Распределенные вычислительные системы

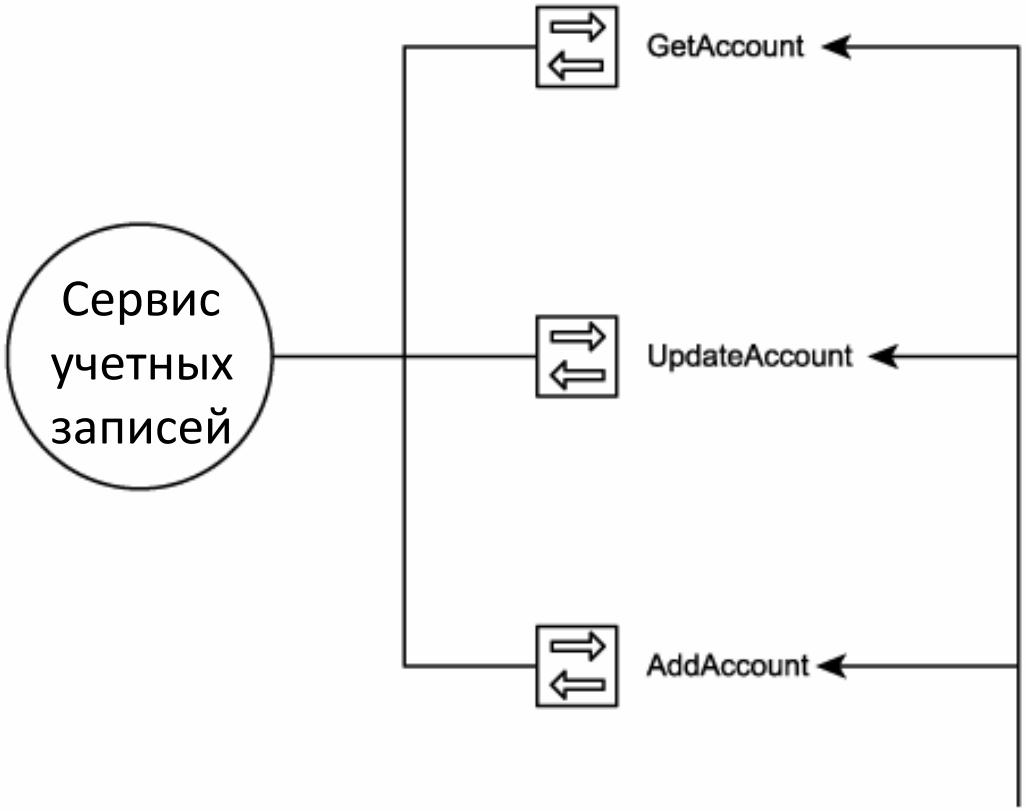
Сервис-ориентированная архитектура

Основные принципы RPC сервисов

1. Сервис должен допускать повторное использование (1)

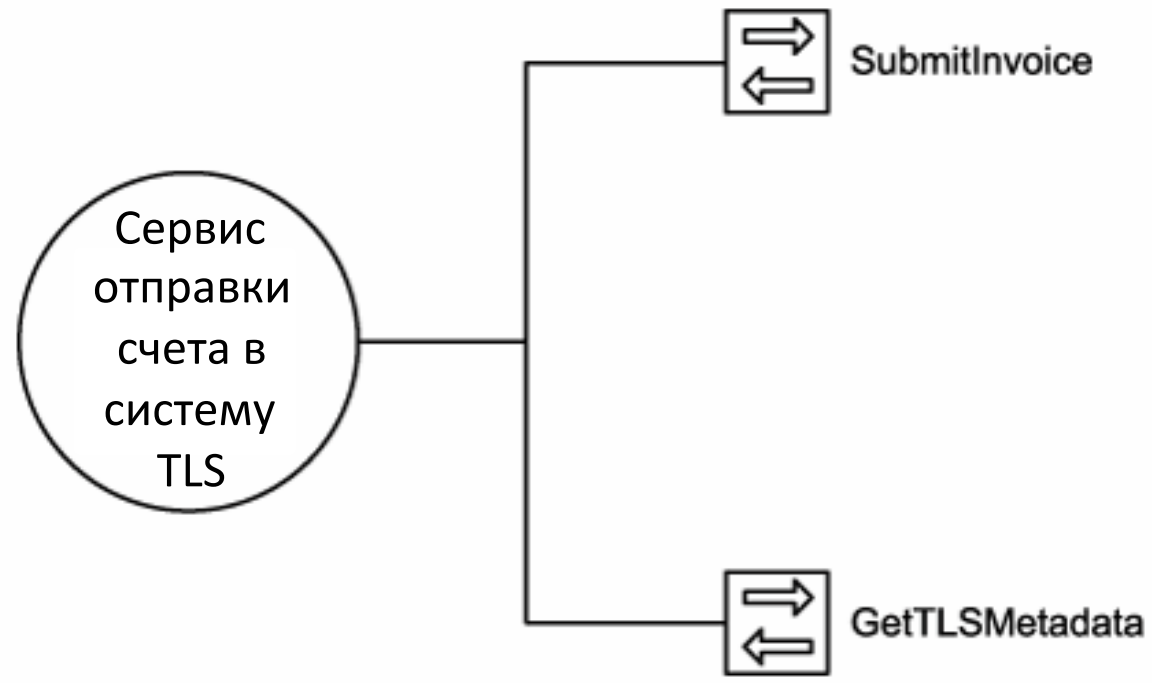
- SOA-системы должны поддерживать повторное использование всех сервисов, независимо от сиюминутных требований к их функциональным особенностям.
  - Это позволит упростить расширение и развитие системы, отказаться от «оберток» над сервисами для их подстройки на решение новых задач.
- Таким образом, каждая операция сервиса должна поддерживать повторное использование

1. Сервис должен допускать повторное использование (2)



