**[SOAP](https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/" \l "soap-use-cases)** is an XML-formatted, highly standardized web communication protocol. Released by Microsoft a year after XML-RPC, SOAP inherited a lot from it. When REST followed, they were first used in parallel, but soon REST won the popularity contest.

SOAP по сути это улучшенный XML-RPC

Если SOAP использовать с протоколом HTTP, то нужно отправлять запрос с методом POST

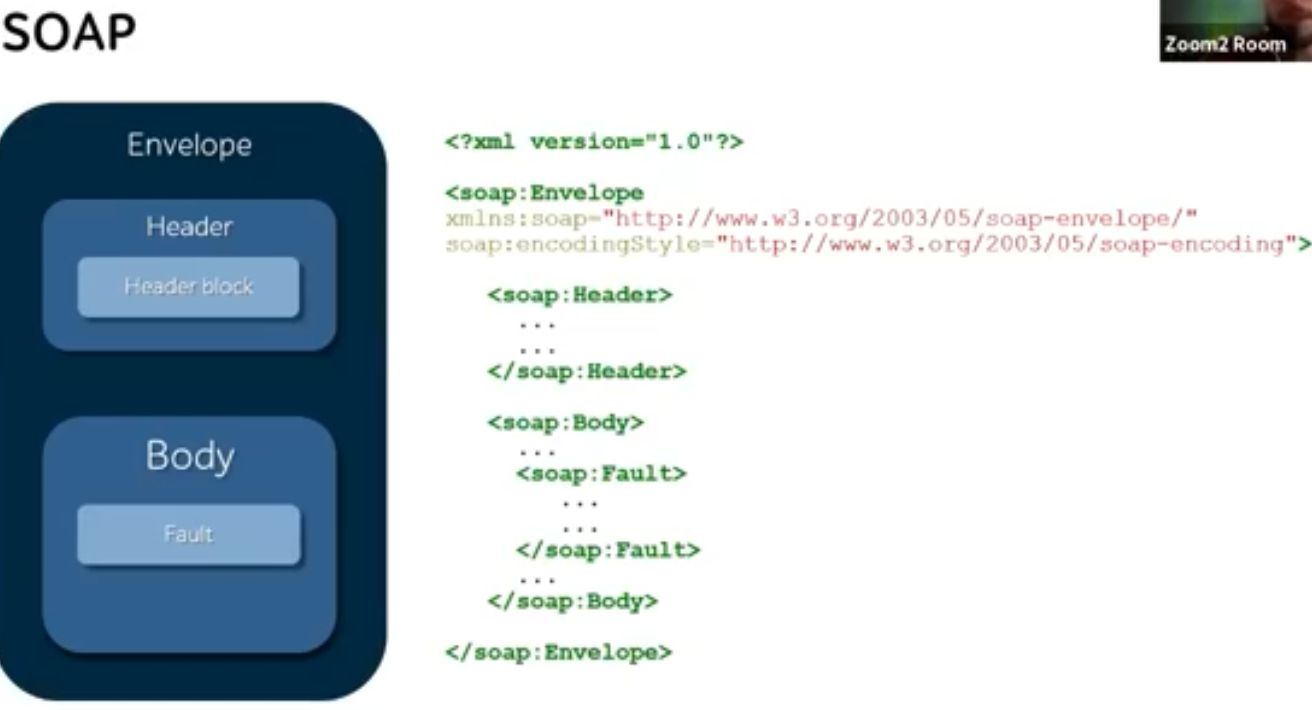


**How SOAP works**

XML data format drags behind a lot of formality. Paired with the massive message structure, it makes SOAP the most verbose API style.

