# Spring Boots

Application.properties

server.port=8080

spring.datasource.type=com.zaxxer.hikari.HikariDataSource

#==== connect to Oracle ======#

#spring.datasource.hikari=oracle.jdbc.pool.OracleDataSource

#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

#spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect

#spring.datasource.url=jdbc:oracle:thin:@14.142.120.194:1521/ORCL

#==== connect to mysql ======#

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/vow\_prod?useSSL=false

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

# HikariCP settings

spring.datasource.hikari.connection-timeout=60000

spring.datasource.hikari.maximum-pool-size=50

# logging

spring.jpa.properties.hibernate.show\_sql=true

spring.jpa.properties.hibernate.use\_sql\_comments=true

spring.jpa.properties.hibernate.format\_sql=true

logging.level.org.hibernate.SQL=DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

POM.xml

<!-- It provides developer tools. These tools are helpful in application

development mode. One of the features of developer tool is automatic restart

of the server for any change in code. -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- HikariCP connection pool -->

<dependency>

<groupId>com.zaxxer</groupId>

<artifactId>HikariCP</artifactId>

</dependency>

<!-- MySQL connector -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- swagger -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.7.0</version>

</dependency>

<!-- swagger ui -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

The entry point of the Spring Boot Application is the class contains **@SpringBootApplication** annotation.

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.

@SpringBootApplication

@RestController

@RequestMapping("/polineitem")

@GetMapping("/getlineitembyponum/{poId}")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

Parameter - @PathVariable("poId") String poId

@PostMapping("/savelineitem")

Parameter - @RequestBody POLineItemDTO polineitem

**return** **new** ResponseEntity<POLineItemDTO>

@Service

@Component

@Transactional

Mapper

@SpringBootApplication

**public** **class** SecurityApplication **extends** SpringBootServletInitializer {

@Override

**protected** SpringApplicationBuilder configure(SpringApplicationBuilder application) {

**return** application.sources(SecurityApplication.**class**);

}

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

SpringApplication.*run*(SecurityApplication.**class**, args);

}

}

@RestController

@RequestMapping("/polineitem")

**public** **class** POLineItemController {

@Autowired

POLineItemServiceImpl polineitemService;

@GetMapping("/getlineitembyponum/{poId}")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

**public** POLineItemDTO getLineItemByPONum(@PathVariable("poId") String poId) {

**return** polineitemService.getLineItemByPoNum(poId);

}

@GetMapping("/getalllineitem")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

**public** List<POLineItemDTO> getAllLineItem(){

**return** polineitemService.getAllPOLineItem();

}

@PostMapping("/savelineitem")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

**public** ResponseEntity<POLineItemDTO> saveLineitem(@RequestBody POLineItemDTO polineitem) {

polineitemService.savePOLineItem(polineitem);

**return** **new** ResponseEntity<POLineItemDTO>(polineitem, HttpStatus.***CREATED***);

}

@PutMapping("/updatepolineitem")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

**public** ResponseEntity<POLineItemDTO> updatePoLineItem(@RequestBody POLineItemDTO polineitem) {

polineitemService.updatePOLineItem(polineitem);

**return** **new** ResponseEntity<POLineItemDTO>(polineitem, HttpStatus.***CREATED***);

}

@DeleteMapping("/deletepolineitem/{id}")

@CrossOrigin(origins = ***USER\_HOST\_SERVER***)

**public** DeleteDTO deletePOLineItem(@PathVariable("id") **long** id) {

polineitemService.deletePOLineItem(id);

DeleteDTO deleteDTO = **new** DeleteDTO();

deleteDTO.setstatus(1);

deleteDTO.setstatusCode(2000);

deleteDTO.setmsg("Deleted successfully.");

**return** deleteDTO;

}

}

@Service

**public** **class** POLineItemServiceImpl **implements** POLineItemService {

@Autowired

POLineItemComponent polineitemDao;

@Override

**public** List<POLineItemDTO> getAllPOLineItem() {

List<POLineItemDTO> polineitemDTO = **new** ArrayList<>();

**try** {

List<POLineItem> polineitemEntity = polineitemDao.getAllPOLineItem();

polineitemEntity.forEach(polineitem->{

polineitemDTO.add(preparePOLineItemDTO(polineitem));

