

Сервисная архитектура приложений. Веб-сервисы

Алексей Островский

Физико-технический учебно-научный центр НАН Украины

21 мая 2015 г.

Сервис-ориентированная архитектура

Определение

Сервис-ориентированная архитектура (англ. *service-oriented architecture, SOA*) — парадигма программирования, в которой для обеспечения модульности применяются распределенные слабо связанные компоненты (*сервисы*), взаимодействующие с помощью стандартизованных протоколов.

Характеристики сервисов:

- ▶ модульность — сервис представляет логически связанные функции в определенной предметной области с заданными входами и выходами;
- ▶ автономность — отсутствие наблюдаемых для пользователей зависимостей;
- ▶ сокрытие реализации — рассматривается как «черный ящик».

Преимущества и недостатки SOA

Достоинства:

- ▶ открытость, стандартизация протоколов доступа;
- ▶ поддержка параллелизма, масштабируемость (напр., за счет прозрачных для клиента балансировщиков нагрузки);
- ▶ отказоустойчивость.

Недостатки:

- ▶ зависимость от состояния сетевых соединений;
- ▶ дополнительные вычислительные ресурсы, ПО и затраты для поддержки масштабирования;
- ▶ проблемы обеспечения безопасности данных, качества обслуживания и т. п.

Веб-сервисы

Определение

Веб-сервис (англ. *web service*) — программная система с возможностью взаимодействия с другими программами через сеть, обладающая заданным интерфейсом и протоколом сообщений для обмена данными.

Характеристики веб-сервиса:

- ▶ **интерфейс** веб-сервиса (\simeq интерфейс компонента): определяемые операции, типы входных и выходных данных;
- ▶ **формат спецификации интерфейса**: на основе формального представления (языка спецификации) или неформального описания;
- ▶ используемый **протокол передачи данных** (HTTP, UDP, ...);
- ▶ **формат представления данных**: на основе XML, JSON, простого текста, ...

Разработка веб-сервисов

Цели разработки:

- ▶ минимизация количества обращений к сервису;
- ▶ скрытие состояния сервиса (хранение состояния — задача клиента; состояние может передаваться в сообщениях).

Этапы разработки:

1. определение функциональности;
2. описание операций и сообщений;
3. имплементация;
4. тестирование;
5. развертывание.

Классификация веб-сервисов

Типы веб-сервисов:

- ▶ **Утилитарные** — реализующие функциональность общего назначения, которая может использоваться в различных предметных областях другими сервисами.

Пример: конвертер валюты.

- ▶ **Бизнес-сервисы** — реализующие функциональность, специфичную для предметной области.

Пример: вычисление кредитного рейтинга.

- ▶ **Координационные** — комплексные бизнес-процессы, зачастую реализуемые с помощью более простых веб-сервисов.

Пример: управление работой магазина (прием заказов, инвентаризация, оплата, ...).

Ориентация веб-сервисов:

- ▶ сущности — поведение, аналогичное объектам в ООП (напр., манипуляции с БД);
- ▶ задания — выполнение действий без привязки к сущностям предметной области.

SOAP-сервисы

Определение

Веб-сервис в узком смысле, **SOAP-сервис** — веб-сервис, в котором спецификация интерфейса и передача данных определены стандартами W3C.

