**Spring boot**

## Why Do I Need Starters?

Spring Boot Starters make the bootstrapping process much easier and faster. The starter brings you required Maven dependencies as well as some predefined configuration bits.

Spring Boot Starter Parent helps with managing dependency versions, the java version used by project and the default configuration for plug-ins.

spring-boot-starter-parent contains configuration meta data, this means, it knows which version of dependency need to be downloaded

@springbootApplication = @Configuration + @EnableAutoConfiguration + @component.

* Create application context.
* Check application type.
* Register the annotated class beans with context.
* Create instance of tomcatemmbededServletcontaine and adds the context.

 Web Starter

First, let’s look at developing the REST service; we can use libraries like Spring MVC, Tomcat and Jackson – a lot of dependencies for a single application.

Spring Boot starters can help to reduce the number of manually added dependencies just by adding one dependency. So instead of manually specifying the dependencies just add one starter as in the following example:

|  |  |
| --- | --- |
|  | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-web</artifactId>  </dependency> |

Now we can create a REST controller. For the sake of simplicity we won’t use the database and focus on the REST controller:

**The Test Starter**

For testing we usually use the following set of libraries: Spring Test, JUnit, Hamcrest, and Mockito. We can include all of these libraries manually, but Spring Boot starter can be used to automatically include these libraries in the following way:

|  |  |
| --- | --- |
|  | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-test</artifactId>      <scope>test</scope>  </dependency> |

Notice that you don’t need to specify the version number of an artifact. Spring Boot will figure out what version to use – all you need to specify is the version of spring-boot-starter-parent artifact. If later on you need to upgrade the Boot library and dependencies, just upgrade the Boot version in one place and it will take care of the rest.

**spring-boot-starter-amqp** It is used for Spring AMQP and Rabbit MQ.

**spring-boot-starter-data-mongodb** It is used for MongoDB document-oriented database and Spring Data MongoDB.

**spring-boot-starter-web-services** It is used for Spring Web Services.

Java 7 feature:

**Strings in Switch:**

In all examples, I tried first to write code as we usually do, and then convert the code to a Java 7 feature. The cool thing about [NetBeans](http://netbeans.org/) is it already have tips to convert the code.

First, will compare some Strings using if-else. NetBeans will popup a warning like this:

And if we accept the change, it will convert the code into this:

|  |
| --- |
| publicvoidtestStringInSwitch(String param){  finalString JAVA5 = "Java 5";  finalString JAVA6 = "Java 6";  finalString JAVA7 = "Java 7";  switch (param) {  case JAVA5:  System.out.println(JAVA5);  break;  case JAVA6:  System.out.println(JAVA6);  break;  case JAVA7:  System.out.println(JAVA7);  break;  }  } |

**Multi-Catch Similar Exceptions:**

|  |
| --- |
| Before 7:  PublicvoidtestMultiCatch(){  try {  thrownewFileNotFoundException("FileNotFoundException");  } catch (FileNotFoundExceptionfnfo) {  fnfo.printStackTrace();  } catch (IOExceptionioe) {  ioe.printStackTrace();  }  After 7  publicvoidtestMultiCatch(){  try {  thrownewFileNotFoundException("FileNotFoundException");  } catch (FileNotFoundException|IOExceptionfnfo) {  fnfo.printStackTrace();  }  } |

**The try-with-resources statement**

Java 7 – allows us to declare resources to be used in a *try*block with the assurance that the resources will be closed when after execution of that block. The resources declared must implement the *AutoCloseable* interface.

|  |
| --- |
| Before  publicvoidtestTryWithResourcesStatement() throwsFileNotFoundException, IOException{  FileInputStreamin=null;  try {  in=newFileInputStream("java7.txt");  System.out.println(in.read());  } finally {  if (in!=null) {  in.close();  }  }  }  Java 7  publicvoidtestTryWithResourcesStatement() throwsFileNotFoundException, IOException{  try (FileInputStreamin=newFileInputStream("java7.txt")) {  System.out.println(in.read());  }  }  Custom and Multiple resource  **publicclass** Demo {  **publicstaticvoid** main(String[] args) {  //note the order of opening the resources  **try**(Demo1 d1 = **new** Demo1(); Demo2 d2 = **new** Demo2())  {  **int**x = 10/0;  d1.show();  d2.show1();  }  **catch**(ArithmeticException e)  {  System.***out***.println(e);  }  }  }  //custom resource 1  **class** Demo1 **implements** AutoCloseable  {  **void** show() {  System.***out***.println("inside show");  }  @Override  **publicvoid** close() {  System.***out***.println("close from demo");  }  }  //custom resource 2  **class** Demo2 **implements** AutoCloseable {  **void** show1() {  System.***out***.println("inside show1");  }  @Override  **publicvoid** close() {  System.***out***.println("close from demo1");  }  } |