});

}**catch**(Exception e) {

e.printStackTrace();

}

**return** polineitemDTO;

}

**private** POLineItemDTO preparePOLineItemDTO(POLineItem polineitem) {

POLineItemDTO polineitemDTO = **new** POLineItemDTO();

polineitemDTO.setActualQuantity(polineitem.getActualQuantity());

polineitemDTO.setAllowableMoisturePercentage(polineitem.getAllowableMoisturePercentage());

polineitemDTO.setApproveFirstDate(polineitem.getApproveFirstDate());

polineitemDTO.setApproverFirst(polineitem.getApproverFirst());

polineitemDTO.setApproverSecond(polineitem.getApproverSecond());

polineitemDTO.setApproveSecondDate(polineitem.getApproveSecondDate());

polineitemDTO.setDiscount(polineitem.getDiscount());

polineitemDTO.setId(polineitem.getId());

polineitemDTO.setIndentId(polineitem.getIndentId());

polineitemDTO.setItemGroupId(polineitem.getItemGroupId());

polineitemDTO.setItemId(polineitem.getItemId());

polineitemDTO.setMarka(polineitem.getMarka());

polineitemDTO.setPoId(polineitem.getPoId());

polineitemDTO.setQualityCode(polineitem.getQualityCode());

polineitemDTO.setQuantity(polineitem.getQuantity());

polineitemDTO.setRate(polineitem.getRate());

polineitemDTO.setStatus(polineitem.getStatus());

polineitemDTO.setTax(polineitem.getTax());

polineitemDTO.setType(polineitem.getType());

polineitemDTO.setUnitId(polineitem.getUnitId());

polineitemDTO.setValueWithoutTax(polineitem.getValueWithoutTax());

polineitemDTO.setValueWithTax(polineitem.getValueWithTax());

polineitemDTO.setItemDesc(polineitem.getItemDesc());

**return** polineitemDTO;

}

@Override

**public** POLineItemDTO getPOLineItemById(**long** id) {

**return** preparePOLineItemDTO(polineitemDao.getPOLineItemById(id));

}

@Override

**public** ResponseEntity<POLineItemDTO> savePOLineItem(POLineItemDTO polineitem) {

POLineItem polineitemEntity = **new** POLineItem();

polineitemEntity.setActualQuantity(polineitem.getActualQuantity());

polineitemEntity.setAllowableMoisturePercentage(polineitem.getAllowableMoisturePercentage());

polineitemEntity.setApproveFirstDate(polineitem.getApproveFirstDate());

polineitemEntity.setApproverFirst(polineitem.getApproverFirst());

polineitemEntity.setApproverSecond(polineitem.getApproverSecond());

polineitemEntity.setApproveSecondDate(polineitem.getApproveSecondDate());

polineitemEntity.setDiscount(polineitem.getDiscount());

polineitemEntity.setId(polineitem.getId());

polineitemEntity.setIndentId(polineitem.getIndentId());

polineitemEntity.setItemGroupId(polineitem.getItemGroupId());

polineitemEntity.setItemId(polineitem.getItemId());

polineitemEntity.setMarka(polineitem.getMarka());

polineitemEntity.setPoId(polineitem.getPoId());

polineitemEntity.setQualityCode(polineitem.getQualityCode());

polineitemEntity.setQuantity(polineitem.getQuantity());

polineitemEntity.setRate(polineitem.getRate());

polineitemEntity.setStatus(polineitem.getStatus());

polineitemEntity.setTax(polineitem.getTax());

polineitemEntity.setType(polineitem.getType());

polineitemEntity.setUnitId(polineitem.getUnitId());

polineitemEntity.setValueWithoutTax(polineitem.getValueWithoutTax());

polineitemEntity.setValueWithTax(polineitem.getValueWithTax());

polineitemEntity.setItemDesc(polineitem.getItemDesc());

polineitemDao.savePoLineItem(polineitemEntity);

