**Сервис-ориентированная архитектура (SOA)** – парадигма программирования, в которой для обеспечения модульности применяются распределенные слабо связанные компоненты (сервисы), взаимодействующие с помощью стандартизированных протоколов.

Проще говоря – проектирование приложений как набора сервисов, которые могут быть распределены в сети.

Принципы SOA:

* Архитектура не привязана к определенной технологии
* Независимость организации системы от используемых платформ
* Независимость организации системы от применяемых ЯП
* Автономность (слабая связность) – между сервисами отсутствует наблюдаемая для пользователей зависимость.
* Сокрытие реализации – сервисы рассматриваются как «черный ящик». Разработчику необходимо знать лишь какие входные и выходные данные нужны и как вызываются эти программы для исполнения.

Недостатки:

* Низкая производительность. Задержки на упаковку-распаковку данных (в json, xml) и передачу их по сети.
* Зависимость от сетевого соединения.

**Веб-сервис –** программная система с возможностью взаимодействия с другими программами через сеть, обладающая заданным интерфейсом и протоколом сообщений для обмена данными.

Они не предназначены для непосредственной работы с пользователем (человеком). Обмен информацией происходит между системами (приложениями). Позволяют легко расширять функционал приложения, за счет использования внешних сервисов.

Веб-сервис отличается от сайта:

* Наличием API. Можем использовать веб-сервис в своем приложении.
* Интероперабельность (функциональная совметимость) – способность системы взаимодействовать с другими системами без каких-либо ограничений доступа и реализации посредством обмена данными.

**Основные стандарты и протоколы**

**SOAP Веб-сервисы** базируются на трех веб стандартах:

**SOAP (Simple Object Access Protocol)** – протокол обмена структурированными сообщениям в распределенной вычислительной среде. Определяет формат XML-сообщений. Он произошел от XML-RPC.

Основная идея в том, чтобы программы, созданные на различных платформах и языках программирования, могли легко обмениваться данными.

Указываем в xml документе куда обратится, с какими параметрами, какой результат ожидаем. На сервере xml парсится, из него достаются соответствующие параметры, и вызывается соответствующий метод с указанными параметрами.

Результат упаковывается в xml и отправляется обратно клиенту.

Чаще всего передача осуществляется с помощью протокола http, но это не принципиально (можно юзать SMTP, FTP).

Для описания интерфейсов (какие есть методы и параметры у веб сервиса) используется WSDL.

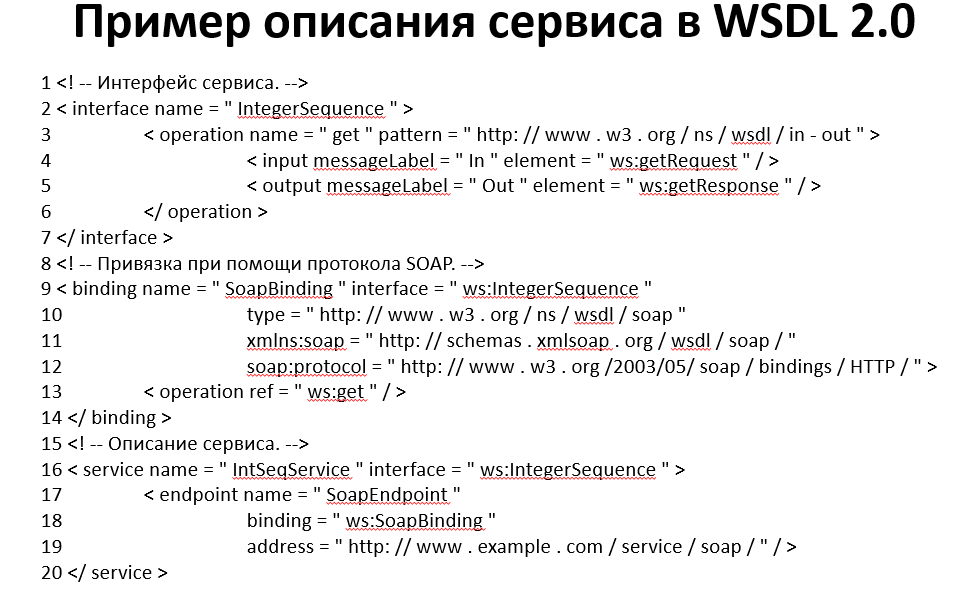
**WSDL (Web Services Description Language)**  - технология, основанная на XML, определяющая интерфейсы веб-сервисов, типы данных и сообщений, а также модели взаимодействия и протоколы связывания.

**Технология UDDI(Universal Description, Discovery and Integration)**  - реестр веб сервисов и механизм поиска. Если нам нужен какой-то веб сервис мы обращаемся к UDDI, который отдает нам WSDL к нужному сервису.



