Spring Web Services

Contents

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?

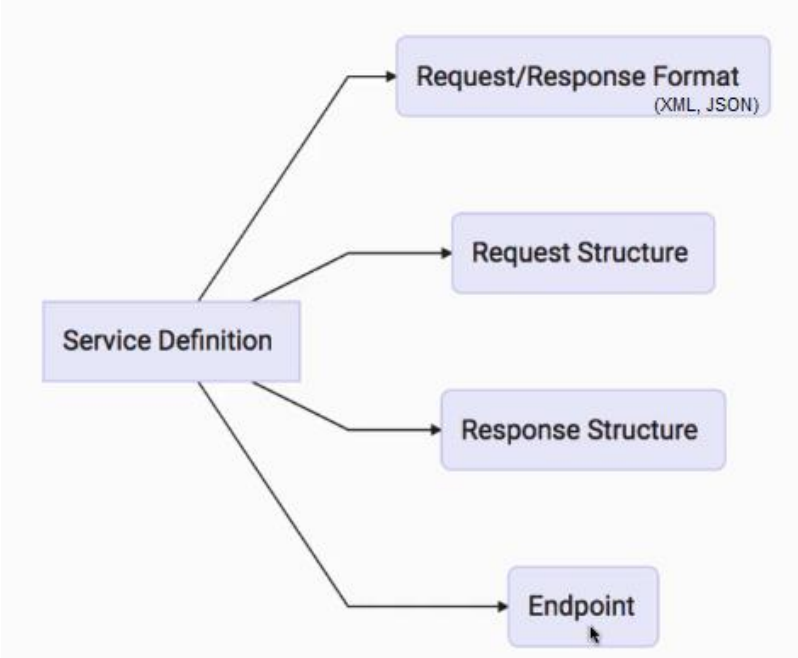
- The input to a web service is called a Request and the output from a web service is called a Response.

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.

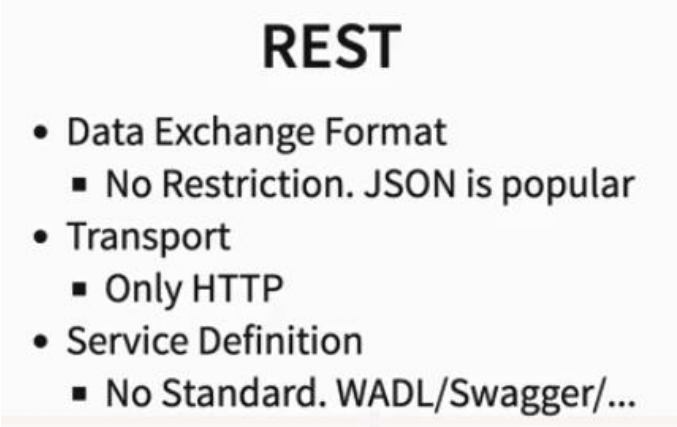
Introduction to SOAP Web Services

- 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.