**return** **new** ResponseEntity<POLineItemDTO>(polineitem, HttpStatus.***CREATED***);

}

@Override

**public** ResponseEntity<POLineItemDTO> updatePOLineItem(POLineItemDTO polineitem) {

POLineItem polineitemEntity = polineitemDao.getPOLineItemById(polineitem.getId());

polineitemEntity.setActualQuantity(polineitem.getActualQuantity());

polineitemEntity.setAllowableMoisturePercentage(polineitem.getAllowableMoisturePercentage());

polineitemEntity.setApproveFirstDate(polineitem.getApproveFirstDate());

polineitemEntity.setApproverFirst(polineitem.getApproverFirst());

polineitemEntity.setApproverSecond(polineitem.getApproverSecond());

polineitemEntity.setApproveSecondDate(polineitem.getApproveSecondDate());

polineitemEntity.setDiscount(polineitem.getDiscount());

polineitemEntity.setId(polineitem.getId());

polineitemEntity.setIndentId(polineitem.getIndentId());

polineitemEntity.setItemGroupId(polineitem.getItemGroupId());

polineitemEntity.setItemId(polineitem.getItemId());

polineitemEntity.setMarka(polineitem.getMarka());

polineitemEntity.setPoId(polineitem.getPoId());

polineitemEntity.setQualityCode(polineitem.getQualityCode());

polineitemEntity.setQuantity(polineitem.getQuantity());

polineitemEntity.setRate(polineitem.getRate());

polineitemEntity.setStatus(polineitem.getStatus());

polineitemEntity.setTax(polineitem.getTax());

polineitemEntity.setType(polineitem.getType());

polineitemEntity.setUnitId(polineitem.getUnitId());

polineitemEntity.setValueWithoutTax(polineitem.getValueWithoutTax());

polineitemEntity.setValueWithTax(polineitem.getValueWithTax());

polineitemEntity.setItemDesc(polineitem.getItemDesc());

polineitemDao.savePoLineItem(polineitemEntity);

**return** **new** ResponseEntity<POLineItemDTO>(polineitem, HttpStatus.***CREATED***);

}

@Override

**public** DeleteDTO deletePOLineItem(**long** id) {

polineitemDao.deletePOLinteItem(id);

DeleteDTO deleteDTO = **new** DeleteDTO();

deleteDTO.setstatus(1);

deleteDTO.setstatusCode(2000);

deleteDTO.setmsg("Deleted successfully.");

**return** deleteDTO;

}

@Override

**public** POLineItemDTO getLineItemByPoNum(String poId) {

**return** preparePOLineItemDTO(polineitemDao.getpolineitemByPoNum(poId));

}

}

@Component

**public** **class** POLineItemComponent {

@Autowired

POLineItemRepository polineitemRepository;

@Transactional

**public** List<POLineItem> getAllPOLineItem(){

**return** polineitemRepository.findAll();

}

@Transactional

**public** POLineItem getPOLineItemById(**long** id) {

**return** polineitemRepository.findOne(id);

}

@Transactional

**public** **void** savePoLineItem(POLineItem polineitem) {

polineitemRepository.save(polineitem);

}

@Transactional

**public** **void** deletePOLinteItem(**long** id) {

polineitemRepository.delete(id);

}

@Transactional

**public** POLineItem getpolineitemByPoNum(String poId) {

**return** polineitemRepository.findByPoId(poId);

}

}

**public** **interface** POLineItemRepository **extends** JpaRepository<POLineItem, Long> {

POLineItem findByPoId(String poId);

List<POLineItem> findByIndentId(String string);

}

@Entity()

@Table(name="SCM\_PO\_LINE\_ITEM")

**public** **class** POLineItem {

@Id

@GeneratedValue(strategy = GenerationType.***SEQUENCE***, generator = "PO\_ITM\_SEQ")

@SequenceGenerator(sequenceName = "SCM\_PO\_ITEM\_SEQ", allocationSize = 1, name = "PO\_ITM\_SEQ")

@Column(name="LINE\_ITEM\_NUM")

**private** **long** id;

@Column(name="PO\_NUM")

**private** String poId;

@Column(name="INDENT\_NO")

**private** String indentId;

}

@Entity

@Table(name = "ITEM\_GROUP\_MASTER")

**public** **class** ItemGroupMaster **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id()

@Column(name = "GROUP\_CODE")

// @GeneratedValue(strategy = GenerationType.SEQUENCE)

**private** String id;

@OneToMany(fetch= FetchType.***EAGER***)

