

SOAP vs REST (differences)

Asked 11 years, 1 month ago Modified 6 months ago

Viewed 1.2m times



1358



I have read articles about the differences between SOAP and REST as a web service communication protocol, but I think that the biggest advantages for REST over SOAP are:

1. REST is more dynamic, no need to create and update UDDI(Universal Description, Discovery, and Integration).
2. REST is not restricted to only XML format. RESTful web services can send plain text/JSON/XML.

But SOAP is more standardized (E.g.: security).

So, am I correct in these points?

rest

http

web-services

soap

protocols

Share

Improve this question

Follow

edited Jul 14, 2023 at 16:41



Mehdi Charife

1,016 ● 3 ● 9 ● 28

asked Nov 9, 2013 at 23:11



Abdulaziz

13.8k ● 3 ● 15 ● 12

219 There's a letter analogy that I liked a lot about SOAP vs REST, [with SOAP you are using an envelope, with REST, it's a postcard](#), so Obviously SOAP has some extra overhead: more bandwidth(more paper), extra work for both parties(wrapping and unwrapping). But that doesn't mean REST is not as secure as SOAP since you can use HTTPS (think of it as replacing the mailman with someone who only speaks foreign languages) – [watashiSHUN](#) Jan 27, 2016 at 18:45

3 [spf13.com/post/soap-vs-rest](#) – [IceCold](#) Mar 16, 2016 at 17:00

[nishantshukla001webservices.blogspot.in/2015/09/...](#)
– [Nico](#) Apr 15, 2016 at 11:33

11 ["In many ways, the World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture."](#) – [Abhijeet](#) May 3, 2016 at 9:15

4 As per [Richardson Maturity Model](#) that breaks down the principal elements of a REST approach into three steps, SOAP is Level 0 REST. – [Sampada](#) Jun 6, 2016 at 11:03

14 Answers

Sorted by:

Highest score (default)



1873



Unfortunately, there are a lot of misinformation and misconceptions around REST. Not only your question and the [answer by @cmd](#) reflect those, but most of the questions and answers related to the subject on Stack Overflow.



SOAP and REST can't be compared directly, since the first is a protocol (or at least tries to be) and the second is



an architectural style. This is probably one of the sources of confusion around it, since people tend to call REST any HTTP API that isn't SOAP.

Pushing things a little and trying to establish a comparison, the main difference between SOAP and REST is the degree of coupling between client and server implementations. A SOAP client works like a custom desktop application, tightly coupled to the server. There's a rigid contract between client and server, and everything is expected to break if either side changes anything. You need constant updates following any change, but it's easier to ascertain if the contract is being followed.

A REST client is more like a browser. It's a generic client that knows how to use a protocol and standardized methods, and an application has to fit inside that. You don't violate the protocol standards by creating extra methods, you leverage on the standard methods and create the actions with them on your media type. If done right, there's less coupling, and changes can be dealt with more gracefully. A client is supposed to enter a REST service with zero knowledge of the API, except for the entry point and the media type. In SOAP, the client needs previous knowledge on everything it will be using, or it won't even begin the interaction. Additionally, a REST client can be extended by code-on-demand supplied by the server itself, the classical example being JavaScript code used to drive the interaction with another service on the client-side.

I think these are the crucial points to understand what REST is about, and how it differs from SOAP:

- REST is protocol independent. It's not coupled to HTTP. Pretty much like you can follow an ftp link on a website, a REST application can use any protocol for which there is a standardized URI scheme.
- REST is not a mapping of CRUD to HTTP methods. Read [this](#) answer for a detailed explanation on that.
- REST is as standardized as the parts you're using. Security and authentication in HTTP are standardized, so that's what you use when doing REST over HTTP.
- REST is not REST without [hypermedia](#) and [HATEOAS](#). This means that a client only knows the entry point URI and the resources are supposed to return links the client should follow. Those fancy documentation generators that give URI patterns for everything you can do in a REST API miss the point completely. They are not only documenting something that's supposed to be following the standard, but when you do that, you're coupling the client to one particular moment in the evolution of the API, and any changes on the API have to be documented and applied, or it will break.
- REST is the architectural style of the web itself. When you enter Stack Overflow, you know what a User, a Question and an Answer are, you know the media types, and the website provides you with the

links to them. A REST API has to do the same. If we designed the web the way people think REST should be done, instead of having a home page with links to Questions and Answers, we'd have a static documentation explaining that in order to view a question, you have to take the URI

`stackoverflow.com/questions/<id>`, replace id with the Question.id and paste that on your browser.

That's nonsense, but that's what many people think REST is.

This last point can't be emphasized enough. If your clients are building URIs from templates in documentation and not getting links in the resource representations, that's not REST. Roy Fielding, the author of REST, made it clear on this blog post: [REST APIs must be hypertext-driven](#).

With the above in mind, you'll realize that while REST might not be restricted to XML, to do it correctly with any other format you'll have to design and standardize some format for your links. Hyperlinks are standard in XML, but not in JSON. There are draft standards for JSON, like [HAL](#).