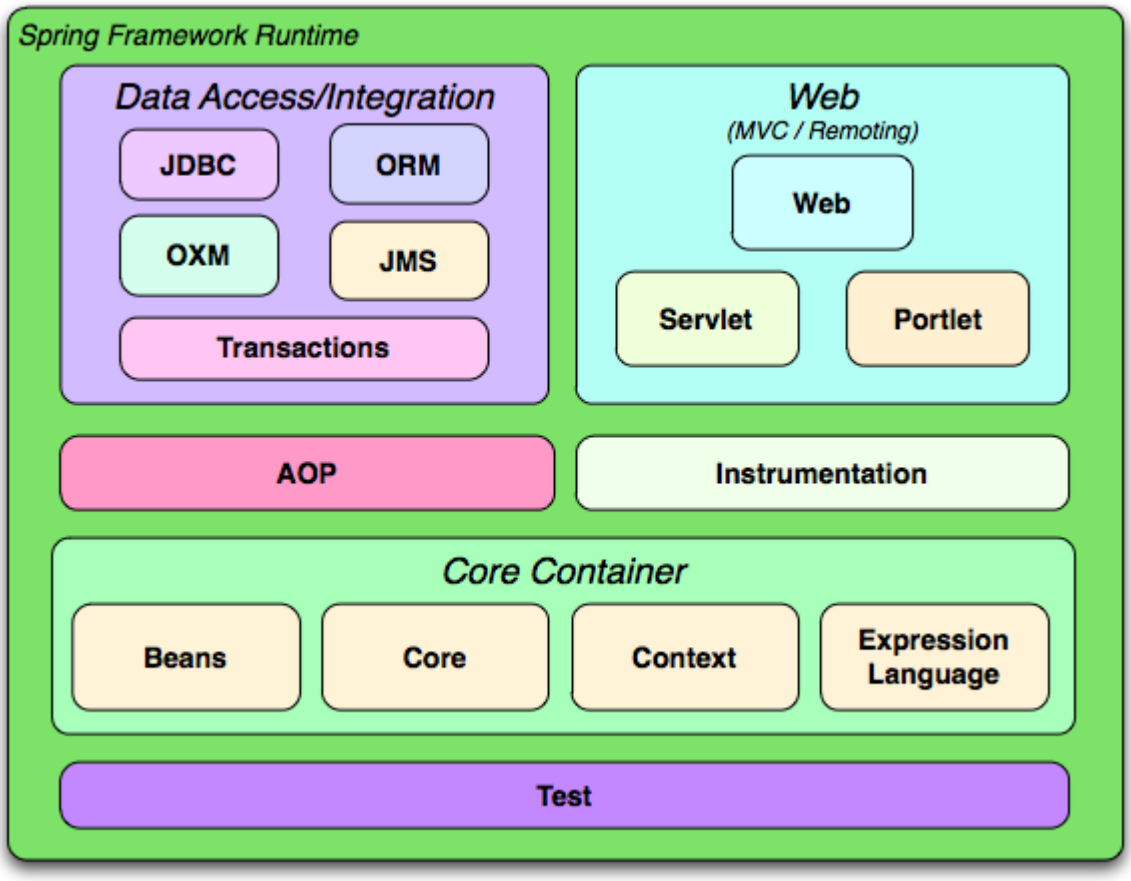
How it works in the background

- 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.

Introduction to Spring Boot

- Spring Boot is the most popular framework to develop Micro services.
- Refer – https://github.com/in28minutes/spring-web-services/tree/master/springboot-in-10-steps

Goals

- The most important goal of Spring Boot is to enable building production ready applications **quickly**.
- The other important goal is to provide all the common non functional features like embedded servers, metrics health checks and externalized configuration.

What Spring Boot is NOT

- There is no code generation at all.
- Spring Boot is neither an application server nor a web server.
- Spring Boot provides great integration with embedded servers like Tomcat, Jetty but by itself Spring Boot is not a web server nor an application server.

Features

- Quick Starter projects with Auto Configuration.
    - For instance, if let's say I want to develop a web application I would need Spring MVC. I would need Spring core. I would need some validation framework. I would need some logging framework. In addition to that, I would need to configure all this stuff that is needed. For example, if I'm using Spring MVC, I would need to configure dispatcher servlet. I would need to configure view resolver and a lot of things like that. However, with Spring Boot starter project it becomes very easy – just add starter called Spring Boot Starter Web.
    - Similarly, for JPA there is a starter called Spring Boot starter JPA. Once we use this starter, you would not only get JPA but also a default implementation of JPA with Hibernate and also auto configuration of that.
- Embedded Servers
    - What you can do is you can package your server. So you can package Tomcat along with your application jar. So I can include the Tomcat server in my application jar. So I don't need to install it.
- Production Ready Features out of the box
    - Spring Boot provides monitoring for your application through something called Spring Boot actuator.

- Externalised configuration
    - The configuration of applications varies between different environments. Your configuration from dev different from your configuration in production. Spring Boot provides these features built in.
- Support for different profiles