@JoinColumn(name = "GROUP\_CODE")

**private** List<ItemMaster> itemmasteres;

@Column(name = "GROUP\_DESC")

**private** String grpDsc;

@Column(name = "ACTIVE\_FLAG")

**private** String activeFlag;

@Column(name = "PURCHASE\_CODE")

**private** Long purchaseCode;

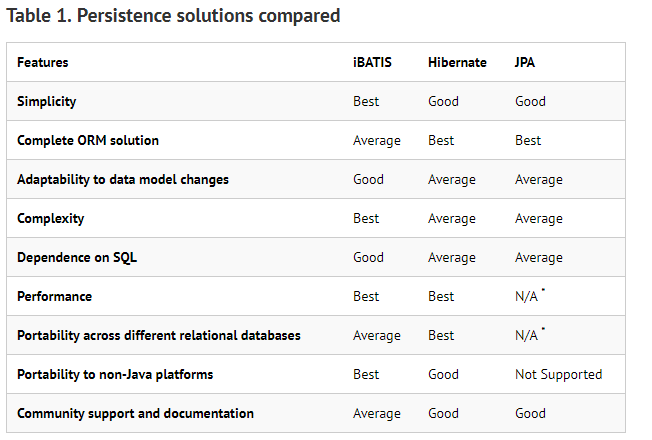
}

# Interview Questions

Java Persistence Framework (MyBatis,Hibernate)

The layer that separates the business logic and the database code is the persistence layer, which keeps the application independent of the underlying database technology.

Hibernate and JPA map Java objects directly to database tables, whereas iBATIS maps Java objects to the results of SQL queries.



SpringSecurity , SpringBoots , MyBatis code sample

<https://github.com/ShawnyXiao/SpringBoot-MyBatis>

Java Bean and EJB

"**Java Beans**" are used to *store*the data *retrieved data*from the database and used as a **container**to carry the data between the Servlets and JSPs in **MVC model**. A class (container) with setters and getters is used to (put) and (get) the data.

"**Enterprise Java Beans**" are similar to "Java Beans" with *added features*such as **Session Management, Security, Transaction and etc.,**with the *aid of different types of EJBs*that are

1.    Session Bean

2.    Entity Beam

3.    Message Driven Beans

Java Bean and POJO

A **JavaBean** follows certain conventions. Getter/setter naming, having a public default constructor, being serializable etc. See [JavaBeans Conventions](http://docstore.mik.ua/orelly/java-ent/jnut/ch06_02.htm) for more details.

A **POJO** (plain-old-Java-object) isn't rigorously defined. It's a Java object that doesn't have a requirement to implement a particular interface or derive from a particular base class, or make use of particular annotations in order to be compatible with a given framework, and can be any arbitrary (often relatively simple) Java object.

Java bean / POJO / EJB

A **JavaBean** is a POJO that is serializable, has a no-argument constructor, and allows access to properties using getter and setter methods. An Enterprise JavaBean is not a single class but an entire component model (again, EJB 3 reduces the complexity of Enterprise JavaBeans).

JavaBeans must implement Serializable and have a no-argument constructor whereas in POJO does not have these restrictions

**EJB3** is the latest standard for developing enterprise applications which replaces EJB2 and is based on concept of taking POJOs and annotating them so that they can be used in enterprise app.

