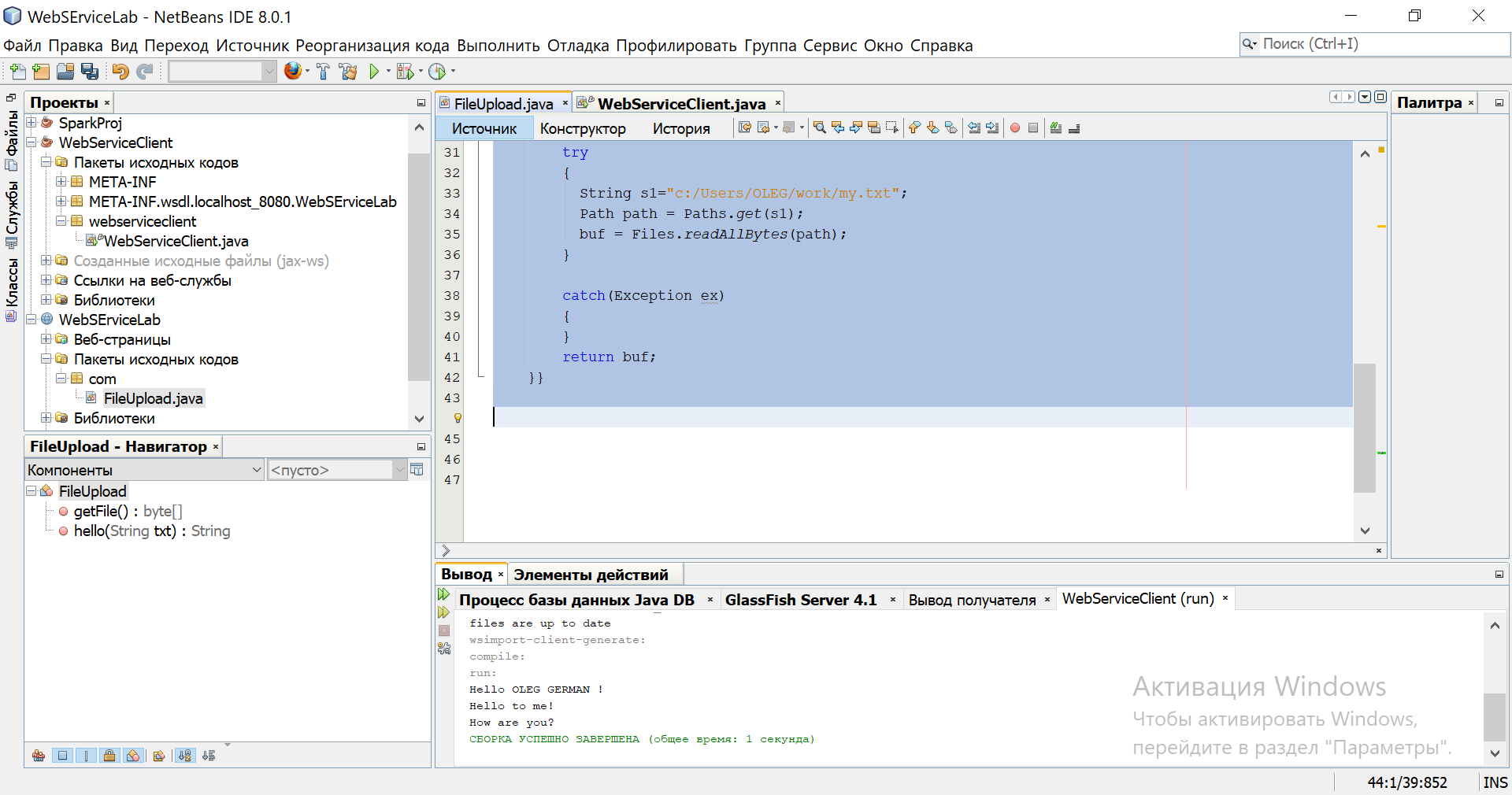
byte [] fromserv

= service.getFileUploadPort().getFile();

Результат работы показан ниже

Hello OLEG GERMAN ! // результат выводится в консоль

ПОСТРОЕНИЕ УСПЕШНО ЗАВЕРШЕНО (общее время: 14 секунд)



Также рассмотрим создание web-сервиса в IntellJIdea. Сначала создайте новый проект, выберите проект WebService и выберите Apache Axis:

