***1. Explain Servlet & JSP life cycle?***

Ans:- Servlet is java class that is executed on server side, when request is done by client and produces result, which is sent to the client as response

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.---classloader is responsible
2. Servlet instance is created.----web container
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

**JSP Life cycle : Server side technology**

step 1. **Translation Stage**: When first request is made for the JSP page, it will be loaded by the web container inside JSP engine. JSP engine translate the JSP page into a servlet's java code. This .java file can be found in tomcat's "work" folder. If there is any error in JSP syntax (e.g. scriptlets), then this stage fails.

step 2. **Compilation Stage**: The translated servlet's .java file will be compiled into a .class file at runtime. If there is any java syntax error, then this stage fails.

step 3. **Instantiation (Loading) Stage**: The .class file will be loaded and object of the translated servlet will be created. Immediately after this init method of the JSP i.e. jspInit() will be executed. If this method throws any exception, this stage fails. This stage is also called as Loading or Initialization stage.

step 4. **Request Handling Stage**: All above stages are done only for the first request of the JSP file; However this stage is executed for each request. For each request, \_jspService() method is executed (which is made up of all the scriptlet and expressions in the JSP file).

step 5. **Destruction Stage**: When servlet object is no longer used or web container is going down, jspDestroy() method will be executed after which servlet object will be garbage collected.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

10.9M

257

How to find Nth Highest Salary in SQL

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

***2. What is difference between Redirect and RequestDispatcher scenario?.***

**Redirection :-**

- Can redirect from any page to any other page of the same application or different application.

- When sendRedirect() is called, a temp response (status code 302 and destination url) is sent to the browser; due to which browser make a new request to the new link. In this case two requests are originated from the browser and hence browser is aware of the navigation. This is slower process.

***Request Dispatcher:-***

- Only one request is originated from the client, and single response is given back.

- This is faster than HTTP redirection. Browser is not aware of the navigation. Navigation can be done only to the pages within the same application

**Forwarding**

Request is forwarded to next servlet from which response will be given to the client.

**Including:**

Request is given to next servlet, which performs some processing and return back to the calling servlet. The response generated by the second servlet will be included into first servlet's response.

***3. What is state management tracking?***

State Management State Management Classfication

**Client Side State Management**

The state/info of client is stored on client machine less memory is needed at the server side Less secure, client can access and/or modify Options: Cookie, QueryString, Hidden Form Fields

**Server Side State Management**

The state/info of client is stored on server machine Large memory is needed at the server side More secure, client cannot directly access Options: Session, Application

***3. What is session tracking? How to do it in java?***

Session Tracking

Each session is identified using a unique session id, which is associated with the client. There are two ways of this association (tracking): cookie based and url rewriting

**Cookie based**

By default, when new session is created (req.getSession() is called first time) a cookie is created and sessionid is sent to client via that cookie. For suqsequent calls to req.getSession() access the appropriate session object by getting sessionid from that cookie.

**Url rewriting**

In case cookies disabled, sessionid can be maintained using url. resp.encodeURL() and resp.encodeRedirectURL() methods are used to embed sessionid into the url. e.g. http://server:port/app/page;jsessionid=374334

***4. What is hibernate? Explain hibernate architecture.***

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

The Hibernate architecture is categorized in four layers.

* Java application layer
* Hibernate framework layer
* Backhand api layer
* Database layer

Hibernate Life Cycle

SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.

It is immutable and thread-safe in nature.

buildSessionFactory() method gathers the meta-data which is in the cfg Object.

From cfg object it takes the JDBC information and create a JDBC Connection.

SessionFactory factory=cfg.buildSessionFactory();

Session:

Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.

It opens the Connection/Session with Database software through Hibernate Framework.

It is a light-weight object and it is not thread-safe.

Session object is used to perform CRUD operations.

Session session=factory.buildSession();

Transaction:

Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.

Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.

Transaction tx=session.beginTransaction();

tx.commit();

Query:

Query is an interface that present inside org.hibernate package.

A Query instance is obtained by calling Session.createQuery().

This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():

A particular page of the result set may be selected by calling setMaxResults(), setFirstResult().

Named query parameters may be used.