Может использоваться несколькими абонентами

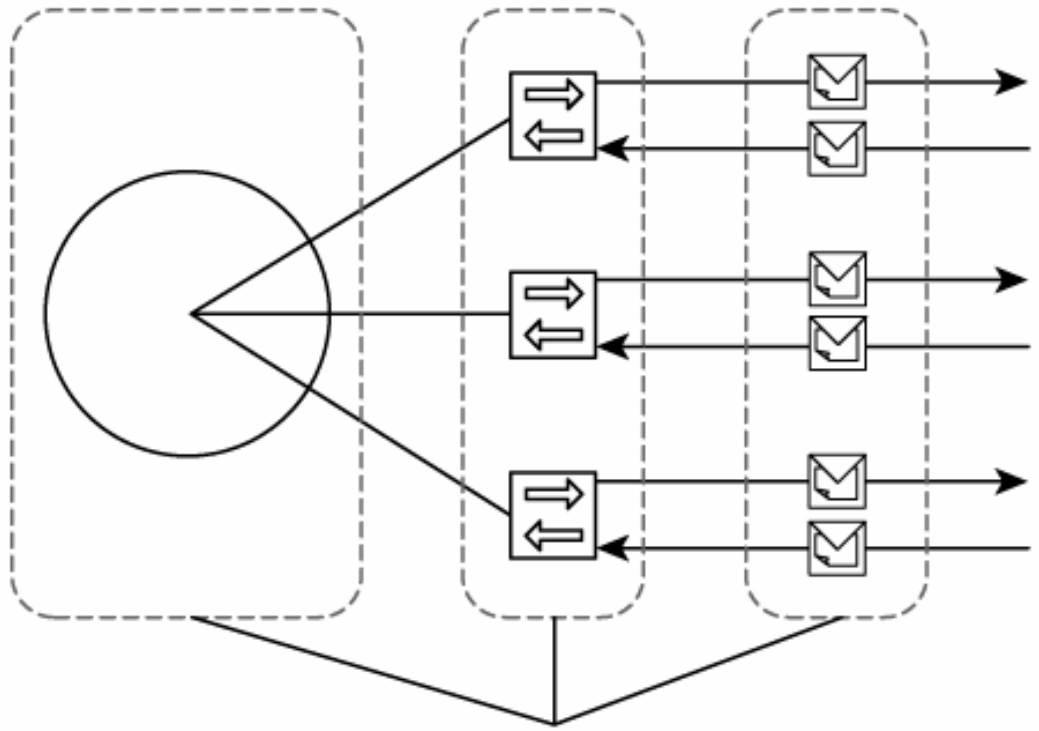
1. Сервис должен допускать повторное использование (3)



2. Сервисы должны обеспечивать формальный контракт использования (1)

- Контракт сервиса предоставляет следующую информацию:
  - Адрес конечной точки (service endpoint)
  - Все операции, предоставляемые сервисом
  - Все сообщения, поддерживаемые каждой операцией
  - Правила и характеристики сервиса и его операций.

2. Сервисы должны обеспечивать формальный контракт использования (2)

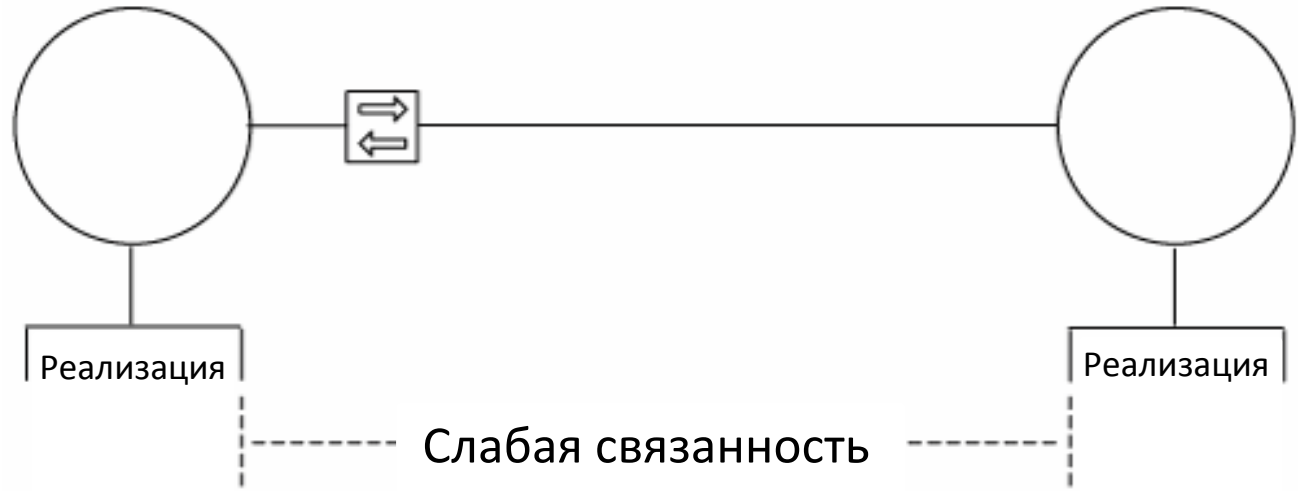


Определяется контрактом сервиса

3. Сервисы должны быть слабосвязанны(1)

