**Лекция 11**

**Первый Web-сервис**

**Установка необходимых приложений**

Для первого примера мы воспользуемся кодом, поставляемым в составе пакета *The Java Web Services Tutorial* (он может быть загружен с сайта фирмы Sun ). Пакет представляет собой архив, содержащий развернутую документацию по разработке Web -сервисов и несколько примеров. Вот одним из этих примеров, называющимся helloservice, мы и воспользуемся в качестве иллюстрации.

Этот пример представляет собой реализацию простейшего Web -сервиса, который содержит единственный метод, принимающий и возвращающий значение типа **String**.Пример располагается в папке *<tutorial.home> /examples/jaxws/helloservice, где <tutorial.home>* - директория, в которую был установлен пакет The Java Web Services Tutorial.

Прежде чем компилировать, устанавливать и выполнять пример, необходимо настроить некоторые параметры, которые определены в файле **build.properties**,лежащем в *<tutorial.home>/examples/common*.Определить нужно следующие параметры:

* **javaee.home** - параметр, указывающий на корень установки сервера приложений;
* **javaee.tutorial.home** - параметр, указывающий на корень установки пакета The Java Web Services Tutorial.

В случае если установка сервера проводилась по умолчанию, никаких других изменений в этом файле делать не нужно. В случае если Sun Java System Application Server установлен в директории *H:/Java/AppServer/*,а *The Java Web Services Tutorial* - в директории *H:/Java/wstutorial20\_new/*,файл **build.properties** будет иметь следующий вид ([пример 10.1](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=3)):

javaee.home=H:/Java/AppServer/

]avaee.tutorial.home= H:/Java/jwstutorial20\_new/

admin.password.file=${javaee.tutorial.home}/exаmples/common/admin-

password.txt

admin.host=localhost

admin.user=admin

admin.port=4848

https.port=8181

domain.resources="domain.resources"

domain.resources.port=8080

# Database properties are in derby.properties

db.vendor=derby

Листинг 10.1. Файл build.properties

Для того чтобы иметь возможность инсталлировать разработанный Web -сервис в сервер приложений, необходимо также в файле **admin-password.txt** (находящемся в *<tutorial.home>/examples/common/)* указать пароль для учетной записи администратора.

**Исходный код Web-сервиса**

После того, как указанные настройки будут закончены, можно приступить непосредственно к примеру.

Файл с исходным кодом Web -сервиса располагается в директории src, называется **Hello.java** и имеет следующий вид ([пример 10.2](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=3#example.10.2)):

1 // Hello.java

2 package helloservice.endpoint;

3

4 import javax.jws.WebMethod;

5 import javax.jws.WebService;

6

7 @WebService()

8 public class Hello {

9 private String message = new String("Hello, ");

10

11 @WebMethod()

12 public String sayHello(String name) {

13 return message + name + ".";

14 }

15 }

Листинг 10.2. Файл Hello.java

Первое, что обращает на себя внимание, - удивительная краткость написанного кода. Но не стоит обольщаться: дело в том, что технология разработки Web -сервисов, которую мы будем использовать, просто скрывает от разработчика большую часть работы по реализации Web -сервиса. Фактически, все, что должен сделать разработчик, - реализовать код самих вызываемых методов; абсолютно всю работу по реализации механизмов, позволяющих вызывать эти методы удаленно, берет на себя используемая нами технология.

Итак, рассмотрим пример подробнее.

Во-первых, определяется пакет, в который будет помещен класс Hello, пакет называется helloservice.endpoint (строка 2). Затем импортируются две аннотации - **javax.jws.WebMethod** (строка 4) и **javax.jws.WebService** (строка 5). Поскольку механизм аннотаций является в java относительно новым, видимо имеет смысл сказать о нем несколько слов.