@Stateless

class MyService {

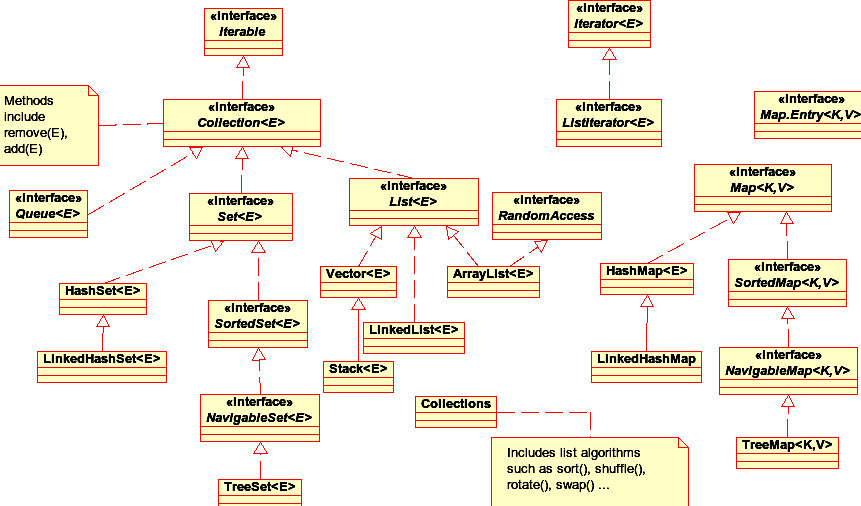
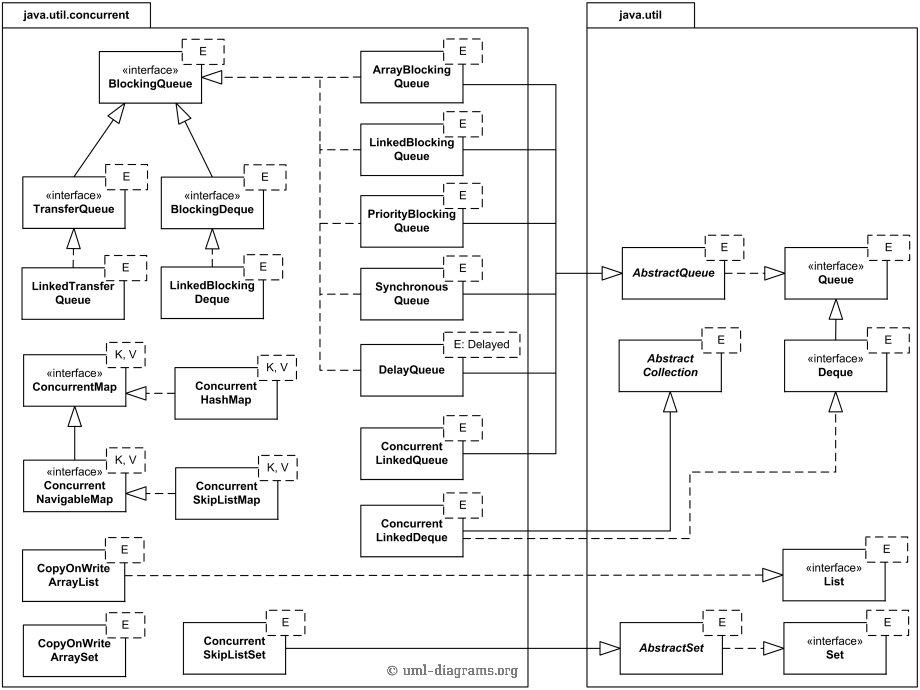
    public String sayHello() { return "hello world"; }

}

As you can see it's very similar to POJOs. In fact most application written for EJB3 can easily be converted to work with Spring, and usually the other way works too.

Benefits of POJO classes :

1. The benefit of using only POJOs is that you do not need an EJB container product such as an application server like WebSphere but you have the option of using only a robust servlet container such as Tomcat or some commercial product

Difference between Enumeration and Iterator interface.

Enumeration is twice as fast as Iterator.

Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object .

Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration

different between Iterator and ListIterator

# WordPress

<https://www.youtube.com/watch?v=OAF-NRVcaP8>

<https://www.youtube.com/watch?v=YslHrObUxTc>

<https://www.youtube.com/playlist?list=PLIl52Z3d-xQN2s3sxrX39KB1XOkKsn9Cr>

SITE TITLE

<?php bloginfo( 'name' ); ?>

Now, our theme will pick up the title that we set in Settings -> General-> Site title inside WP-admin panel.

MENU

Method 1:

We need to enable Menu first by adding following line of code in functions.php file.

<?php add\_theme\_support( 'menus' ); ?>

In our theme, our index.php contains the navigation menu. Find following lines of code:

<li><a class="active" href="index.html">Home Page</a></li>

<li><a href="news.html">Our News</a></li>

<li><a href="services.html">Our Services</a></li>

<li><a href="products.html">Our Products</a></li>

<li class="last-item"><a href="contacts.html">Contact Us</a></li>

Replace above lines with:

<?php wp\_nav\_menu( array( 'sort\_column' => 'menu\_order', 'container\_class' => 'menu-header' ) ); ?>

https://developer.wordpress.org/reference/functions/wp\_nav\_menu/

Method 2:

Add the following line of code in functions.php file.

