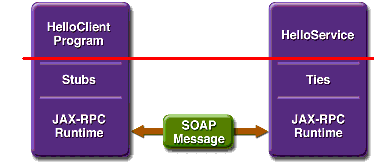
**Лекция 12**

При написании Web -сервисов с помощью инструментария, предоставляемого java, не оставляет ощущение какого-то подвоха - уж слишком просто оказывается это сделать! На самом деле, как и при использовании любого другого хорошего промежуточного программного обеспечения ( middleware ), простота эта кажущаяся - очень много действий за разработчика делает сама среда. И в данном случае процесс взаимодействия между клиентом и Web -сервисом в реальности выглядит следующим образом1 (см. [рис. 11.1](https://www.intuit.ru/studies/courses/633/489/lecture/11088?page=1#image.11.1))



**Рис. 11.1.** Процесс взаимодействия между клиентом и Web-сервисом

Причем разработчик реализует только функциональную начинку (на схеме - слои, находящиеся над красной линией). **Stubs** и **Ties** формируются автоматически утилитами **wsgen** и **wsimport**.Сгенерированные этими утилитами классы используют библиотеки поддержки **jax-rpc**,входящие в состав java. Сервлет, который принимает http -запросы, содержащие SOAP -пакеты, генерируется автоматически сервером приложений.

**Компиляция и инсталляция вручную**

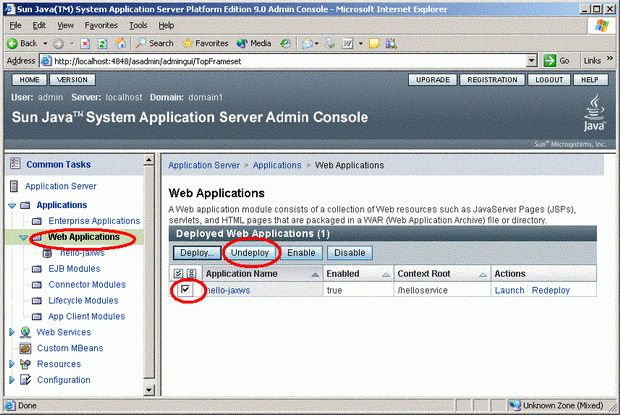
Следующим шагом мы должны были бы создать клиентское приложение, обращающееся к этому Web -сервису. Однако прежде чем это сделать, видимо, стоит подробнее разобраться с процессом компиляции и установки Web -сервисов. В самом деле, поскольку мы воспользовались скриптами, предоставленными разработчиками *The Java Web Services Tutorial*,реальные шаги, которые необходимо выполнить при создании Web -сервиса, оказались скрыты2. Таким образом, сейчас мы собираемся проделать те же самые действия - компиляция, генерация вспомогательных файлов, подготовка модуля развертывания и инсталляция сервиса, только теперь уже "вручную".

Первое, что мы проделаем, - деинсталлируем сервис и уничтожим все ранее созданные откомпилированные файлы.

Для деинсталляции выполняем команду:

asant undeploy

или пользуемся интерфейсом административной консоли ([рис. 11.2](https://www.intuit.ru/studies/courses/633/489/lecture/11088?page=1#image.11.2)).



**Рис. 11.2.** Деинсталляция

Выбираем, соответственно, пункт меню "Web Applications\*, указав в правой части экрана нужное приложение и нажав кнопку Undeploy.

После деинсталляции список установленных Web -сервисов станет пустым.

Команда

asant clean

удаляет вместе со всем содержимым директории *./build* и **./assemble**,приводя таким образом наш проект в исходное состояние.

Итак, первый шаг, который мы должны сделать, - откомпилировать исходный файл с нашим классом. Чтобы не смешивать результирующие файлы с исходными, по примеру разработчиков исходных скриптов компиляции создадим директорию *newbuild*.

Тогда команда компиляции будет выглядеть примерно следующим образом:

H:\Java\AppServer\jdk\bin\javac -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\]2ee-svc.jar;H:\Java\AppServer\lib\]2ee.]ar;

H:\Java\AppServer\lib\javaee.]ar;H:\Java\AppServer\lib\]hall.]ar;

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\new\_build -d

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\new\_build

./src/Hello.java

Выглядит, конечно, несколько устрашающе, но на самом деле ничего страшного не происходит - вызывается компилятор **javac**,в качестве classpath3 перечисляются все библиотеки, входящие в комплект j2ee, с помощью ключа - d указывается, что результат компиляции нужно разместить во вновь созданной папке *newbuild*,а компилировать нужно файл **./src/Hello.java**.