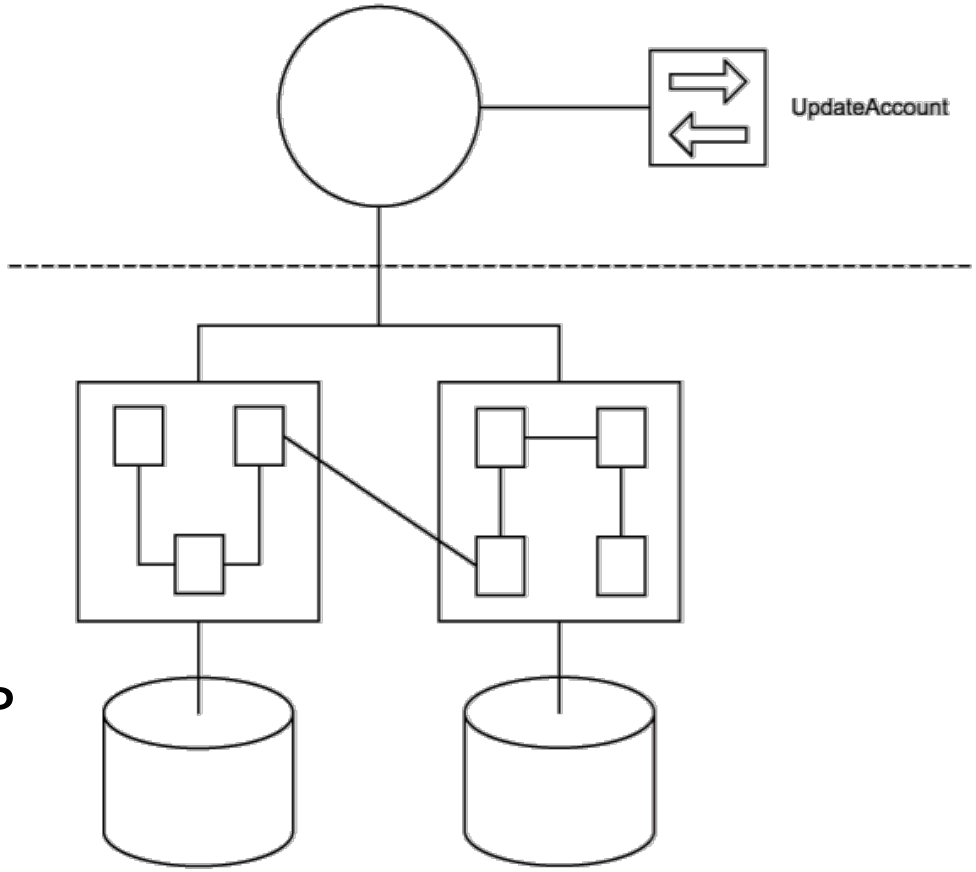
- Необходимо обеспечить целостность системы в рамках развития системы, независимо от пути развития
- Система сервисов является слабо связанной если сервис может приобретать знания о другом сервисе, оставаясь независимым от внутренней реализации логики данного сервиса.

3. Сервисы должны быть слабосвязанны(2)



4. Сервисы должны абстрагировать внутреннюю логику

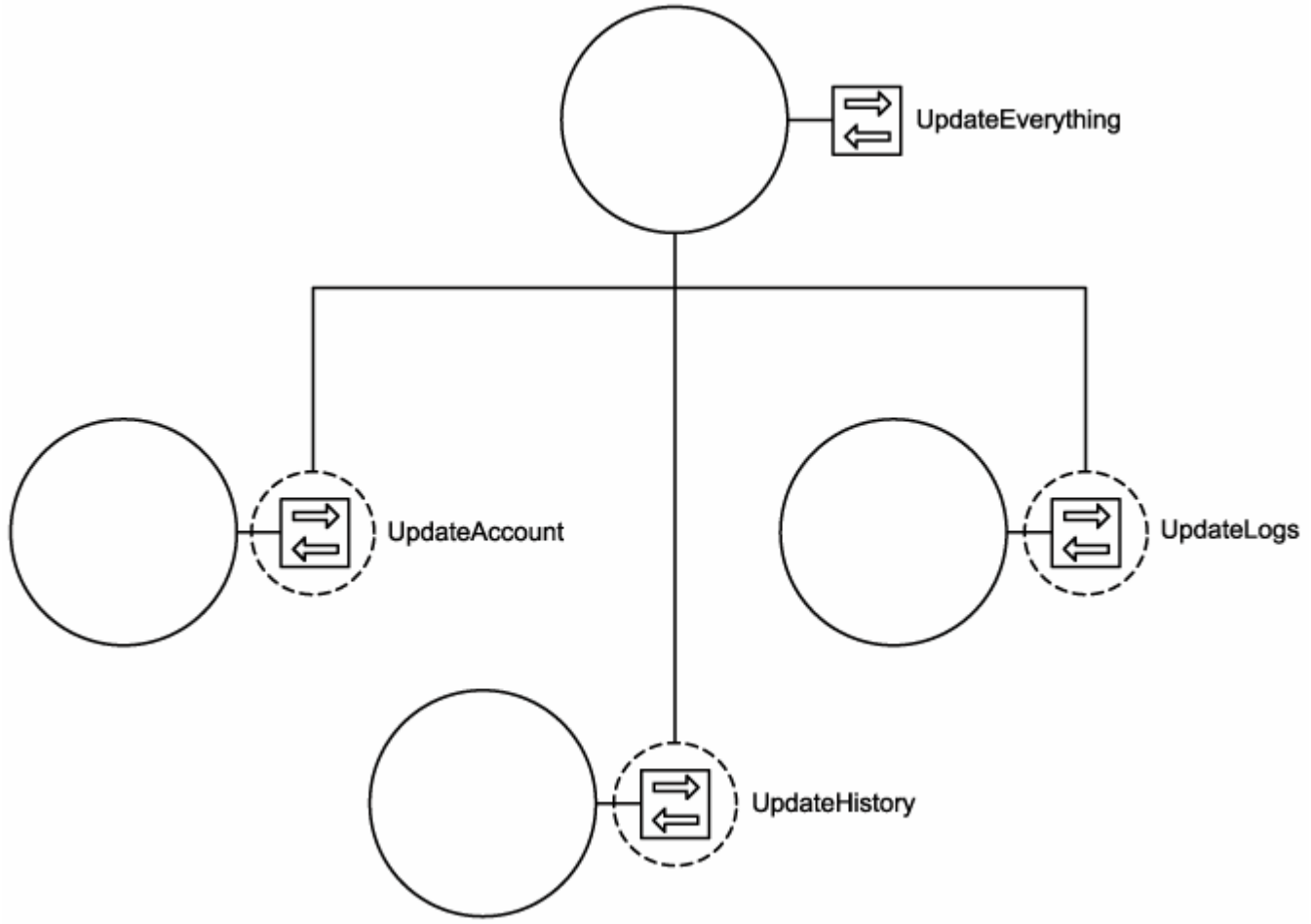
- Каждый сервис должен действовать как «черный ящик»
- Это одно из требований, обеспечивающих слабосвязанность сервисов.



5. Сервисы должны быть совместимы (1)

- Сервис может как самостоятельно реализовывать логику, так и применять другие сервисы для ее реализации
- Сервисы должны быть спроектированы таким образом, чтобы поддерживать возможность их использования в качестве элементов другого сервиса
- Такой процесс называется «Оркестрацией сервисов» (Service orchestration).