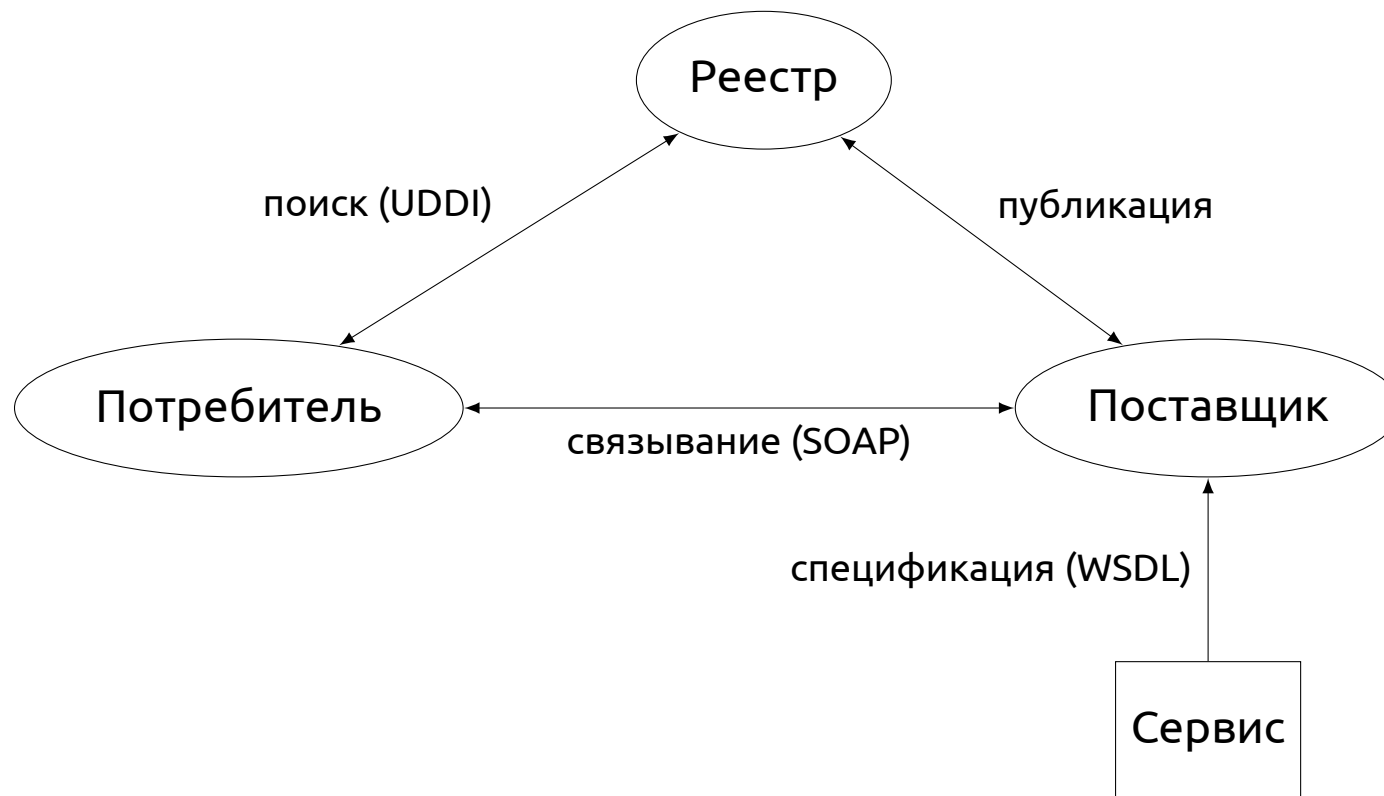


Схема взаимодействия с веб-сервисом

Стандарты SOAP-сервисов

Основные стандарты:

- ▶ **SOAP** — протокол передачи данных для вызова операций, определенных интерфейсом сервиса;
- ▶ **WSDL** — стандарт для определения интерфейса сервиса;
- ▶ **UDDI** (universal description, discovery, and integration) — стандарт для обнаружения активных сервисов в сети (расположение WSDL-описания интерфейса и т. п.);
- ▶ **WS-BPEL** — стандарт для высокоуровневого описания программ, использующих веб-сервисы.

Вспомогательные стандарты:

- ▶ защита данных (WS-Security);
- ▶ транзакции в распределенных сервисах (WS-Transactions);
- ▶ контроль передачи сообщений (WS-Reliable Messaging), ...

WSDL

Определение

WSDL (web service description language) — язык спецификации интерфейса веб-сервисов, использующий XML.

Содержимое спецификации:

- ▶ Операции, предоставляемые сервисом (\simeq методы в ООП), соответствующие входные и возвращаемые данные;
- ▶ формат сообщений для взаимодействия с сервисом;
- ▶ (необязательно) типы данных, используемые в сообщениях;
- ▶ определение конкретных протоколов доступа к операциям (с помощью SOAP или других методов).

Понятия WSDL 2.0

- ▶ **Интерфейс** — набор операций для веб-сервиса.
- ▶ **Операция** — определение способа обращения к сервису; ~ вызов функции или метода в ЯП.
- ▶ **Типы данных** — определения используемой структуры входных / выходных сообщений для операций с помощью XML Schema.
- ▶ **Привязка** (англ. *binding*) — спецификация способа доступа к определенному интерфейсу, в частности, протокол связи.
- ▶ **Конечная точка** (англ. *endpoint*) — адрес доступа к веб-сервису (чаще всего — простой HTTP-адрес), соответствующий некоторой привязке.
- ▶ **Сервис** — набор конечных точек, обладающих общим интерфейсом.

