Бесплатная электронная книга

учусь soap

Free unaffiliated eBook created from
Stack Overflow contributors.

#soap

1

1: 2

2

2

Examples 2

2

4

SOAP 1.1 1.2 5

- 5

WSDL 6

WSDL 1.1 6

WSDL 2.0 8

WSDL 1.1 2.0 10

12

RPC / 13

RPC / 14

/ 15

/ 15

/ () 16

UDDI 17

: 18

Java- webservice http://www. 19

- JAX-WS (/) 20

2: - SOAP 24

24

24

Examples 24

Stub Java 24

26

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: soap

It is an unofficial and free soap ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official soap.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с мылом

замечания

В этом разделе представлен обзор того, что такое мыло, и почему разработчик может захотеть его использовать.

Следует также упомянуть любые крупные предметы в мыле и ссылки на связанные темы. Поскольку документация для мыла новая, вам может потребоваться создать начальные версии этих связанных тем.

Версии

Версия	Дата выхода
1,1	2000-05-08
1.2	2003-06-24

Examples

Основная информация

SOAP - это аббревиатура для *Simple Object Access Protocol,* который определяет протокол, который используется для обмена данными посредством удаленного вызова процедур (RPC) с другими службами или клиентами SOAP. Он доступен в двух версиях:

- SOAP 1.1 [IETF]
- SOAP 1.2 [IETF]

SOAP 1.2 устаревает SOAP 1.1, поэтому рекомендуется использовать SOAP 1.2, если это возможно.

Он часто строится поверх HTTP / S и редко на SMTP или FTP, хотя он будет поддерживать его в соответствии с протоколом. Хотя HTTP часто используется в качестве основного транспортного протокола, SOAP использует только ограниченное подмножество. Для отправки запросов он почти полностью опирается на операцию `POST` HTTP. `GET` вызовы теоретически возможны с 1.2, хотя документ должен быть передан как параметр URI и, следовательно, может превышать границу примерно 3000 символов, которая отклоняется большинством фреймворков. Кроме того, параметры, связанные с безопасностью, обычно определяются в специальном заголовке SOAP.

Хотя SOAP и REST называются веб-сервисами, они по своей природе очень разные. В некоторых структурах различают WS (для служб на основе SOAP) и RS (для служб на основе REST).

В следующей таблице приведен краткий обзор различий между типами веб-сервисов.

аспект	МЫЛО	ОСТАЛЬНОЕ
стандарт	SOAP , WSDL	Нет стандартного, просто архитектурного стиля
Реализация ресурсов	Косвенные операции SOAP	через уникальные идентификаторы ресурсов (URI)
Обработка ошибок	Сообщение об ошибке SOAP	Коды ответа HTTP-кода и, возможно, тело ответа
Представление данных	XML	все доступные кодировки в HTTP
Использование HTTP	В качестве транспортного протокола	Действия с ресурсами (CRUD), отображаемые по HTTP-методам (GET, POST, PUT, DELETE, ...)
Транзакционная поддержка	через заголовок SOAP	путем моделирования транзакции как ресурса
государственный	Stateful (действие SOAP является частью приложения)	Без гражданства (автономные запросы)
Обнаружение службы	UDDI / WSDL	На самом деле нет; Start-URI API должен возвращать список дополнительных API-интерфейсов, хотя
метод	Внутри корпуса SOAP	Метод HTTP
Аргументы метода	Определено XML-схемой в WSDL	Либо через HTTP-заголовки, либо параметры Path / Query или Matrix в URI
Переход государства	Трудно определить, как не напрямую на основе данных	Следующий вызов URI

аспект	МЫЛО	ОСТАЛЬНОЕ
Поддержка кеширования	Кэширование часто нежелательно,	Простой, как определено HTTP

МЫЛО

Запросы SOAP состоят из SOAP-конверта, который должен содержать элемент body и может содержать необязательный элемент заголовка. Элемент заголовка используется для передачи определенных конфигураций службе, например, WS-Security может определить, что сообщение зашифровано, или WS-Coordination / WS-Transaction может определить, что сообщение должно выполняться в транзакции.

Простые запросы SOAP 1.2 через HTTP, который добавляет два значения, могут выглядеть так:

```
POST /calculator HTTP/1.1
Host: http://example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 224

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
  <m:AddValues xmlns:m="http://example.org/calculator">
   <m:FirstValue>1</m:FirstValue>
   <m:SecondValue>2</m:SecondValue>
  </m:AddValues>
 </env:Body>
</env:Envelope>
```

Ответ на вышеупомянутый запрос образца может выглядеть так:

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 329

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
 <soap:Body xmlns:m="http://www.example.org/calculator">
  <m:AddValuesResponse>
   <m:Result>3</m:Result>
  </m:AddValuesResponse>
 </soap:Body>
</soap:Envelope>
```

В приведенном выше примере определен запрос, который вызывается методом `AddValues` с двумя аргументами, `FirstValue` установлен `FirstValue 1` а `SecondValue` - `2` . Запрос привел к выполнению этого метода на удаленном сервере SOAP, который в результате вычислил

значение `3` которое инкапсулировано в отдельный элемент ответа, который по соглашению часто является именем вызываемого метода плюс завершающей строкой `Response` так что любой, кто проверка ответа может заключаться в том, что это ответ предыдущего `AddValue` метода `AddValue` .

Различия между SOAP 1.1 и 1.2

SOAP 1.2 позволяет другим транспортным протоколам, а затем HTTP, если инфраструктура привязки поддерживается протоколом.

SOAP 1.1 основан на XML 1.0, в то время как 1.2 основан на XML Infoset, который позволяет сериализовать SOAP-сообщения с другими сериализаторами, а затем по умолчанию XML 1.0-сериализатор, используемый SOAP 1.1. Это позволяет, например, сериализовать сообщения как двоичные сообщения и, следовательно, предотвратить некоторые накладные расходы XML-характера сообщения. В дополнение к этому механизм сериализации используемого базового протокола может быть определен посредством привязки данных.

Аспект совместимости также был поддержан SOAP 1.2, определяя более конкретную модель обработки, а затем ее предшественницу, которая устранила множество возможностей для интерпретации. SOAP с API-интерфейсом вложений (SAAJ), который позволяет работать с сообщениями SOAP 1.1 и 1.2, помог многим разработчикам среды обрабатывать и создавать сообщения.

W3C опубликовал краткий обзор основных изменений между SOAP 1.1 и 1.2

Взаимодействие веб-сервисов

Взаимодействие Web-сервисов (также известный как WS-I) - это руководство по совместимости, управляемое некоторыми известными предприятиями, такими как IBM, Microsoft, Oracle и HP, чтобы назвать лишь некоторые из них. Эти рекомендации, среди других, рекомендуют использовать только один корневой элемент внутри тела SOAP, даже если SOAP позволяет содержать несколько элементов в теле.

WS-I состоит из

- WS-I Basic Profile aka WSI-BP
- Базовый профиль безопасности WS-I
- Простой профиль связывания мыла

WSI-BP доступен в 4 различных версиях v1.0 (2004) , v1.1 (2006) , v1.2 (2010) , v2.0 (2010)) и определяет рекомендации по совместимости для основных спецификаций веб-сервисов, таких как SOAP, WSDL и UDDI. Благодаря использованию языка описания веб-сервисов (WSDL) службы SOAP могут описывать поддерживаемые операции и методы в связном

наборе с другими конечными точками. WSI-BP использует WSDL для определения более узкого множества, тогда полная схема WSDL или SOAP будет определять и, таким образом, устраняет некоторую двусмысленность в самой спецификации и тем самым улучшает взаимодействие между конечными точками.

WSDL

Чтобы рекламировать доступные операции SOAP, их параметры, а также соответствующие конечные точки для вызова клиентам, для дальнейшего использования используется другой XML-документ, называемый `Web Services Description Language` или WSDL.

WSDL описывает конечную точку службы, привязку SOAP-сообщений к операциям, интерфейс операций, а также их типы для клиентов. WSDL, как и SOAP, доступен в двух версиях, которые отличаются своим синтаксисом, хотя и выражают почти ту же семантику клиенту.

WSDL 1.1

Описание WSDL 1.1 содержит `service` , `binding` , `portType` и раздел `message` . Он может дополнительно импортировать или определять схемы в файле WSDL, как видно из примера файла WSDL, который соответствует образцу калькулятора, показанному выше:

```
<wsdl:definitions xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:calc="http://example.org/calculator"
                  xmlns:tns="http://example.org/calculatorService"
                  targetNamespace="http://example.org/calculatorService">

    <!--
      Abstract type definitions
    -->

    <wsdl:types>
        <!--
        <xs:schema>
            <xs:import namespace="http://example.org/calculator"
schemaLocation="calc/calculator.xsd" />
        </xs:schema>
        -->
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                   xmlns:tns="http://example.org/calculator"
                   targetNamespace="http://example.org/calculator"
                   elementFormDefault="qualified"
                   attributeFormDefault="qualified">

            <xs:element name="AddValuesRequest" type="tns:AddValuesType" />
            <xs:element name="AddValuesResponse" type="tns:AddValuesResponseType" />

            <xs:complexType name="AddValuesType">
```

```
                <xs:sequence>
                    <xs:element name="FirstValue" type="xs:int" minOccurs="1" maxOccurs="1" />
                    <xs:element name="SecondValue" type="xs:int" minOccurs="1" maxOccurs="1"
/>
                </xs:sequence>
            </xs:complexType>

            <xs:complexType name="AddValuesResponseType">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="Result" type="xs:int" />
                </xs:sequence>
            </xs:complexType>

            <xs:attribute name="Timestamp" type="xs:dateTime" />
            <xs:element name="CalculationFailure">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ErrorCode" type="xs:int" />
                        <xs:element name="Reason" type="xs:string" />
                    </xs:sequence>
                    <xs:attribute ref="tns:Timestamp" use="required" />
                </xs:complexType>
            </xs:element>

        </xs:schema>
    </wsdl:types>

    <!--
        Abstract message definitions
    -->

    <wsdl:message name="AddValuesRequest">
        <wsdl:part name="in" element="calc:AddValuesRequest" />
    </wsdl:message>
    <wsdl:message name="AddValuesResponse">
        <wsdl:part name="out" element="calc:AddValuesResponse" />
    </wsdl:message>
    <wsdl:message name="CalculationFault">
        <wsdl:part name="fault" element="calc:CalculationFailure" />
    </wsdl:message>