Query query=session.createQuery();

Criteria:

Criteria is a simplified API for retrieving entities by composing Criterion objects.

The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions.

Criteria criteria=session.createCriteria();

Dialect

• RDBMS have specific features like data types, stored procedures, primary key generation, etc.

• Hibernate support all RDBMS. • Most of code base of Hibernate is common.

• Database level changes are to be handled specifically and appropriate queries should be generated. This is handled by Dialect.

• Hibernate have dialects for all RDBMS. Programmer should configure appropriate dialect to utilize full features of RDBMS.

Flow of working during operation in Hibernate Framework :

Suppose We want to insert an Object to the database. Here Object is nothing but persistence logic which we write on java program and create an object of that program. If we want to insert that object in the database or we want to retrieve the object from the database. Now the question is that how hibernate save the Object to the database or retrieve the object from the database. There are several layers through which Hibernate framework go to achieve the above task. Let us understathe nd the layers/flow of Hibernate framework during performing operations:

Stage I: In first stage, we will write the persistence logic to perform some specific operations to the database with the help of Hibernate Configuration file and Hibernate mapping file. And after that we create an object of the particular class on which we wrote the persistence logic.

Stage II:In second stage, our class which contains the persistence logic will interact with the hibernate framework where hibernate framework gives some abstraction do perform some task. Now here the picture of java class is over. Now Hibernate is responsible to perform the persistence logic with the help of layers which is below of Hibernate framework or we can say that the layers which are the internal implementation of Hibernate.

Stage III:In third stage, our hibernate framework interact which JDBC, JNDI, JTA etc to go to the database to perform that persistence logic.

Stage IV & V:In fourth & fifth stage, hibernate is interact with Database with the help of JDBC driver. Now here hibernate perform that persistence logic which is nothing but CRUD operation. If our persistence logic is to retrieve an record then in the reverse order it will display on the console of our java program in terms of Object.

***Hibernate is used to overcome the of limitations of JDBC like***:

JDBC code is dependent upon the Database software being used i.e. our persistence logic is dependent, because of using JDBC.

1. Here we are inserting a record into Employee table but our query is Database software-dependent i.e. Here we are using MySQL.

But if we change our Database then this query won’t work.

2.If working with JDBC, changing of Database in middle of the project is very costly.

3.JDBC code is not portable code across the multiple database software.

To overcome the above problems we use ORM tool i.e. nothing but Hibernate framework.

By using Hibernate we can avoid all the above problems and we can enjoy some additional set of functionalities.

***Functionalities supported by Hibernate framework***

Hibernate framework support Auto DDL operations.

1.In JDBC manually we have to create table and declare the data-type for each and every column.

But Hibernate can do DDL operations for you internally like creation of table,drop a table,alter a table etc.

2.Hibernate supports Auto Primary key generation. It means in JDBC we have to manually set a primary key for a table.

But Hibernate can this task for you.

3.Hibernate framework is independent of Database because it supports HQL (Hibernate Query Language) which is not specific to any database, whereas JDBC is database dependent.

In Hibernate, Exception Handling is not mandatory, whereas In JDBC exception handling is mandatory.

***Java Beans***

Java beans are simple java classes which contain constructor, fields, getters/setters and one/more business logic methods

***MVC design pattern*** makes Java EE web applications extendable & maintainable. • Typical MVC implementation

• Model: Java bean/POJO

• View: JSP page

• Controller: Servlet (forwarding request)

• Popular MVC frameworks: Struts, JSF, Spring.

1. ***Explain hibernate entity life cycle?***

• **Transient** : New Java object of entity class. • This object is not yet associated with hibernate.

• **Persistent** • Object in session cache. • For all objects created by hibernate or associated with hibernate. • State is tracked by hibernate and updated in database during commit. • Never garbage collected.

• **Detached** • Object removed from session cache.

• **Removed** • Object whose corresponding row is deleted from database.

