Spring Core

Sun Tech

JSE

Standard Edition 1.18 (Standalone Applications)

Core Java

JDBC java.sql

JEE

Components

Servlet (Web Applications Development)

JSP (Java Server Page)

Java Bean

EJB 2/3 Enterprise Java Bean (Service/ Domain)

JNDI Java Naming and Directory Interface

JAX WS Java API for XML Web Services (XML/SOAP)

JAX RS Java API for XML RESTful Service

JDBC Javax.sql (Javax is the Extension of core java)

Framework

Built on top of JEE

Persistency 🡪 JDBC 🡪 ORM (Object Relational Mapping)

Different vendors for ORM are

Hibernate

MyBatis / IBatis

Toplink

Spring

Open source freeware under the PCF (Pivotal Cloud Foundry)

Prerequisites for spring

JDK 1.8

Database

MySQL

Build tool

Maven / Gradel

IDE

Java Eclipse JEE

STS (Spring Tool Suite)

JDeveloper (Oracle)

IntelliJ

Spring is a framework to develop Java EE applications using the Java resources

IOC – Inversion Of Control

If we are using the spring we need to avoid the keyword “new”. The spring will take care of object referencing and dereferencing.

Spring uses two ways to achieve the IOC

Configuration 🡪 <filename.xml> 🡪 XML based

@Annotation 🡪 Java based

DI – Dependency Injection

AOP – Aspect Oriented Programming

SPEL – Spring Expression Language ${}

XML BASED CONFIGARATION

Any .java file associated with spring, that java file is called as bean which is managed by the Spring IOC

Container (Life cycle of the bean)

BeanFactory (deprecated)

ApplicationContext extends BeanFactory

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("StudentConfig.xml");

Injection (How to pass the values to the bean)

Constructor Injection

Setter Injection

Setter injection is best compared to the constructor injection. Why? Constructor is called once at the time of creation. Whereas the object is created once, then the setter methods can be called n times.

Spring based project can be done using

IDE 🡪 Eclipse 🡪 Build Tool 🡪 Maven

STS 🡪 Spring tool suite

If our Java objects are having HAS a relationship

Dependency Injection

Using the ref tag in xml file we can establish the relations between the beans

Ex: Trainer will be related to the students.

<List> </List>

<Set> </Set>

<Map> </Map>

These tags can be used to refer the multiple objects.

Txt = abc.properties (File 🡪 New 🡪 File\*)

<bean id=*"student"* class=*"com.model.Student"*>

<property name=*"studentId"* value = *'${aaa}'*></property>

<property name=*"studentName"* value = *'${bbb}'*></property>

<property name=*"studentMark"* value = *'${ccc}'*></property>

</bean>

<bean id=*"student1"* class=*"com.model.Student"*>

<property name=*"studentId"* value = *'${aa}'*></property>

<property name=*"studentName"* value = *'${bb}'*></property>

<property name=*"studentMark"* value =*'${cc}'*></property>

</bean>

<bean id=*"trainer"* class=*"com.model.Trainer"* autowire=*"byName"*>

<property name=*"trainerId"* value=*"1587"*></property>

<property name=*"trainerName"* value=*"JKD"*></property>

<property name=*"trainerDept"* value=*"Training"*></property>

<!-- <property name="students">

<list>

<ref bean = "student"/>

<ref bean = "student1"/>

<ref bean = "student1"/>

<ref bean = "student1"/>

</list>

<set>

<ref bean = "student"/>

<ref bean = "student1"/>

</set></property> -->

</bean>

<bean class=*"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"*>

<property name=*"locations"*>

<list>

<value>classpath\*:/myproperties/abc.properties</value>

<value>classpath\*:/myproperties/efg.properties</value>

</list>

</property>

</bean>

Wiring

ByType 🡪 Referencing the one or more java files with the type of the files

ByName 🡪 Referencing the one or more java files with the NAME of the files

Spring Annotation @ Meta Info

Using @ symbol can be at variable /method/class level

@Repository (The DAO layer annotation)

@Service (Service layer annotation)

@Component (The models can be aliased by using the attribute names) //POJO models annotation

@Component (value = “<ALIASED NAME>”)

Bean calling will be the similar to the XML configuration

ApplicationContext = **new** AnnotationConfigApplicationContext(MyConfig.**class**);

package com.stg.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.stg.config.MyConfig;

import com.stg.model.Address;

public class AddressMain {

public static void main(String[] args) {

ApplicationContext applicationContext = new AnnotationConfigApplicationContext(MyConfig.class);

//Address address = (Address) applicationContext.getBean(Address.class); Calling the bean using the .class extension

Address address = (Address) applicationContext.getBean(Address.class); // Java instantiated the class and referred the instantiated class name with address

System.out.println(address.getCityName());

}}

XML + Annotation

WEB Application JEE (Web components 🡪 Reusable)

Servlet (Controller)

JSP (Java server page) 🡪 View

POJO (Model)

Design Pattern

MVC (Model View Controller)

WEB SERVICES

JEE

JAX WS Java API for XML Web Services

JAX RS Java API for RESTful Web Services

Service?