A SOAP message is composed of:

* an envelope tag that begins and ends every message,
* a body containing the request or response
* a header if a message must determine any specifics or extra requirements, and
* a fault informing of any errors that can occur throughout the request processing



**Envelope** – корневой элемент в котором определяется пространство имен

**Header** – содержатся атрибуты сообщения. Заголовки могут содержать токены, информацию безопасности, маршрутизации. Чаще всего через него будет авторизация

**Body** – содержаться сообщения которыми приложения обмениваются

**Fault** – не обязательный элемент, который содержит инфу об ошибке



В отличие от REST запроса, где мы используем методы для указания что сделать с EndPoint. В SOAP действие прописано в тегэ. В данном случае *getProductDetails*



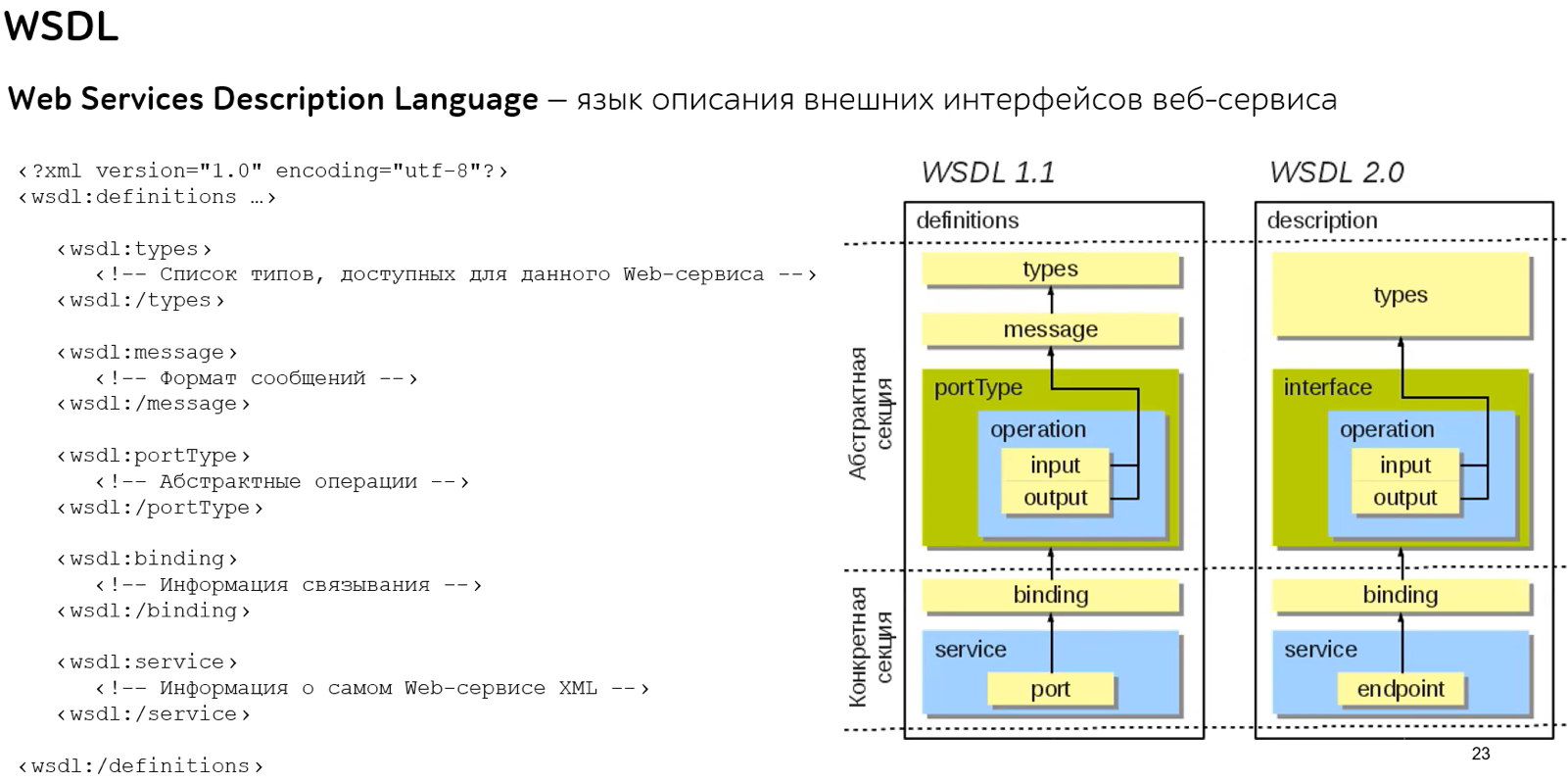


faultcode – ошибка

faultsting – ошибка в легко читаемой для человека форме

faultactor – ресурс который вызывает ошибку

The SOAP API logic is written in Web Service Description Language (WSDL). This API description language defines the endpoints and describes all processes that can be performed. This allows different programming languages and IDEs to quickly set up communication.