Структура файлов WSDL 2.0

```
1 <description xmlns="http://www.w3.org/ns/wsd1"
2     xmlns:ws="http://example.com/service"
3     targetNamespace="http://example.com/service">
4     <types>
5         <!-- Определения типов данных с помощью XML schema. -->
6     </types>
7     <!-- Определение интерфейса веб-сервиса. -->
8     <interface name="Foo">
9         <!-- Описание операций. -->
10    </interface>
11    <!-- Привязки. -->
12    <binding name="SoapBinding" interface="ws:Foo"
13        type="http://www.w3.org/ns/wsd1/soap" ...>
14        <!-- Ссылки на операции, определенные в интерфейсе. -->
15    </binding>
16    <!-- Декларация веб-сервиса. -->
17    <service name="FooService" interface="ws:Foo">
18        <!-- Точки доступа к веб-сервису. -->
19    </service>
20 </description>
```

Пример: типы данных в WSDL 2.0

```
1 public interface IntegerSequence {
2     public BigInteger get(String sequence, int index);
3 }

1 <types>
2     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3         targetNamespace="http://example.com/service">
4         <xs:element name="getRequest">
5             <xs:complexType><xs:sequence>
6                 <xs:element name="sequence" type="xs:string"/>
7                 <xs:element name="index" type="xs:int" />
8             </xs:sequence></xs:complexType>
9         </xs:element>
10        <xs:element name="getResponse">
11            <xs:complexType><xs:sequence>
12                <xs:element name="number" type="xs:integer" />
13            </xs:sequence></xs:complexType>
14        </xs:element>
15    </xs:schema>
16 </types>
```

Пример: описание сервиса в WSDL 2.0

```
1 <!-- Интерфейс сервиса. -->
2 <interface name="IntegerSequence">
3     <operation name="get" pattern="http://www.w3.org/ns/wsdل/in-out">
4         <input messageLabel="In" element="ws:getRequest"/>
5         <output messageLabel="Out" element="ws:getResponse"/>
6     </operation>
7 </interface>
8 <!-- Привязка при помощи протокола SOAP. -->
9 <binding name="SoapBinding" interface="ws:IntegerSequence"
10     type="http://www.w3.org/ns/wsdل/soap"
11     xmlns:soap="http://schemas.xmlsoap.org/wsdل/soap/"
12     soap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
13     <operation ref="ws:get" />
14 </binding>
15 <!-- Описание сервиса. -->
16 <service name="IntSeqService" interface="ws:IntegerSequence">
17     <endpoint name="SoapEndpoint"
18         binding="ws:SoapBinding"
19         address="http://www.example.com/service/soap/" />
20 </service>
```

SOAP

Определение

SOAP (simple object access protocol) — протокол для обмена структурированными данными с веб-сервисами через сеть (напр., поверх HTTP-соединения).

Сообщение сервису:

- ▶ заголовок сообщения — нефункциональные характеристики запроса (приоритетность, время обработки, ...);
- ▶ тело сообщения — список операций веб-сервиса и соответствующих параметров.

Ответное сообщение:

- ▶ тело сообщения — список с результатами выполнения операций;
- ▶ отказы — информация об отказах при проведении операций.

Пример SOAP-сообщения

Запрос:

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2     xmlns:ws="http://example.com/service">
3     <soapenv:Body>
4         <ws:get>
5             <ws:sequence>fib</ws:sequence>
6             <ws:index>100</ws:index>
7         </ws:get>
8     </soapenv:Body>
9 </soapenv:Envelope>
```

Ответ:

```
1 <soapenv:Envelope
2     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
3     <soapenv:Body>
4         <response xmlns="http://example.com/service">
5             <number>354224848179261915075</number>
6         </response>
7     </soapenv:Body>
8 </soapenv:Envelope>
```

BPEL

Определение

WS-BPEL (web service business process execution language) — язык на основе XML для описания бизнес-процессов, координирующих веб-сервисы.

Базовые инструкции:

- ▶ ветвление (**if** — **elseif** — **else**);
- ▶ цикл (**while**);
- ▶ последовательность действий (**sequence**);
- ▶ параллельные действия (**flow**).

Возможности:

- ▶ обмен данными с веб-сервисами, извлечение сведений из ответов с помощью XPath;
- ▶ синхронизация параллельных действий;
- ▶ обработка событий и исключительных ситуаций.

Достоинства и недостатки BPEL

Достоинства:

- ▶ высокий уровень абстракции;
- ▶ не зависит от парадигмы программирования (ООП, структурное программирование, ...);
- ▶ ориентация на специфичную для веб-сервисов функциональность (параллельные запросы, разбор данных, ...);

Недостатки:

- ▶ чрезмерная абстрактность побуждает к созданию дополнений для BPEL, несовместимых между собой (что противоречит сути стандарта);
- ▶ отсутствие встроенной поддержки новых технологий (WSDL 2.0, REST-сервисов, ...);
- ▶ централизованная модель управления.

Разработка с SOAP-сервисами

Способы разработки сервисов:

- ▶ top-down — вначале разрабатывается WSDL-описание сервиса, затем на его основе — реализация на ЯП;
- ▶ bottom-up — WSDL-описание генерируется на основе готовых интерфейсов и классов.

Вспомогательные инструменты: обработка поступающих запросов и их трансляция в вызовы методов имплементации (напр., при помощи Apache Axis в Java EE).

Разработка клиента:

- ▶ автоматическая генерация интерфейса и клиентского стаба на основе WSDL-описания сервиса;
- ▶ клиентский стаб позволяет обращаться к сервису как к локальному объекту.

Пример: SOAP-сервис в JavaEE