В результате выполнения компиляции в директории *newbuild* будет создана соответствующая нашей структуре пакетов структура каталогов и в нее будет помещен файл **Hello.class**.

Следующим шагом необходимо выполнить запуск утилиты **wsgen**,которая сгенерирует необходимые вспомогательные модули.

Эта команда выглядит следующим образом:

H:\Java\AppServer\bin\wsgen -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\]sf-impl.]ar;

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\]axws\helloservice\ new\_bild -d

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

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\new\_build

-verbose helloservice.endpoint.Hello

и должна выполняться в директории *newbuild*.

По-прежнему в качестве **classpath** указываются все библиотеки, а также наша директория, в которой лежит откомпилированный класс с Web -сервисом. Ключ **-d** указывает каталог, в который необходимо разместить результирующие файлы; ключ **-s** указывает директорию для размещения сгенерированных исходных кодов; ключ **-keep** указывает на то, что после компиляции файлы с исходными кодами нужно сохранить; ключ **-verbose** обязывает утилиту генерировать отчет о своих действиях. Последний параметр задает обрабатываемый класс. В результате работы утилиты мы получаем два дополнительных сервисных класса, расположенных в пакете **helloservice.endpoint.jaxws**.

Следующим этапом будет инсталляция нашего Web -сервиса в сервер приложений. Для этого сначала нужно создать стандартную структуру каталогов для Web -приложения. Структура эта такая:

* в корне лежат файлы \*.html, \*.jsp и т.д. - файлы, которые должны быть видны и доступны клиенту при их наборе в строке адреса браузера;
* /WEB-INF/web.xml - Web Application Deployment Descriptor для приложения. Это XML файл, описывающий сервлеты и другие компоненты приложения, их параметры инициализации и т.д.;
* /WEB-INF/sun-web.xml - описывает параметры приложения, специфические для сервера приложений Sun ;
* /WEB-INF/sun-jaxws.xml - описывает параметры, необходимые серверу приложений для создания Web -сервиса;
* /WEB-INF/classes/ - директория содержит классы приложения (именно классы, а не архивы jar );
* /WEB-INF/lib/ - директория содержит архивы jar, необходимые для работы приложения.

Создадим новую директорию newassemble и в ней соответствующую структуру каталогов:

new\_assemble\WEB-INF\classes new\_assemble\WEB-INF\sun-jaxws.xml

new\_assemble\WEB-INF\sun-web.xml new\_assemble\WEB-INF\web.xml

new\_assemble\WEB-INF\classes\helloservice\endpoint\Hello.class

new\_assemble\WEB-INF\classes\helloservice\endpoint\jaxws\

SayHello.class

new\_assemble\WEB-INF\classes\helloservice\endpoint\jaxws\SayHello.java

new\_assemble\WEB-INF\classes\helloservice\endpoint\jaxws\SayHelloResponse.class

new\_assemble\WEB-INF\classes\helloservice\endpoint\jaxws\SayHelloResponse.java

И теперь инсталлируем наш сервис командой:

asadmin deploydir H:\Java\jwstutorial20\_new\examples\jaxws\

helloservice\new\_assemble

После ее успешного завершения новый сервис инсталлирован, в чем можно убедиться, войдя в административную консоль.

**Клиент**

Теперь, когда Web -cервис готов к работе, можно приступить к реализации клиента для него. Для этого снова воспользуемся кодом примера, поставляемого с **The Java Web Services Tutorial** (он располагается в каталоге simpleclient ).

Класс клиента будет выглядеть следующим образом ([пример 11.1](https://www.intuit.ru/studies/courses/633/489/lecture/11088?page=2#example.11.1)).