A behaviour/function/business logic which is exposed to www (world wide web) and reusable by the users without recreating it.

Frame work – JAX RS / JAX WS

RMI🡪 Remote method invocation

COM

CORBA

Web Service

MyApp 🡪 Java 🡪 Standard XML 🡪 which can be used in the other platforms like .net, .php etc

API 🡪 JAXP 🡪 Java API for XML Processing

JAXB 🡪 JAVA API for XML binding

As http won’t carry the XML we use the SOAP to send the xml through the http

Java 🡪 expose as a service 🡪 XML (SOAP – envelop) 🡪 http

JAX WS 🡪 XML WEB SERVICE 🡪 SOAP web service (Deprecated)

SOAP -> Simple Object Access Protocol

WEB Service can be create in two ways

Top Down Approach -> wsdl 🡪 .java

Tags in wsdl (tmpbs)

Types

Message

Port type

Binding

Service

Bottom up Approach 🡪

Demo to create the JAX WS

Pre Requisites

JDK 1.8

Web Server -🡪 APACHE Tom Cat

Web Service Server 🡪 AXIS (Apache Tom cat inbuilt)

IDE Eclipse

How to Test the Web services

Testing tool: SOAP UI

RESTful web services

**RESTful** Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications. This tutorial will teach you the basics of RESTful Web Services and contains chapters discussing all the basic components of RESTful Web Services with suitable examples.

REST stands for Representational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

**HTTP methods**

Following four HTTP methods are commonly used in REST based architecture.

**GET** − Provides a read only access to a resource.

**POST** − Used to create a new resource.

**DELETE** − Used to remove a resource.

**PUT** − Used to update an existing resource or create a new resource.

Every method is called as resource and accessed by the URI (Uniform Resource Identifier) (End Point)

A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

Frameworks to create the RESTful services:

Jersey Framework

RESTeasy

Spring REST

Prerequisites

JDK 1.8

Build tool 🡪 Maven

Web Service 🡪 Apache Tomcat

Spring web .\*jar

IDE 🡪 Eclipse

Step-1:

Set path and set JAVA\_HOME in environment variables for JDK

Set path and M2\_HOME in environment variables for maven

To check cmd

java-version

javac-version

mvn-version

echo%path%

Step – 2:

Create a simple maven project

Step-3:

Configure the dependencies in the pom.xml

<properties>

<spring.version>5.3.19</spring.version>

<jackson.databind-version>2.9.0</jackson.databind-version>

<org.slf4j-version>1.7.5</org.slf4j-version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring dependencies -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Jackson -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.databind-version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${org.slf4j-version}</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

Note: Jackson core

What is Jackson core used for?

Data Binding API is used **to convert JSON to and from POJO (Plain Old Java Object) using property accessor or using annotations**. It is of two type. Simple Data Binding - Converts JSON to and from Java Maps, Lists, Strings, Numbers, Booleans and null objects.

sTEP-4:

Create WEB-INF

STEP-5:

Create a front controller web.xml under the WEB-INF

Step-6:

Configure the spring configuration file dispatcher-servlet.xml under the WEB-INF

Step-7:

Create the necessary packages

Step-8:

Create the required beans (POJO Files)

Step-9:

Create the REST controller src/main/java/com.controller

package com.controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.stg.model.User;

@RestController

public class UserController {

@RequestMapping (value = "/getuser")

public User getUser() {

return new User(10, "Bharath", "Chinna@239");

}

}

**SOA (Service Oriented Architecture)**

What Is Service-Oriented Architecture?