Spring Boot Auto Configuration

- You can create REST services very easily in Spring boot. But How did all the things that are needed for the REST service to be up and running get configured automatically?
- The annotation **@SpringBootApplication** indicates that this is a Spring context file/class.
- @SpringBootApplication enables auto configuration and component scan.
- Component scan would start automatically scanning the classes in this package (so that package in which the class annotated with @SpringBootApplication is) and its sub package for any beans. (Bean is a class which is annotated with @Component. Also @RestController in the end extends @Component).
- SpringApplication.run() is used to run the Spring Context by passing SpringContext to it.
- One of the dependencies you can see in your Spring boot project is the Spring-Boot-Autoconfigure jar.
- At application startup, Spring Boot framework would trigger the auto configuration jar. And it would loop through classes which are on the classpath (including spring related jars). E.g. If it sees a jar called spring-webmvc jar in classpath, it would configure the dispatcher servet, view resolver, etc.
- Spring Boot looks at A) the frameworks which are available on the classpath and also, B) It looks at the existing beans which are configured for the application. And based on that, it provides the configuration needed.
- Run the application in DEBUG logging to see all the details about Auto configuration. You can turn it on by adding this line in application.properties file. logging.level.org.springframework=DEBUG
- Auto configuration is one of the fundamental things why Spring Boot is so famous and so easy to use.

Spring VS Spring MVC VS Spring Boot

- Refer – https://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring

Spring (Framework)

- Most important feature of Spring Framework is Dependency Injection. At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.

- It helps you to build loosely coupled applications. Loosely coupled applications can be easily unit tested.
- Spring Framework solves the problem of duplication/plumbing/boilerplate code.
- Spring Framework allows good integration with other frameworks.

Spring MVC

- Spring MVC is concerned with developing web applications. Spring MVC provides a simple way of developing web applications.
- Spring MVC makes it very easy to develop your web applications as well as RESTful services.

Spring Boot

- For faster development.
- Provides great start projects like Spring Boot Web, etc.

Spring Boot Actuator

- To use spring boot actuator, just include spring-boot-starter-actuator dependency.
- Actuator bring in a lot of monitoring around your application.
- Actuator actually exposes a lot of the rest services and these services are compliant with the standard called "HAL" standard.
  (More details on HAL - https://www.baeldung.com/spring-rest-hal)
- We also need to add dependency for the HAL browser as spring-data-rest-hal-browser. When you add HAL browser and when you hit localhost:8080 you can browse through the actuators.
- Now once your application is running, you can monitor the health of your application via /actuator endpoint. E.g. http://localhost:8080/actuator
- There many properties related to actuator available which you can use to get different info from the actuator.
- One of the important things to remember is when you enable a lot of tracking a lot of auditing, there would be a performance impact as well.
- One of the recommendations is to identify which of the things that you would want and only enable those.

Spring Boot Developer Tools

- This is very helpful dependency to have. With this you don't need to keep on restarting the server in order to verify your changes as you are coding it.
- The dependency is **spring-boot-devtools**. It restarts the server very efficiently when it notices new changes. Basically it loads only application beans and not the framework beans.

SOAP Web Services with Spring and Spring Boot

- Refer – https://github.com/in28minutes/spring-web-services/tree/master/soap-web-services

WSDL

These are the different details that are typically present in all the WSDLs.

- At a high level **types** defines all the different elements.
- **messages** define what are the requests and responses.
- **portType** defines the interface. Basically, what are the operations that are available.
- **binding** maps the operations to how you are exposing them. Whether you're using SOAP over HTTP or SOAP over MQ.
- And **service** maps it to an end point. What is the URL which customers can use to call this web service.

WS Security

- WS Security is basically a set of specifications which actually try to provide security around your SOAP based web services.
- WS security is transport independent. Whether you're using HTTP, whether using MQ, it does not really matter.
- WS security provides specifications for authentication based on passwords, based on digital signatures, and when you're using certificates.
- **XWSS** is one of the important implementations of WS security. It stands for XML and Web service security.

RESTful Web Services with Spring and Spring Boot

- Refer – https://github.com/in28minutes/spring-web-services/tree/master/restful-web-services

Behind the Scenes

- Spring Boot auto configuration does most of the things in the background.
- E.g. it configures the dispatcher servlet, initializes the message converters (e.g. Jackson converter to convert from java to json and vice versa, etc.), configures the error mappings, etc. Also maps dispatcher servlet to application root (/) to handle all requests.
- Dispatcher servlet is the front controller for spring web MVC framework.
- @RestController annotation is itself annotated with @ResponseBody annotation.
- When we put @ResponseBody annotation on a controller, the response from that controller class would be mapped by a message convertor into some other format. E.g. Jackson conveter.