    <!--
        Abstract portType / interface definition
    -->

    <wsdl:portType name="CalculatorEndpoint">
        <wsdl:operation name="AddValues">
            <wsdl:documentation>Adds up passed values and returns the
result</wsdl:documentation>
            <wsdl:input message="tns:AddValuesRequest" />
            <wsdl:output message="tns:AddValuesResponse" />
            <wsdl:fault name="CalculationFault" message="tns:CalculationFault" />
        </wsdl:operation>
    </wsdl:portType>

    <!--
      Concrete binding definition
    -->

    <wsdl:binding name="CalculatorBinding" type="tns:CalculatorEndpoint">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
        <wsdl:operation name="AddValues">
            <soap:operation soapAction="http://example.org/calculator/AddValuesMessage" />
            <wsdl:input>
                <soap:body parts="in" use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="out" use="literal" />
            </wsdl:output>
            <wsdl:fault name="CalculationFault">
                <soap:fault name="CalculationFault" use="literal" />
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>

    <!--
        Concrete service definition
    -->

    <wsdl:service name="CalculatorService">
        <wsdl:port name="CalculatorServicePort" binding="tns:CalculatorBinding">
            <soap:address location="http://localhost:8080/services/calculator" />
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```

Раздел `service` определяет конкретные конечные точки, которые служба будет прослушивать входящие запросы. Раздел `binding` привязывает операцию к конкретному стилю и определяет, какие форматы сообщений ожидает сервер или что клиент может ожидать.

Абстрактный раздел состоит из блока `portType` который определяет операции, предлагаемые службой, и какие сообщения обмениваются. Сообщения указываются в своем блоке и привязаны к типам схем, аргументы и возвращаемые значения являются экземплярами. Сообщения могут объявлять параметры или возвращаемые значения , чтобы быть `in` , `out` или `inout` . В то время как первые два довольно просто понять, последний имитирует поведение аргументов, переданных по ссылке. Поскольку pass-by-ref не поддерживается на некоторых языках, этот эффект часто моделируется с помощью определенных обработчиков.

WSDL 2.0

Тот же калькулятор можно описать в WSDL 2.0 следующим образом:

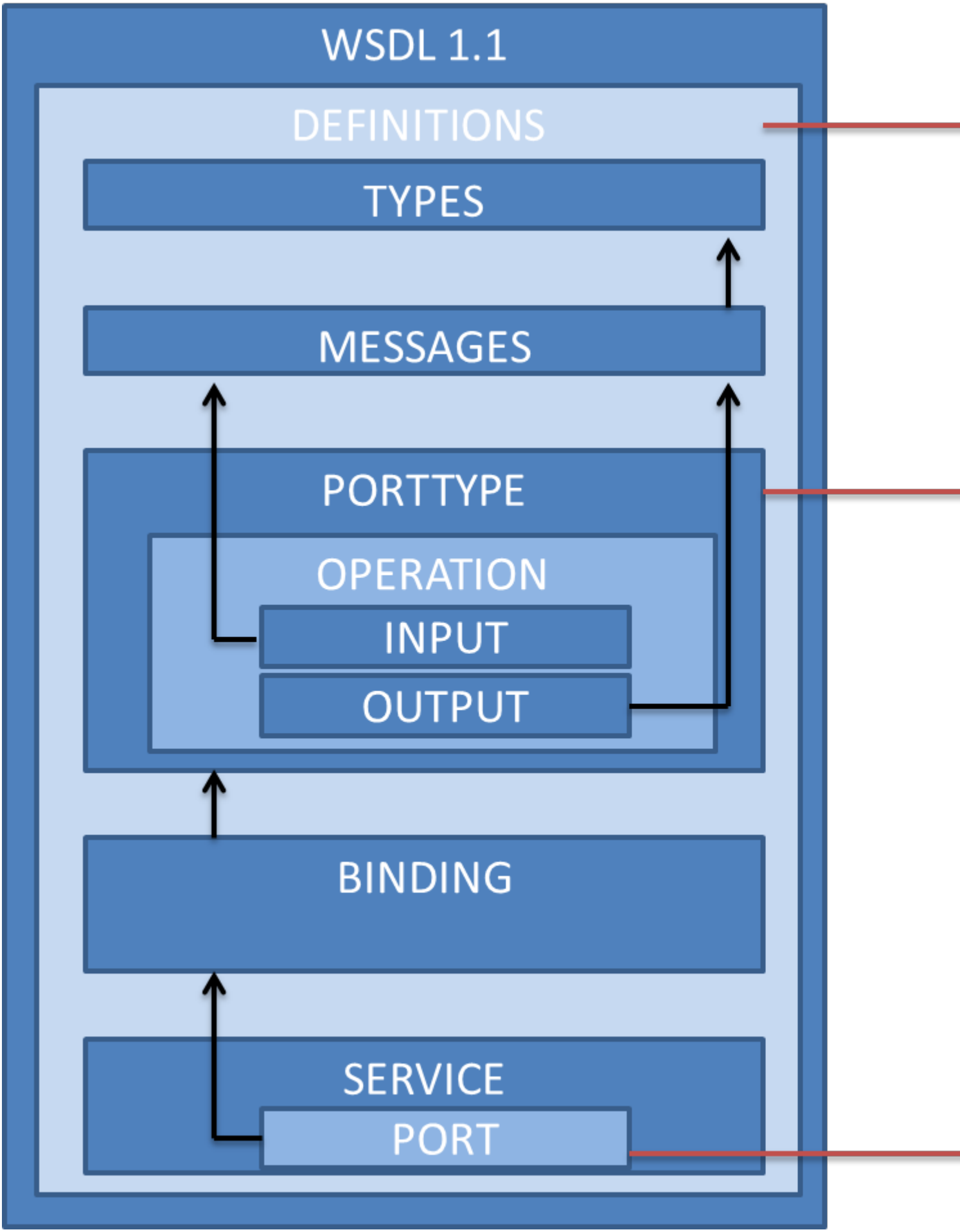
```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:description xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://www.w3.org/ns/wsdl"
                  xmlns:soap="http://www.w3.org/ns/wsdl/soap"
                  xmlns:calc="http://example.org/calculator"
                  xmlns:tns="http://example.org/calculatorService"
                  targetNamespace="http://example.org/calculatorService">

    <!--
```

```
      Abstract type definitions
    -->

    <wsdl:types>
        <!--
        <xs:schema>
            <xs:import namespace="http://example.org/calculator"
schemaLocation="calc/calculator.xsd" />
        </xs:schema>
        -->
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                   xmlns:tns="http://example.org/calculator"
                   targetNamespace="http://example.org/calculator"
                   elementFormDefault="qualified"
                   attributeFormDefault="qualified">

            <xs:element name="AddValuesRequest" type="tns:AddValuesType" />
            <xs:element name="AddValuesResponse" type="tns:AddValuesResponseType" />

            <xs:complexType name="AddValuesType">
                <xs:sequence>
                    <xs:element name="FirstValue" type="xs:int" minOccurs="1" maxOccurs="1" />
                    <xs:element name="SecondValue" type="xs:int" minOccurs="1" maxOccurs="1"
/>
                </xs:sequence>
            </xs:complexType>

            <xs:complexType name="AddValuesResponseType">
                <xs:sequence minOccurs="1" maxOccurs="1">
                    <xs:element name="Result" type="xs:int" />
                </xs:sequence>
            </xs:complexType>

            <xs:attribute name="Timestamp" type="xs:dateTime" />
            <xs:element name="CalculationFault">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="ErrorCode" type="xs:int" />
                        <xs:element name="Reason" type="xs:string" />
                    </xs:sequence>
                    <xs:attribute ref="tns:Timestamp" use="required" />
                </xs:complexType>
            </xs:element>

        </xs:schema>
    </wsdl:types>

    <!--
        Abstract interface
    -->

    <wsdl:interface name="CalculatorInterface">
        <wsdl:fault name="fault" element="calc:CalculationFault" />
        <wsdl:operation name="AddValues" pattern="http://www.w3.org/ns/wsdl/in-out"
style="http://www.w3.org/ns/wsdl/style/iri" wsdl:safe="true">
            <wsdl:documentation>Adds up passed values and returns the
result</wsdl:documentation>
            <wsdl:input messageLabel="in" element="calc:AddValuesRequest" />
            <wsdl:output messageLabel="out" element="calc:AddValuesResponse" />
            <wsdl:outfault messageLabel="fault" ref="tns:fault" />
        </wsdl:operation>
```

```
    </wsdl:interface>

    <!--
        Concrete binding definition
    -->

    <wsdl:binding name="CalculatorBinding" interface="tns:CalculatorInterface"
type="http://www.w3.org/ns/wsdl/soap"
soap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
        <wsdl:operation ref="tns:AddValues" soap:mep="http://www.w3.org/2003/05/soap/mep/soap-
response" />
        <wsdl:fault ref="tns:fault" soap:code="soap:Sender" />
    </wsdl:binding>

    <!--
        Concrete service definition
    -->

    <wsdl:service name="CalculatorService" interface="tns:CalculatorInterface">
        <wsdl:endpoint name="CalculatorEndpoint" binding="tns:CalculatorBinding"
address="http://localhost:8080/services/calculator" />
    </wsdl:service>

</wsdl:description>
```

Различия между WSDL 1.1 и 2.0

Графический обзор различий между обеими версиями можно увидеть на рисунке ниже.


WSDL 1.1
DEFINITIONS
TYPES
MESSAGES
PORTTYPE
OPERATION
INPUT
OUTPUT
BINDING
SERVICE
PORT

(Источник)

Как видно из изображения, раздел `message` удалился, который теперь содержится в разделе `interface` . Кроме того, некоторые элементы были переименованы, другие имеют другой синтаксис, но, как правило, версия WSDL в основном аналогична версии 2.0, требующей немного меньшего количества накладных расходов по сравнению с 1.1.