1 package simpleclient;

2

3 import javax.xml.ws.WebServiceRef;

4 import helloservice.endpoint.HelloService;

5 import helloservice.endpoint.Hello;

6

7 public class HelloClient {

8 @WebServiceRef(wsdlLocation="http://localhost:8080/helloservice/hello?wsdl")

9 static HelloService service;

10

11 /\*\*

12 \* @param args the command line arguments

13 \*/

14 public static void main(String[] args) {

15 try {

16 HelloClient client = new HelloClient();

17 client.doTest(args);

18 } catch(Exception e) {

19 e.printStackTrace();

20 }

21 }

22

23 public void doTest(String[] args) {

24 try {

25 System.out.println("Retrieving the port from the following service: " + service);

26 Hello port = service.getHelloPort();

27 System.out.println("Invoking the sayHello operation on the port.");

28

29 String name;

30 if (args.length > 0) {

31 name = args[0];

32 } else {

33 name = "No Name";

34 }

35

36 String response = port.sayHello(name);

37 System.out.println(response);

38 } catch(Exception e) {

39 e.printStackTrace();

40 }

41 }

42 }

Листинг 11.1. Клиентское приложение

Клиент объявляется принадлежащим пакету **simpleclient** (строка 1). Импортируется класс **javax.xml.ws.WebServiceRef** (строка 3), являющийся аннотацией. Далее импортируются два класса из пакета **helloservice.endpoint** - класс **Hello** (строка 5) и класс **HelloService** (строка 4). Эти классы (а также несколько других) будут сгенерированы автоматически, на основании анализа WSDL -файла. **Hello** - интерфейс, в котором объявлены все методы Web -сервиса. Клиентская заглушка ( proxy класс), передающий вызовы Web -сервису, реализует этот интерфейс, и со стороны клиента все выглядит так, как будто он непосредственно вызывает методы Web -сервиса. **HelloService** - сервисный класс, который создает клиентский proxy. Кроме того, для каждого метода и возвращаемого значения генерируются свои вспомогательные классы.

Для того чтобы вспомогательные классы могли быть сгенерированы, нужно указать файл WSDL, в котором описан нужный Web -сервис. Для этого служит специальная аннотация **WebServiceRef** (строка 8), свойство **wsdlLocation** которой указывает на местоположение этого файла.

При старте клиента (метод **main**,строка 14) создается объект типа **HelloClient** (строка 16) и запускается его метод **doTest** (строка 17). Этот метод получает ссылку на клиентский класс proxy, вызывая метод **getHelloPort** (строка 26) класса **HelloService**.Начиная с этого момента, клиент может вызывать методы Web -сервиса (строка 36). Таким образом, как и ожидалось, транспортные детали взаимодействия с Web -сервисом полностью скрыты от клиента - для него все выглядит как обычный локальный вызов метода **sayHello** локального объекта **port**.

Для того чтобы откомпилировать пример, достаточно набрать команду

asant

(компиляция является целью по умолчанию). Ниже приведен вывод этой команды

Buildfile: build.xml

clean:

javaee-home-test:

init:

prepare:

[echo] Creating the required directories ...

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

jaxws\simpleclient\build

generate-client:

[wsimport] command line: wsimport

H:\Java\AppServer\jdk\jre\bin\java.exe -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-ext.jar;

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

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

H:\Java\AppServer\lib\appserv-deployment-client.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\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 com.sun.tools.ws.WsImport -d

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

H:\Java\jwstutorial20\_new\examples\jaxws\simpleclient\ src -verbose

http://localhost:8080/helloservice/hello?wsdl -p

helloservice.endpoint

[wsimport] helloservice\endpoint\Hello.java

[wsimport] helloservice\endpoint\HelloService.java

[wsimport] helloservice\endpoint\ObjectFactory.java

[wsimport] helloservice\endpoint\SayHello.java

[wsimport] helloservice\endpoint\SayHelloResponse.java

[wsimport] helloservice\endpoint\package-info.java

[wsimport] helloservice\endpoint\Hello.java

[wsimport] helloservice\endpoint\HelloService.java

[wsimport] helloservice\endpoint\ObjectFactory.java

[wsimport] helloservice\endpoint\SayHello.java

[wsimport] helloservice\endpoint\SayHelloResponse.java

[wsimport] helloservice\endpoint\package-info.java

compile-client:

[echo] Compiling the client source code ...

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

jwstutorial20\_new\examples\jaxws\simpleclient\build

prepare-dist:

[echo] Creating the required directories ...

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

examples\jaxws\simpleclient\dist

package-client:

[echo] Building the client JAR file ...

[jar] Building jar: H:\Java\jwstutorial20\_new\

examples\jaxws\simpleclient\dist\simpleclient.jar

build-client:

build:

BUILD SUCCESSFUL

Total time: 9 seconds

Следует отметить, что одной командой выполнено несколько действий. Во-первых, сгенерированы необходимые вспомогательные файлы с помощью утилиты **wsimport**.Во-вторых, исходный код клиента откомпилирован (вместе с автоматически сгенерированными файлами). И в-третьих, полученные откомпилированные модули собраны в архив.