***6. What is the difference between get() and load() of hibernate?***

load()

Only use load() method if you are sure that the object exists.

load() method will throw an exception if the unique id is not found in the database.

load() just returns a proxy by default and database won't be hit until the proxy is first invoked.

get()

If you are not sure that the object exist, then use one of get() methods.

get() method will return null if the unique id is not found in the database.

get() will hit the database immediately.

save()

It return generated id and return type is serializable

It can save object within boundaries and outside boundaries

It will create a new row in the table for detached object

It is only supported by Hibernate

persist()

It does not return anything. Its void return type.

It can only save object within the transaction boundaries

It will throw persistence exception for detached object

It is also supported by JPA

diff between save save or update()

1) The main difference between the save() and saveOrUpdate() method is that the save() method performs an INSERT operation to store the object into the database,

but INSERT will fail if the primary key is already persistent i.e. object already exists in the database.

This is why you should only call save() with an absolutely new object which doesn't have any database identifier. Calling save() with the detached object will fail.

This is the opposite of the saveOrUpdate() method, which can do either INSERT or UPDATE SQL query depending upon whether an object exists in the database or not.

The saveOrUpdate() method first executes a SELECT query to determine if it needs to do an INSERT or UPDATE operation.

2) Another key difference between the save() and saveOrUpdate() method is that the former is used to bring a transient object to a persistent state but saveOurUpdate() can bring both transient (new) and detached (existing) object into the persistent state.

It is often used to re-attach a detached object into a Session.

7. ***How can we call stored procedure in hibernate?***

In Hibernate, there are three approaches to call a database store procedure.

. Native SQL – createSQLQuery

You can use createSQLQuery() to call a store procedure directly.

Query query = session.createSQLQuery(

"CALL GetStocks(:stockCode)")

.addEntity(Stock.class)

.setParameter("stockCode", "7277");

List result = query.list();

2. NamedNativeQuery in annotation

Declare your store procedure inside the @NamedNativeQueries annotation.

@NamedNativeQueries({

@NamedNativeQuery(

name = "callStockStoreProcedure",

query = "CALL GetStocks(:stockCode)",

resultClass = Stock.class

)

})

@Entity

@Table(name = "stock")

public class Stock implements java.io.Serializable {

Call it with getNamedQuery().

Query query = session.getNamedQuery("callStockStoreProcedure")

.setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

Stock stock = (Stock)result.get(i);

System.out.println(stock.getStockCode());

}

3. sql-query in XML mapping file

Declare your store procedure inside the “sql-query” tag.

Call it with **getNamedQuery()**.

8. ***What is IOC and Dependancy injection***?

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.

Dependency Injection makes our programming code loosely coupled.

in sw we build objects .those objects may be depend on other object

di allows us to use external objects,variable in our class without creating object

it create the object dynamically at runtime and inject it in our class.

Inversion of Control(IoC) is a principle by which the control of objects is transferred to a container or framework. Dependency injection is a pattern through which IoC is implemented and the act of connecting objects with other objects or injecting objects into objects is done by container rather than by the object themselves

**Need for Dependency Injection:**

Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be dependent on class Two. Now though it might appear okay to depend a module on the other but, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided.

Spring IOC resolves such dependencies with Dependency Injection, which makes the code easier to test and reuse.

**Two ways to perform Dependency Injection in Spring framework**

Spring framework provides two ways to inject dependency

By Constructor

By Setter method

**Advantages:**

* Helps to enable loose coupling architecture, there is no need to modify the code if our logic is moved to a new environment.
* Makes Unit testing easy.
* Boilerplate code is reduced, as initializing of dependencies is done by the injector component.
* Extending the application becomes easier.
* Allows code to be more reusable, testable, and readable.
* Reduce unnecessary dependencies.
* Allows concurrent or independent development.
* Reduced module complexity
* Increased system maintainability, because logic changes in the domain affect fewer modules.
* Increased module reusability.

***9. What is auto­wiring? Which are the types of auto wiring? What if prototype bean is auto­wired in a*** singleton ***bean?***

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Wiring allows the Spring container to automatically resolve dependencies between collaborating beans by inspecting the beans that have been defined. we can inject the dependencies by making them with **@Autowired** annotation which is called field-based dependency injection.