Finally, REST isn't for everyone, and a proof of that is how most people solve their problems very well with the HTTP APIs they mistakenly called REST and never venture beyond that. REST is hard to do sometimes, especially in the beginning, but it pays over time with easier evolution on the server side, and client's resilience

to changes. If you need something done quickly and easily, don't bother about getting REST right. It's probably not what you're looking for. If you need something that will have to stay online for years or even decades, then REST is for you.

Share Improve this answer

edited Jul 10, 2017 at 15:46

Follow

answered Nov 10, 2013 at 0:45



Pedro Werneck

41.8k ● 10 ● 66 ● 87

8 Either one is fine. The issue is how the users get the URLs, not how they use them. They should get the search url from a link in some other document, not from documentation. The documentation may explain how to use the search resource. – [Pedro Werneck](#) Jun 2, 2014 at 14:40

2 @CristiPotlog I never said SOAP is dependent on any particular protocol, I merely emphasize how REST isn't. The second link you sent says REST requires HTTP, which is wrong. – [Pedro Werneck](#) Jan 21, 2015 at 16:41

5 Lets repeat that once more: HATEOAS is a **constraint** if you wanna call your API Restful! – [Orestis](#) Aug 11, 2016 at 16:14

3 @SachinKainth There's an answer for that [here](#). You can map CRUD ops to HTTP methods, but that's not REST, because it's not the intended semantics of those methods as documented in the RFCs. – [Pedro Werneck](#) Mar 14, 2017 at 20:43

3 Last 4 lines are gem and should be fully understood by the person in development. Doing pure rest is time consuming

but gives rewards in longer run. So better for medium sized or big sized projects. Not good for prototyping and small projects. – [Rajan Chauhan](#) Oct 28, 2017 at 17:55



REST VS SOAP is **not** the right question to ask.

317

REST, unlike SOAP is **not** a protocol.



REST is an *architectural style* and a *design* for network-based software architectures.



REST concepts are referred to as resources. A representation of a resource must be stateless. It is represented via some media type. Some examples of media types include XML, JSON, and RDF. Resources are manipulated by components. Components request and manipulate resources via a standard uniform interface. In the case of HTTP, this interface consists of standard HTTP ops e.g. GET, PUT, POST, DELETE.

@Abdulaziz's question does illuminate the fact that REST and HTTP are often used in tandem. This is primarily due to the simplicity of HTTP and its very natural mapping to RESTful principles.

Fundamental REST Principles

Client-Server Communication

Client-server architectures have a very distinct separation of concerns. All applications built in the RESTful style

must also be client-server in principle.

Stateless

Each client request to the server requires that its state be fully represented. The server must be able to completely understand the client request without using any server context or server session state. It follows that all state must be kept on the client.

Cacheable

Cache constraints may be used, thus enabling response data to be marked as cacheable or not-cacheable. Any data marked as cacheable may be reused as the response to the same subsequent request.

Uniform Interface

All components must interact through a single uniform interface. Because all component interaction occurs via this interface, interaction with different services is very simple. The interface is the same! This also means that implementation changes can be made in isolation. Such changes, will not affect fundamental component interaction because the uniform interface is always unchanged. One disadvantage is that you are stuck with the interface. If an optimization could be provided to a specific service by changing the interface, you are out of luck as REST prohibits this. On the bright side, however, REST is optimized for the web, hence incredible popularity of REST over HTTP!

The above concepts represent defining characteristics of REST and differentiate the REST architecture from other architectures like web services. It is useful to note that a REST service is a web service, but a web service is not necessarily a REST service.

See this blog [post](#) on **REST Design Principles** for more details on **REST** and the above stated bullets.

EDIT: update content based on comments

Share Improve this answer

Follow

edited Oct 12, 2017 at 16:40



BobRodes

6,155 ● 2 ● 27 ● 29

answered Nov 9, 2013 at 23:19



cmd

11.8k ● 8 ● 52 ● 63

-
- 7 REST does not have a predefined set of operations that are CRUD operations. Mapping HTTP methods to CRUD operations blindly is one of the most common misconceptions around REST. The HTTP methods have very well defined behaviors that have nothing to do with CRUD, and REST isn't coupled to HTTP. You can have a REST API over ftp with nothing but RETR and STOR, for instance.
– [Pedro Werneck](#) Nov 10, 2013 at 0:51

-
- 10 Also, what do you mean by 'REST services are idempotent'? As far as I know, you have some HTTP methods that by default are idempotent, and if a particular operation in your service needs idempotence, you should use them, but it doesn't make sense to say the service is idempotent. The service may have resources with actions that may be

effected in an idempotent or non-idempotent fashion.

– [Pedro Werneck](#) Nov 10, 2013 at 0:53

2 @cmd :please remove fourth point - "A RESTful architecture may use HTTP or SOAP as the underlying communication protocol". its a misinformation you are conveying.

– [Bruce_Wayne](#) Apr 16, 2015 at 18:25 ✎



256



SOAP (**Simple Object Access Protocol**) and REST (**Representation State Transfer**) both are beautiful in their way. So I am not comparing them. Instead, I am trying to depict the picture, when I preferred to use REST and when SOAP.