Помимо меньшего размера определения SOAP-сервисов через WSDL 2.0, более новая версия также предоставляет возможности для определения служб REST, хотя WSDL 2.0 или даже WADL НЕ рекомендуются для служб RESTful, поскольку они противоречат фактической идее, стоящей за ней.

Какой стиль предпочитают

Раздел привязки WSDL описывает, как служба привязана к протоколу обмена сообщениями SOAP. В приведенном выше примере использовался `document` как стиль привязки, который позволяет структурировать тело SOAP так, как мы хотим, пока результирующий вывод является допустимым экземпляром XML. Это стиль привязки по умолчанию и часто упоминается как `Message-Oriented style` на `Message-Oriented style` .

В отличие от стиля `document` , тела запроса стиля `RPC` должны содержать как имя операции, так и набор параметров метода. Поэтому структура экземпляра XML предопределена и не может быть изменена.

В дополнение к стилю привязки секция связывания также определяет модель трансляции для привязок к сообщениям SOAP в имени `literal` или `encoded` . Разница между ними заключается в том, что эта `literal` модель должна соответствовать пользовательской структуре XSD, которая может использоваться для проверки запросов и ответов, в то время как `encoded` модель должна использовать типы данных XSD, такие как `xs:integer` или `xs:string` но взамен поэтому не должен соответствовать какой-либо пользовательской схеме. Однако это затрудняет проверку тела сообщения или преобразование сообщения через XSLT в другой формат.

Сочетание стиля привязки с используемой моделью позволяет фактически 4 разных результата сообщения. 5-я запись добавляется в список, который обычно используется (хотя и не является частью стандарта).

- RPC / закодированный
- RPC / литерал
- Документ / кодировка
- Документ / литерал
- Документ / литерал (завернутый)

  В текстовом / литеральном стиле обмена сообщениями существует шаблон, который известен как wrapped-document / literal. Это всего лишь шаблон и не входит в спецификацию WSDL. Этот шаблон упоминается в JSR 224 (JAX-WS:

Java API для веб-сервисов на основе XML). (Источник)

В приведенном ниже разделе представлен обзор различий в отношении объявления WSDL или схемы и их влияние на результирующий формат сообщения SOAP при изменении стиля привязки или определения модели использования.

RPC / закодированный

WSDL:

```
...
<wsdl:message name="AddValues">
  <wsdl:part name="FirstValue" type="xsd:int" />
  <wsdl:part name="SecondValue" type="xsd:int" />
</wsdl:message>
<wsdl:message name="AddValuesResponse">
  <wsdl:part name="Result" type="xsd:int" />
</wsdl:message>

<wsdl:portType name="CalculatorEndpoint">
  <wsdl:operation="AddValues">
    <wsdl:input message="AddValues" />
    <wsdl:output message="AddValuesResponse" />
  </wsdl:operation>
</wsdl:portType>

<!-- binding style set to 'RPC' and use to 'encoded' -->
...
```

Запрос SOAP

```
<soap:envelope>
  <soap:body>
    <AddValues>
      <FirstValue xsi:type="xsd:int">1</FirstValue>
      <SecondValue xsi:type="xsd:int">2</SecondValue>
    </AddValues>
  </soap:body>
</soap:envelope>
```

Ответ SOAP

```
<soap:envelope>
  <soap:body>
    <AddValuesResponse>
      <Result xsi:type="xsd:int">3</Result>
    </AddValuesResponse>
  </soap:body>
</soap:envelope>
```

Pros

- простой WSDL
- Название операции и элементов, доступных в запросе и ответе

Cons

- Явное объявление типов XSI
- Трудно проверить
- Не соответствует WS-I

RPC / литерал

WSDL:

```
...
<wsdl:message name="AddValues">
  <wsdl:part name="FirstValue" type="xsd:int" />
  <wsdl:part name="SecondValue" type="xsd:int" />
</wsdl:message>
<wsdl:message name="AddValuesResponse">
  <wsdl:part name="Result" type="xsd:int" />
</wsdl:message>

<wsdl:portType name="CalculatorEndpoint">
  <wsdl:operation="AddValues">
    <wsdl:input message="AddValues" />
    <wsdl:output message="AddValuesResponse" />
  </wsdl:operation>
</wsdl:portType>

<!-- binding style set to 'RPC' and use to 'literal' -->
...
```

Запрос SOAP

```
<soap:envelope>
  <soap:body>
    <AddValues>
      <FirstValue>1</FirstValue>
      <SecondValue>2</SecondValue>
    </AddValues>
  </soap:body>
</soap:envelope>
```

Ответ SOAP

```
<soap:envelope>
  <soap:body>
    <AddValuesResult>
      <Result>3</Result>
    </AddValuesResult>
  </soap:body>
</soap:envelope>
```

Pros

- простой WSDL
- Название операции и элементов, доступных в запросе и ответе

- Нет необходимости в спецификации типа XSI
- Соответствие WS-I

Cons

- Трудно проверить

Документ / кодировка

Поэтому смысл не опускается.