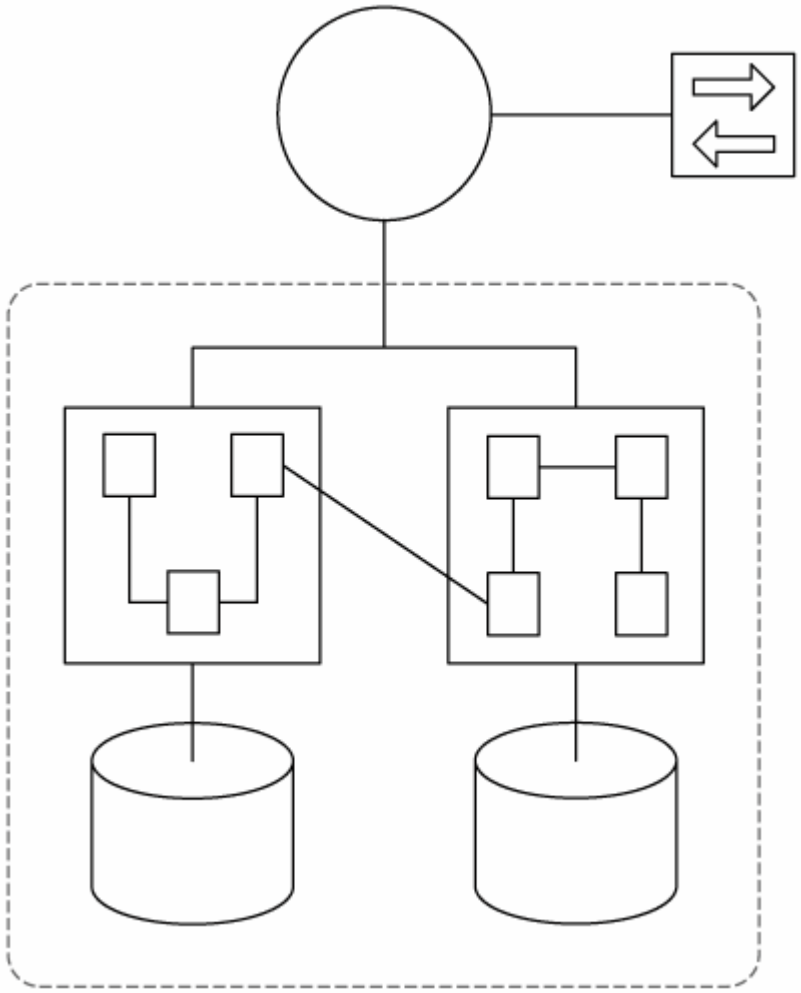
5. Сервисы должны быть совместимы (2)



6. Сервисы должны быть автономны (1)

- Область бизнес-логики и ресурсов, используемых сервисом должны быть ограничены явными пределами
- Вопрос автономности – наиболее важный аргумент при распределении бизнес-логики на отдельные сервисы
- Выделяют 2 типа автономности:
  - *Автономность на уровне сервиса*: границы ответственности сервисов отделены, но они могут использовать общие ресурсы
  - *Чистая автономность:* бизнес-логика и ресурсы находятся под полным контролем сервиса

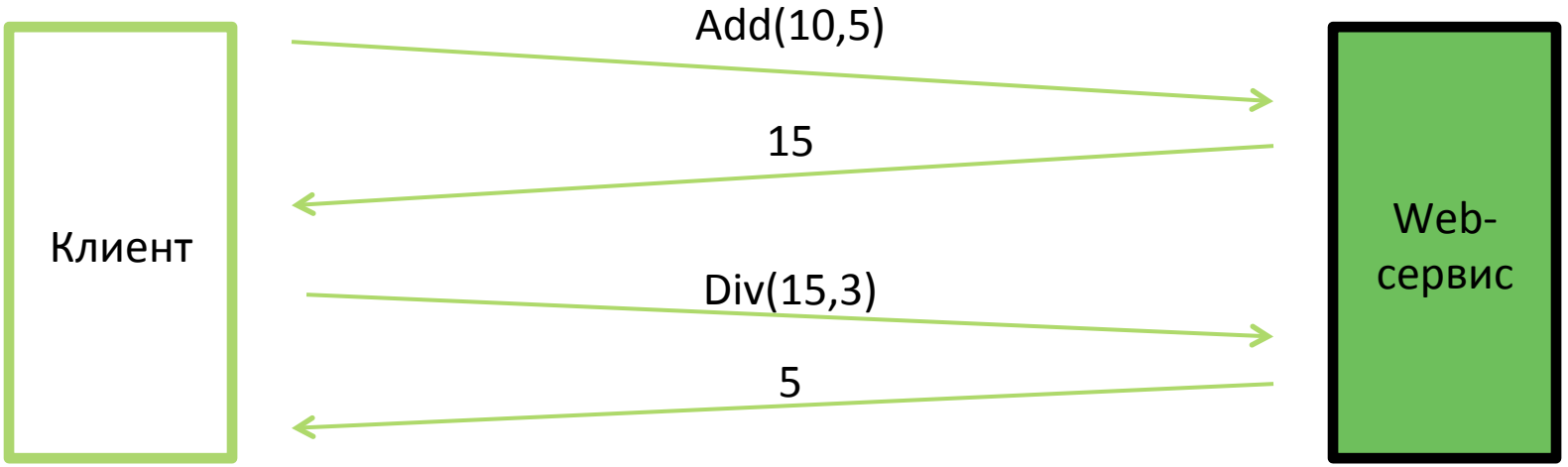
6. Сервисы должны быть автономны (2)