Annotations used in REST services created using Spring Boot

- @RestController
- @GetMapping, @PostMapping, etc.
- @PathVariable
- @RequestBody
- @RequestParam
- @ResponseStatus
- @ControllerAdvice – used when we have multiple controller classes and we would want to share things across them.
- @ExceptionHandler
- @Valid – part of Java Validation API (javax.validation.*).

HATEOAS (Hypermedia as the Engine of Application State)

- Include spring-boot-starter-hateoas project.
- Spring Boot HATEOAS starter enables us to easily add links using the methods.
- Useful classes – WebMvcLinkBuilder

Notes

- It would be really good if you can actually define a standard exception structure and that is followed across all your restful services.
- You can extend ResponseEntiryExceptionHandler class to build your customized exception handling and annotate that class with @RestController along with details about to which resources it is applicable using @ControllerAdvice annotation.

Internationalization

- We need to configure LocaleResolver and ResourceBundleMessageSource.
- Use AcceptHeaderLocaleResolver if we are expecting that the locale will come in "Accept-Language" header.
- Also we need to autowire MessageSource.
- Also instead of creating bean for MessageSource, we can add this line in application.properties file so that spring boot will create a bean for us.
  spring.messages.basename=messages
  Assuming we have messages.properties files.
- And also need to accept header called "Accept-Language".
- LocaleContextHolder is very useful class provided by Spring Boot.

Content Negotiation

- Content negotiation is the mechanism that is used for serving different representations of a resource at the same URI, so that the user agent can specify which is best suited for the user.
- Content negotiation is returning response in the format as requested by the client.
- Include Jackson-dataformat-xml in your pom.
- Now based on "Accept" header in the request, your service can return JSON or XML response back.

Gotcha

- Disable XML Support if not required. This is because browsers give preference to XML over JSON (They send Accept header with application/xml).

Swagger

- Swagger is one of the popular documentation format for RESTful services.
- Add swagger dependencies in your pom.xml –
  springfox-swagger2, springfox-swagger-ui
- Next, create a class for Swagger configuration and annotate it with @Configuration.
- Enable swagger for that class using swagger's annotation @EnableSwagger2.
- Create a method to get Docket. (Annotate it with @Bean).
- Once you build this, it will expose couple of new endpoints like /v2/api-docs and also you can access /swagger-ui.html.
- There are lots of APIs and annotations provided by swagger dependency to customize our documentation.
- Some of the annotations are @ApiModel, @ApiModelProperty and many more.

Monitoring APIs with Spring Boot Actuator

- Include spring-boot-starter-actuator project in your pom.
- Also include spring-data-rest-hal-browser.
- HAL stands for Hypertext Application Language. It specifies a simple format to specify how to hyperlink between resources in your API.
- Spring boot actuator API is actually in the HAL format. So what the HAL browser does is it looks at those APIs, identifies the links and shows them on the screen so that you can easily browse through the API by just clicking the links.
- So the HAL browser makes it easy to consume the HAL services which are being exposed by the Spring boot actuator.

Static Filtering in REST

- Suppose you don't want to return certain attributes/properties of your bean in the response, this is called as filtering.
- Just use fasterxml Jackson annotation @JsonIgnore to exclude a field of a bean from being sent to the client. This is static filtering.
- Another approach of static filtering is using annotation @JsonIgnoreProperties on the bean class and pass it list of fields which you want to exclude.
- Recommended to use @JsonIgnore instead of @JsonIgnoreProperties so that you don't have to hard code property names in @JsonIgnoreProperties.

Dynamic Filtering in REST

- With dynamic filtering we cannot configure filtering directly on the bean. We need to start configuring the filtering where we are retrieving the values.
- We need to create object of MappingJacksonValue class and configure a filter.
- Classes involved – SimpleBeanPropertyFilter, FilterProvider, SimpleFilterProvider, MappingJacksonValue.
- Also list of valid filters should be defined on the bean using @JsonFilter annotation on the bean class.

Versioning RESTful services

- There are multiple option to version our APIs.
- Versioning using URIs
  e.g. /v1/persons, /v2/persons. E.g. Twitter
- Versioning using request parameters.
  e.g. /persons/version=1, /persons/version=2. E.g. Amazon
- Versioning using custom header parameters.
  Pass some custom header from client e.g. X-API-VERSION=1. E.g. Microsoft
- Versioning using "produces" / Versioning using Accept header / MIME Type versioning / Content Negotiation Versioning.
  E.g. "application/vnd.company.app-v1+json", "application/vnd.company.app-v2+json".
  So the client will send the "Accept" header with one of these 2 formats.
  E.g. Github
- There is no one correct way as there are tradeoffs between these available options.
- There are some factors which can help us to decide which way to use like URI pollution, easy of using, ease of caching, ability to generate API documentation from code.
- Before you build your first API, have your versioning strategy ready.