--**Problem with injecting prototype scoped bean into singleton scoped bean**

When you inject prototype bean to singleton bean, prototype bean still behave like a singleton bean

**--Solutions for injecting prototype scoped bean into singleton scoped bean**

. Implementing the ApplicationContextAware interface

Lookup method injection in Spring

aop:scoped-proxy

Using ObjectFactory interface

***10. Explain spring bean life cycle. Explain spring bean scopes.***

the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom **init()** method and the **destroy()** method.

**Ways to implement the life cycle of a bean**

1. **By XML:** In this approach, in order to avail custom init() and destroy() method for a bean we have to register these two methods inside Spring XML configuration file while defining a bean.
2. **By Programmatic Approach:** To provide the facility to the created bean to invoke custom **init()** method on the startup of a spring container and to invoke the custom **destroy()** method on closing the container, we need to implement our bean with two interfaces namely **InitializingBean**, **DisposableBean** and will have to override **afterPropertiesSet()** and **destroy()** method. **afterPropertiesSet()** method is invoked as the container starts and the bean is instantiated whereas, the **destroy()** method is invoked just after the container is closed. **Note:** To invoke destroy method we have to call a **close()** method of ConfigurableApplicationContext.
3. **Using Annotation:** To provide the facility to the created bean to invoke custom **init()** method on the startup of a spring container and to invoke the custom **destroy()** method on closing the container, we need annotate **init()** method by **@PostConstruct** annotation and **destroy()** method by **@PreDestroy** annotation.  
   **Note:** To invoke the **destroy()** method we have to call the **close()** method of ConfigurableApplicationContext.

Bean scope can be set in XML or annotation.

• • @Scope("singleton|prototype|request|session") •

**singleton**

• Single bean object is created and accessed throughout the application. • XMLBeanFactory creates object when getBean() is called for first time for that bean. • ApplicationContext creates object when ApplicationContext is created. • For each sub-sequent call to getBean() returns same object reference. • Reference of all singleton beans is managed by spring container. • During shutdown, all singleton beans are destroyed (@PreDestroy will be called).

**prototype** • No bean is created during startup. • Reference of bean is not maintained by ApplicationContext. • Beans are not destroyed automatically during shutdown. • Bean object is created each time getBean() is called. • request and session: scope limited to current request and session

***11. What is use of @Transactional? Why it should be used on service layer?***

It's important to notice that JPA on itself does not provide any type of declarative transaction management. When using JPA outside of a dependency injection container, transactions need to be handled programatically by the developer which is more prone to the errors and may become difficult to handle and debug the errors.

@Transactional annotation makes our task easy.

By using @Transactional, many important aspects such as transaction propagation are handled automatically.

**@Transactional on service method**

Service layer may call different DAOs to perform DB operations. Lets assume a situation where you have 3 DAO operations in a service method.

If your 1st DAO operation failed, other two may be still passed and you will end up with an inconsistent DB state. Annotating Service layer can save you from such situations.

You are going to want your services to be transactional.

If your DAOs are transactional, and you call different DAOs in each service, then you would have multiple transactions, which is not what you want.

Make the service calls transactional, and all DAO calls inside those methods will participate in the transactions for the method.

Transaction management is done by platform transaction manager e.g. datasource, hibernate or jpa transaction manager

***12. Explain Spring MVC life cycle?***

• DispatcherServlet(front controller) receives the request. • DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet. • DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter. • HandlerAdapter calls the business logic process of Controller. • Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter. • DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name. • DispatcherServlet dispatches the rendering process to returned View. • View renders Model data and returns the response

***13. What is the difference between SOAP and REST?***

* SOAP stands for Simple Object Access Protocol whereas REST stands for Representational State Transfer.
* SOAP is a protocol whereas REST is an architectural pattern.
* SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer. while REST uses Uniform Service locators to access to the components on the hardware device.
* SOAP needs more bandwidth for its usage whereas REST doesn’t need much bandwidth.
* Comparing SOAP vs REST API, SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON.
* SOAP cannot make use of REST whereas REST can make use of SOAP.