```
1 @Stateless @WebService(  
2     serviceName = "IntSequence",  
3     targetNamespace = "http://example.com/int-sequence/")  
4 public class IntSequence {  
5     @WebMethod  
6     public List<Description> listSequences() { /* ... */ }  
7  
8     @WebMethod  
9     public Description getDetails(String seq) throws IntSequenceException {  
10         if (/* Последовательность не зарегистрирована */) {  
11             throw new IntSequenceException(/* ... */);  
12         }  
13         // Вернуть информацию о последовательности  
14     }  
15  
16     @WebMethod  
17     public BigInteger getDetails(String seq, int index) {  
18         // Вернуть член последовательности  
19     }  
20 }
```

REST

Определение

Передача репрезентативного состояния (англ. *representational state transfer, REST*) — архитектура распределенных приложений, предназначенная для создания масштабируемых веб-сервисов, которая определяется как набор ограничений.

Задача

Соответствуют ли операции сервиса методам одного объекта?

Если да, то:

- ▶ как обрабатывать одновременные запросы?
- ▶ как масштабировать сервис?

Решение: отсутствие состояния сервиса; каждая операция выполняется независимо от других (но может модифицировать данные, с которыми работает сервис).

REST

Ограничения REST-архитектуры:

- ▶ модель «клиент — сервер» для разделения ответственности;
- ▶ отсутствие хранимого состояния при взаимодействии клиента и сервера;
- ▶ кэшируемость запросов к веб-сервисам (согласно спецификации HTTP-протокола);
- ▶ прозрачная многослойная архитектура (напр., для подключения балансировщиков нагрузки);
- ▶ унифицированный интерфейс сервисов:
- ▶ доступ к ресурсам с помощью различных URI-адресов;
- ▶ режим обработки возвращенных данных определяется в ответе (напр., как спецификатор MIME).

Стандарты в REST

NB. REST не определяет стандартов для взаимодействия, определения интерфейса и т. п. Веб-сервис на основе SOAP теоретически может удовлетворять ограничениям REST-архитектуры.

Часто используемые стандарты:

Протокол передачи данных:	HTTP
Идентификация операции и параметров:	с помощью URI (<code>http://example.com/api/add/2,3</code>) и / или параметров HTTP;
Возвращаемые данные:	XML, JSON, plain text (может определяться в запросе при помощи параметра HTTP Accept);
Спецификация интерфейса:	неформальная, с помощью документации на API; определение допустимых операций для ресурсов согласно HATEOAS .

Сравнение SOAP- и REST-сервисов

Преимущества SOAP-сервисов:

- ▶ стандартизация всех аспектов сервисов;
- ▶ наличие вспомогательных технологий (безопасность информации, транзакции, ...)

Преимущества REST-сервисов:

- ▶ отсутствие дополнительной нагрузки, связанной с использованием «тяжелых» протоколов (SOAP, WSDL);
- ▶ более высокая скорость разработки за счет использования неявных соглашений (англ. *convention over configuration*);
- ▶ легкость доступа и создания клиентов.

HTTP-доступ к REST-сервисам

Метод	Коллекция ресурсов (напр., http://example.com/api/ books)	Отдельный ресурс (напр., http://example.com/api/ books/1000)
GET	получение списка ресурсов (возможно, с доп. информацией)	получение представления запрашиваемого ресурса
PUT	замена коллекции целиком	замена или (при отсутствии) создание нового ресурса с заданным URI
POST	создание нового ресурса в коллекции	не используется
DELETE	удаление коллекции целиком	удаление запрашиваемого ресурса

Пример интерфейса REST-сервиса

Сервис: целочисленные последовательности (напр., числа Фибоначчи).

Базовый URL: `http://example.com/api/`

► **GET** `http://example.com/api/`

Возвращает список зарегистрированных целочисленных последовательностей в формате JSON.

Запрос: GET `http://example.com/api/`

Ответ: HTTP 200 OK; Content-Type: application/json