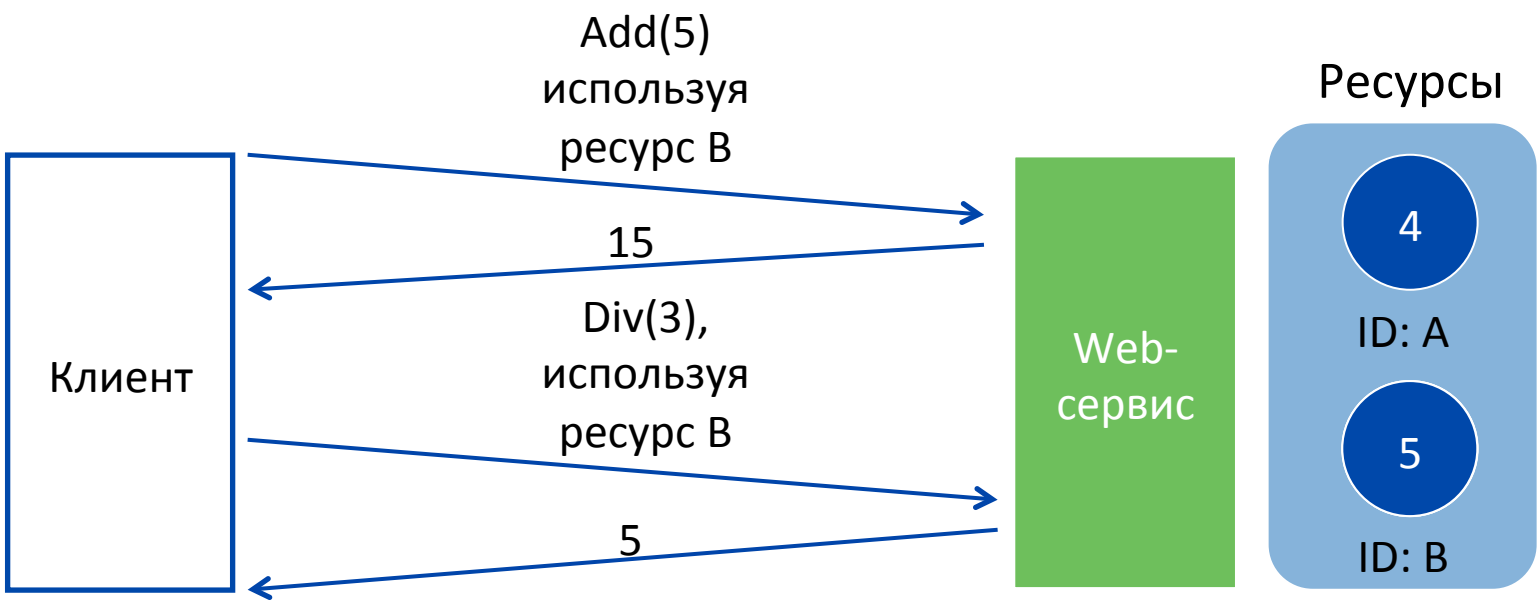
7. Сервисы не должны использовать информацию о состоянии (1)

- «Чистые» сервисы должны значительно ограничивать объем и время хранения информации (в идеале – только на время вычислений)
- Не зависимость от состояния (Statelessness) позволяет повысить возможности масштабируемости и повторного использования сервисов
- Но это очень жесткое ограничение, которое не всегда удается удовлетворить.

7. Сервисы не должны использовать информацию о состоянии (2)



7. Сервисы не должны использовать информацию о состоянии (3)



8. Сервисы должны поддерживать обнаружение

- Обнаружение сервисов позволяет избежать случайного создания избыточного сервиса, обеспечивающего избыточную логику
- Все методы сервиса должны содержать метаданные, описывающие их возможности в системе поиска
- Каждый сервис должен предоставлять как можно больше информации о своих возможностях

Методы организации RPC

- Есть 2 основных подхода для описания контрактов таких сервисов:
  - *XML Веб-сервисы:* наиболее стандартный подход, использующий язык WSDL (Web Services Description Language) и спецификации WS-Policy и WS-Security для определения различных методов авторизации, шифрования и др.
  - Не-XML Веб-сервисы (RPC): это целый спектр различных протоколов RPC, начиная со стандарта JSON-RPC и заканчивая множеством пропиретарных или открытых стандартов от различных поставщиков.

JSON RPC

JSON-RPC

- JSON-RPC (JavaScript Object Notation Remote Procedure Call — JSON-вызов удаленных процедур) — протокол удаленного вызова процедур, использующий JSON для кодирования сообщений.
- JSON-RPC работает отсылая запросы к серверу, реализующему протокол. Клиентом обычно является программа, которой нужно вызвать метод на удаленной системе. Все передаваемые данные — простые объекты, сериализованные в JSON

JSON-RPC – Запрос и ответ

- **Запрос** должен содержать три обязательных свойства:
  - *method* — Строка с именем вызываемого метода.
  - *params* — Массив объектов, которые должны быть переданы методу, как параметры.
  - *id* — Значение любого типа, которое используется для установки соответствия между запросом и ответом.

- **Ответ** должен содержать следующие свойства:
  - *result* — Данные, которые вернул метод. Если произошла ошибка во время выполнения метода, это свойство должно быть установлено в null.
  - *error* — Код ошибки, если произошла ошибка во время выполнения метода, иначе null.
  - *id* — То же значение, что и в запросе, к которому относится данный ответ.

JSON-RPC – пример

--> обозначает данные, отправленные серверу (запрос)

<-- обозначает ответ

```
...
--> {"method": "postMessage", "params": ["Hello all!"], "id": 99}
<-- {"result": 1, "error": null, "id": 99}
<-- {"method": "handleMessage", "params": ["user1", "Ok!"], "id": null}
<-- {"method": "handleMessage", "params": ["user3", "gotta go"], "id": null}
--> {"method": "postMessage", "params": ["I have a question:"], "id": 101}
<-- {"method": "userLeft", "params": ["user3"], "id": null}
<-- {"result": 1, "error": null, "id": 101}
...
```

