**Spring Framework**

* **Introduction to Spring Framework**

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

1. Make a Java method execute in a database transaction without having to deal with transaction APIs.
2. Make a local Java method a remote procedure without having to deal with remote APIs.
3. Make a local Java method a management operation without having to deal with JMX APIs.
4. Make a local Java method a message handler without having to deal with JMS APIs.

* **Dependency Injection and Inversion of Control**

Java applications -- a loose term that runs the gamut from constrained applets to n-tier server-side enterprise applications -- typically consist of objects that collaborate to form the application proper. Thus the objects in an application have dependencies on each other.

Although the Java platform provides a wealth of application development functionality, it lacks the means to organize the basic building blocks into a coherent whole, leaving that task to architects and developers. True, you can use design patterns such as Factory, Abstract Factory, Builder, Decorator, and Service Locator to compose the various classes and object instances that make up an application. However, these patterns are simply that: best practices given a name, with a description of what the pattern does, where to apply it, the problems it addresses, and so forth. Patterns are formalized best practices that you must implement yourself in your application.

* **Spring Core**

Spring relies on an XML file which holds the definitions of various been that we can configure. The advantage is that we can fetch the object of the bean any number of times without having to worry how to create it. Spring internally uses XML Bean factory or classpath application context in order to instantiate objects of beans.

**Code**: Injecting simple bean object

pom.xml

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.0.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.0.8.RELEASE</version>

</dependency>

</dependencies>

Circle.java

package com;

public class Circle {

String name;

int radius;

public Circle(String name, int radius) {

this.name = name;

this.radius = radius;

}

@Override

public String toString() {

return "Circle [name=" + name + ", radius=" + radius + "]";

}

}

Main Class

package com;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support. ClassPathXmlApplicationContext;

public class TestSpringMain {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

Circle c = (Circle) context.getBean("circle");

System.out.println(c);

}

}

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="circle" class="com.Circle">

<constructor-arg value="10" type="int"/>

<constructor-arg value="Basic Circle" />

</bean>

</beans>

**Code**: Injecting object inside another object

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"

"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

<bean id="PointTriangle" class="containment.PointTriangle">

<property name="a" ref="pointA"></property>

<property name="b" ref="pointB"></property>

<property name="c">

<bean class="containment.Point">

<property name="x" value="5"></property>

<property name="y" value="5"></property>

</bean>

</property>

</bean>

<bean id="pointA" class="containment.Point">

<property name="x" value="0"></property>

<property name="y" value="0"></property>

</bean>

<bean id="pointB" class="containment.Point">

<property name="x" value="10"></property>

<property name="y" value="10"></property>

</bean>

</beans>

PointTriangle.java

package containment;

public class PointTriangle {

Point a,b,c;

public PointTriangle() {

}

public PointTriangle(Point a, Point b, Point c) {

super();

this.a = a;

this.b = b;

this.c = c;

}

public Point getA() {

return a;

}

public void setA(Point a) {

this.a = a;

}

public Point getB() {

return b;

}

public void setB(Point b) {

this.b = b;

}

public Point getC() {

return c;

}

public void setC(Point c) {

this.c = c;

}

public void draw() {

System.out.println("Triangle is drawn:");

System.out.println("Point 1: " + a.getX() + "," + a.getY());

System.out.println("Point 2: " + b.getX() + "," + b.getY());

System.out.println("Point 3: " + c.getX() + "," + c.getY());

}

}

Point.java

package containment;

public class Point {

int x,y;

public Point() {

}

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

}

Main Class

package com;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import containment.\*;

public class DrawingApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

PointTriangle triangle = (PointTriangle)context.getBean("PointTriangle");

triangle.draw();

}

}

Code: Injecting list inside another object

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"

"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

<bean id="PointTriangle" class="containment.PointTriangle">

<property name="points">

<list>

<ref bean="pointA" />

<ref bean="pointB" />

<ref bean="pointC" />

</list>

</property>

</bean>

<bean id="pointC" class="containment.Point">

<property name="x" value="5"></property>

<property name="y" value="5"></property>

</bean>

<bean id="pointA" class="containment.Point">

<property name="x" value="0"></property>

<property name="y" value="0"></property>

</bean>

<bean id="pointB" class="containment.Point">

<property name="x" value="10"></property>

<property name="y" value="10"></property>

</bean>

</beans>

Bean Class

public class PointTriangle {

List<Point> points;

public PointTriangle() {}

// rest of the class

}

Code: Injecting map inside another object

Spring.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"