Документ / литерал

WSDL:

```
...
<types>
  <schema>
    <element name="FirstValueElement" type="xsd:int" />
    <element name="SecondValueElement" type="xsd:int" />
    <element name="ResultValueElement" type="xsd:int" />
  </schema>
</types>

<wsdl:message name="AddValues">
  <wsdl:part name="FirstValue" element="FirstValueElement" />
  <wsdl:part name="SecondValue" element="SecondValueElement" />
</wsdl:message>
<wsdl:message name="AddValuesResponse">
  <wsdl:part name="Result" element="ResultValueElement" />
</wsdl:message>

<wsdl:portType name="CalculatorEndpoint">
  <wsdl:operation="AddValues">
    <wsdl:input message="AddValues" />
    <wsdl:output message="AddValuesResponse" />
  </wsdl:operation>
</wsdl:portType>

<!-- binding style set to 'Document' and use to 'literal' -->
...
```

Запрос SOAP

```
<soap:envelope>
  <soap:body>
    <FirstValueElement>1</FirstValueElement>
    <SecondValueElement>2</SecondValueElement>
  </soap:body>
</soap:envelope>
```

Ответ SOAP

```
<soap:envelope>
```

```
  <soap:body>
    <ResultElement>3</ResultElement>
  </soap:body>
</soap:envelope>
```

Pros

- Нет кодировки типа XSI
- Возможность проверки тела
- WS-I, соответствующий ограничениям

Cons

- WSDL более сложна из-за дополнительного определения XSD
- Название операции потеряно
- WS-I разрешает только одному ребенку в теле SOAP

Документ / литерал (завернутый)

WSDL:

```
...
<types>
  <schema>
    <element name="AddValues">
      <complexType>
        <sequence>
          <element name="FirstValue" type="xsd:int" />
          <element name="SecondValue" type="xsd:int" />
        </sequence>
      </complexType>
    </element>
    <element name="AddValuesResponse">
      <complexType>
        <sequence>
          <element name="ResultValue" type="xsd:int" />
        </sequence>
      </complexType>
    </element>
  </schema>
</types>

<wsdl:message name="AddValues">
  <wsdl:part name="in" element="AddValues" />
</wsdl:message>
<wsdl:message name="AddValuesResponse">
  <wsdl:part name="out" element="AddValuesResponse" />
</wsdl:message>

<wsdl:portType name="CalculatorEndpoint">
  <wsdl:operation="AddValues">
    <wsdl:input message="AddValues" />
    <wsdl:output message="AddValuesResponse" />
  </wsdl:operation>
</wsdl:portType>
```

```
<!-- binding style set to 'Document' and use to 'literal' -->
...
```

Запрос SOAP

```
<soap:envelope>
  <soap:body>
    <AddValues>
      <FirstValue>1</FirstValue>
      <SecondValue>2</SecondValue>
    </AddValues>
  </soap:body>
</soap:envelope>
```

Ответ SOAP

```
<soap:envelope>
  <soap:body>
    <AddValuesResponse>
      <Result>3</Result>
    </AddValuesResponse>
  </soap:body>
</soap:envelope>
```

Pros

- Нет кодировки типа XSI
- Возможность проверки тела
- Название операции и элементов, доступных в запросе и ответе
- Соответствие WS-I

Cons

- WSDL более сложна из-за дополнительного определения XSD

UDDI

`Universal Description, Discovery and Integration (UDDI)` - это открытая отраслевая инициатива, выпущенная в 2000 году, которая служит реестром на основе желтых страниц на базе XML для веб-сервисов, который помогает находить службы, которые решают конкретные задачи. Чтобы найти подходящую услугу, служба сначала должна быть зарегистрирована в реестре веб-службы, например UDDI.

UDDI работает с обменом сообщениями SOAP и предоставляет доступ к документам WSDL, которые могут использоваться для вызова фактической веб-службы.

UDDI предоставляет критерии поиска, такие как

- бизнес-идентификатор

- Наименование фирмы
- расположение бизнеса
- бизнес-категория
- тип услуги по имени
- URL-адреса обнаружения

Однако большой недостаток нынешнего UDDI заключается в том, что он позволяет использовать только один критерий в инструкции поиска. Некоторые разработчики, таким образом, модулировали свои реализации UDDI, чтобы разрешить запросы, порождающие несколько UDDI одновременно, а затем агрегировать возвращенные результаты.

На практике, однако, UDDI часто не используется. Некоторые даже говорят, что UDDI мертв, так как IBM, Microsoft и SAP закрыли свои UDDI-сервисы в 2005 году .

Дальнейшие примечания:

SOAP / WSDL предоставляют широкий спектр поддержки инструментальных средств, а также позволяют динамически генерировать классы-заглушки для клиентов и серверов, поскольку тип сообщений и обмен данными хорошо определяется через встроенные или связанные схемы XSD.

В то время как WSDL 2.0 имеет меньше накладных расходов на определение веб-сервисов, некоторые языки все еще не приняли новый стандарт. Т.е. в Java популярные инструменты, такие как `wsimport` (от Oracle / Sun) или `wsdl2java` (от Apache CXF), не могут правильно обрабатывать описания WSDL 2.0. Поэтому по соображениям совместимости по-прежнему рекомендуется использовать WSDL 1.1. Если вам нужно разработать службу SOAP на основе WSDL 2.0 на Java, посмотрите на `wsdl2java` из проекта Apache Axis2 .