***What is significance of RestController?***

The @RestController annotation in Spring MVC is nothing but a combination of the @Controller and the @ResponseBody annotation. It was added into Spring 4.0 to make the development of RESTful Web Services in Spring framework easier. If you are familiar with the [REST web services](http://www.java67.com/2017/04/3-great-books-to-learn-java-web-services-soap-and-restful.html) you know that the fundamental difference between a web application and a REST API is that the response from a web application is a generally view of HTML + CSS + JavaScript while REST API just return data in form of JSON or XML. This difference is also obvious in the @Controller and the @RestController annotation. The job of the @Controller is to create a Map of model object and find a view but the @RestController simply returns the object and object data is directly written into HTTP response as JSON or XML.

ere are some important differences between these two annotations.

1. The @Controller is a common annotation which is used to mark a class as Spring MVC Controller while the @RestController is a special controller used in [RESTFul web services](https://javarevisited.blogspot.sg/2015/08/difference-between-soap-and-restfull-webservice-java.html" \t "_blank) and the equivalent of @Controller + @ResponseBody.
2. The @RestController is relatively new, added only on Spring 4.0 but @Controller is an old annotation, exists since Spring started supporting annotation, and officially it was added on Spring 2.5 version.
3. The @Controller annotation indicates that the class is a “Controller” e.g. a web controller while the @RestController annotation indicates that the class is a controller where @RequestMapping methods assume @ResponseBody semantics by default i.e. servicing REST API.
4. The @Controller is a specialization of @Component annotation while @RestController is a specialization of @Controller annotation. It is actually a convenience controller annotated with @Controller and @ResponseBody as shown below.
5. One of the key difference between @Controler and @RestCotroller in Spring MVC is that once you mark a class as @RestController then every method is written a domain object instead of a view. You can see Bryan Hassen’s [Introduction to Spring MVC 4](https://www.shareasale.com/m-pr.cfm?merchantID=53701&userID=880419&productID=557072989) to learn more about how to use the @RestController annotation in your Spring based application.
6. Another key difference between @RestController and @Controller is that you don’t need to use @ResponseBody on every handler method once you annotate the class with @RestController

***14. What is Spring Boot? What do you mean by opinionated defaults? How auto­configuration works?***

pring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications. This chapter will give you an introduction to Spring Boot and familiarizes you with its basic concepts.

## What is Micro Service?

Micro Service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications.

### Advantages

Micro services offers the following advantages to its developers −

* Easy deployment
* Simple scalability
* Compatible with Containers
* Minimum configuration
* Lesser production time

## What is Spring Boot?

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**. You can get started with minimum configurations without the need for an entire Spring configuration setup.

### Advantages

Spring Boot offers the following advantages to its developers −

* Easy to understand and develop spring applications
* Increases productivity
* Reduces the development time

### Goals

Spring Boot is designed with the following goals −

* To avoid complex XML configuration in Spring
* To develop a production ready Spring applications in an easier way
* To reduce the development time and run the application independently
* Offer an easier way of getting started with the application

## Why Spring Boot?

You can choose Spring Boot because of the features and benefits it offers as given here −

* It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
* It provides a powerful batch processing and manages REST endpoints.
* In Spring Boot, everything is auto configured; no manual configurations are needed.
* It offers annotation-based spring application
* Eases dependency management
* It includes Embedded Servlet Container

## How does it work?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

## Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include **spring-boot-starter-data-jpa** dependency in your project.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** \*, where \* indicates that it is a type of the application.

## Auto Configuration

Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add **@EnableAutoConfiguration** annotation or **@SpringBootApplication** annotation to your main class file. Then, your Spring Boot application will be automatically configured.

If you added **@SpringBootApplication** annotation to the class, you do not need to add the **@EnableAutoConfiguration, @ComponentScan** and **@SpringBootConfiguration** annotation. The **@SpringBootApplication** annotation includes all other annotations.

### What is Spring-Boot’s Opinionated Strategy?

Spring-Boot’s [Opinionated Defaults Configuration](https://stackoverflow.com/questions/47018071/what-is-meant-by-spring-boot-follows-opinionated-defaults-configuration-approa/47018551) is more of a strategy to eliminate boilerplate and configurations meant to improve unit testing, development, and integration test procedures. It decides the defaults to use for configuration and the packages to install based on the dependencies requirement. You only need to include the [Spring Boot starter](https://www.baeldung.com/spring-boot-starters) ‘pom’ to enable autoconfigure capability to get started quickly on their projects. For example, including the Spring Boot starter ‘pom’ for ‘[JPA](https://www.javatpoint.com/spring-boot-jpa)’ will autoconfigure an in-memory database, a data source, and a hibernate entity manager. You can still override this opinionated default configuration depending on the needs of your project.