Аннотации java предоставляют разработчику механизм, позволяющий включать в код некие *метаданные*.Метаданные - это дополнительная информация, которая может быть добавлена к классам, интерфейсам, полям и методам - так называемые "данные о данных". Метаданные доступны как на этапе компиляции, так и во время исполнения через механизм рефлексии. Вот эти данные, которые мы можем добавить к классам, интерфейсам, полям и методам, и называются аннотациями.

Для аннотации вводится понятие области видимости, или времени жизни. Область видимости выбирается из перечисления **java.lang.annotation. RetentionPolicy** и может принимать следующие значения:

* SOURCE - аннотация будет удалена компилятором;
* CLASS - аннотация будет записана в байт-код класса, но не будет использоваться Java -машиной;
* RUNTIME - аннотация будет записана в байт-код и будет доступна во время исполнения через механизм рефлексии.

Одна из замечательных возможностей, которую открывает перед разработчиками механизм аннотаций, заключается в том, что с их помощью могут быть построены средства автоматического внесения в код нужной функциональности.

В нашем конкретном примере аннотации используются именно с этой целью. Ориентируясь на них, специальная утилита **wsgen** генерирует классы-обертки, выполняющие всю техническую работу.

Первая аннотация, которая используется применительно ко всему классу, - аннотация **WebService** (строка 7). Эта аннотация нужна для того чтобы указать, что данный класс в дальнейшем будет опубликован в качестве Web -сервиса. Есть некоторые ограничения, связанные с применением этой аннотации. Методы, которые предполагается опубликовать, должны быть аннотированы как **javax.jws.WebMethod**,они должны быть объявлены как **static** или **final**.В качестве принимаемых и возвращаемых параметров должны использоваться параметры совместимых с JAX-B типов (все примитивные типы могут использоваться в качестве параметров). Класс, реализующий Web -сервис, не может быть объявлен как **final**,кроме того, он не должен быть абстрактным, должен иметь публичный конструктор по умолчанию и не переопределять метод **finalize**. Все эти ограничения связаны с тем, что разрабатываемый класс будет выполняться не самостоятельно, а будет, фактически, встроен в сервер, следовательно, сервер должен полностью контролировать процесс его создания (загрузки) и уничтожения. В случае если при загрузке и/или выгрузке объекта необходимо предпринимать какие-то дополнительные действия, могут применяться аннотации **javax.annotation.PostConstruct** и **javax.annotation.PreDestroy**.Метод, аннотированный как **PostConstruct**,будет вызван сервером приложений до того, как объекту первый раз будет передан запрос клиента. Метод, аннотированный как **PreDestroy**,вызывается перед уничтожением объекта.

Рассматриваемый класс **Hello** удовлетворяет всем указанным ограничениям. Кроме того, он объявляет единственный метод, аннотированный как **WebMethod** (строка 11), который принимает параметр типа **String** и возвращает его же с присоединенной в начале константной строкой.

Собственно, на этом разработка Web -сервиса заканчивается. Следующее, что необходимо сделать, - откомпилировать его, пропустить через утилиту **wsgen** для генерации вспомогательных классов, создать **war-file**,содержащий в себе откомпилированное приложение и необходимые ресурсы, и затем разместить и зарегистрировать его на сервере приложений.

В комплекте с примерами, поставляемыми в пакете *The Java Web Services Tutorial*,поставляются также скрипты для их компиляции. Эти скрипты предназначены для специального инструментального средства компиляции, которое называется ant3 (исполняющая часть **ant** устанавливается вместе с *Sun Java System Application Server)*.

Разработчики примеров для пакета *The Java Web Services Tutorial* постарались на славу, и теперь для компиляции и установки приложения необходимо выполнить лишь несколько простых команд. Мы воспользуемся этим обстоятельством, а затем подробно рассмотрим, что стоит за каждой из этих простых команд и какие действия при этом выполняются.

**Компиляция и инсталляция на сервере приложений**

Итак, первое, что предстоит сделать, - откомпилировать приложение. Для компиляции в настройках сборки определена специальная цель *(target)* - *build*.