"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

<bean id="pointA" class="containment.Point">

<property name="x" value="0"></property>

<property name="y" value="0"></property>

</bean>

<bean id="pointB" class="containment.Point">

<property name="x" value="10"></property>

<property name="y" value="10"></property>

</bean>

<bean id="pointC" class="containment.Point">

<property name="x" value="5"></property>

<property name="y" value="5"></property>

</bean>

<bean id="pointTriangle" class="containment.PointTriangle">

<property name="type" value="Scalene"></property>

<property name="points">

<map>

<entry key="1" value-ref="pointA"></entry>

<entry key="2" value-ref="pointB"></entry>

<entry key="3" value-ref="pointC"></entry>

</map>

</property>

</bean>

</beans>

* Web Services and Spring BOOT

The term web service is either of the following:

(**generic**): a service offered by an electronic device to another electronic device, communicating with each other via the World Wide Web, or

(**specific**): a web service implemented in the particular technology or brand, W3C Web Services.

In a web service, the Web technology such as HTTP—originally designed for human-to-machine communication—is utilized for machine-to-machine communication, more specifically for transferring machine-readable file formats such as XML and JSON.

In practice, a web service commonly provides an object-oriented web-based interface to a database server, utilized for example by another web server, or by a mobile app, that provides a user interface to the end user. Many organizations that provide data in formatted HTML pages will also provide that data on their server as XML or JSON, often through a web service to allow syndication. Another application offered to the end user may be a mashup, where a web server consumes several web services at different machines, and compiles the content into one user interface.

**REST:**

REST (REpresentational State Transfer) is an architectural style for developing web services. REST is popular due to its simplicity and the fact that it builds upon existing systems and features of the internet's HTTP in order to achieve its objectives, as opposed to creating new standards, frameworks and technologies.

**Advantages of REST**

A primary benefit of using REST, both from a client and server's perspective, is REST-based interactions happen using constructs that are familiar to anyone who is accustomed to using the internet's Hypertext Transfer Protocol (HTTP).

An example of this arrangement is REST-based interactions all communicate their status using standard HTTP status codes. So, a 404 means a requested resource wasn't found; a 401 code means the request wasn't authorized; a 200 code means everything is OK; and a 500 means there was an unrecoverable application error on the server.

Similarly, details such as encryption and data transport integrity are solved not by adding new frameworks or technologies, but instead by relying on well-known Secure Sockets Layer (SSL) encryption and Transport Layer Security (TLS). So, the entire REST architecture is built upon concepts with which most developers are already familiar.

REST is also a language-independent architectural style. REST-based applications can be written using any language, be it Java, Kotlin, .NET, AngularJS or JavaScript. As long as a programming language can make web-based requests using HTTP, it is possible for that language to be used to invoke a RESTful API or web service. Similarly, RESTful web services can be written using any language, so developers tasked with implementing such services can choose technologies that work best for their situation.

The other benefit of using REST is its pervasiveness. On the server side, there are a variety of REST-based frameworks for helping developers create RESTful web services, including RESTlet and Apache CXF. From the client side, all of the new JavaScript frameworks, such as JQuery, Node.js, Angular and EmberJS, all have standard libraries built into their APIs that make invoking RESTful web services and consuming the XML- or JSON-based data they return a relatively straightforward endeavor.

**Disadvantages of REST**

The benefit of REST using HTTP constructs also creates restrictions, however. Many of the limitations of HTTP likewise turn into shortcomings of the REST architectural style. For example, HTTP does not store state-based information between request-response cycles, which means REST-based applications must be stateless and any state management tasks must be performed by the client.

Similarly, since HTTP doesn't have any mechanism to send push notifications from the server to the client, it is difficult to implement any type of services where the server updates the client without the use of client-side polling of the server or some other type of web hook.

From an implementation standpoint, a common problem with REST is the fact that developers disagree with exactly what it means to be REST-based. Some software developers incorrectly consider anything that isn't SOAP-based to be RESTful. Driving this common misconception about REST is the fact that it is an architectural style, so there is no reference implementation or definitive standard that will confirm whether a given design is RESTful. As a result, there is discourse as to whether a given API conforms to REST-based principles.