What is payload?

When data is sent over the Internet, each unit transmitted includes both header information and the actual data being sent. The header identifies the source and destination of the packet, **while the actual data is referred to as the payload.**

In general, the payload is the data that is carried on behalf of an application and the data received by the destination system.

Now, for example, I have to send a **Telegram** and we all know that the cost of the telegram will depend on some words.

So tell me among below mentioned these two messages, which one is cheaper to send?

```
<name>Arin</name>
```

or

```
"name": "Arin"
```

I know your answer will be the second one although both representing the same message second one is cheaper regarding cost.

So I am trying to say that, **sending data over the network in JSON format is cheaper than sending it in XML format regarding payload.**

Here is the first benefit or advantages of REST over SOAP. SOAP only supports XML, but REST supports different format like text, JSON, XML, etc. And we already know, if we use Json then definitely we will be in better place regarding payload.

Now, SOAP supports only XML, **but it also has its advantages.**

Really! How?

SOAP relies on XML in three ways Envelope – that defines what is in the message and how to process it.

A set of encoding rules for data types, and finally the layout of the procedure calls and responses gathered.

This envelope is sent via a transport (HTTP/HTTPS), and an RPC (Remote Procedure Call) is executed, and the envelope is returned with information in an XML formatted document.

The important point is that **one of the advantages of SOAP** is the use of the “**generic**” transport but **REST uses HTTP/HTTPS**. SOAP can use almost any transport to send the request but REST cannot. So here we got an advantage of using SOAP.

As I already mentioned in above paragraph “**REST uses HTTP/HTTPS**”, so go a bit deeper on these words.

When we are talking about REST over HTTP, all security measures applied HTTP are inherited, and this is known as **transport level security** and it secures messages only while **it is inside the wire** but once you delivered it on the other side you don't know how many stages it will have to go through before reaching the real point where the data will be processed. And of course, all those stages could use something different than HTTP. **So Rest is not safer completely, right?**

But SOAP **supports SSL** just like REST additionally it **also supports WS-Security** which adds some enterprise security features. WS-Security offers protection from the **creation of the message to it's consumption**. So for transport level security whatever loophole we found that can be prevented using WS-Security.

Apart from that, as **REST is limited by its HTTP protocol** so its transaction support is neither ACID compliant nor can provide two-phase commit across distributed transactional resources.

But SOAP has comprehensive support for both **ACID based transaction management** for short-lived transactions and compensation based transaction management for long-running transactions. It also supports **two-phase commit across distributed resources**.

I am not drawing any conclusion, but I will prefer SOAP-based web service while security, transaction, etc. are the main concerns.

Here is the "The Java EE 6 Tutorial" where they have said [A RESTful design may be appropriate when the following conditions are met](#). Have a look.

Hope you enjoyed reading my answer.

Share Improve this answer

Follow

edited Jun 8 at 13:54



Sajad

2,363 ● 11 ● 53 ● 95

answered Jun 14, 2015 at 19:48



Bacteria

8,566 ● 11 ● 56 ● 70

17 Great answer but remember REST can use any transport protocol. For example, it can use FTP. – [Bhargav Nanekalva](#)
Aug 23, 2015 at 3:35

13 Who said REST can't use SSL? – [Osama Aftab](#) Sep 7, 2015 at 12:45

1 @ Osama Aftab REST supports SSL, but SOAP supports SSL **just like REST** additionally it also supports WS-Security. – [Bacteria](#) Sep 7, 2015 at 16:01

3 To reference the point about size of XML data, when compression is enabled, XML is quite small.
– [GaTechThomas](#) Nov 8, 2016 at 18:59

2 The point about the size of the payload should be deleted, it is such a one-dimensional comparison between JSON and XML and is only possible to detect in seriously optimized setups, which are far between. – [ThomasRS](#) Apr 20, 2017 at 21:51



[SOAP or REST for Web Services?](#)

143



REST(**RE**presentational **S**tate **T**ransfer)

REpresentational **S**tate of an Object is **T**ransferred is REST i.e. we don't send Object, we send state of Object. REST is an architectural style. It doesn't define so many standards like SOAP. REST is for exposing Public APIs(i.e. Facebook API, Google Maps API) over the internet to handle CRUD operations on data. REST is focused on accessing named resources through a single consistent interface.



SOAP(**S**imple **O**bject **A**ccess **P**rotocol)

SOAP brings its own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing

named operations, each operation implement some business logic. Though SOAP is commonly referred to as **web services** this is misnomer. SOAP has a very little if anything to do with the Web. REST provides true **Web services** based on URIs and HTTP.

Why REST?

- Since REST uses standard HTTP it is much simpler in just about every way.
- REST is easier to implement, requires less bandwidth and resources.
- REST permits many different data formats whereas SOAP only permits XML.
- REST allows better support for browser clients due to its support for JSON.
- REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.
- If security is not a major concern and we have limited resources. Or we want to create an API that will be easily used by other developers publicly then we should go with REST.
- If we need Stateless CRUD operations then go with REST.
- REST is commonly used in social media, web chat, mobile services and Public APIs like Google Maps.