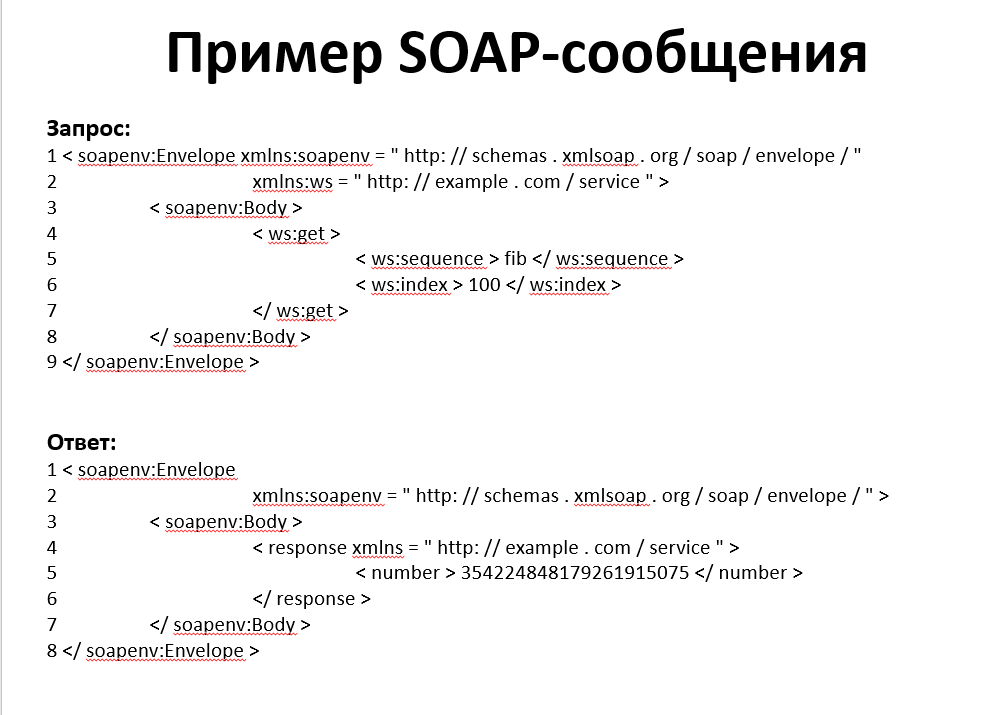
Основные элементы WSDL:

* **<types>** – описание всех типов данных, предлагаемых веб-сервисом.
* **<operation> -**  тег, описывающий функции. То есть он указывает на имя функции и то, как должен выглядеть запрос и ответ.
* Вложенные в operation теги <**input>** и <**output>** содержат информацию о входных и выходных параметрах функции.
* **<message>** - определение формата обмена запросами и ответами данного метода
* **<service>** указывает расположение веб-сервиса.
* **<portType> -** список операций, которые могут быть выполнены с сообщением. Объединяет например сообщение запроса и сообщение ответа в одну операцию.
* **<binding>** указывает как передавать сообщение через SOAP, HTTP GET, HTTP POST.
* **<service> -** характеристики веб-сервиса, как получить доступ к нему, его URL, порты.



SOAP сообщение состоит из следующих элементов:

* **Envelope**. Корневой обязательный элемент. Определяет начало и конец сообщения.
* **Header**. Необязательный элемент – заголовок. Содержит элементы, необходимые для обработки самого сообщения. Например, идентификатор сессии.
* **Body**. Основной элемент, содержит основную информацию сообщения. Обязательный.
* **Fault**. Элемент, содержащий информацию об ошибках, возникающих в процессе обработки сообщения. Необязательный.



**Плюсы**

* В большинстве систем описание wsdl генерируется автоматически.
* Имеет четкую структуру.
* Автоматическая валидация.
* Может работать с любым протоколом транспортного уровня.
* Возможность сохранять состояние объекта.
* Большая безопасность за счет использования WS Security. В заголовке сообщения можно определить ключ безопасности. Если какой-либо элемент в теле SOAP зашифрован, заголовок будет содержать необходимые ключи шифрования.

**Минусы**

* Низкая производительность, так как используется XML, а у него большой объем, сложность парсинга.
* Автоматическая смена описания веб-сервиса может сломать все клиенты.
* Безопасность. Нужно шифровать SOAP-сообщения.

WS-Security определяет, как обеспечивать целостность и конфиденциальность сообщений. В частности, он позволяет использовать различные типы шифрования сообщений, данных и паролей пользователей. Также прикреплять токены для идентификации отправителя.

**XML-RPC (XML Remote Procedure Call) –** протокол вызов удаленных процедур. При обращении к нему посылаем XML-запрос.

**REST**

Rest-сервисы построены на основе архитектурного стиля REST, который основан на теории манипуляции объектами CRUD. Функциональность сводится к манипуляции ресурсами, а не методами. Каждый ресурс определяется своим URI.

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

В отличии от SOAP не подкреплен стандартом, основывается на соглашениях. Используется для простых задач.

* Сервера легкие, достаточно легко масштабируются.