По факту нужен для того что бы сказать приложению что умеет делать веб сервис



**definitions** – корневой элемент, определяет название веб службы, пространство имен

* является контейнером всех других элементов.
* указывает, что этот документ называется *HelloService* .
* указывает атрибут *targetNamespace* . *TargetNamespace* — это соглашение XML-схемы, которое позволяет документу WSDL ссылаться на себя. В этом примере мы указали *targetNamespace* http://www.examples.com/wsdl/HelloService.wsdl
* определяет пространство имен по умолчанию: xmlns = http: //schemas.xmlsoap.org/wsdl/. Поэтому все элементы без префикса пространства имен, такие как *message* или *portType* , считаются частью пространства имен WSDL по умолчанию.
* определяет многочисленные пространства имен, которые используются в оставшейся части документа.

**types** – определение типа данных. Содержит описание всех типов данных, предлагаемый веб сервисом

* Элемент *types* описывает все типы данных, используемые между клиентом и сервером.
* WSDL не привязан исключительно к конкретной системе ввода.
* WSDL использует спецификацию XML-схемы W3C в качестве выбора по умолчанию для определения типов данных.
* Если служба использует только простые встроенные типы XML-схемы, такие как строки и целые числа, то элемент *types* не требуется.
* WSDL позволяет определять типы в отдельных элементах, чтобы их можно было повторно использовать в нескольких веб-службах.

**message** – сообщения используемые веб сервисом (данные которыми мы обмениваемся)

* Каждый веб-сервис имеет два сообщения: вход и выход.
* Входные данные описывают параметры для веб-службы, а выходные данные описывают возвращаемые данные из веб-службы.
* Каждое сообщение содержит ноль или более параметров **<part>** , по одному для каждого параметра функции веб-службы.
* Каждый параметр **<part>** связывается с конкретным типом, определенным в элементе контейнера **<types>** .

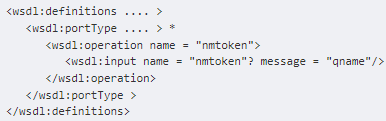
**portType** – Элемент **<portType>** объединяет несколько элементов сообщения, чтобы сформировать полную одностороннюю или двустороннюю операцию.

* **operation** - это абстрактное определение операции для сообщения, например, присвоение имени методу, очереди сообщений или бизнес-процессу, которое примет и обработает сообщение.
* Элемент portType определяет одну операцию, называемую sayHello .
* Операция состоит из одного входного сообщения SayHelloRequest и
* выходное сообщение SayHelloResponse .

Есть 4 вида операции

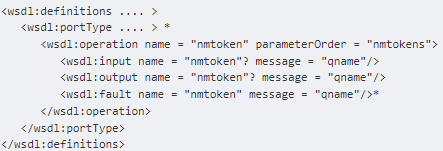
**В одну сторону**

Служба получает сообщение. Следовательно, операция имеет один элемент *ввода*



**Ответ на запрос**

Служба получает сообщение и отправляет ответ. Поэтому операция имеет один элемент *ввода* , за которым следует один элемент *вывода* . Для инкапсуляции ошибок также может быть указан необязательный элемент *неисправности*



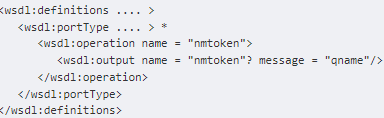
**Запрашивать-ответ**

Служба отправляет сообщение и получает ответ. Поэтому операция имеет один элемент *вывода* , за которым следует один элемент *ввода* . Для инкапсуляции ошибок также может быть указан необязательный элемент *неисправности*



**уведомление**

Служба отправляет сообщение. Следовательно, операция имеет один *выходной* элемент



**binding** – Элемент **<binding>** предоставляет конкретные сведения о том, как операция *portType* будет передаваться по проводам.

* Привязки могут быть доступны через несколько транспортов, включая HTTP GET, HTTP POST или SOAP.
* Привязки предоставляют конкретную информацию о том, какой протокол используется для передачи операций *portType* .
* Привязки предоставляют информацию о том, где находится сервис.
* Для протокола SOAP привязка — это **<soap: binding>** , а транспорт — это сообщения SOAP поверх протокола HTTP.
* Вы можете указать несколько привязок для одного *portType* .