Теперь осталось выполнить команду

asant run

и убедиться в том, что написанный клиент успешно обратился к Web -сервису:

Buildfile: build.xml

run:

[echo] To set the name, modify the client.arg property

[echo] in build.properties. If client.arg is unset,

[echo] the default name sent to the service is No Name.

appclient-command-common:

[exec] Retrieving the port from the following service:

helloservice.endpoint.HelloService@9770a3

[exec] Invoking the sayHello operation on the port.

[exec] Hello, Web service test.

BUILD SUCCESSFUL

Total time: 9 seconds

Без использования скриптов процесс компиляции выглядел бы так:

1. создаем директорию *./build* (в нее будем складывать откомпилированные файлы);
2. с помощью утилиты **wsimport** создаем вспомогательные файлы на основании анализа WSDL.

Для этого введем команду:

wsimport -d ./build -s ./build -keep http://localhost:8080/

helloservice/hello?wsdl

здесь ключ **-d** указывает директорию, в которую следует помещать результирующие откомпилированные файлы;

ключ **-s** указывает директорию для размещения сгенерированных исходных файлов;

ключ **-keep** указывает на то, что после компиляции исходные файлы следует сохранить;

последний параметр - адрес WSDL -файла, для которого нужно сгенерировать вспомогательные файлы.

В результате работы этой утилиты в папке *./build* будет сформирована соответствующая структуре пакетов структура каталогов и в ней будут размещены исходные и откомпилированные файлы - результат работы утилиты wsimport ;

1. компилируем приложение:
2. javac -classpath H:\Java\AppServer\lib\activation.jar;
3. H:\Java\ AppServer\lib\admin-cli.jar;
4. H:\Java\AppServer\lib\ appserv-admin.jar;
5. H:\Java\AppServer\lib\appserv-cmp.jar;
6. H:\Java\AppServer\lib\appserv-deployment-client.jar;
7. H:\Java\AppServer\lib\appserv-ext.jar;
8. H:\Java\AppServer\lib\appserv-jstl.jar;
9. H:\Java\AppServer\lib\appserv-jwsacc.jar;
10. H:\Java\AppServer\lib\appserv-launch.jar;
11. H:\Java\AppServer\lib\appserv-rt.jar;
12. H:\Java\AppServer\lib\appserv-tags.jar;
13. H:\Java\AppServer\lib\appserv-upgrade.jar;
14. H:\Java\AppServer\lib\appserv-ws.jar;
15. H:\Java\AppServer\lib\com-sun-commons-launcher.jar;
16. H:\Java\AppServer\lib\com-sun-commons-logging.jar;
17. H:\Java\AppServer\lib\dbschema.jar;
18. H:\Java\AppServer\lib\j2ee-svc.jar;
19. H:\Java\AppServer\lib\j2ee.jar;
20. H:\Java\AppServer\lib\javaee.jar;
21. H:\Java\AppServer\lib\ jhall.jar;
22. H:\Java\AppServer\lib\jmxremote\_optional.jar;
23. H:\Java\AppServer\lib\jsf-impl.jar;
24. H:\Java\AppServer\lib\mail.jar;
25. H:\Java\AppServer\lib\sun-appserv-ant.jar;
26. H:\Java\AppServer\lib\toplink-essentials-agent.jar;
27. H:\Java\AppServer\lib\toplink-essentials.jar;
28. H:\Java\AppServer\jdk\lib\tools.jar;
29. H:\Java\jwstutorial20\_new\examples\jaxws\simpleclient\
30. build -d H:\Java\jwstutorial20\_new\examples\jaxws\simpleclient\

build ./src/HelloClient.java

1. и запускаем (для этого нужно перейти в папку *./build)*:

appclient simpleclient.HelloClient "WEB service test "

Полученный вывод

Retrieving the port from the following service: helloservice.end-

point.HelloService@801059

Invoking the sayHello operation on the port.

Hello, WEB service test

говорит о том, что клиент успешно откомпилирован и выполнен.

Командный файл **appclient** располагается в каталоге *bin* сервера приложений и представляет собой обертку над интерпретатором java, которая определяет некоторые переменные окружения, *classpath* и другие параметры виртуальной машины.