Однако более популярными в настоящее время являются либо службы HTTP на основе API, которые смешивают вызовы HTTP-операций с чистыми понятными для пользователя URI и определенными настройками протокола, чтобы выполнить свою работу, службы на основе REST , которые полностью соответствуют фактическим рекомендациям, или собственные протоколы байтового уровня, такие как OFTP2 .

SOAP по-прежнему полезен в настоящее время, если вы не можете напрямую сопоставить свою задачу с ресурсами, например, базовыми службами HTTP / REST, поскольку задача выполнения представляет собой естественное действие или должна определять определенную семантику транзакций. Кроме того, если у вас нет ресурсов для определения или реализации собственного протокола, вы, вероятно, лучше используете SOAP. SOAP особенно полезен, если вам приходится иметь дело с оркестровкой, поскольку описание WSDL в сочетании с UDDI позволяет динамически комбинировать службы.

Java-клиент для службы погоды с открытым исходным кодом webservice доступен на http://www.webserviceX.NET

```
 package com.test.ws.example;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;

/*
 * WSDL url :  http://www.webservicex.com/globalweather.asmx?WSDL
 * Endpoint URL: http://www.webservicex.com/globalweather.asmx   */

   public class WSClient {

  public static void main(String args[]) {

   try {
       SOAPConnectionFactory soapConnectionFactory = SOAPConnectionFactory.newInstance();
       SOAPConnection soapConnection = soapConnectionFactory.createConnection();

       // Generate SOAP request XML

       MessageFactory messageFactory = MessageFactory.newInstance();
       SOAPMessage soapMessage = messageFactory.createMessage();
       MimeHeaders header = soapMessage.getMimeHeaders();
       header.setHeader("SOAPAction", "http://www.webserviceX.NET/GetCitiesByCountry");
       SOAPPart soapPart = soapMessage.getSOAPPart();
       SOAPEnvelope envelope = soapPart.getEnvelope();
       envelope.addNamespaceDeclaration("web", "http://www.webserviceX.NET");
       SOAPBody soapBody = envelope.getBody();
       SOAPElement soapBodyElem = soapBody.addChildElement("GetCitiesByCountry", "web");
       SOAPElement soapBodyElem1 = soapBodyElem.addChildElement("CountryName", "web");
       soapBodyElem1.addTextNode("INDIA");
       soapMessage.saveChanges();
       soapMessage.writeTo(System.out);


       // Call webservice endpint
       String url = "http://www.webservicex.com/globalweather.asmx";
       SOAPMessage soapResponse = soapConnection.call(soapMessage, url);
       Source sourceContent = soapResponse.getSOAPPart().getContent();

       // Print SOAP response
       TransformerFactory transformerFactory = TransformerFactory.newInstance();
       Transformer transformer = transformerFactory.newTransformer();
       System.out.println("Response SOAP Message \n");
       StreamResult result = new StreamResult(System.out);
       transformer.transform(sourceContent, result);
```

```
        soapConnection.close();

    } catch (Exception e) {

        e.printStackTrace();
    }
}
```

}

Создание простого веб-сервиса и клиентов с помощью JAX-WS (документ / литерал)

Это каталог проекта.

1. **Интерфейс конечной точки службы**

Сначала мы создадим интерфейс конечной точки службы. Аннотации javax.jws.WebService `@WebService` определяют класс как конечную точку веб-службы.

```
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
import javax.jws.soap.SOAPBinding.Use;

// Service Interface with customize targetNamespace
@WebService(targetNamespace = "http://hello-soap/ws")
@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL) //optional
public interface HelloSoap {

    @WebMethod String getHelloSoap(String name);

}
```

2. **Реализация конечной точки службы (SEI)**

Затем мы создадим реализацию конечных точек службы. Мы создадим явный интерфейс, добавив `endpointInterface` элемент в `@WebService` аннотаций в классе реализации. Вот несколько правил 28.1.1 Требования конечной точки JAX-WS, которые должны следовать

конечным точкам JAX-WS. Метод getHelloSoap возвращает приветствие клиенту с именем, переданным ему.

```
import javax.jws.WebService;

// Customized Service Implementation (portName,serviceName,targetNamespace are optional)

@WebService(portName = "HelloSoapPort", serviceName = "HelloSoapService",
endpointInterface = "com.wonderland.hellosoap.HelloSoap", targetNamespace = "http://hello-
soap/ws")
public class HelloSoapImpl implements HelloSoap {

    @Override
    public String getHelloSoap(String name) {
        return "[JAX-WS] Hello : " + name;
    }

}
```

3. **Издатель конечных точек веб-службы**

```
import javax.xml.ws.Endpoint;

public class HelloSoapPublisher {

    public static void main(String[] args) {
        // creating web service endpoint publisher
        Endpoint.publish("http://localhost:9000/ws/hello-soap", new HelloSoapImpl());
    }

}
```

4. В следующих шагах мы будем запускать `HelloSoapPublisher.java` качестве приложения java. Затем мы рассмотрим файл WSDL, запросив URL-адрес `http://localhost:9000/ws/hello-soap?wsdl` в веб-браузере.