Реализация JSON-RPC

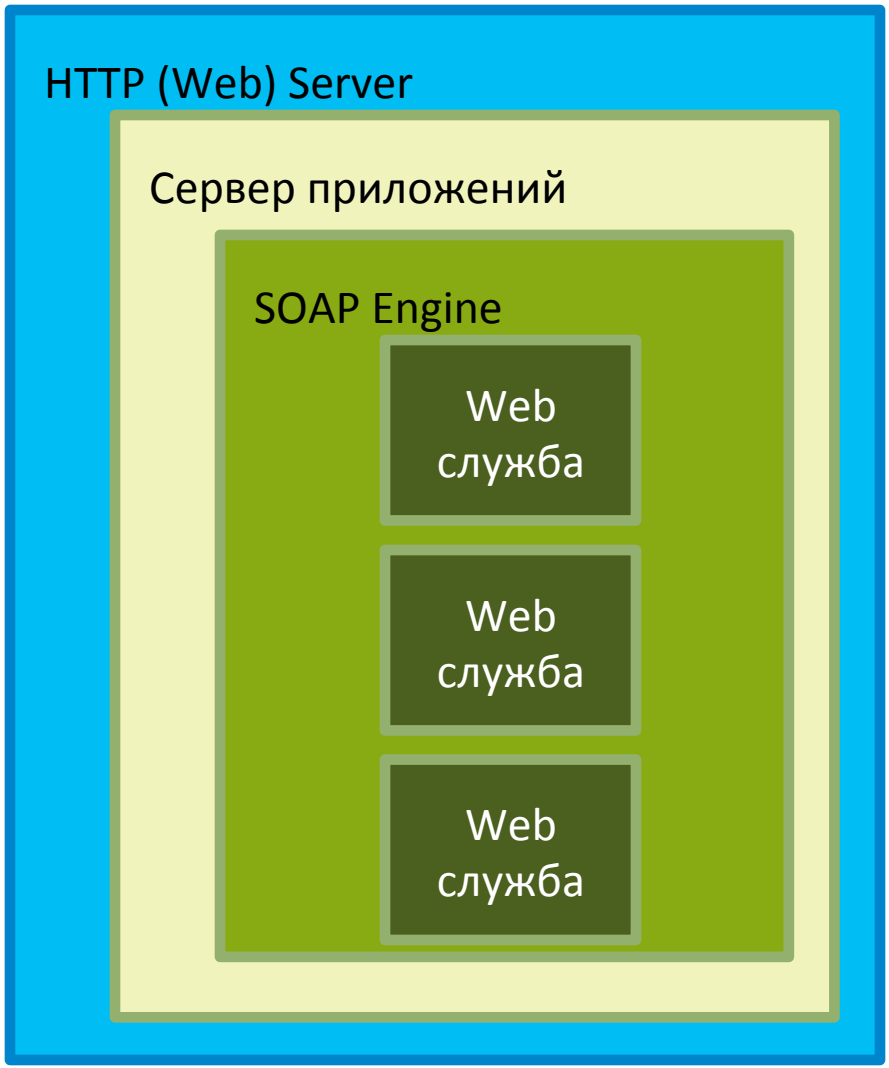
Название	версия JSON-RPC	Описание	Язык(и), Платформы
JSON-RPC.NET	2.0	Быстрый JSON-RPC сервер. Поддерживает сокеты, именованные сокеты и HTTP с помощью ASP.NET требует Mono или .NET Framework 4.0.	.NET
Jayrock	1.0	Серверная реализация JSON-RPC 1.0 для версий 1.1 и 2.0 Microsoft .NET Framework.	.NET
jsonrpc-c	2.0	Реализация JSON-RPC через TCP сокеты (только сервер).	C
libjson-rpc-cpp	2.0	C++ JSON-RPC фрэймворк, поддерживающий клиентскую и серверную стороны через HTTP.	C++
Phobos	2.0	Реализация для Qt/C++. Абстрагирует уровень передачи данных (готовые к использованию классы для TCP и HTTP).	C++
qjsonrpc	2.0	Реализация для Qt/C++. Поддерживает соединения между сообщениями и QObject слотами (как QDBus, qxtrpc) использует новые JSON классы, включенные в Qt5.	C++
JSON Toolkit	2.0	Реализация на Delphi	Delphi
go/net/rpc	?	Реализация JSON-RPC стандартной библиотеки Go	Go
jsonrpc4j	2.0	Java реализация JSON-RPC 2.0 поддерживает как сокеты, так и HTTP соединение.	Java
json-rpc	1.0	Базовая Java/JavaScript реализация, которая хорошо интегрируется в Android/Servlets/Standalone Java/JavaScript/App-Engine приложения.	Java / JavaScript
jpoxy	2.0	Простая Java реализация JSON-RPC созданная для упрощения реализации доступа к POJOs через сырой RPC фрэймворк.	Java
JSON Service	2.0	JSON-RPC серверная реализациясс поддержкой Service Mapping Description. Хорошо интегрируется с Dojo Toolkit и Spring Framework.	Java
JSON-RPC 2.0	2.0	Легкая Java library для разбора и сериализации JSON-RPC 2.0 сообщений (open source). Несколько реализация на сайте. (Base, Client, Shell, ...)	Java
java-json-rpc	2.0	Реализация для J2EE серверов.	Java
lib-json-rpc	2.0	Реализация servlet, client, JavaScript	Java
simplejsonrpc	2.0	Простой JSON-RPC 2.0 Servlet, обслуживающий методы класса.	Java
gson-rmi	2.0	Легковесный, независимый от способа передачи RMI фрэймворк разработанный для распределенных вычислений.	Java
jsonrpcjs	2.0	JavaScript клиентская библиотека для JSON-RPC 2.0, Не имеет зависимостей.	JavaScript
easyXDM	2.0	Библиотека для cross-domain соединений с поддержкой RPC. Поддерживает все браузеры postMessage, nix, frameElement, window.name, и FIM, очень проста в использовании.	JavaScript
Dojo Toolkit	1.0+	Предоставляет поддержку JSON-RPC	JavaScript
Pmrpc	2.0	JavaScript библиотека для использования в HTML5 браузерах. Реализация JSON-RPC, используя HTML5 postMessage API для передачи сообщений.	JavaScript
qooxdoo	2.0	Имеет JSON-RPC реализацию с опциональными бэк-эндами на Java, PHP, Perl и Python.	JavaScript, Java, PHP, PERL, & Python
JSON-RPC Реализация на JavaScript	2.0	Поддерживает JSON-RPC через HTTP и TCP/IP.	JavaScript
jabsorb	2.0	Легковесный Ajax/Web 2.0 JSON-RPC Java фрэймворк, расширяющий JSON-RPC протокол дополнительной ORB функциональностью, такой как поддержка циклических зависимостей.	JavaScript, Java