Элемент привязки имеет два атрибута: атрибут *name* и *type* .



Атрибут *name* определяет имя привязки, а атрибут type указывает на порт для привязки, в данном случае это порт «tns: Hello\_PortType».

* **operation name** - Этот элемент указывает на привязку конкретной операции к конкретной реализации SOAP. *Атрибут soapAction* указывает, что HTTP-заголовок SOAPAction будет использоваться для идентификации службы.
* **body** - Этот элемент позволяет указать детали входных и выходных сообщений. В случае HelloWorld элемент body определяет стиль кодирования SOAP и URN пространства имен, связанный с указанным сервисом.

**service** – Элемент **<service>** определяет порты, поддерживаемые веб-службой. Для каждого из поддерживаемых протоколов существует один элемент порта. Служебный элемент представляет собой набор портов.

* гдеполучить доступ к услуге
* через какой порт получить доступ к веб-сервису
* как определяютсякоммуникационные сообщения.

Реквесты идут к URL. Что бы получить WSLD, нужно добавить к URL - **?WSLD**

<http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL>

SOAP supports both stateful and stateless messaging. In a stateful scenario, the server stores the received information that can be really heavy. But it’s justified for operations involving multiple parties and complex transactions.

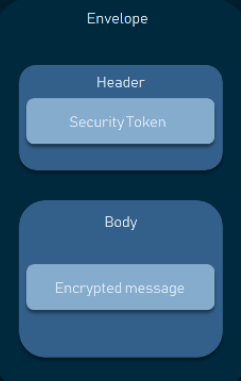
### SOAP pros

**Language- and platform-agnostic.**The built-in functionality to create web-based services allows SOAP to handle communications and make responses language- and platform-independent.

**Bound to a variety of transport protocols.**SOAP is flexible in terms of transfer protocols to accommodate for multiple scenarios.

**Built-in error handling.**SOAP API specification allows for returning the Retry XML message with error code and its explanation

**A number of security extensions.**Integrated with the WS-Security protocols, SOAP meets an enterprise-grade transaction quality. It provides privacy and integrity inside the transactions while allowing for encryption on the message level.



**Remembers State**  - received information is stored on a server, so next request can be based on the previous one

### SOAP cons

These days, many developers shudder at the idea of having to integrate a SOAP API for several reasons.

**XML only.**SOAP messages contain a lot of metadata and only support verbose XML structures for requests and responses.

**Heavyweight.**Due to the large size of XML-files, SOAP services require a large bandwidth.

**Narrowly specialized knowledge.** Building SOAP API servers requires a deep understanding of all protocols involved and their highly restricted rules.

**Tedious message updating.**Requiring additional effort to add or remove the message properties, rigid SOAP schema slows down adoption.

### SOAP use cases

Right now, the SOAP architecture is most commonly used for internal integration within enterprises or with their trusted partners.

**Highly standardized operations: billing, navigation, facilities.** All use cases where you have to eliminate any kind of misinterpretation are a good fit for SOAP communication

**Bank transactions and payment systems**. Firstly SOAP provides level of security using WS-Security protocol. Secondly SOAP is stateful (Server can store information on previous requests)

**Non-HTTP messaging and legacy environment**

**SOAP стоит использовать, если**:

* Asynchronous processing and subsequent invocation
* A formal means of communication
* Stateful operations (если нужно сохранять состояние от одного запроса к другому.) Например если мы отправляем серверу запрос, а какая погода будет в Москве 20 ноября 2022 года, то он отвечает и сохраняет дату у себя. После, если мы спросим у него, а какая погода завтра, то сервер, сохранив прошлый запрос, поймет контекст, что мы имеем ввиду 21 ноября 2022 года.

**Challenges in SOAP**:

* **WSDL** - The tight contract between the client and the server. One change could cause a large impact on the whole client application. (Если мы переименуем функцию в WSDL, то все клиенты, которые подключены к этому API, должны будут сделать соответствующие изменения)
* **Document Size** - Because of the large messages, using SOAP in places where bandwidth is a constraint can be issue