[cid:image005.png@01D4CEC6.C221ABA0]

[cid:image007.png@01D4CEC7.88D967B0]

# Spring Annotation and MyBatis

<mvc:annotation-driven />

Configures the annotation-driven Spring MVC Controller programming model. Note that, with Spring 3.0, this tag works in Servlet MVC only!

<context:component-scan base-package="com.walgreens.bdm,com.walgreens.cos" />

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

WEB

-------

<context:annotation-config /> -

Activates various annotations to be detected in bean classes: Spring's @Required and @Autowired, as

well as JSR 250's @PostConstruct, @PreDestroy and @Resource (if available), JAX-WS's

@WebServiceRef (if available), EJB3's @EJB (if available), and JPA's @PersistenceContext and

@PersistenceUnit (if available). Alternatively, you may choose to activate the individual

BeanPostProcessors for those annotations. Note: This tag does not activate processing of Spring's

@Transactional or EJB3's @TransactionAttribute annotation. Consider the use of the tag for that

purpose.

<context:component-scan base-package="com.walgreens.bdm" />

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location">

<value>database.properties</value>

</property>

</bean>

<bean id="DataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName"

value="oracle.jdbc.OracleDriver" />

<property name="url"

value="jdbc:oracle:thin:@${dbServer}:${dbPort}:${dbService}" />

<property name="username" value="${dbUser}" />

<property name="password" value="${dbPwd}" />

</bean>

<!-- Configure MyBatis SqlSessionFactory -->

<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="DataSource" />

<property name="configLocation" value="classpath:com/walgreens/bdm/common/mybatis/mybatis-config.xml" />

<property name="mapperLocations" value="classpath\*:com/walgreens/bdm/common/mybatis/\*Mapper.xml" />

<property name="typeAliasesPackage" value="com.walgreens.bdm.common.domain" />

</bean>

<!-- Scan for MyBatis Mappers to allow Autowiring -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.walgreens.bdm.common.persistence" />

<property name="sqlSessionFactoryBeanName" value="sqlSessionFactory" />

</bean>

<!-- Configure Spring Transaction Management -->

<bean id="dbTransactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="DataSource" />

</bean>

<tx:annotation-driven transaction-manager="dbTransactionManager" />

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

context.refresh();

BeanFactory beanFactory = context.getBeanFactory();

public class COSPTBController{

@Component

public class COSPTBBatchBO

{

@Autowired

private COSPTBBatchMapper cosPtbBatchMapper=null;

id A unique identifier in this namespace that can be used to reference this statement.

parameterType The fully qualified class name or alias for the parameter that will be passed into this statement. This attribute is optional because MyBatis can calculate the TypeHandler to use out of the actual parameter passed to the statement. Default is unset.

parameterMap This is a deprecated approach to referencing an external parameterMap. Use inline parameter mappings and the parameterType attribute.

resultType The fully qualified class name or alias for the expected type that will be returned from this statement. Note that in the case of collections, this should be the type that the collection contains, not the type of the collection itself. Use resultType OR resultMap, not both.

resultMap A named reference to an external resultMap. Result maps are the most powerful feature of MyBatis, and with a good understanding of them, many difficult mapping cases can be solved. Use resultMap OR resultType, not both.

flushCache Setting this to true will cause the local and 2nd level caches to be flushed whenever this statement is called. Default: false for select statements.

useCache Setting this to true will cause the results of this statement to be cached in 2nd level cache. Default: true for select statements.

timeout This sets the number of seconds the driver will wait for the database to return from a request, before throwing an exception. Default is unset (driver dependent).

fetchSize This is a driver hint that will attempt to cause the driver to return results in batches of rows numbering in size equal to this setting. Default is unset (driver dependent).

statementType Any one of STATEMENT, PREPARED or CALLABLE. This causes MyBatis to use Statement, PreparedStatement or CallableStatement respectively. Default: PREPARED.

resultSetType Any one of FORWARD\_ONLY|SCROLL\_SENSITIVE|SCROLL\_INSENSITIVE. Default is unset (driver dependent).