The Wakanda platform	2.0	Поддерживает JSON-RPC 2.0 клиент внутри Ajax Framework и JSON-RPC 2.0 сервис в серверном JavaScript	JavaScript
Deimos	1.0+2.0	Серверная реализация для Node.js/JavaScript.	JavaScript
Barracuda Web Server	2.0	Barracuda Web Server's интегрированный	Lua
DeferredKit	1.0	Поддерживает JSON-RPC 1.0 клиент.	Objective-C
Demiurgic	2.0	JSON-RPC 2.0 клиент для Objective-C	Objective-C
Oxen iPhone Commons JSON components	1.0	JSON-RPC 1.0 клиент для Objective-C	Objective-C
objc-JSONRpc	2.0	Objective-c JSON RPC клиент. Поддерживает уведомления, простые вызовы и множественные вызовы.	Objective-C
JSON::RPC	2.0	Реализация сервера JSON RPC 2.0	Perl
json-rpc-perl6	2.0	Клиент и сервер.	Perl 6
php-json-rpc	2.0	Простая PHP реализация JSON-RPC 2.0 через HTTP клиента.	PHP
JQuery JSON-RPC Server	2.0	JSON-RPC сервер, специально сделанный для работы с Zend Framework JSON RPC Server.	PHP, JavaScript
jsonrpc2php	2.0	PHP5 JSON-RPC 2.0 базовый класс и пример сервера	PHP
tivoka	1.0 + 2.0	Универсальный клиент/серверная JSON-RPC библиотека для PHP 5+.	PHP
junior	2.0	Client/server библиотека для JSON-RPC 2.0	PHP
json-rpc-php	2.0	Client/server библиотека для JSON-RPC 2.0	PHP
JSONRpc2	2.0	Реализация с «dot magic» для PHP (= поддержка группировки методов и разделения точками)	PHP
GetResponse jsonRPCClient	2.0	Объектно-ориентированная реализация клиента	PHP
zoServices	2.0	PHP, Node.js и JavaScript реализация JSON-RPC 2.0	PHP, JavaScript, Node.js
json-rpc2php	2.0	Серверная и клиентская реализация для PHP. Содержит JavaScript клиент, использующий jQuery	PHP, JavaScript
jsonrpc-php	2.0	JSON-RPC реализация для PHP	PHP
php-json-rpc	2.0	Реализация JSON-RPC 2.0.	PHP
Django JSON-RPC 2.0	2.0	JSON-RPC сервер для Django	Python
Pyjamas		JSON-RPC клиентская реализация.	Python
Zope 3	1.1	JSON RPC реализация для Zope 3	Python
jsonrpclib	2.0	JSON-RPC клиентский модуль для Python.	Python
tornadorpc	2.0	Поддерживает JSON-RPC требует Tornado web server.	Python
tinyrpc	2.0	Поддерживает JSON-RPC через TCP, WSGI, ZeroMQ и др. Разделяет передачу данных от обработки сообщений, может работать без пересылки сообщений.	Python
jsonrpc	2.0	JSON-RPC 2.0 для Python + Twisted.	Python
bjsonrpc	1.0+	Реализация через TCP/IP (асинхронная, двунаправленная)	Python
Barrister RPC	2.0	JSON-RPC реализация клиента и сервера	Python, Ruby, JavaScript (Node.js + web browser), PHP, Java
pyramid_rpc	2.0	Гибкая JSON-RPC реализация интегрированная в Pyramid web application. Работает с Pyramid's системой авторизации.	Python
rjr	2.0	JSON-RPC через TCP/UDP, HTTP, WebSockets, AMQP, и прочие.	Ruby (EventMachine) сервер с Ruby и JavaScript клиентами.
jimson	2.0	Клиент и сервер для Ruby	Ruby
JSON-RPC Objects	1.0+	Реализация только объектов (без клиента и сервера).	Ruby
JSON-RPC RT	2.0	Полная поддержка JSON-RPC 2.0 через TCP.	Windows Runtime (WinRT)
XINS	2.0	С Версии 2.0, поддерживает JSON и JSON-RPC.	XML