Securing your REST services from unauthorized access

- There are different ways to allow only authorized access to our REST services.
- Basic authentication using username and password.
- Digest authentication where the password Digest is created and the Digest is sent across so the actual password is not sent to the server.
- OAuth or OAuth2 authenication.

Basic authentication

- For basic authentication, add spring-boot-starter-security project. It will automatically configure basic security.
- When you start your server, you will see a console log with "default security password". This is the default password. And the default username is "user".
- By default, everytime you start the server, the default password will change. However we can configure both username and password in application.properties file via below properties.
  spring.security.user.name=<username>
  spring.security.user.password=<password>

Introduction to JPA – An Overview

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

The Problem (Object Relational Impedance Mismatch)

- How we design our Java objects is different from how we design our tables. And that's really the problem that all persistence frameworks have been trying to solve.
- Some of the frameworks are JDBC,Spring JDBC, myBatis.

World before JPA

- With **plain JDBC**, we have to write a lot of code just for simple database operations like select, update, etc.
- Then **Spring JDBC** came, which is kind of a wrapper over JDBC which abstracted lot of things and made easier to perform database operations compared to plain JDBC. E.g. Using JDBCTemplate, RowMapper, etc.
- myBatis maps objects to queries.
- The fundamental thing for all these three approaches was the fact that they are still based on queries. The **problem** with queries is when the relationships between tables change then you have to modify all your queries which is lot of overhead.

Introduction to JPA

- JPA solves the problem of object relational mapping.
- JPA provides a mapping. You can map application classes to tables.
- JPA is a interface or a specification. Hibernate implements JPA.
- JPA defines all the annotations and hibernate provides implementations to all these concepts.
- Hibernate is one of the most popular ORM frameworks. ORM is object relational mapping and JPA standardizes ORM.

Some Notes

- Spring-boot-starter-data-jpa starter project.
- @Repository indicates something which interacts with the database.
- @Transactional provided by JPA to do declarative transaction. Can be used on class as well as method.
- Entity manager is actually an interface to the persistence context.
- Only those objects which are in the persistence context are tracked by the entity manager.
- @PersistenceContext – Entity manager should be annotated with @PersistenceContext in order for it to track the objects.

- Spring Boot based on the jars which you have configures the Entity Manager, Data Source and Transaction Manager for us.

Introduction to Spring Data JPA

- With Spring Data JPA, we just need to write an interface which extends JpaRepository and provide the entity name and the primary key type of that entity which you want to manage (insert, delete, update, etc.), also annotate that interface with @Repository. Then spring will provide an implementation for that interface.
- The JPA repository kind of acts as a shortcut to manage your entities. Instead of writing a lot of DAO services, with very similar logic (same insert, persist, merge, find and all that kind of stuff), Spring data JPA provides a common abstraction called JPA repository.
- If you extend the JpaRepository and provide the entity you'd want to manage, you'd get all the different methods that are provided. A varied range of methods that are provided for doing all kind of operations with your entities.
- Once you have defined entities, Spring data JPA makes it really easy to manage those entities.

REST Best Practices

Richardson Maturity Model

- Richardson Maturity Model helps you evaluate “how RESTful are you?”.
- It has 4 levels. Google it. ☺

Best Practices

- Consumer First.
- Have great documentation.
- Make best use of HTTP methods.
- Make best use of correct Response HTTP status codes.
- Use plurals for REST endpoints.
- Think about nouns for resources.

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`
- JPA defines how ORM applications or ORM frameworks should work.
- In your IntelliJ Idea, install “Spring Assistant” plugin to get auto completion in the properties/yaml files. For eclipse it is something called “Spring Tools”.
- Java Validation API standard is “validation-api.xxx.jar” and most popular implementation of this Java Validation API is hibernate-validator.
- Spring boot looks at the jars which are available in the classpath and it decides the best configuration for you.