**Hash code and equals methods:**

If two objects are equal according to the *equals(Object)* method, then calling the **hashcode()** method on each of the two objects must produce the same integer result.

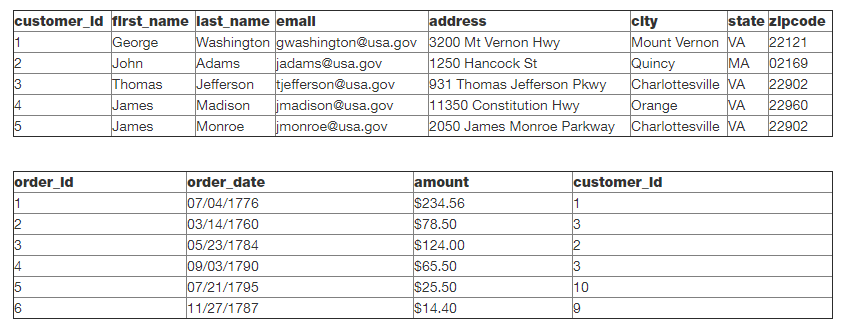
**Spring Boot with WebLogic server**

1. First of all, we will need to change the default packaging in pom.xml file from jar to war like below:
2. 2. Still in pom.xml, add the plugin maven-war-plugin failOnMissingWebXml = false, because we don’t have web.xml in the project
3. Create a weblogic.xml inside src/main/webapp/WEB-INF with context-root:
4. Create a empty file dispatcherServlet-servlet.xml in WEB-INF folder like this:
5. Run mvn clean and mvn install
6. If everything is fine, in the target folder there is a war. file generated.
7. Go to weblogic server console and do a new deployment

SQL Querys

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN



**INNER JOIN:**

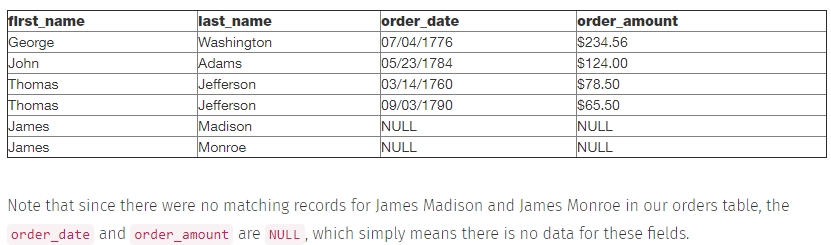
The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

|  |
| --- |
| **select first\_name, last\_name, order\_date, order\_amount from customers c inner join orders o on c.customer\_id = o.customer\_id** |

**LEFT JOIN:**

This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

|  |
| --- |
| **select first\_name, last\_name, order\_date, order\_amount**  **from customers c**  **left join orders o**  **on c.customer\_id = o.customer\_id** |



**RIGHT JOIN:**

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

|  |
| --- |
| **select first\_name, last\_name, order\_date, order\_amount**  **from customers c**  **right join orders o**  **on c.customer\_id = o.customer\_id** |

**FULL JOIN:**

 FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

|  |
| --- |
| **select first\_name, last\_name, order\_date, order\_amount**  **from customers c**  **full join orders o**  **on c.customer\_id = o.customer\_id** |

Self Join

The SQL SELF JOIN is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement

|  |
| --- |
| **SELECT E.name, ME.name AS manager**  **FROM dbo.Employees E**  **LEFT JOIN dbo.Employees ME**  **ON ME.employeeid = E.managerid** |

**ASN1**

Abstract Syntax Notation number One is a standard that defines a formalism for the specification of abstract data types.