Service-Oriented Architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network. Its principles are independent of vendors and other technologies. In[**service oriented architecture**](http://santexgroup.com/), a number of services communicate with each other, in one of two ways: through passing data or through two or more services coordinating an activity. This is just one definition of Service-Oriented Architecture. [An article on Wikipedia goes into much more detail.](https://en.wikipedia.org/wiki/Service-oriented_architecture#Event-driven_architectures)

Characteristics Of Service-Oriented Architecture

While the defining concepts of Service-Oriented Architecture vary from company to company, there are six key tenets that overarch the broad concept of Service-Oriented Architecture. These core values include:

Business value

Strategic goals

Intrinsic inter-operability

Shared services

Flexibility

Evolutionary refinement

Each of these core values can be seen on a continuum from older format distributed computing to Service-Oriented Architecture to cloud computing (something that is often seen as an offshoot of Service-Oriented Architecture).

Service-Oriented Architecture Patterns



There are three roles in each of the Service-Oriented Architecture building blocks: service provider; service broker, service registry, service repository; and service requester/consumer.

The service provider works in conjunction with the service registry, debating the whys and hows of the services being offered, such as security, availability, what to charge, and more. This role also determines the service category and if there need to be any trading agreements.

The service broker makes information regarding the service available to those requesting it. The scope of the broker is determined by whoever implements it.

The service requester locates entries in the broker registry and then binds them to the service provider. They may or may not be able to access multiple services; that depends on the capability of the service requester.

Implementing Service-Oriented Architecture

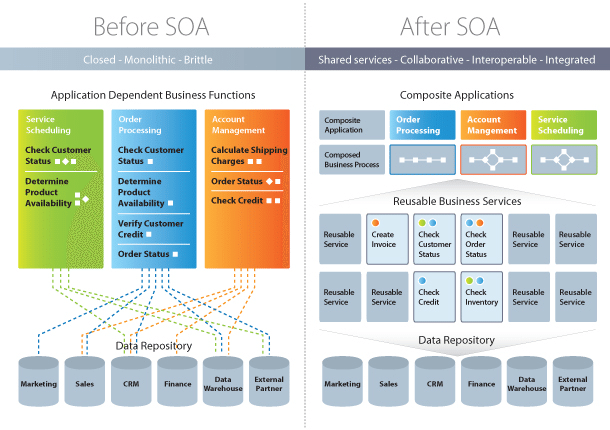
When it comes to implementing service-oriented architecture (SOA), there is a wide range of technologies that can be used, depending on what your end goal is and what you’re trying to accomplish.

Typically, Service-Oriented Architecture is implemented with web services, which makes the “functional building blocks accessible over standard internet protocols.”

An example of a web service standard is [SOAP](https://en.wikipedia.org/wiki/SOAP), which stands for Simple Object Access Protocol. In a nutshell, SOAP “is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. Although SOAP wasn’t well-received at first, since 2003 it has gained more popularity and is becoming more widely used and accepted. Other options for implementing Service-Oriented Architecture include Jini, COBRA, or REST.

It’s important to note that architectures can “operate independently of specific technologies,” which means they can be implemented in a variety of ways, including messaging, such as ActiveMQ; Apache Thrift; and SORCER.

Why Service-Oriented Architecture Is Important



There are many [benefits to service-oriented architecture](https://www.bmc.com/blogs/service-oriented-architecture-overview/), especially in a web service based business. We’ll outline a few of those benefits here, in brief:

Use Service-Oriented Architecture to create reusable code: Not only does this cut down on time spent on the development process, but there’s no reason to reinvent the coding wheel every time you need to create a new service or process. Service-Oriented Architecture also allows for using multiple coding languages because everything runs through a central interface.

Use Service-Oriented Architecture to promote interaction: With Service-Oriented Architecture, a standard form of communication is put in place, allowing the various systems and platforms to function independent of each other. With this interaction, Service-Oriented Architecture is also able to work around firewalls, allowing “companies to share services that are vital to operations.”

Use Service-Oriented Architecture for scalability: It’s important to be able to scale a business to meet the needs of the client, however certain dependencies can get in the way of that scalability. Using Service-Oriented Architecture cuts back on the client-service interaction, which allows for greater scalability.

Use Service-Oriented Architecture to reduce costs: With Service-Oriented Architecture, it’s possible to reduce costs while still “maintaining a desired level of output.” Using Service-Oriented Architecture allows businesses to limit the amount of analysis required when developing custom solutions.

How Service-Oriented Architecture And Cloud Computing Work Together

First, it’s important to note that Service-Oriented Architecture can work with or without cloud computing, although more and more businesses are moving file storage to the cloud so it makes sense to use cloud computing and Service-Oriented Architecture together.