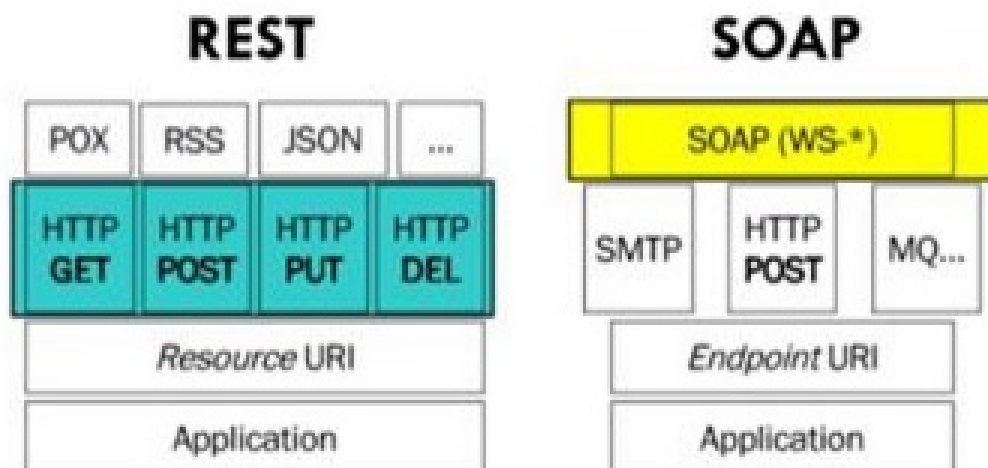
- RESTful service return various MediaTypes for the same resource, depending on the request header parameter "Accept" as `application/xml` or `application/json` for POST and `/user/1234.json` or GET `/user/1234.xml` for GET.
- REST services are meant to be called by the client-side application and not the end user directly.
- **ST** in REST comes from **State Transfer**. You transfer the state around instead of having the server store it, this makes REST services scalable.

Why SOAP?

- SOAP is not very easy to implement and requires more bandwidth and resources.
- SOAP message request is processed slower as compared to REST and it does not use web caching mechanism.
- **WS-Security:** While SOAP supports SSL (just like REST) it also supports WS-Security which adds some enterprise security features.
- **WS-AtomicTransaction:** Need ACID Transactions over a service, you're going to need SOAP.
- **WS-ReliableMessaging:** If your application needs Asynchronous processing and a guaranteed level of reliability and security. Rest doesn't have a standard messaging system and expects clients to deal with communication failures by retrying.

- If the security is a major concern and the resources are not limited then we should use SOAP web services. Like if we are creating a web service for payment gateways, financial and telecommunication related work then we should go with SOAP as here high security is needed.

Protocol Layering



[source1](#)

[source2](#)

Share Improve this answer

edited Sep 16, 2023 at 3:02

Follow

answered Dec 8, 2015 at 23:38



Premraj

74.4k ● 26 ● 243 ● 184

REST verbs/methods don't have a 1 to 1 relation to CRUD methods although, it can help in the beginning to understand the REST style. – [Santiago Martí Olbrich](#) Feb 27, 2016 at 20:30

5 REST does not support SSL ? the uniform resource url for rest can not be start with https:// ? – [Mou](#) Nov 7, 2016 at 14:33



IMHO you can't compare SOAP and REST where those are two different things.

26



SOAP is a protocol and **REST** is a software architectural pattern. There is a lot of misconception in the internet for **SOAP vs REST**.



SOAP defines XML based message format that web service-enabled applications use to communicate each other over the internet. In order to do that the applications need prior knowledge of the message contract, datatypes, etc..

REST represents the state(as resources) of a server from an URL.It is stateless and clients should not have prior knowledge to interact with server beyond the understanding of hypermedia.

Share Improve this answer

answered Jan 17, 2016 at 0:17

Follow



[marvelTracker](#)

4,959 ● 4 ● 38 ● 50



22

First of all: officially, the correct question would be `web services + WSDL + SOAP` VS `REST`.



Because, although the *web service*, is used in the loose sense, when using the HTTP protocol to transfer *data* instead of web pages, [officially](#) it is a very specific form of that idea. According to the definition, REST is not "web service".

In practice however, everyone ignores that, so let's ignore it too

There are already technical answers, so I'll try to provide some intuition.

Let's say you want to call a function in a remote computer, implemented in some other programming language (this is often called *remote procedure call/RPC*). Assume that function can be found at a specific URL, provided by the person who wrote it. You have to (somehow) send it a message, and get some response. So, there are two main questions to consider.

- what is the format of the message you should send
- how should the message be carried back and forth

For the first question, the official definition is [WSDL](#). This is an XML file which describes, in detailed and strict format, what are the parameters, what are their types, names, default values, the name of the function to be called, etc. [An example WSDL](#) here shows that the file is human-readable (but not easily).