**Alternatives to REST**

Alternate technologies for creating SOA-based systems or creating APIs for invoking remote microservices include XML over HTTP (XML-RPC), CORBA, RMI over IIOP and the Simple Object Access Protocol (SOAP).

Each technology has its own set of benefits and drawbacks, but the compelling feature of REST that sets it apart is the fact that, rather than asking a developer to work with a set of custom protocols or to create a special data format for exchanging messages between a client and a server, REST insists the best way to implement a network-based web service is to simply use the basic construct of the network protocol itself, which in the case of the internet is HTTP.

This is an important point, as REST is not intended to apply just to the internet; rather, its principles are intended to apply to all protocols, including WEBDAV, FTP and so on.

**REST vs. SOAP**

The two competing styles for implementing web services are REST and SOAP. The fundamental difference between the two is the philosophical approach the two have to remotely invocations.

REST takes a resource-based approach to web-based interactions. With REST, you locate a resource on the server, and you choose to either update that resource, delete it or get some information about it.

With SOAP, the client doesn't choose to interact directly with a resource, but instead calls a service, and that service mitigates access to the various objects and resources behind the scenes.

SOAP has also built a large number of frameworks and APIs on top of HTTP, including the Web Services Description Language (WSDL), which defines the structure of data that gets passed back and forth between the client and the server.

Some problem domains are served well by the ability to stringently define the message format, or can benefit from using various SOAP-related APIs, such as WS-Eventing, WS-Notification and WS-Security. There are times when HTTP cannot provide the level of functionality an application might require, and in these cases, using SOAP is preferable.

**REST URIs and URLs**

Most people are familiar with the way URLs and URIs work on the web. A RESTful approach to developing applications asserts that requesting information about a resource should be as simple as invoking its URL.

For example, if a client wanted to invoke a web service that listed all of the quizzes available here at TechTarget, the URL to the web service would look something like this:

www.techtarget.com/restfulapi/quizzes

When invoked, the web service might respond with the following JSON string listing all of the available quizzes, one of which is about DevOps:

{ "quizzes" : [ "Java", "DevOps", "IoT"] }

To get the DevOps quiz, the web service might be called using the following URL:

www.techtarget.com/restfulapi/quizzes/DevOps

Invoking this URL would return a JSON string listing all of the questions in the DevOps quiz. To get an individual question from the quiz, the number of the question would be added to the URL. So, to get the third question in the DevOps quiz, the following RESTful URL would be used:

www.techtarget.com/restfulapi/quizzes/DevOps/3

Invoking that URL might return a JSON string such as the following:

{ "Question" : {"query":"What is your DevOps role?", "optionA":"Dev", "optionB":"Ops"} }

As you can see, the REST URLs in this example are structured in a logical and meaningful way that identifies the exact resource being requested.

**JSON and XML REST data formats**

The example above shows JSON used as the data exchange format for the RESTful interaction. The two most common data exchange formats are JSON and XML, and many RESTful web services can use both formats interchangeably, as long as the client can request the interaction to happen in either XML or JSON.

Note that while JSON and XML are popular data exchange formats, REST itself does not put any restrictions on what the format should be. In fact, some RESTful web services exchange binary data for the sake of efficiency. This is another benefit to working with REST-based web services, as the software architect is given a great deal of freedom in terms of how best to implement a service.

**REST and the HTTP methods**

The example above only dealt with accessing data.

The default operation of HTTP is GET, which is intended to be used when getting data from the server. However, HTTP defines a number of other methods, including PUT, POST and DELETE.

The REST philosophy asserts that to delete something on the server, you would simply use the URL for the resource and specify the DELETE method of HTTP. For saving data to the server, a URL and the PUT method would be used. For operations that are more involved than simply saving, reading or deleting information, the POST method of HTTP can be used.

**Use of a uniform interface (UI).**

As stated earlier, resources in REST-based systems should be uniquely identifiable through a single URL, and only by using the underlying methods of the network protocol, such as DELETE, PUT and GET with HTTP, should it be possible to manipulate a resource.

**Client-server-based.**

In a REST-based system, there should be a clear delineation between the client and the server. UI and request-generating concerns are the domain of the client. Meanwhile, data access, workload management and security are the domain of the server. This separation allows loose coupling between the client and the server, and each can be developed and enhanced independent of the other.