Communication protocols describe the sequence, the content and the encoding of messages exchanged between computers communicating with each other. ASN.1 is a language for describing the content and the encoding of such messages. For example:

Contact ::= SEQUENCE {  
name VisibleString,  
phone NumericString  
}

. ASN.1 is independent of the programming languages used to implement communications. Whether you prefer to work in C, C++, Java, C#, or even COBOL, you can still transfer information. One end of transfer could be written in Java while the other end is written in C, and they will still be able to exchange information using ASN.1 with its encoding rules.

The notation provides a certain number of pre-defined basic types such as:

integers (INTEGER),

booleans (BOOLEAN),

character strings (IA5String, UniversalString...),

bit strings (BIT STRING),

etc.,

Enum:

The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

|  |
| --- |
| 1. **class** EnumExample1{ 2. //defining enum within class 3. **public** **enum** Season { WINTER, SPRING, SUMMER, FALL } 4. //creating the main method 5. **public** **static** **void** main(String[] args) { 6. //printing all enum 7. **for** (Season s : Season.values()){ 8. System.out.println(s); 9. } 10. System.out.println("Value of WINTER is: "+Season.valueOf("WINTER")); 11. System.out.println("Index of WINTER is: "+Season.valueOf("WINTER").ordinal()); 12. System.out.println("Index of SUMMER is: "+Season.valueOf("SUMMER").ordinal()); 14. }}   output:  WINTER  SPRING  SUMMER  FALL  Value of WINTER is: WINTER  Index of WINTER is: 0  Index of SUMMER is: 2 |

Question: What is out of the program

|  |
| --- |
| **class** A {  **publicvoid** printData() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **publicvoid** printData() {  System.***out***.println("B");  }  }  **class** C **extends**B{  **public** C() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **new** C();  }}  OutPut:  C |

Case 2

|  |
| --- |
| **class** A {  **public** A() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **public** B() {  System.***out***.println("B");  }  }  **class** C **extends** B {  **public** C() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **new** C();  }}  Output:  A  B  C |

Case 3

|  |
| --- |
| **class** A {  **public** A() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **public** B() {  System.***out***.println("B");  }  }  **class** C **extends** B {  **publicvoid** print() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **null**;  obj.print();  }}  Output: compilation is fine but we will get null pointer exception |

Case 4:

|  |
| --- |
| **class** SampleDemo {  SampleDemo obj = **new** SampleDemo();  }  **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  SampleDemo obj = **new** SampleDemo();  }  }  Output:  Exception in thread "main" java.lang.StackOverflowError  at com.mng.java.SampleDemo.<init>(javainterviewcase4.java:5)  at com.mng.java.SampleDemo.<init>(javainterviewcase4.java:5) |

Case 5:

|  |
| --- |
| **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  Stringstr = **new**String(**null**);  }}  Note: in this case compilation ambiguity error occurred because internally String class contains two constructors  Public String(String str)  {--  }  Public String(StringBuffer sb){  ---  }  Like  **class** SampleDemo {  **public** SampleDemo() {}  **public** SampleDemo(String str) {  System.***out***.println("String Constructor");  }  **public** SampleDemo(StringBuffer str) {  System.***out***.println("StringBuffer Constructor");  }  }  **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  SampleDemo obj = **new** SampleDemo(**null**);  }  }  Above also ambiguity error |

Case 5:

|  |
| --- |
| **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  Integer it = **new** Integer(**null**);  }}  In internally Interger class chcking null  Like  **if** (s == **null**) {  **thrownew** NumberFormatException("null");  }  Output:  Exception in thread "main" java.lang.NumberFormatException: null  at java.lang.Integer.parseInt(Integer.java:542)  at java.lang.Integer.<init>(Integer.java:867) |

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 this chapter, we are going to learn in detail about How to Enable Cross-Origin Requests for a RESTful Web Service application.

homeInit() method will be accessible only from domain http://example.com. Rest other methods in HomeController will be accessible from all domains.

@Controller

@CrossOrigin(origins = "\*", allowedHeaders = "\*")

@CrossOrigin(origins = "[http://example.com](http://example.com/)")

public class HomeController

{

    @CrossOrigin(origins = "[http://example.com](http://example.com/)")

    @GetMapping(path="/")

    public String homeInit(Model model) {

        return "home";

    }

}