Spring Web Services

# Introduction to Web Services

## What is Web Service?

* A software system designed to support **interoperable** **machine to machine (or application to application)** **interaction** **over a network**.
* Three Keys Points –
  + A web service should support application to application interaction.
  + It should be interoperable (platform independent).
  + It should allow communication over the network.

## Hows?

### How does data exchange between applications take place?

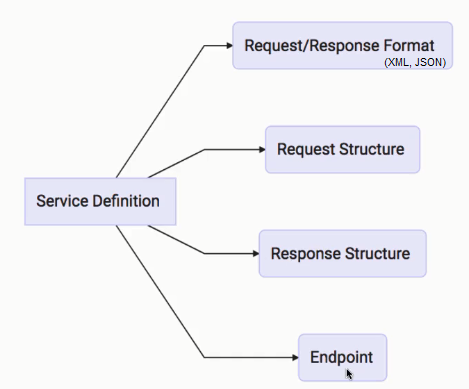
* The input to a web service is called a Request and the output from a web service is called a Response.

### How to make web services platform independent?

* The request and the response also should be platform independent.
* They should be in formats which are supported by all different kinds of platforms.
* E.g. XML, JSON

### How does an application know the format of request and response?

* Via Service definition. Every web service offers a service definition.

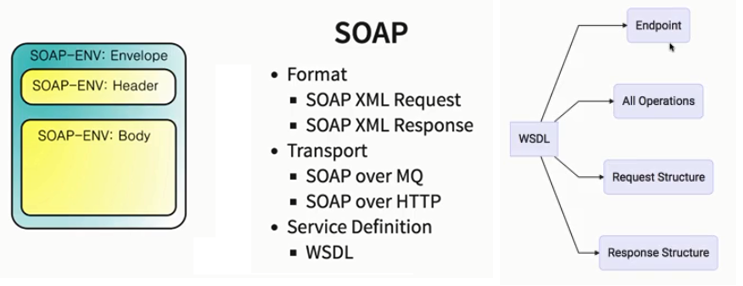


## Web Services – Key Terminologies

* Request is the input of our service and response is the output from a web service.
* Message exchange format is the format of the request and the response. E.g. XML, JSON.
* Service provider (Server) is the one which hosts the web service.
* Service consumer (Client) is the one which is consuming the web service.
* Service definition is the contract between the service provider and the service consumers.
* Transport defines how a service is called. E.g. HTTP, MQ.

## Introduction to SOAP Web Services

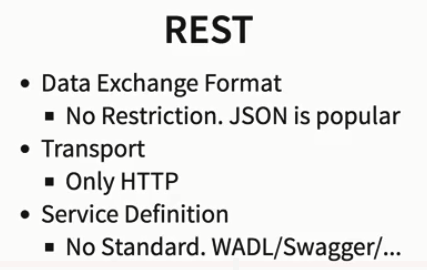
* Simple Object Access Protocol.
* In SOAP we use XML as the request exchange format.
* SOAP defines a specific XML request and response structure (SOAP XML format).
* If you're using SOAP then you have to use this structure. So you have to create a SOAP and envelope which contains a SOAP header and a SOAP body.
* The header contains meta information like authentication, authorization, signatures and things like that. SOAP Header can be empty.
* Body is where you really put the real content of your request or your response.



* In SOAP, the service definition is typically done using WSDL (Web service definition language).
* In summary, SOAP is all about adhering to the services XML structure. Adhering to the envelop header and the body.

## Introduction to RESTful Web Services

* REST stands for REpresentational State Transfer.
* It’s a term which is coined by Roy Fielding. Roy Fielding is the guy who also developed HTTP protocol.
* The key thing about REST services is the fact that they would want to make best use of his HTTP.
* RESTful web services try to define services using the different concepts that are already present in HTTP.
* The most important abstraction in the REST is something called a resource.
* A **resource** is anything that you'd want to expose to the outside world through your application.
* When we are talking about talking about RESTful services, we are always thinking about resources.
* **WADL** (web application definition language) is one of the formats in which you can specify your RESTful Web services. Another option is **swagger**.
* REST focuses on your resources and how do you perform actions on them making best use of HTTP.



## SOAP vs REST

* SOAP and REST are not really comparable. It's not really an apple to apple comparison.
* REST defines an architectural approach.
* Whereas SOAP poses restrictions on the format of XML which is exchanged between your service provider and the service consumer.
* RESTful services are typically easier to implement than SOAP.
* RESTful services are typically based on JSON which is an easy format to pass and do things with it and also with RESTful services, we don't really need to mandate really define a service definition. But with SOAP you have to define WSDL and there are a lot of complexities associated with parsing your XMLs as well.

# Introduction to Spring Framework

* Refer – <https://github.com/in28minutes/spring-web-services/tree/master/spring-in-10-steps>

## Using Spring to Manage Dependencies

* In order for Spring to manage dependencies, we need to answer 3 questions –
  + What are the beans?
  + What are the dependencies of the bean?
  + Where to search for the beans?
* Spring application context is the one which would maintain all the beans.
* SpringApplication.run() returns the spring application context (ApplicationContext).
* To access/get the spring managed bean, we can use the Spring application context as e.g. BinarySearchImpl bean = applicationContext.getBean(BinarySearchImpl.class);

### What are the beans?

* We annotate a class with **@Component** annotation to let Spring know that the class is a bean.
* Classes annotated with @Component will be managed by Spring.