**Stateless operations.**

All client-server operations should be stateless, and any state management that is required should happen on the client, not the server.

**RESTful resource caching.**

The ability to cache resources between client invocations is a priority in order to reduce latency and improve performance. As a result, all resources should allow caching unless an explicit indication is made that it is not possible.

Code: Creating Web Service using Jersey implementation (JAX/RS)

package org.posts;

@Path("posts")

public class MyResource {

PostsDAOImpl impl = new PostsDAOImpl();

@Path("all")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public List<Post> getPosts() {

ArrayList<Post> list = (ArrayList<Post>) impl.getPosts();

return list;

}

@Path("/{title}")

@GET

@Produces(MediaType.APPLICATION\_JSON)

public ArrayList<Post> getPost(@PathParam("title") String title) {

ArrayList<Post> posts = impl.getPost(title);

return posts;

}

@Path("/add")

@POST

@Produces(MediaType.APPLICATION\_JSON)

@Consumes(MediaType.APPLICATION\_JSON)

public Post addPost(Post post) {

post = impl.addPost(post);

System.out.println("Added Posts's Id: " + post.id);

return post;

}

@Path("update/{id}")

@PUT

@Produces(MediaType.APPLICATION\_JSON)

@Consumes(MediaType.APPLICATION\_JSON)

public Status updatePost(@PathParam("id") int id, Post post) {

post.setId(id);

Status s = impl.updatePost(post);

return s;

}

@Path("delete/{id}")

@DELETE

@Produces(MediaType.APPLICATION\_JSON)

public Status deletePost(@PathParam("id") int id) {

Post post = new Post();

post.setId(id);

Status s = impl.deletePost(post);

return s;

}

}

**Spring Boot**

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". Most Spring Boot applications need very little Spring configuration.

1. Create stand-alone Spring applications
2. Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
3. Provide opinionated 'starter' dependencies to simplify your build configuration
4. Automatically configure Spring and 3rd party libraries whenever possible
5. Provide production-ready features such as metrics, health checks and externalized configuration
6. Absolutely no code generation and no requirement for XML configuration

**Using BOOT**

**Developing Your First Spring Boot Application**

This section describes how to develop a simple “Hello World!” web application that highlights some of Spring Boot’s key features. We use Maven to build this project, since most IDEs support it.

**[Tip]**

The spring.io web site contains many “Getting Started” guides that use Spring Boot. If you need to solve a specific problem, check there first.

You can shortcut the steps below by going to start.spring.io and choosing the "Web" starter from the dependencies searcher. Doing so generates a new project structure so that you can start coding right away. Check the Spring Initializr documentation for more details.

Before we begin, open a terminal and run the following commands to ensure that you have valid versions of Java and Maven installed:

$ java -version

java version "1.8.0\_102"

Java(TM) SE Runtime Environment (build 1.8.0\_102-b14)

Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)

$ mvn -v

Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T16:41:47+00:00)

Maven home: /usr/local/Cellar/maven/3.3.9/libexec

Java version: 1.8.0\_102, vendor: Oracle Corporation

[Note]

This sample needs to be created in its own folder. Subsequent instructions assume that you have created a suitable folder and that it is your current directory.

1. **Creating the POM**

We need to start by creating a Maven pom.xml file. The pom.xml is the recipe that is used to build your project. Open your favorite text editor and add the following:

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>myproject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.4.RELEASE</version>

</parent>

<!-- Additional lines to be added here... -->

</project>

The preceding listing should give you a working build. You can test it by running mvn package (for now, you can ignore the “jar will be empty - no content was marked for inclusion!” warning).

[Note]

At this point, you could import the project into an IDE (most modern Java IDEs include built-in support for Maven). For simplicity, we continue to use a plain text editor for this example.

2. **Adding Classpath Dependencies**

Spring Boot provides a number of “Starters” that let you add jars to your classpath. Our sample application has already used spring-boot-starter-parent in the parent section of the POM. The spring-boot-starter-parent is a special starter that provides useful Maven defaults. It also provides a dependency-management section so that you can omit version tags for “blessed” dependencies.