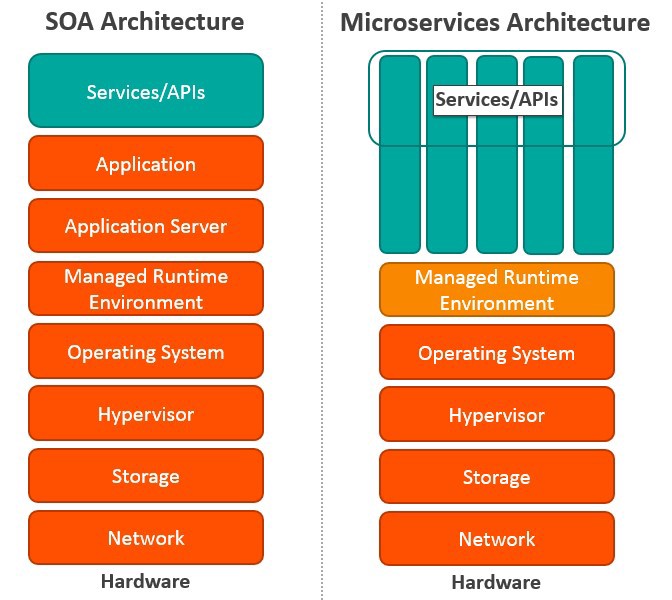
In a nutshell, [using cloud computing](https://www.oracle.com/technetwork/articles/soa/ind-soa-cloud-2190513.html) allows users to easily and immediately implement services tailored to the requirements of their clients, “without needing to consult an IT department.” One downfall of using Service-Oriented Architecture and cloud computing together is that some aspects of it are not evaluated, such as security and availability. When using cloud computing, users are often at the mercy of the provider. There is one fairly major challenge businesses face when merging cloud computing and Service-Oriented Architecture is the integration of existing data and systems into the cloud solution. There needs to be continuity from beginning to end in order for there to be a seamless transition. It’s also important to keep in mind that not every IT aspect can be outsourced to the cloud — there are some things that still need to be done manually. You can read more about how [service-oriented architecture and cloud computing work together right here](https://www.oracle.com/technetwork/articles/soa/ind-soa-cloud-2190513.html).

The Difference Between Service-Oriented Architecture and SaaS



We’ve talked quite a bit about what Service-Oriented Architecture is and how it can be used to advance your business. But there’s also [SaaS (Software as a Service)](https://apprenda.com/library/architecture/soa-vs-saas-whats-the-difference/), which can also be used to advance your business. You may be wondering what SaaS is and how it differs from Service-Oriented Architecture. In brief, the resources available through SaaS are software applications. A key component is that the SaaS infrastructure is “available to, but hidden, from users.” An advantage of SaaS is that users don’t have to both install and maintain software, which eliminates any complex requirements. With SaaS, the customer also doesn’t require any up-front licensing, which leads to lower costs because providers are only maintaining a single application.

Differences Between Service-Oriented Architecture and Micro services



Microservices, also known as Microservice Architecture, is an “architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.”

While microservices and Service-Oriented Architecture are similar in some ways, the key differences come in their functionality. Services is, obviously, the main component of both. There are four basic types of services:

Functional service: these define core business operations

Enterprise service: these implement the functionality defined by the functional services

Application service: these are confined to specific application content

Infrastructure service: implements non-functional tasks such as authentication, auditing, security, and logging

As you can see, each of these services builds on the one before it, creating a system that is not only easy to use, but provides you with a variety of ways to manage your business. As with any functionality, it’s a matter of figuring out what works best for you and your business.

[You can read more about service-oriented architecture and microservices here](https://www.quora.com/What-is-the-difference-between-SOA-and-microservices).

Some Final Thoughts On Service-Oriented Architecture



As you can imagine, service-oriented architecture can be a bit of a tough nut to crack, but once you understand the nuts and bolts of it and the benefits it can provide your company, you’ll be thrilled you discovered it.

We have provided several excellent service-oriented architecture resources throughout this article and have a couple more articles forthcoming that will take a deeper dive into the concept and provide you with even more resources to help you push your business to the next level.

No matter which direction you decide to go when providing services to your clients, it’s important to keep in mind that different things will work for different people. While you can’t provide custom services for every single client you’ve taken on, you can provide a range of services that will fit the most common needs of your clients.

In future articles, we will look at Web 2.0 and dive deeper into Service-Oriented Architecture Microservices.

5