For the second question, there are various answers. However, the only one used in practice is [SOAP](#). Its main idea is: wrap the previous XML (the actual message) into yet another XML (containing encoding info and other helpful stuff), and send it over HTTP. The POST method of the HTTP is used to send the message, since *there is always a body*.

The main idea of this whole approach is that you *map a URL to a function*, that is, to **an action**. So, if you have a list of customers in some server, and you want to view/update/delete one, you must have 3 URLs:

- `myapp/read-customer` and in the body of the message, pass the id of the customer to be read.
- `myapp/update-customer` and in the body, pass the id of the customer, as well as the new data
- `myapp/delete-customer` and the id in the body

The REST approach sees things differently. A URL should not represent an action, but **a thing** (called *resource* in the REST lingo). Since the HTTP protocol (which we are already using) supports verbs, *use those verbs to specify what actions* to perform on the thing.

So, with the REST approach, customer number 12 would be found on URL `myapp/customers/12`. To view the customer data, you hit the URL with a GET request. To delete it, **the same** URL, with a DELETE verb. To update it, **again, the same** URL with a POST verb, and the new content in the request body.

For more details about the requirements that a service has to fulfil to be considered truly RESTful, see the [Richardson maturity model](#). The article gives examples, and, more importantly, explains why a (so-called) SOAP service, is a level-0 REST service (although, level-0 means low compliance to this model, it's not offensive, and it is still useful in many cases).

Share Improve this answer

edited Jun 20, 2020 at 9:12

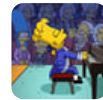
Follow



Community Bot

1 • 1

answered Aug 15, 2018 at 18:19




aris

29k • 10 • 82 • 107

What do you mean REST is not web service?? Whats JAX-RS then?? – [Ashish Kamble](#) Sep 17, 2019 at 10:41

1 @AshishKamble: I provided the link of the rest services specification. The official definition contains only the WS-* protocols (roughly the ones we call "SOAP") and rest is not part of it officially – [aris](#) Sep 17, 2019 at 10:46

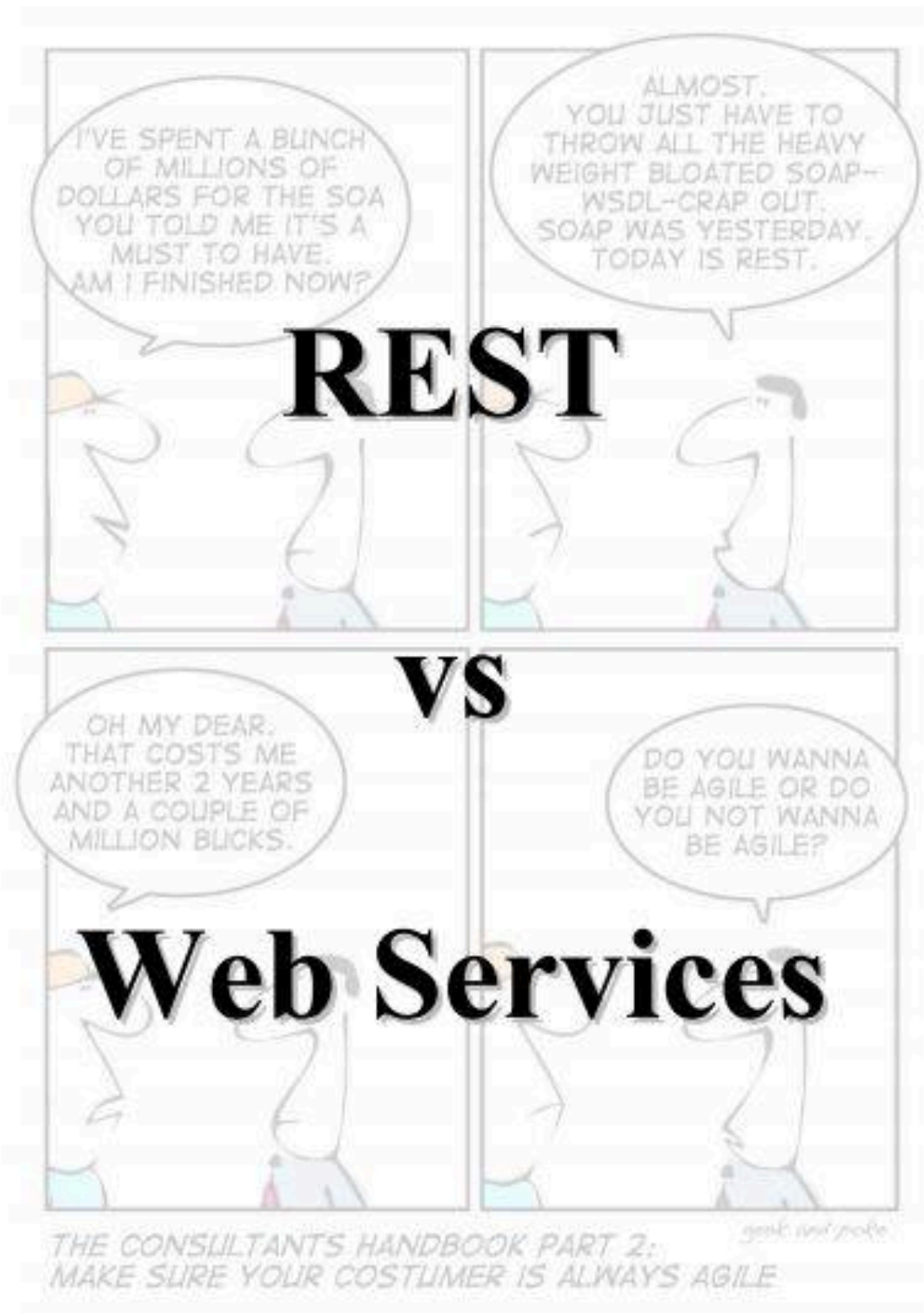
1 @AshishKamble: Also, note that there's also a JAX-WS, which means "web services", differentiated from "rest services". Anyway, the distinction is not important for any practical purposes, as I also noted. – [aris](#) Sep 17, 2019 at 10:47 