***15. What is significance of @CrossOrigin? How it works***

Cross-Origin Resource Sharing (CORS) is a security concept that allows restricting the resources implemented in web browsers. It prevents the JavaScript code producing or consuming the requests against different origin.

For example, your web application is running on 8080 port and by using JavaScript you are trying to consuming RESTful web services from 9090 port. Under such situations, you will face the Cross-Origin Resource Sharing security issue on your web browsers.

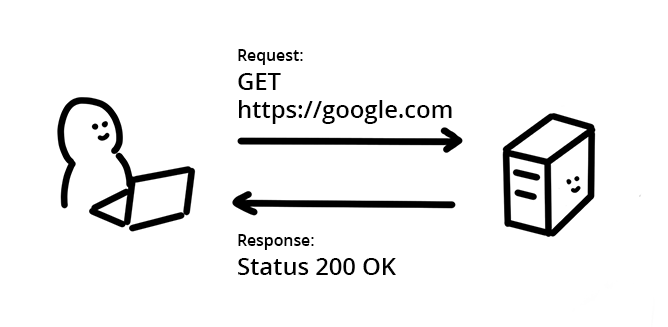
Two requirements are needed to handle this issue −

* RESTful web services should support the Cross-Origin Resource Sharing.
* RESTful web service application should allow accessing the API(s) from the 8080 port.
* In other words, there are public resources that should be available for anyone to read, but the same-origin policy blocks that. Developers have used work-arounds such as [JSONP](https://stackoverflow.com/questions/2067472/what-is-jsonp-all-about), but Cross-Origin Resource Sharing (CORS) fixes this in a standard way.
* Enabling CORS lets the server tell the browser it's permitted to use an additional origin.

## Enable CORS in Controller Method

We need to set the origins for RESTful web service by using **@CrossOrigin** annotation for the controller method. This @CrossOrigin annotation supports specific REST API, and not for the entire application.

## How does a resource request work on the web? [#](https://web.dev/cross-origin-resource-sharing/#how-does-a-resource-request-work-on-the-web)

Figure: Illustrated client request and server response

A browser and a server can exchange data over the network using the **Hypertext Transfer Protocol** (HTTP). HTTP defines the communication rules between the requester and the responder, including what information is needed to get a resource.

The HTTP header is used to negotiate the type of message exchange between the client and the server and is used to determine access. Both the browser's request and the server's response message are divided into two parts: **header** and **body**:

## *How does CORS work?*[*#*](https://web.dev/cross-origin-resource-sharing/#how-does-cors-work)

Remember, the same-origin policy tells the browser to block cross-origin requests. When you want to get a public resource from a different origin, the resource-providing server needs to tell the browser "This origin where the request is coming from can access my resource". The browser remembers that and allows cross-origin resource sharing.

### Step 1: client (browser) request [#](https://web.dev/cross-origin-resource-sharing/#step-1:-client-(browser)-request)

When the browser is making a cross-origin request, the browser adds an Origin header with the current origin (scheme, host, and port).

### Step 2: server response [#](https://web.dev/cross-origin-resource-sharing/#step-2:-server-response)

On the server side, when a server sees this header, and wants to allow access, it needs to add an Access-Control-Allow-Origin header to the response specifying the requesting origin (or \* to allow any origin.)

### Step 3: browser receives response [#](https://web.dev/cross-origin-resource-sharing/#step-3:-browser-receives-response)

When the browser sees this response with an appropriate Access-Control-Allow-Origin header, the browser allows the response data to be shared with the client site.

***Que…Annotation vs XML config***

XML based config

* Need not to recompile the whole project.
* If config has mistake, then deployment will fail. (No compilation in adavance).

Annotation based config

* Java code is compiled for syntax errors in advanced, so less chances of failure while deployment.
* Need to recompile the project. Nowadays preferred in industry.