Набрав в командной строке команду **asant build (asant** - вызов командного файла, запускающего **ant, build** - имя цели, которую он должен выполнить), получим следующий вывод:

Buildfile: build.xml

javaee-home-test:

init:

prepare:

[echo] Creating the required directories....

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

jaxws\helloservice\build

compile-service:

[echo] Compiling the server-side source code ...

[javac] Compiling 1 source file to H:\Java\

jwstutorial20\_new\examples\jaxws\helloservice\build

[wsgen] command line: wsimport -classpath

H:\Java\AppServer\lib\activation.jar;

H:\Java\AppServer\lib\admin-cli.jar;

H:\Java\AppServer\lib\appserv-admin.jar;

H:\Java\AppServer\lib\appserv-cmp.jar;

H:\Java\AppServer\lib\appserv-deployment-client.jar;

H:\Java\AppServer\lib\appserv-ext.jar;

H:\Java\AppServer\lib\appserv-jstl.jar;

H:\Java\AppServer\lib\appserv-jwsacc.jar;

H:\Java\AppServer\lib\appserv-launch.jar;

H:\Java\AppServer\lib\appserv-rt.jar;

H:\Java\AppServer\lib\appserv-tags.jar;

H:\Java\AppServer\lib\appserv-upgrade.jar;

H:\Java\AppServer\lib\appserv-ws.jar;

H:\Java\AppServer\lib\com-sun-commons-launcher.jar;

H:\Java\AppServer\lib\com-sun-commons-logging.jar;

H:\Java\AppServer\lib\dbschema.jar;

H:\Java\AppServer\lib\j2ee-svc.jar;

H:\Java\AppServer\lib\j2ee.jar;

H:\Java\AppServer\lib\javaee.jar;

H:\Java\AppServer\lib\jhall.jar;

H:\Java\AppServer\lib\jmxremote\_optional.jar;

H:\Java\AppServer\lib\jsf-impl.jar;

H:\Java\AppServer\lib\mail.jar;

H:\Java\AppServer\lib\sun-appserv-ant.jar;

H:\Java\AppServer\lib\toplink-essentials-agent.jar;

H:\Java\AppServer\lib\toplink-essentials.jar;

H:\Java\AppServer\jdk\lib\tools.jar;

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\build -d

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\ build -keep -s

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\ build

-verbose helloservice.endpoint.Hello [wsgen] Note: ap round: 1

[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]

[wsgen] [should process method: sayHello hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WrapperGen - method: sayHello(java.lang.String)]

[wsgen] [method.getDeclaringType():

helloservice.endpoint.Hello]

[wsgen] [requestWrapper: helloservice.endpoint.jaxws.SayHello]

[wsgen] [ProcessedMethods Class: java.lang.Object]

[wsgen] helloservice\endpoint\jaxws\SayHello.java

[wsgen] helloservice\endpoint\jaxws\SayHelloResponse.java

[wsgen] Note: ap round: 2

[wsgen] [completing model for endpoint:

helloservice.endpoint.Hello]

[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]

[wsgen] [should process method: sayHello hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WebServiceReferenceCollector - method:

sayHello(java.lang.String)]

[wsgen] [ProcessedMethods Class: java.lang.Object]

build-service:

build:

BUILD SUCCESSFUL

Total time: 9 seconds

Поскольку мы намеренно включили опцию вывода отладочной информации для **ant**,вывод получился довольно обширный.

Первое, что делается для компиляции программы, - создается специальная директория *build*,в которую будут помещены откомпилированные модули. Она создается в текущей директории. Затем вызывается компилятор javac, который компилирует наш класс **Hello.java**,а результат компиляции кладет в директорию *build*.Поскольку класс **Hello** определен в пакете **helloservice.endpoint**,в директории *build* будет создана соответствующая система каталогов и файл **Hello.class** будет помещен в каталог *./build/ helloservice/endpoint*.