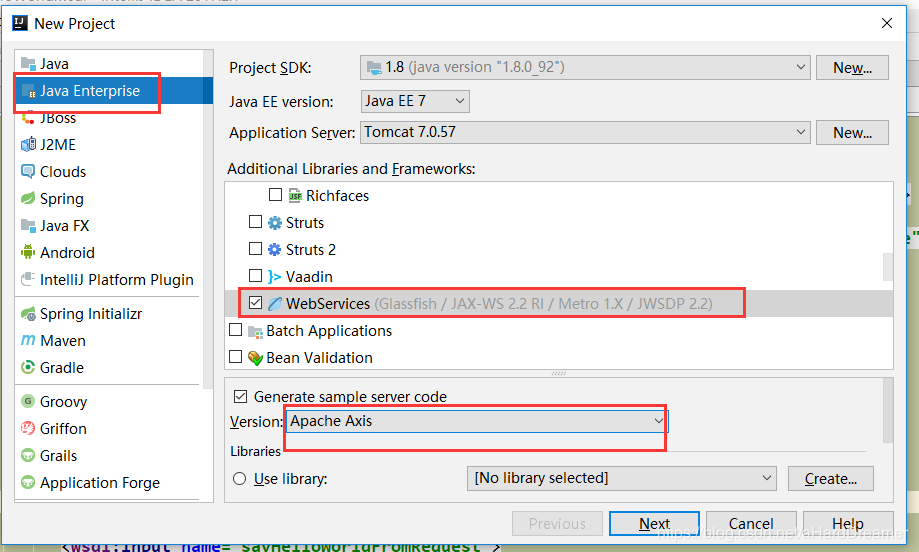


Рисунок 11. Создание web-сервиса в IntellJIdea.

Нажмите «Далее» и перейдите к имени проекта, например, веб-сервис.

После завершения будут некоторые образцы файлов, по умолчанию есть класс HelloWorld, и в этом классе есть метод sayHelloWorldFrom (String from)- рисунок 12.



Рисунок 12. Класс сервиса

Это очень простой Java-метод, давайте рассмотрим структуру и использование WebService с помощью этого простого Java-метода.

Щелкните правой кнопкой мыши этот класс -> WebService, чтобы сгенерировать файл wsdl (рисунок 13):

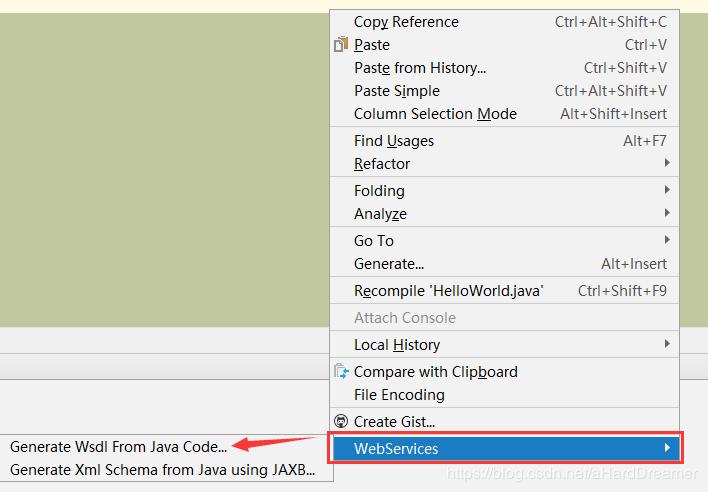


Рисунок 13. Класс сервиса с wsdl

После того, как он будет создан, в том же пакете, что и HelloWorld.java, будет файл .wsdl, откройте этот файл.

Этот WSDL состоит из пяти частей: службы, привязки, типа порта, сообщения и типов. Служба в основном описывает путь к интерфейсу службы, который эквивалентен API, определенному в REST:

<wsdl:service name="HelloWorldService">  
  
   <wsdl:port binding="impl:HelloWorldSoapBinding" name="HelloWorld">  
  
      <wsdlsoap:address location="http://localhost:8080/services/HelloWorld"/>  
  
   </wsdl:port>  
  
</wsdl:service>

Привязка связывает запрос и ответ с SoapAction вместе:

<wsdl:binding name="HelloWorldSoapBinding" type="impl:HelloWorld">

   <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

   <wsdl:operation name="sayHelloWorldFrom">

      <wsdlsoap:operation soapAction="test"/>

      <wsdl:input name="sayHelloWorldFromRequest">

         <wsdlsoap:body use="literal"/>

      </wsdl:input>

      <wsdl:output name="sayHelloWorldFromResponse">

         <wsdlsoap:body use="literal"/>

      </wsdl:output>

   </wsdl:operation>

 Тип терминала описывает вход (сообщение запроса) и выход (сообщение ответа) и параметры этого интерфейса (метод).

<wsdl:portType name="HelloWorld">

   <wsdl:operation name="sayHelloWorldFrom" parameterOrder="from">

      <wsdl:input message="impl:sayHelloWorldFromRequest" name="sayHelloWorldFromRequest"/>

      <wsdl:output message="impl:sayHelloWorldFromResponse" name="sayHelloWorldFromResponse"/>

   </wsdl:operation>

</wsdl:portType>

Сообщение определяет параметры запроса Request и тип ответа Response.

<wsdl:message name="sayHelloWorldFromResponse">

   <wsdl:part element="impl:sayHelloWorldFromReturn" name="sayHelloWorldFromReturn"/>

</wsdl:message>

<wsdl:message name="sayHelloWorldFromRequest">

   <wsdl:part element="impl:from" name="from"/>

</wsdl:message>

 Типы в основном перечисляют типы запросов и ответов, вплоть до основных типов данных.