databaseId In case there is a configured databaseIdProvider, MyBatis will load all statements with no databaseId attribute or with a databaseId that matches the current one. If case the same statement if found with and without the databaseId the latter will be discarded.

resultOrdered This is only applicable for nested result select statements: If this is true, it is assumed that nested results are contained or grouped together such that when a new main result row is returned, no references to a previous result row will occur anymore. This allows nested results to be filled much more memory friendly. Default: false.

resultSets This is only applicable for multiple result sets. It lists the result sets that will be returned by the statement and gives a name to each one. Names are separated by commas.

<select>

{call

DECLARE

end }

</select>

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

WEB

-------

<context:annotation-config />

<context:component-scan base-package="com.walgreens.bdm" />

<import resource="bounce\_common.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/accelerate/CommonBeans.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/authrules.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/HTTPMapper.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/RequestHandlers.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/ApplicationContext.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/DataAccessors.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/DataSources.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/CacheBeans.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/Converters.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/WebFiles.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/AppProfiling.xml" />

<import resource="classpath\*:com/walgreens/bdm/common/spring/Interceptors.xml" />

<bounceweb:action id="/searchPlanogram" authRuleBeanRef="bdmRefreshLogin"

requestBean="com.walgreens.bdm.beans.searchplanogram.PlanogramSearchRequestBean"

webPreProcessorBeanRef="searchPlanogramPreProcessor" webPostProcessorBeanRef=""

serviceBeanRef="planogramSearchService"

method="showSearchPlanogram">

<bounceweb:result name="success" type="tiles" value="searchPlanogram" />

<bounceweb:result name="denied" type="tiles" value="noAccess"/>

</bounceweb:action>

@Service

public class PlanogramSearchBO {

List<Integer> planoNumber = new ArrayList<Integer>();

/\*\*

\* Auto wiring PlanogramSearchMapper

\*/

@Autowired

private PlanogramSearchMapper planogramMapper = null;

@Controller

public class PlanogramSearchService {

/\*\*

\* Autowiring PlanogramSearchBO

\*/

@Autowired

private PlanogramSearchBO planogramSearchBO = null;

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

Spring MVC Framework

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

web.xml

<web-app id = "WebApp\_ID" version = "2.4"

xmlns = "http://java.sun.com/xml/ns/j2ee"

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

xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>Spring MVC Application</display-name>

<!-------- DispatcherServlet definition goes here----->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<servlet>

<servlet-name>HelloWeb</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>HelloWeb</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

</web-app>

HelloWeb-servlet.xml

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

xmlns:context = "http://www.springframework.org/schema/context"

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

http://www.springframework.org/schema/context

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

<context:component-scan base-package = "com.tutorialspoint" />

<bean class = "org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name = "prefix" value = "/WEB-INF/jsp/" />

<property name = "suffix" value = ".jsp" />

</bean>

</beans>

@Controller

public class StudentController {

@RequestMapping(value = "/student", method = RequestMethod.GET)

public ModelAndView student() {

return new ModelAndView("student", "command", new Student());

}

@RequestMapping(value = "/addStudent", method = RequestMethod.POST)

public String addStudent(@ModelAttribute("SpringWeb")Student student,

ModelMap model) {

model.addAttribute("name", student.getName());

model.addAttribute("age", student.getAge());

model.addAttribute("id", student.getId());

return "result";

}

}

<%@taglib uri = "http://www.springframework.org/tags/form" prefix = "form"%>

<html>

<head>

<title>Spring MVC Form Handling</title>

</head>

<body>

<h2>Student Information</h2>

<form:form method = "POST" action = "/HelloWeb/addStudent">

<table>

<tr>

<td><form:label path = "name">Name</form:label></td>

<td><form:input path = "name" /></td>

</tr>

<tr>

<td><form:label path = "age">Age</form:label></td>

<td><form:input path = "age" /></td>

</tr>

<tr>

<td><form:label path = "id">id</form:label></td>

<td><form:input path = "id" /></td>

</tr>

<tr>

<td colspan = "2">

<input type = "submit" value = "Submit"/>

</td>

</tr>

</table>

</form:form>

</body>

</html>