16



Among many others already covered in the many answers, I would highlight that SOAP enables to define a contract, the WSDL, which define the operations supported, complex types, etc. SOAP is oriented to operations, but REST is oriented at resources. Personally I would select SOAP for complex interfaces between internal enterprise applications, and REST for public, simpler, stateless interfaces with the outside world.



Share Improve this answer

edited May 25, 2019 at 15:30

Follow

answered May 23, 2018 at 15:41



Jose Manuel Gomez Alvarez

2,218 ● 25 ● 33

About five years later, we should know that REST vs SOAP is winning in popularity (Google results) by a factor of 6x in favor of REST and considered by many a legacy (see stackoverflow.com/questions/38463/...). In addition, OpenAPI is filling the gap that existed originally about documentation of REST APIs (which was done through WSDL in the case of SOAP). – [Jose Manuel Gomez Alvarez](#) Dec 4, 2023 at 11:56



10



Addition for:

++ A mistake that's often made when approaching REST is to think of it as "web services with URLs"—to think of REST as another remote procedure call (RPC) mechanism, like SOAP, but invoked through plain HTTP URLs and without SOAP's hefty XML namespaces.

++ On the contrary, REST has little to do with RPC. Whereas RPC is service oriented and focused on actions and verbs, REST is resource oriented, emphasizing the things and nouns that comprise an application.

Share Improve this answer

answered Sep 20, 2016 at 8:02

Follow



[Quan Nguyen](#)

696 ● 1 ● 10 ● 21



9

A lot of these answers entirely forgot to mention hypermedia controls (HATEOAS) which is completely fundamental to REST. A few others touched on it, but didn't really explain it so well.



[This article](#) should explain the difference between the concepts, without getting into the weeds on specific SOAP features.



Share Improve this answer

answered Jan 5, 2018 at 0:17

Follow



Phil Sturgeon

30.7k ● 12 ● 80 ● 117



REST API

1

RESTful APIs are the most famous type of API. REST stands REpresentational State Transfer.



REST APIs are APIs that follow standardized principles, properties, and constraints.



You can access resources in the REST API using HTTP verbs.

REST APIs operate on a simple request/response system. You can send a request using these HTTP methods:

- GET
- POST
- PUT
- PATCH
- DELETE
- TRACE

- OPTIONS
- CONNECT
- HEAD

Here are the most common HTTP verbs

- GET (read existing data)
- POST (create a new response or data)
- PATCH (update the data)
- DELETE (delete the data)

The client can make requests using HTTP verbs followed by the endpoint.

The endpoint (or route) is the URL you request for. The path determines the resource you're requesting.

When you send a request to an endpoint, it responds with the relevant data, generally formatted as JSON, XML, plain text, images, HTML, and more.

REST APIs can also be designed with many different endpoints that return different types of data. Accessing multiple endpoints with a REST API requires various API calls.

An actual RESTful API follows the following five constraints:

1. Client-Server Architecture

The client requests the data from the server with no

third-party interpretation.

2. Statelessness

Statelessness means that every HTTP request happens in complete isolation. Each request contains the information necessary to service the request. The server never relies on information from previous requests. There's no state.

3. Cacheability

Responses can be explicitly or implicitly defined as cacheable or non-cacheable to improve scalability and performance. For example, enabling the cache of GET requests can improve the response times of requests for resource data.

4. Layering

Different layers of the API architecture should work together, creating a scalable system that is easy to update or adjust.

5. Uniform Interface

Communication between the client and the server must be done in a standardized language that is independent of both. This improves scalability and flexibility.

REST APIs are a good fit for projects that need to be

- Flexible
- Scalable
- Fast

SOAP API

SOAP is a necessary protocol that helped introduce the widespread use of APIs.

SOAP is the acronym for Simple Object Access Protocol.

SOAP is a standardized protocol that relies on XML to make requests and receive responses.

Even though SOAP is based on XML, the SOAP protocol is still in wide usage.