Базовые принципы:

* Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат, аналогичный структуре каталогов.
* Единый интерфейс между компонентами, который включает 4 правила:

1. Каждый ресурс должен однозначно идентифицироваться с помощью URI.
2. Манипуляция над ресурсами через представления (представлением может являться XML либо JSON описание этого ресурса).
3. Само-документируемые сообщения. Запрос и ответ должны хранить в себе всю необходимую информацию для их обработки.
4. *HATEOAS (hypermedia as the engine of application state)*. Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запроса и запрашиваемый URI. В случае необходимости ссылки могут содержаться в теле ответа, чтобы можно было извлечь подобъекты (Например в объекте автомобиль будет URI двигателя)

* Явное использование HTTP методов.

GET – получить ресурс.

POST – создать ресурс.

PUT – изменить ресурс. Нужно отправлять все тело ресурса.

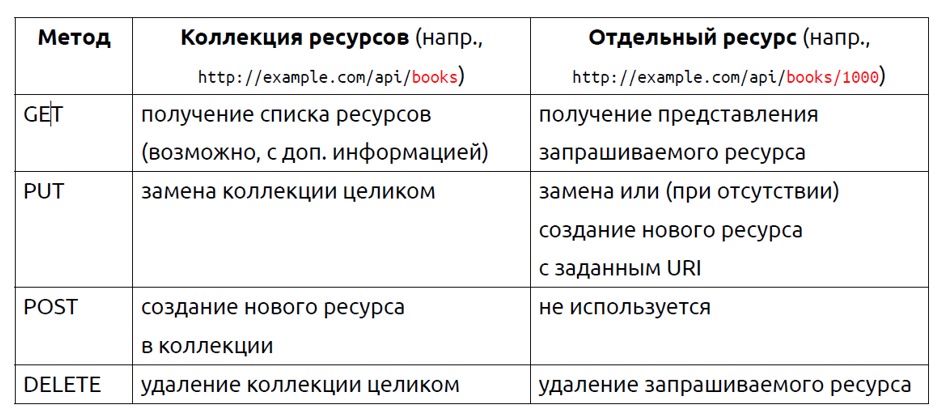
PATCH – частично изменить ресурс.

DELETE – удалить ресурс.

OPTION – можно использовать для описания API REST-сервиса.

* Несохранение состояния. Информация о состоянии сессии хранится у клиента.
* Передача данных в любых форматах (XML, JSON, YAML и др.)
* Архитектура клиент-сервер
* Кэширование
* Разбиение приложения на слои. Каждый слой может видеть компоненты только непосредственно следующего слоя
* Код по требованию, позволяющий серверам расширять функциональность клиента путем передачи исполняемого кода.

Рекомендации по URI



Для каких-то специфических операций, например поиск по имени, а не по id, лучше использовать параметры запроса:

<http://example.com/api/books?name=bookname>

**Плюсы REST**

* Надежность (за счет отсутствия необходимости сохранять информацию о состоянии клиента, которая может быть утеряна);
* Производительность (в т.ч. за счет кэширования);
* Масштабируемость (часть URL может обрабатываться на одних физических серверах, часть на других).
* Простота инетрфейсов.
* Портативность компонентов.
* Легкость внесения изменений (Проще, чем переписывать XML и WSDL).
* Более высокая скорость, за счет использования неявных соглашений. (В soap wsdl надо)

**Минусы**

* Чаще всего привязаны к HTTP (допустим если сеть закрытая, и там нет HTTP то бан).
* Отсутствие хранимого состояния при взаимодействии клиента и сервера (сложность реализации авторизации)
* Не стандартизированы механизмы безопасности, гарантированной доставки сообщений и т.п.
* Кэширование (можем получать устаревшие данные)
* Трудно тестировать

**REST vs SOAP**

* SOAP более тяжелый, меньшая производительность из-за формата XML. Он больше весит, его нужно парсить. Производительность REST также повышается за счет кэширования.
* В SOAP все запросы идут на один урл, а в XML указывается какой метод вызвать. В REST много эндпоинтов, в зависимости от URL и от метода будет выполнено определенное действие.
* REST поддерживает различные форматы: text, JSON, XML, YAML. SOAP – только XML.
* В REST мы работаем с ресурсами, в SOAP с операциями, которые реализуют какую-либо бизнес логику.
* В REST легче вносить изменения, без изменения кода клиента. В случае с SOAP надо заново генерировать WSDL, и переделывать под него клиента.
* SOAP специфицирован, содержит WSDL и меньше подвержен ошибкам. REST использует единообразный интерфейс (URL)
* SOAP поддерживает SSL и WS-security, в то время как REST только SSL. REST идеально подходит для общедоступных URL.
* SOAP поддерживает более надежные транзакции по стандарту ACID.

REST следует использовать если:

* Ограниченные ресурсы и пропускная способность.
* Если нет необходимости хранить состояние между запросами.
* Если есть необходимость в кэшировании большого количества запросов.
* Хотите быстро реализовать функционал, REST написать проще.

SOAP следует использовать если:

* Нужна большая безопасность.
* Нужна большая надежность. SOAP стандартизирован, и все взаимодействия четко определены в WSDL и жестко фиксируются.
* Нужно хранить состояние.