Other “Starters” provide dependencies that you are likely to need when developing a specific type of application. Since we are developing a web application, we add a spring-boot-starter-web dependency. Before that, we can look at what we currently have by running the following command:

$ mvn dependency:tree

[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT

The mvn dependency:tree command prints a tree representation of your project dependencies. You can see that spring-boot-starter-parent provides no dependencies by itself. To add the necessary dependencies, edit your pom.xml and add the spring-boot-starter-web dependency immediately below the parent section:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

If you run mvn dependency:tree again, you see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

3. **Writing the Code**

To finish our application, we need to create a single Java file. By default, Maven compiles sources from src/main/java, so you need to create that folder structure and then add a file named src/main/java/Example.java to contain the following code:

import org.springframework.boot.\*;

import org.springframework.boot.autoconfigure.\*;

import org.springframework.web.bind.annotation.\*;

@RestController

@EnableAutoConfiguration

public class Example {

@RequestMapping("/")

String home() {

return "Hello World!";

}

public static void main(String[] args) throws Exception {

SpringApplication.run(Example.class, args);

}

}

Although there is not much code here, quite a lot is going on. We step through the important parts in the next few sections.

3.1 **The @RestController and @RequestMapping Annotations**

The first annotation on our Example class is @RestController. This is known as a stereotype annotation. It provides hints for people reading the code and for Spring that the class plays a specific role. In this case, our class is a web @Controller, so Spring considers it when handling incoming web requests.

The @RequestMapping annotation provides “routing” information. It tells Spring that any HTTP request with the / path should be mapped to the home method. The @RestController annotation tells Spring to render the resulting string directly back to the caller.

**[Tip]**

The @RestController and @RequestMapping annotations are Spring MVC annotations. (They are not specific to Spring Boot.) See the MVC section in the Spring Reference Documentation for more details.

3.2 **The @EnableAutoConfiguration Annotation**

The second class-level annotation is @EnableAutoConfiguration. This annotation tells Spring Boot to “guess” how you want to configure Spring, based on the jar dependencies that you have added. Since spring-boot-starter-web added Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

**Starters and Auto-configuration**

Auto-configuration is designed to work well with “Starters”, but the two concepts are not directly tied. You are free to pick and choose jar dependencies outside of the starters. Spring Boot still does its best to auto-configure your application.

3.3 **The “main” Method**

The final part of our application is the main method. This is just a standard method that follows the Java convention for an application entry point. Our main method delegates to Spring Boot’s SpringApplication class by calling run. SpringApplication bootstraps our application, starting Spring, which, in turn, starts the auto-configured Tomcat web server. We need to pass Example.class as an argument to the run method to tell SpringApplication which is the primary Spring component. The args array is also passed through to expose any command-line arguments.

4. **Running the Example**

At this point, your application should work. Since you used the spring-boot-starter-parent POM, you have a useful run goal that you can use to start the application. Type mvn spring-boot:run from the root project directory to start the application. You should see output similar to the following:

$ mvn spring-boot:run

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.0.4.RELEASE)

....... . . .

....... . . . (log output here)

....... . . .

........ Started Example in 2.222 seconds (JVM running for 6.514)

If you open a web browser to localhost:8080, you should see the following output:

Hello World!

Code: Posts Web Service

package posts;

import java.util.ArrayList;

import java.util.List;

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

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

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

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

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

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

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

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

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

import com.posts.Post;

import service.PostsDAOImpl;

import service.Status;

@RestController

@CrossOrigin(methods = { RequestMethod.GET, RequestMethod.DELETE, RequestMethod.POST, RequestMethod.PUT })

@RequestMapping("/posts")

public class PostController {

PostsDAOImpl impl = new PostsDAOImpl();

@RequestMapping("/all")

public List<Post> getPosts() {

ArrayList<Post> list = (ArrayList<Post>) impl.getPosts();

return list;

}

@RequestMapping("/{title}")

public List<Post> getPost(@PathVariable String title) {

ArrayList<Post> list = (ArrayList<Post>) impl.getPost(title);

return list;

}

@RequestMapping("/add")

@PostMapping

public Post addPost(@RequestBody Post post) {

post = impl.addPost(post);

return post;

}

@RequestMapping("/update/{id}")

@PutMapping

public Status updatePost(@PathVariable int id, @RequestBody Post post) {

post.setId(id);

Status s = impl.updatePost(post);

return s;

}

@RequestMapping("/delete/{id}")

@DeleteMapping

public Status deletePost(@PathVariable("id") int id) {

Post post = new Post();

post.setId(id);

Status s = impl.deletePost(post);

return s;

}

}