SOAP APIs make data available as a service and are typically used when performing transactions involving multiple API calls or applications where security is the primary consideration.

SOAP was initially developed for Microsoft in 1998 to provide a standard mechanism for integrating services on the internet regardless of the operating system, object model, or programming language.

The “S” in SOAP stands for Simple, and for a good reason — SOAP can be used with less complexity as it requires less coding in the app layer for transactions, security, and other functions.

SOAP has three primary characteristics:

1. Extensibility of SOAP API

SOAP allows for extensions that introduce more

robust features, such as Windows Server Security, Addressing, and more.

2. Neutrality of SOAP API

SOAP is capable of operating over a wide range of protocols, like UDP, JMS, SMTP, TCP, and HTTP.can operate.

3. Independence of SOAP API

SOAP API responses are purely based on XML.

Therefore SOAP APIs are platform and language independent.

Developers continue to debate the pros and cons of using SOAP and REST. The best one for your project will be the one that aligns with your needs.

- SOAP APIs remain a top choice for corporate entities and government organizations that prioritize security, even though REST has largely dominated web applications.
- SOAP is more secure than REST as it uses WS-Security for transmission along with Secure Socket Layer
- SOAP also has more excellent transactional reliability, which is another reason why SOAP historically has been favored by the banking industry and other large entities.

Share Improve this answer

answered Nov 11, 2021 at 10:53

Follow



Pratham

537 ● 4 ● 7



1



Web service architectures are frameworks for building and integrating software systems that can communicate over the internet. Here are some of the commonly used web service architectures

SOAP (Simple Object Access Protocol)

SOAP is an XML-based messaging protocol used for exchanging structured data between web services. It defines a standardized set of rules and protocols for the format of the request and response messages, as well as the operations that can be performed on the web service.

Benefits:

- Supports various transport protocols (such as HTTP, SMTP, TCP, and others)
- Provides reliable messaging and security features
- Can be used with any programming language

Disadvantages:

- Complex messaging structure can make it difficult to use and debug
- Performance can be slower compared to other architectures

- Requires more bandwidth and resources

examples:

- Financial systems that require secure and reliable messaging for transactions
- Healthcare systems that require strict compliance with regulatory standards

REST (Representational State Transfer)

REST is an architectural style that uses HTTP (Hypertext Transfer Protocol) to exchange data between clients and servers. RESTful web services use standard HTTP methods (such as GET, POST, PUT, and DELETE) to perform operations on resources (such as data objects) identified by URLs (Uniform Resource Locators).

Benefits:

- Lightweight and easy to use
- Provides greater scalability and performance
- Can be used with any programming language

Disadvantages:

- Limited support for transaction management and security
- Can be difficult to design a proper API for complex systems
- No standard messaging format

examples:

- Social media platforms that require quick and efficient data access for large user bases
- Mobile applications that require low latency and high performance

RPC (Remote Procedure Call)

RPC is a protocol used for communication between applications running on different systems. It allows a client to call a procedure (or method) on a remote server as if it were a local method call.

Benefits:

- Enables easy integration between different systems and programming languages
- Provides a simpler and more direct approach to remote method invocation
- Can be used with any transport protocol

Disadvantages:

- Limited support for distributed transactions and security
- Can be difficult to scale and manage for large systems
- May require more complex error handling

examples:

- IoT systems that require communication between different devices and protocols
- Gaming systems that require real-time communication between players

GraphQL

GraphQL is a query language and runtime for APIs that allows clients to specify exactly what data they need and get back only that data. It provides a flexible and efficient way of fetching data from a server, reducing the amount of data transferred over the network.

Benefits:

- Provides more flexibility and control over data queries
- Allows for efficient data retrieval and reduces over-fetching
- Can be used with any programming language

Disadvantages:

- Can be difficult to design and optimize queries for complex systems Limited support for caching and security
- May require additional development effort compared to traditional REST APIs

examples:

- E-commerce systems that require efficient and flexible data queries for product catalogs and recommendations
- Analytics platforms that require complex and customized data queries for business intelligence and reporting

WebSocket

WebSocket is a protocol for two-way communication between a client and a server over a single, long-lived connection. It allows real-time, event-driven communication between a client and a server, enabling applications such as chat, gaming, and real-time collaboration.

Benefits:

- Provides real-time and event-driven communication between client and server
- Enables efficient communication with low latency and overhead
- Can be used with any programming language

Disadvantages:

- Requires support for bidirectional communication in the client and server Limited support for routing and security
- Can be more complex to implement compared to traditional REST APIs

examples:

- Chat and messaging systems that require real-time communication between users
- Real-time collaboration platforms for video conferencing and document editing

Share Improve this answer

answered Mar 21, 2023 at 9:56

Follow



Kavinda

66 ● 3



0



What is REST

REST stands for representational state transfer, it's actually an architectural style for creating Web API which treats everything(data or functionality) as resource. It expects; exposing resources through URI and responding in multiple formats and representational transfer of state of the resources in stateless manner. Here I am talking about two things:

1. Stateless manner: Provided by HTTP.
2. Representational transfer of state: For example if we are adding an employee. . into our system, it's in POST state of HTTP, after this it would be in GET state of HTTP, PUT and DELETE likewise.