   HTTP: // локальный: 9000 / WS / привет-мыло WSDL

Если формат данных XML отображается в веб-браузере, мы готовы перейти к следующему шагу.

> **Замечания:**
> Если вы получаете какое-то сообщение об ошибке, возможно, вам нужно использовать инструмент `wsgen` для создания необходимых переносных артефактов JAX-WS. Здесь мы не `wsgen` инструмент `wsgen` .

5. **Клиент веб-сервиса**

Заключительный шаг, мы создадим клиента, который обращается к нашей опубликованной службе.

```
import java.net.URL;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

public class HelloSoapClient {

    public static void main(String[] args) throws Exception {

        // create wsdl url
        URL wsdlDocumentUrl = new URL("http://localhost:8000/ws/hello-soap?wsdl");
        QName helloSoapService = new QName("http://hello-soap/ws", "HelloSoapService");
        // create web service
        Service service = Service.create(wsdlDocumentUrl, helloSoapService);
        // get object of pointed service port
        HelloSoap helloSoap = service.getPort(HelloSoap.class);
        // testing request
        System.out.println(helloSoap.getHelloSoap("Soap "));

    }
```

```
}
```

Выход: `[JAX-WS] Hello : Soap`

Примечание. Номер порта изменился на `8000` в нашем клиенте веб-сервисов. Причина в том, что я использовал Eclipse IDE, встроенный инструмент `TCP/IP monitor` для отслеживания сообщений (Дополнительная информация: как трассировать SOAP-сообщение в Eclipse IDE). Для функционального тестирования попробуйте SoapUI | Функциональное тестирование для API SOAP и REST .

Прочитайте Начало работы с мылом онлайн: https://riptutorial.com/ru/soap/topic/5375/начало-работы-с-мылом

глава 2: Потребление веб-службы SOAP

Вступление

В этом разделе должны быть представлены все возможные способы использования веб-службы SOAP.

параметры

параметр	подробности
Название страны	Струны, такие как Великобритания

Examples

Без создания файлов Stub или Java

```
public String getCitiesByCountry(String countryName) throws MalformedURLException, IOException
{

    //Code to make a webservice HTTP request
    String responseString = "";
    String outputString = "";
    String wsURL = "http://www.webservicex.com/globalweather.asmx";// Endpoint of the
webservice to be consumed
    URL url = new URL(wsURL);
    URLConnection connection = url.openConnection();
    HttpURLConnection httpConn = (HttpURLConnection)connection;
    ByteArrayOutputStream bout = new ByteArrayOutputStream();
    String xmlInput =
        "<soap:Envelope xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\">
          <soap:Body>
            <GetCitiesByCountry xmlns=\"http://www.webserviceX.NET\">
              <CountryName>" + countryName + "</CountryName>
            </GetCitiesByCountry>
          </soap:Body>
        </soap:Envelope>"; //entire SOAP Request

    byte[] buffer = new byte[xmlInput.length()];
    buffer = xmlInput.getBytes();
    bout.write(buffer);
    byte[] b = bout.toByteArray();
    String SOAPAction = "http://www.webserviceX.NET/GetCitiesByCountry"; // SOAP action of the
webservice to be consumed
    // Set the appropriate HTTP parameters.
    httpConn.setRequestProperty("Content-Length",
    String.valueOf(b.length));
    httpConn.setRequestProperty("Content-Type", "text/xml; charset=utf-8");
```

```
    httpConn.setRequestProperty("SOAPAction", SOAPAction);
    httpConn.setRequestMethod("POST");
    httpConn.setDoOutput(true);
    httpConn.setDoInput(true);
    OutputStream out = httpConn.getOutputStream();
    //Write the content of the request to the outputstream of the HTTP Connection.
    out.write(b);
    out.close();
    //Ready with sending the request.

    //Read the response.
    InputStreamReader isr = null;
    if (httpConn.getResponseCode() == 200) {
      isr = new InputStreamReader(httpConn.getInputStream());
    } else {
      isr = new InputStreamReader(httpConn.getErrorStream());
    }

    BufferedReader in = new BufferedReader(isr);

    //Write the SOAP message response to a String.
    while ((responseString = in.readLine()) != null) {
        outputString = outputString + responseString;
    }
    //Parse the String output to a org.w3c.dom.Document and be able to reach every node with
the org.w3c.dom API.
    Document document = parseXmlFile(outputString);
    NodeList nodeLst = document.getElementsByTagName("GetCitiesByCountryResult"); // TagName
of the element to be retrieved
    String elementValue = nodeLst.item(0).getTextContent();
    System.out.println(elementValue);

    return elementValue;
}

public Document parseXmlFile(String in) {
    try {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        InputSource is = new InputSource(new StringReader(in));
        return db.parse(is);
    } catch (ParserConfigurationException e) {
        throw new RuntimeException(e);
    } catch (SAXException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Прочитайте Потребление веб-службы SOAP онлайн: https://riptutorial.com/ru/soap/topic/8292/потребление-веб-службы-soap

кредиты

S. No	Главы	Contributors
1	Начало работы с мылом	Alice, Community, Ray, Roman Vottner, ssanrao
2	Потребление веб-службы SOAP	RAS