Следующим шагом вызывается утилита **wsgen**,которая формирует вспомогательные классы. По умолчанию исходные коды этих классов после компиляции уничтожаются, однако, выставив опцию **keep=true** (эта и другие опции могут быть установлены в файле **build.properties)**,исходные коды можно сохранить. Помещаются они в пакет **jaxws** того же пакета, которому принадлежит и класс. Соответственно, для нашего примера исходные файлы (а затем и откомпилированные классы) будут располагаться в директории *./build/ helloservice/endpoint/jaxws*. После того как утилита wsgen отработала, мы имеем откомпилированный пакет **helloservice.endpoint.jaxws**,содержащий необходимые вспомогательные классы. На этом шаге компиляция нашего Web -сервиса закончена. Следующим этапом необходимо подготовить модуль развертывания. В нашем случае это делается с помощью команды:

asant create-war

Вывод получаем следующий:

Buildfile: build.xml

prepare-assemble:

[echo] Creating the assemble directory....

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

jaxws\helloservice\assemble

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

jaxws\helloservice\assemble\war

create-war:

[echo] Creating the WAR ...

[war] Building war: H:\Java\jwstutorial20\_new\examples\

jaxws\helloservice\assemble\war\hello-jaxws.war

BUILD SUCCESSFUL

Total time: 3 seconds

Создается отдельный каталог *assemble*,в нем создается каталог war, в котором формируется файл **hello-jaxws.war**.Этот файл представляет собой архив, в который помещены откомпилированные файлы нашего приложения и некоторые вспомогательные файлы. Теперь у нас полностью готов модуль развертывания, который мы можем инсталлировать в сервере приложений. Инсталляция может быть выполнена командой:

asant deploy

Результат выполнения команды следующий:

Buildfile: build.xml deploy:

admin\_command\_common:

[echo] Doing admin task deploy assemble/war/hello-jaxws.war

[sun-appserv-admin] Executing: deploy --port 4848 --host

localhost --passwordfile "H:\Java\jwstutorial20\_new\examples\

common\admin-password.txt" --user admin assemble/war/

hello-jaxws.war

[sun-appserv-admin] Command deploy executed successfully.

BUILD SUCCESSFUL

Total time: 43 seconds

Чтобы убедиться в том, что инсталляция приложения прошла успешно, можно воспользоваться консолью администратора:



**Рис. 10.4.** Проверка инсталляции приложения

В процессе инсталляции, кроме прочего, был сгенерирован WSDL -файл, описывающий установленный Web -сервис. Как уже говорилось, этот файл содержит полное описание Web -сервиса, включая названия его методов, а также количество и типы передаваемых и возвращаемых параметров. Этот файл является важной составляющей частью технологии, поскольку он позволяет строить приложения, осуществляющие динамические вызовы методов Web -сервисов. Кроме того, этот файл может быть использован для автоматической генерации вспомогательных классов (классов- proxy ) для обращения к Web -сервису.

WSDL -файл для нашего Web -сервиса может быть получен по адресу <http://localhost:8080/helloservice/hello?wsdl> - [пример 10.3](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=3#example.10.3).

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

- <definitions xmlns:tns="http://endpoint.helloservice/"

 xmlns:xsd="http://www.w3.org/2001/XMLSchema"

