An introduction to SOAP and REST for web services

Overview

When building web services, it's important to carefully evaluate different information exchange protocols and API architecture paradigms. The choice at the design stage affects various aspects of the web service, and the clients interacting with it. This article helps you decide when to use SOAP or REST for developing web services, based on their use-case specific advantages and trade-offs.

SOAP

Simple Object Access Protocol (**SOAP**) is a W3C recommended, XML-based message exchange protocol for web services. Most SOAP-based API implementations use Web Services Description Language (WSDL) to define the following in a machine-readable format:

- Resources, operations, procedure calls, and responses
- Encapsulation, encoding, and structure of the message
- Bindings with lower-level protocols in the network stack

There are tools to automatically generate a SOAP API based on the server-side code. Alternatively, you can define a SOAP API and use tools that automatically generates a template for your application code. SOAP-based implementations are customizable to a large extent, but for most developers the resulting code is difficult to develop and maintain.

Advantages

- SOAP-based architectures strongly enforce a pre-defined contract between the server and the client, which improves service reliability and data integrity.
- SOAP offers enhanced security when extended using WS-Security.
- You may define a custom set of operations that the client can use to interact with the service.

Disadvantages

- The tight coupling between server and client implementations greatly reduces the flexibility of the web service.
- Mandatory use of XML often makes it inconvenient to develop and maintain the client, which may prefer HTML or JSON as the payload format.
- SOAP messages are verbose and contains significant overhead, which makes parsing them computationally expensive.

REST

Representational State Transfer (**REST**) is an extremely popular paradigm for designing stateless web service architectures. In a RESTful web service:

- URIs represent resource endpoints, and the API serves data formatted as JSON, HTML, or XML.
- A client can request an HTTP method based operation on the data; REST supports `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, `DELETE`, `CONNECT`, `OPTIONS`, and `TRACE`.
- If a request is valid and authentic, the web service responds with the result of the requested HTTP method call.

Similar to SOAP-based implementations, you can use tools to generate a REST API and related documentation from your application code. In addition, there are third-party services that connects various REST APIs using common standards.

Advantages

- Support for multiple data formats such as JSON, HTML, and XML gives flexibility to clients consuming the web service. In particular, REST APIs serving JSON data helps develop leaner browser-based client applications by reducing boilerplate code.
- Less constraints and lack of rigid rules encourage rapid iterations and faster implementations for both server and client. However, drastic changes in the data models and schemas without proper API versioning will break the client.
- A RESTful web service can mark parts of its response as cacheable, which results in faster client applications and reduced number of calls to the server.
- Requests and responses are simpler and contains less overhead as compared to SOAP messages, which makes them easier to process.
- Most modern web services are RESTful, and there is a large and helpful community to support your development and maintenance efforts.

Disadvantages

- Most RESTful web services use the insecure HTTP as the default application layer protocol. To enhance security, enforce use of HTTPS.
- Unlike SOAP that offers provision for custom operations and procedure calls, REST APIs allow only standard HTTP methods as permissible operations for interacting with API endpoints. However, for the majority of use-cases a combination of HTTP methods is sufficient.
- Point-to-point communication between multiple clients and a server exposing a REST API endpoint is tricky to scale.

Recommendation

Both SOAP and REST allows a stateless architecture and asynchronous client-server messaging. They are not mutually exclusive, but in practice no API uses them simultaneously. You can serve XML data from a REST API endpoint without the restrictions imposed by SOAP.

Consider designing a SOAP-based API for contexts that are sensitive to security, and demands high degree of reliability and data integrity. For example, web services handling financial transactions and sensitive databases often benefits from the rigor that SOAP provides. It also helps if both server and client implementations are under your control, because even a trivial change in the backend mandates modifying the client.

REST is a better choice for most other types of web service, especially for a public API aiming widespread adoption and third-party integration. However, to avoid exploitation of public API servers, consider imposing limits on the rate of API calls. It's also a standard practice to strengthen the security of API endpoints with token-based authentications.

References

- **Web Service** - https://en.wikipedia.org/wiki/Web_service
- **REST** - https://en.wikipedia.org/wiki/Representational_state_transfer
- **SOAP** - https://en.wikipedia.org/wiki/SOAP