REST can use SOAP web services because it is a concept and can use any protocol like HTTP,

SOAP.SOAP uses services interfaces to expose the business logic. REST uses URI to expose business logic.

REST is not REST without HATEOAS. This means that a client only knows the entry point URI and the resources are supposed to return links the client should follow. Those fancy documentation generators that give URI patterns for everything you can do in a REST API miss the point completely. They are not only documenting something that's supposed to be following the standard, but when you do that, you're coupling the client to one particular moment in the evolution of the API, and any changes on the API have to be documented and applied, or it will break.

HATEOAS, an abbreviation for Hypermedia As The Engine Of Application State, is a constraint of the REST application architecture that distinguishes it from most other network application architectures. The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. By contrast, in some service-oriented architectures (SOA), clients and servers interact through a fixed interface shared through documentation or an interface description language (IDL).

[Reference 1](#) [Reference 2](#)

Share Improve this answer

edited Sep 16, 2020 at 19:35

Follow

answered Sep 16, 2020 at 19:30



MAQ

144 ● 2 ● 9



0



Although SOAP and REST share similarities over the HTTP protocol, SOAP is a more rigid set of messaging patterns than REST. The rules in SOAP are relevant because we can't achieve any degree of standardization without them. REST needs no processing as an architecture style and is inherently more versatile. In the spirit of information exchange, both SOAP and REST depend on well-established laws that everybody has decided to abide by. The choice of SOAP vs. REST is dependent on the programming language you are using the environment you are using and the specifications.

Share Improve this answer

answered Nov 17, 2020 at 17:52

Follow



Laura Nutt

313 ● 3 ● 7



0



To answer this question it's useful to understand the evolution of the architecture of distributed applications from simple layered architectures, to object & service based, to resources based, & nowadays we even have event based architectures. Most large systems use a combination of styles.



The first distributed applications had layered architectures. I'll assume everyone here knows what layers are. These structures are neatly organized, and can be stacks or cyclical structures. Effort is made to maintain a unidirectional data flow.

Object-based architectures evolved out of layered architectures and follow a much looser model. Here, each component is an object (often called a distributed object). The objects interact with one another using a mechanism similar to remote procedure calls - when a client binds to a distributed object it loads an implementation of the objects interface into its address space. The RPC stub can marshal a request & receive a response. Likewise the objects interface on the server is an RPC style stub. The structure of these object based systems is not as neatly organized, it looks more like an object graph.

The interface of a distributed object conceals its implementation. As with layered components, if the interface is clearly defined the internal implementation can be altered - even replaced entirely. Object-based architectures provide the basis for encapsulating services. A service is provided by a self-contained entity, though internally it can make use of other services. Gradually object-based architectures evolved into service-oriented architectures (SOAs).

With SOA, a distributed application is composed of services. These services can be provided across administrative domains - they may be available across

the web (i.e. a storage service offered by a cloud provider).

As web services became popular, and more applications started using them, service composition (combining services to form new ones) became more important. One of the problems with SOA was that integrating different services could become extremely complicated.

While SOAP is a protocol, its use implies a service oriented architecture. SOAP attempted to provide a standard for services whereby they would be composable and easily integrated.

Resource-based architectures were a different approach to solving the integration problems of SOA. The idea is to treat the distributed system as a giant collection of resources that are individually managed by components. This led to the development of RESTful architectures. One thing that characterizes RESTful services is stateless execution. This is different than SOA where the server maintains the state.

So... how do service-specific interfaces, as provided by service-oriented architectures (including those that use SOAP) compare with resource-based architecture like REST?

While REST is simple, it does not provide a simple interface for complex communication schemes. For example, if you are required to use transactions REST is not appropriate, it is better to keep the complex state

encapsulated on the server than have the client manage the transaction. But there are many scenarios where the orthogonal use of resources in RESTful architectures greatly simplifies integration of services in what would otherwise mean an explosion of service interfaces. Another tradeoff is resource-based architectures put more complexity on the client & increase traffic over the network while service-based increase the complexity of the server & tax its memory & CPU resources.

Some people have also mentioned common HTTP services or other services that do not satisfy the requirements of RESTful architecture or SOAP. These too can be categorized as either service-based or resource-based. These have the advantage of being simpler to implement. You'd only use such an approach if you knew your service will never need to be integrated across administrative domains since this makes no attempt at fixing the integration issues that arise.

These sorts of HTTP-based services, especially Pseudo-RESTful services are still the most common types. Implementing SOAP is complicated and should only be used if you really need it - i.e. you need a service that's easily integrated across domains and you want it to have a service-interface. There are still cases where this is needed. A true RESTful service is also difficult to implement, though not as difficult as SOAP.

Share Improve this answer

edited Oct 18, 2021 at 5:24

Follow

answered Oct 17, 2021 at 4:17



ibrust

180 ● 1 ● 8



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.