Spring Boot

Remove all XML based configuration

support micro-services

Integrated server - Tomcat,Jetty

Embedded DB : H2, Apache Derby

Imp : -

Repository concept : - Spring data provide repository, that are implemented at run-time by the spring container.

Spring boot = (Spring framework - XML Configuration) + Integrated Server

Types of interface are used to connect DB

1. Repository -- Marker interface -- Here we have to define our methods to work with.

2. CrudRepository

3. JpaRepository

4. PagingAndSortingRepository

CrudRepository Vs JpaRepository

Hibernate Annotation

@DynamicUpdate

Ways to create Spring boot application

Spring boot CLI

Spring Initializer

STS

Major components of Spring Boot

Spring boot starter

Spring boot AutoConfigration

Spring boot Actuator

Spring boot Starter

it is a major feature of spring boot.

The main responsibility is to combine group of common dependencies into single dependency.

Example : When we add "spring-boot-starter-security"

jar file dependency to our build file, then Spring boot framework

will automatically download all required jars and add to our project class path.

Spring boot AutoConfigration

To develop a spring based application required lots of configration (Either XML or Annotation Configuration)

To solve this problem Spring boot AutoConfigration came in picture.

@AutoConfigration / = @Configuration + @ComponentScan + @EnableAutoConfigration

@SpringBootApplication

@Configuration : used to indicate that class declare one or more @Beans methods. These classes are

processed by the spring container to generate bean definitions and service request.

@ComponentScan :

@EnableAutoConfigration :

To pass value in spring boot

--------------------------------------------

Main class must implements CommandLineRunner interface.

and override

public void run(String... args) throws Exception {}

Annotation Used

---------------------------

@SpringBootApplication

@RestController

@GetMapping(value = '/home')

@PostMapping(value = "/create")

@RequestBody

Jpa

@NamedQuery

@NativeNamedQuery

@Query

@Async: when u want to perform the take by separate thread. Method marked with @async

Used for Unique result.

@Modifying: Used to modify the DB based on selected query. Using this Annotation . EntityManager will flush the information.

To Achieve this . Method must be marked Transaction

@Transaction

@Modifying(clearAutomaticall = true)

@Query("update User u set u.firstname = ?1 where u.lastname = ?2")

int setFixedFirstnameFor(String firstname, String lastname);

Logging Level

logging.level.root = debug/info/warn/error

private static final Logger LOGGER = LoggerFactory.getLogger(Classname.class);

Security

1 Add maven dependencies -- spring-boot-starter-security

by default

username : user

password : provide by application. every time when the server start.

Customize username and password

application.proerties adding few properties

spring.security.user.name = admin

spring.secuity.user.password = admin

BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();

@GetMapping("/topics")

public ResponseEntity<List<Topic>> getAllTopics() {

List<Topic> list = topicService.getAllTopics();

return new ResponseEntity<List<Topic>>(list, HttpStatus.OK);

}

@PostMapping("/topic")

public ResponseEntity<String> addTopic(@RequestBody Topic topic, UriComponentsBuilder builder) {

boolean flag = topicService.addTopic(topic);

if (flag == false) {

return new ResponseEntity<String>("This Topic already exist", HttpStatus.CONFLICT);

}

HttpHeaders headers = new HttpHeaders();

headers.setLocation(builder.path("/topic/{id}").buildAndExpand(topic.getTopicId()).toUri());

return new ResponseEntity<String>(headers, HttpStatus.CREATED);

}

public interface TopicService {

@Secured ({"ROLE\_ADMIN", "ROLE\_USER"})

public abstract List<Topic> getAllTopics();

................

}

@Service

public class TopicServiceImpl implements TopicService {

@Autowired

private TopicDAO topicDAO;

@Override

public List<Topic> getAllTopics(){

return topicDAO.getAllTopics();

}

...............

}

@autowire JdbcTemplate jdbcTemplate

public void createEmployee(Employee emp){

String createEmp\_SQL = "insert into emp table (emp\_name,salary) values (?,?)";

int update = jdbcTemplate(createEmp\_SQL, emp.getName, emp.getSalary());

if(update == 1){

syso("updated");

}

}

RowMapper

public class EmployeeRowMapper implemets RowMapper<Employee>{

public Employee mapRow(ResultSet rs, int rowNum){

Employee emp = new Employee();

emp.setEmployeeId(rs.getInt("employee\_id"));

emp.setEmployeeName(rs.getString("employee\_name"));

return emp;

}

}

Excluding Dependencies from POM

<exclusions>

<exclusion>

<groupId>------</groupId>

<artifactId> ----- </artifactId>

<exclusion>

<exclusions>

Types of connection pool

Apache tomcat

Hikari

DBCP 2 ( DBCp 1 is depricated)

Application Properties

-----------------------------------

spring.profiles.active = dev

for Database connection

spring.datasource.url =

spring.datasource.username =

spring.datasource.password =

spring.datasource.driver-class-name =

spring.jpa.proerties.hibernate.format\_sql = true / false

spring.jpa.properties.hibernate.dialect =

spring.jpa.hibernate.ddl-auto =

spring.jpa.show-sql =

For Connection Pooling

spring.datasource.tomcat.max-wait = 20000 ms

spring.datasource.tomcat.max-active = 50

spring.datasource.tomcat.max-idle = 20

spring.datasource.tomcat.min-idle = 20

For Execute sql schema/data from classpath

spring.datasource.schema = classpath:schema.sql

spring.datasource.data = classpath:data.sql

For Security Management

endpoints.metrics.sensitive=false (Default true)

Spring boot Cache Management (JCacheCacheManager / EhCacheManager / CaffeineCacheManager )

// This default cache is for testing or for POC not for production

1 . @EnableCacheing

2. Define cache manager

@Bean

public CacheManager cacheManager(){

return new concurrentMapCacheManager("ticketsCache");

}

if u r not specifying the method then u have to write same property in application.properties file

spring.cache.cache-names = ticketsCache

spring.cache.type = simple / none / Ehcache......

3 . controller class modification

@RestController

@RequestMapping(value="/api")

public class TicketBookingController {

@Autowired

private TicketBookingService ticketBookingService;

// use unless when result is null , else it will increase the cache for null value also

@Cacheable(value = "ticketManager" , key = "#ticketId" , unless = "#result == null")

@GetMapping(value="/tickets/ticketId/{ticketId}")

public Ticket getTicketById(@PathVariable("ticketId")Integer ticketId){

return ticketBookingService.getTicketById(ticketId);

}

@Cacheable(value = "ticketManager")

@GetMapping(value="/admin/tickets/alltickets")

public Iterable<Ticket> getAllBookedTickets(){

return ticketBookingService.getAllBookedTickets();

}

}

Transaction Management :

@EnableTransactionanManagement(proxyTargetClass = true) // optional

@Transaction : In Service methods

Batch processing

Spring Batch ItemReader ItemProcessor ItemWriter

| | | |

|-----Read()-------| | |

|-----item---------| | |

| | |

|---------------process()-----------| |

|---------------Transformed Item----| |

| |

| |

| |

|-------------------write(transfored item)----------|

| |

work of chunks of data : small data

Swager 2 : Rest Documentation tool