```
1 [ { "name": "Fibonacci",  
2     "description": "Fibonacci numbers [...]",  
3     "uri": "http://example.com/api/fib"  
4 },  
5 { "name": "Powers of two",  
6     "description": "Integer powers of two [...]",  
7     "uri": "http://example.com/api/pow2"  
8 }, ... ]
```

Пример интерфейса REST-сервиса (продолжение)

► GET `http://example.com/api/<id>`

Возвращает сведения о последовательности `<id>` в формате JSON.

Ошибки:

- Если `<id>` не является зарегистрированной последовательностью, возвращается ошибка HTTP 404 с сообщением в формате `text/plain`.

Запрос: GET `http://example.com/api/fib`

Ответ: HTTP 200 OK; Content-Type: `application/json`

```
1 { "name": "Fibonacci",  
2   "description": "Fibonacci numbers [...]",  
3   "uri": "http://example.com/api/fib",  
4   "maxIndex": 1000000,  
5   "sequence": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] }
```

Запрос: GET `http://example.com/api/non-existent`

Ответ: HTTP 404 Not Found; Content-Type: `text/plain`

Unknown integer sequence identifier: 'non-existent'

Пример интерфейса REST-сервиса (продолжение)

► GET `http://example.com/api/<id>/<index>`

Возвращает член последовательности `<id>` в формате `text/plain`.

Ошибки:

- Если `<id>` не является зарегистрированной последовательностью, возвращается ошибка HTTP 404 с сообщением в формате `text/plain`.
- Если `<index>` не является числом или не выполняются ограничения на индекс, возвращается ошибка HTTP 400 с телом, содержащем описание ошибки в формате `text/plain`.

Запрос: GET `http://example.com/api/fib/100`

Ответ: HTTP 200 OK; Content-Type: `text/plain`

354224848179261915075

Запрос: GET `http://example.com/api/fib/1000000000`

Ответ: HTTP 400 Bad Request; Content-Type: `text/plain`

Index too large: 1000000000

Разработка с REST-сервисами

Способы разработки сервисов:

- ▶ как часть веб-приложений с использованием архитектуры MVC;
- ▶ как составляющая модулей для сервера приложений (напр., [JAX-RS](#) в рамках Java EE).

Способы разработки клиентов:

- ▶ с помощью специализированных API (напр., [JAX-RS Client API](#) для Java EE);
- ▶ при помощи API общего назначения для отправки и обработки HTTP-запросов наподобие [libcurl](#) + средства для сериализации / десериализации данных.

Пример: REST-сервис в JavaEE

```
1 @Path("/")
2 public class IntegerSequenceContainer {
3     @GET @Produces("application/json")
4     public String listSequences() { /* ... */ }
5
6     @GET @Path("{seq}/") @Produces({"application/json","text/plain"})
7     public Response getDetails(@PathParam("seq") String seq) {
8         if (/* Последовательность не зарегистрирована */) {
9             return Response.status(404).entity("Unknown sequence: " + seq)
10                .type("text/plain").build();
11        }
12        // Вернуть информацию о последовательности
13    }
14
15    @GET @Path("{seq}/{index}") @Produces("text/plain")
16    public Response getDetails(
17        @PathParam("seq") String sequenceID,
18        @PathParam("index") int index) {
19        // Вернуть член последовательности
20    }
21 }
```


Выводы

1. Веб-сервисы — один из способов реализации компонентов в распределенных приложениях. Существуют два типа веб-сервисов: SOAP-сервисы и REST-сервисы.
2. SOAP-сервисы используют стандарты доступа и описания интерфейса, предложенные W3C, — SOAP и WSDL, соответственно. SOAP-сервисы громоздки, зато обладают дополнительными возможностями, отсутствующими в REST (напр., адресация). Для создания композиций сервисов есть язык BPEL.
3. REST-сервисы используют для передачи данных средства протокола HTTP (напр., кэширование и определение типа содержимого). В отличие от SOAP-сервисов интерфейс REST-сервисов определяется неформально в документации.

Материалы

 **Sommerville, Ian**
Software Engineering.

Pearson, 2011. — 790 p.

 **Лавріщева К. М.**
Програмна інженерія (підручник).

К., 2008. — 319 с.

Спасибо за внимание!