### What are the dependencies of the bean?

* Use **@Autowired** annotation for the properties of a class so that Spring will manage those dependencies.
* Three options for using autowiring - constructor, setter and neither setter nor constructor. Setter and “neither setter nor constructor” autowiring is the same.
* With Earlier versions of Spring, the recommendation was if you have mandatory dependencies then use constructor injection. For all other dependencies, use setter injection.

### Where to search for the beans?

* Basically we need to tell Spring where (in which package) our beans (components) are.
* In order to do so, we use another annotation called **@ComponentScan**.
* However what **@SpringBootApplication** annotation does is, it by default scans the package and the sub packages of the package where this class (annotated with @SpringBootApplication) is. So we may not need to use @ComponentScan.

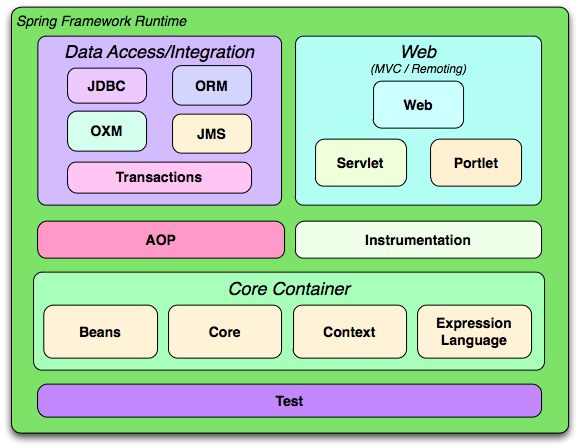
## How it works in the background

* Once you run your Spring boot application the class (annotated with @SpringBootApplication) with main method runs.
* Then spring does the Component Scan as spring needs to know where the components are. So it searches for the packages mentioned in @ComponentScan or package/subpackages where the class annotated with @SpringBootApplication is.
* In the component scan, spring searches for the classes annotated with @Component.
* One Spring has figured out what components it needs to manage, then it would start creating the beans and trying to identify the dependencies.
* Once spring identifies dependencies for a component, it will create a bean i.e. it will create an instance of the class by setting the dependencies.

## Dynamic Autowiring

* For instance there is QuickSort and BubbleSort classes which implement Sort interface and the Sort interface is used as a dependency in some other class like BinarySearch.
* So what if you have two components on classpath? What does Spring do to resolve that?
* By default, in this case, the Spring boot application will fail to start as it can see the conflicting classes (QuickSort and BubbleSort) and it does not know which one to inject into BinarySearch class/bean.
* **Solution** – use @Primary annotation for one the conflicting classes.
* If you have more than one component matching a specific type, you can use **@Primary** to give more importance to one of those components.

## Spring Modules



* One of the important things about Spring is that it's not one big framework. So we have lot of small jars with dedicated purposes.
* This enables you to use specific modules without using the other models of Spring.
* All the Spring modules have the same release version as the Spring Framework.

## Spring Projects

* Refer – <https://spring.io/projects>
* There are other things Spring does other then the Spring Framework and it's modules. These are called Spring projects. E.g. Spring boot, Spring batch, Spring Security, Spring Data, Spring Cloud, etc.
* These Spring projects provide solutions for different problems faced by enterprise applications
* Spring Boot has become the most popular framework used for developing microservices.
* Spring Boot makes it very very easy to develop applications quickly. With features like start up projects, auto configuration, actuator Spring Boot makes developing applications especially micro service a cakewalk.
* Spring Cloud can be used to develop Cloud native applications. We would want to be able to dynamically configure applications, we would be able to dynamically connect them. We would want to be able to dynamically deploy applications.
* Spring data provides a consistent data access through different types of databases (SQL, NoSQL).
* Spring integration addresses problems with application integration. Spring integration helps us in connecting enterprise applications very easily.
* Spring Batch helps in developing batch processing applications.
* Spring Security provides solutions for securing your applications whether it's a web application or whether it's a REST service. Spring Security has support for multiple security options like basic authentication, OAuth authentication, OAuth2 for example.
* Spring HATEOAS enables you to develop HATEOAS compatible services.

## Why Spring is so popular

* Spring has survived for more than 15 years (as of March 2021).
* The most important reasons Spring is so popular are
  + It enables writing testable code.
  + There is no plumbing/boilerplate code at all. E.g. Spring makes all its exceptions unchecked so you don’t need to handle those
  + Spring brings in the architecture flexibility. Spring is very modular. There are Spring modules and Spring projects for very specific purposes. And I can use a specific Spring module without using all others. If I use Spring in my project my options are not really restricted.
  + It is able to stay with the trend.

# Tips and Tricks

* To use Spring boot logging, add below line in application.properties file with relevant logging level.

logging.level.<package-name> = <log-level>

e.g. logging.level.org.springframework = debug