XML (SOAP) Веб-сервисы

XML Веб-сервисы

- XML Веб-сервисы – это основанная на языке XML платформо-независимая технология, поддерживающая разработку распределенных приложений.
- XML Веб-сервисы обеспечивает создание независимых, масштабируемых, слабосвязанных приложений.
- Они основаны на протоколах HTTP, XML, XSD, SOAP, WSDL.
- Реализации SOAP/ WSDL:
  - The Java API for XML Web Services (JAX-WS)
  - Apache CXF
  - Microsoft’s Windows Communication Foundation (WCF)
  - …

Серверная часть Web служб



Borja Sotomayor . The Globus Toolkit 4 Programmer's Tutorial. University of Chicago Department of Computer Science

WSDL: Web Service Definition Language

- WSDL – это стандартный XML-документ, описывающий фундаментальные свойств Web службы, как то:
  - **Что это** – описание методов, предоставляемые Web службой;
  - **Как осуществляется доступ** – формат данных и протоколы;
  - **Где он расположен** – сетевой адрес службы (URI).

Адресация Web служб

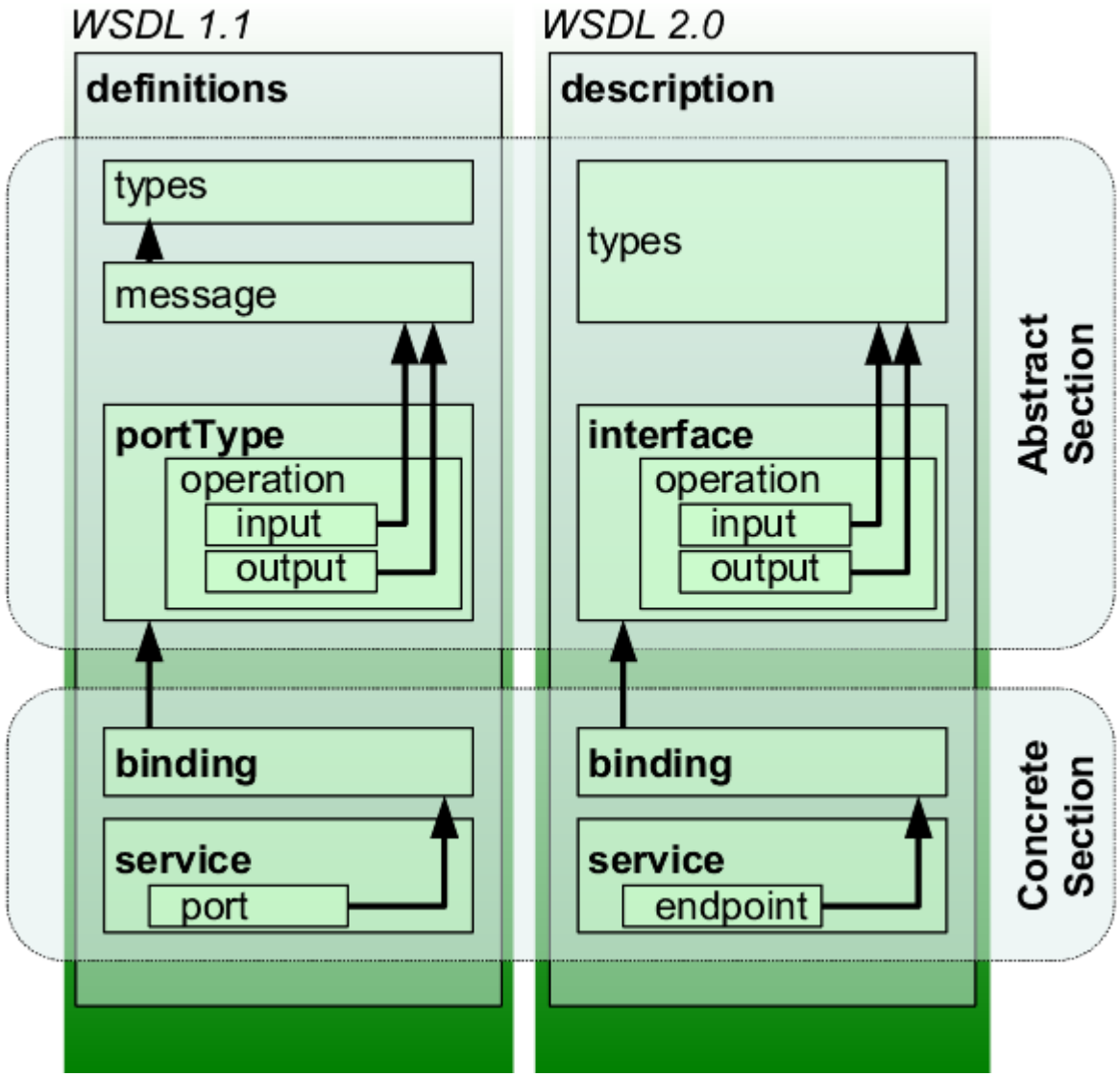
- Адрес Web службы – это стандартный URI (Uniform Resource Identifer).

```
http://live.capescience.com/ccx/GlobalWeather
```

  - BTW: URL (Uniform Resource Locator) является одним из видов URI, и является прародителем URI.

http://www.service-repository.com/

WSDL 1.1 VS WSDL 2.0



Пример Web службы

- Простой пример Web службы (Java)

```
@WebService
public class MyMath
     {
          @WebMethod
          public int squared(int x)
          {
               return x * x;
          }
     }
```

Элементы WSDL-документа

- *Блок types* – типы данных, используемые Web-службой
- *Блок message* – сообщения, используемые Web-службой
- *Блок portType* – методы, предоставляемые Web-службой
- *Блок binding* – протоколы связи, используемые Web-службой

WSDL-документ

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://DefaultNamespace"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://DefaultNamespace"
  xmlns:intf="http://DefaultNamespace"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:message name="squaredRequest">
       <wsdl:part name="in0" type="xsd:int"/>
    </wsdl:message>
    <wsdl:message name="squaredResponse">
       <wsdl:part name="squaredReturn" type="xsd:int"/>
    </wsdl:message>
    <wsdl:portType name="MyMath">
       <wsdl:operation name="squared" parameterOrder="in0">
          <wsdl:input message="impl:squaredRequest" name="squaredRequest"/>
          <wsdl:output message="impl:squaredResponse" name="squaredResponse"/>
       </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="MyMathSoapBinding" type="impl:MyMath">
       <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/ soap/http"/>
       <wsdl:operation name="squared">
          <wsdlsoap:operation soapAction=""/>
          <wsdl:input name="squaredRequest">
             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
                namespace="http://DefaultNamespace" use="encoded"/>
          </wsdl:input>
          <wsdl:output name="squaredResponse">
             <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
                namespace="http://DefaultNamespace" use="encoded"/>
          </wsdl:output>
       </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MyMathService">
       <wsdl:port binding="impl:MyMathSoapBinding" name="MyMath">
          <wsdlsoap:address location="http://localhost:8080/axis/testaccount/MyMath"/>
       </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Namespaces

messages - сообщения

portTypes - методы

binding - связи

Определение службы

Порты WSDL <portType>

- Элемент <portType> является наиболее важным элементом WSDL.
- Он определяет саму Web-службу, предоставляемые ей операции и используемые сообщения.
- Можно сравнить с библиотекой функций, в которой указаны входные параметры и результаты работы функции.

Пример блока <portType>

```
<wsdl:portType name="MyMath">
  <wsdl:operation name="squared"
parameterOrder="in0">
    <wsdl:input message="impl:squaredRequest"
        name="squaredRequest"/>
    <wsdl:output message="impl:squaredResponse"
        name="squaredResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

Сообщения WSDL <message>

- Элемент <message> определяет элементы данных операции.
- Каждое сообщение может содержать одну или несколько частей. Эти части можно сравнить с параметрами вызываемых функций в традиционных языках программирования.

Пример блока <message>

```
<wsdl:message name="squaredRequest">
  <wsdl:part name="in0" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="squaredResponse">
  <wsdl:part name="squaredReturn"
type="xsd:int"/>
</wsdl:message>
```

Связи WSDL <binding>

- Элемент <binding> определяет формат сообщения и детали протокола для каждого порта.

- Отвечает за то, каким образом элементы абстрактного интерфейса в блоке <portType> преобразуются в массивы информации в формате протоколов взаимодействия, например SOAP.

Пример блока <binding>

```
<wsdl:binding name="MyMathSoapBinding" type="impl:MyMath">
  <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/ soap/http"/>
  <wsdl:operation name="squared">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="squaredRequest">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
          namespace="http://DefaultNamespace" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="squaredResponse">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/ encoding/"
          namespace="http://DefaultNamespace" use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Блоки <port> и <service>

- Данные блоки определяют, где находится служба.

- port – описывает расположение и способ доступа к конечной точке
- service – именованная коллекция портов

Пример блока <service>

```
<wsdl:service name="MyMathService">
  <wsdl:port binding="impl:MyMathSoapBinding"
      name="MyMath">
    <wsdlsoap:address
location="http://localhost:8080/axis/myaccount/
MyMath"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```