 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"

 xmlns="http://schemas.xmlsoap.org/wsdl/"

 targetNamespace="http://endpoint.helloservice/"

 name="HelloService">

- <types>

- <xsd:schema>

 <xsd:import namespace="http://endpoint.helloservice/"

 schemaLocation=

 "http://localhost:8080/helloservice/hello?xsd=1" />

 </xsd:schema>

 </types>

- <message name="sayHello">

 <part element="tns:sayHello" name="parameters" />

 </message>

- <message name="sayHelloResponse">

 <part element="tns:sayHelloResponse"

 name="parameters" />

 </message>

- <portType name="Hello">

- <operation name="sayHello">

 <input message="tns:sayHello" />

 <output message="tns:sayHelloResponse" />

 </operation>

 </portType>

- <binding type="tns:Hello"

 name="HelloPortBinding">

 <soap:binding style="document"

 transport= "http://schemas.xmlsoap.org/soap/http" />

- <operation name="sayHello">

 <soap:operation soapAction="" />

- <input>

 <soap:body use="literal" />

 </input>

- <output>

 <soap:body use="literal" />

 </output>

 </operation>

 </binding>

- <service name="HelloService">

- <port binding="tns:HelloPortBinding"

 name="HelloPort">

 <soap:address location="http://localhost:8080/helloservice/

 hello" />

 </port>

 </service>

 </definitions>

Листинг 10.3. WSDL-файл Web-сервиса Hello

**Тестирование Web-сервиса**

Итак, наш Web -сервис успешно инсталлирован. Осталось только убедиться в том, что он действительно работает. Чуть позже мы напишем специальное приложение-клиент, которое будет обращаться к нашему Web -сервису, а пока воспользуемся средствами, предоставляемыми нам *Sun Java System Application Server*.Дело в том, что этот сервер приложений способен самостоятельно динамически выстроить среду для вызова методов инсталлированных в нем Web -сервисов. Всей необходимой информацией, а именно: имена публикуемых методов, количество и тип принимаемых и возвращаемых методами параметров - он обладает.



**Рис. 10.5.** Описание Web-сервиса в административной консоли

Для того чтобы воспользоваться указанной возможностью, нужно выбрать нужный нам сервис в списке сервисов (в правой части окна браузера) и нажать кнопку "Test".



**Рис. 10.6.** Тестирование Web-сервиса

Откроется новое окно браузера ([рис. 10.6](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=4#image.10.6)), в котором отобразится динамически построенная сервером страница. На этой странице перечислены все опубликованные методы Web -сервиса (в нашем случае - один метод **sayHello)** и реализован интерфейс для их вызова. Если ввести в соответствующее поле строку и нажать кнопку - вызовется метод Web -сервиса и введенное значение будет передано ему в качестве параметра. Кроме всего прочего, на результирующей странице отобразятся SOAP -сообщения, соответственно, отправленные Web -сервису и пришедшие от него в качестве ответа.

Результирующая страница будет иметь следующий вид ([пример. 10.4](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=4#example.10.4)).

На странице ([пример. 10.4](https://www.intuit.ru/studies/courses/633/489/lecture/11087?page=4#example.10.4)) видны значения и типы переданных параметров, ответ, который возвратил метод Web -сервиса, - как и ожидалось, ответ представляет собой строку "Hello, Web-service test", - а также отправленный и полученный пакеты.

Таким образом, разработанный нами Web -сервис успешно инсталлирован в сервере приложений и может обрабатывать запросы клиентов, в чем мы убедились, использовав тестовое окружение, предоставляемое сервером приложений.

sayHello Method invocation

Method parameter(s)

Type Value

java.lang.String Web-service test

Method returned

java.lang.String : "Hello, Web-service test."

SOAP Request

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

<soapenv:Envelope xmlns:soapenv=

 "http://schemas.xmlso-ap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:ns1="http://endpoint.helloservice/">

<soapenv:Body>

<ns1:sayHello>

<arg0>Web-service test</arg0>

SOAP Response

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

<soapenv:Envelope

xmlns:soapenv="http://schemas.xmlsoap.org/ soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:ns1="http://endpoint.helloservice/">

<soapenv:Body>

<ns1:sayHelloResponse>

<return>Hello, Web-service test.</return>

</ns1:sayHelloResponse>

</soapenv:Body>

</soapenv:Envelope>

</ns1:sayHello>

</soapenv:Body>

</soapenv:Envelope>

Листинг 10.4. Результат