<wsdl:types>

 <schema elementFormDefault="qualified" targetNamespace="http://example" xmlns="http://www.w3.org/2001/XMLSchema">

  <element name="from" type="xsd:string"/>

  <element name="sayHelloWorldFromReturn" type="xsd:string"/>

 </schema>

</wsdl:types>

Теперь настройте Tomcat и разверните проект на Tomcat. Обратите внимание, что вручную добавьте эту зависимость в проект (рисунок 14), иначе при доступе будет сообщение об ошибке.

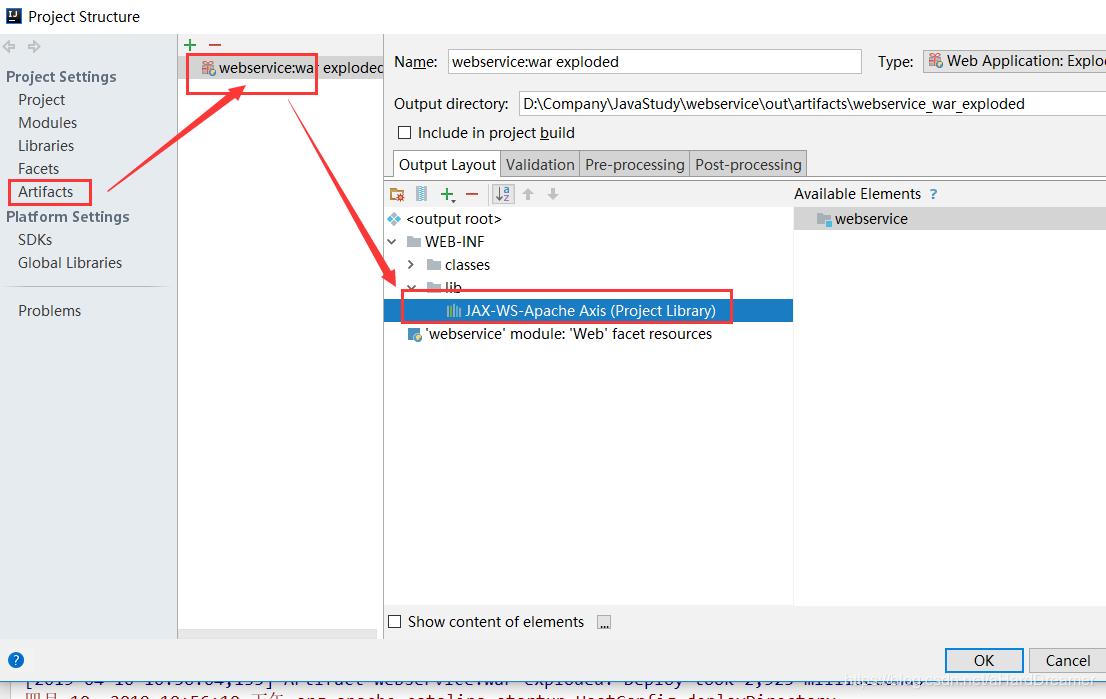


Рисунок 14. Добавление зависимости (артефакта)

Посетите wsdl HelloWorld в браузере:

<http://localhost:8080/services/HelloWorld?wsdl>

### Задание (общее)

Приведенное приложение следует усложнить. Клиенту web-сервис передает список всех текстовых файлов (для простоты – txt), находящихся в электронной библиотеке. Этот список раскрывается в окне клиента. Клиент должен произвести выбор книги, а затем повторно обратиться к сервису за самой книгой. Ясно, что приложение клиента следует выполнить как оконное.

***Порядок выполнения работы***

1. Изучить теоретическую часть.
2. Получить индивидуальное задание у преподавателя из прилагаемого списка.
3. Создать и отладить приложение .
4. Написать отчет.
5. Защитить работу.

***Индивидуальные задания***

**Вариант 1.**

Напишите web-сервис, который возвращает результат вычисления простых бесскобочных арифметических операций, например. 2+4\*7.

**Вариант 2.**

Напишите web-сервис, который возвращает результат перевода с русского языка на английский или наоборот

**Вариант 3.**

Напишите web-сервис, который возвращает файл с музыкой. На стороне клиента обеспечьте проигрывание файла на медиаплейере.

**Вариант 4.**

Напишите web-сервис, который возвращает файл Html. На стороне клиента обеспечьте открытие этого файла в браузере.

**Вариант 5.**

Напишите web-сервис, который возвращает результат поиска в фале xml. Например, пусть это будет котировка валюты. Котировки записаны в тегах xml-документа.

**Контрольные вопросы**.

1. Что такое управляемый web-сервис, как он связывается с клиентом?
2. Расскажите, как создать web-сервис.
3. Расскажите, как создать службу клиента.