Spring work ::

Routes all request through a disparcherServlet.. if someone goes the browser , tries to do go first web app, Spring sent request to this disparcherServlet .. And disparcherServlet gonna go and look for a controller , then tell it what to do with that request ,, what view to display using what data.. In web.xml…

View Resolver for JSPs

Xuất thong báo từ validation sang jsp dung csslass:

<sf:errors path=*"email"* cssClass=*"error"*>

Spring – form- error

1. path=”\*” – display all error messages associated with any fields.
2. path=”username” – display error messages associated with the “username” field only.

<p><a href=*"*${pageContext.request.contextPath}*/offers"*>Show offers</a></p>

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

[SqlParameterSource](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/SqlParameterSource.html) implementation that holds a given Map of parameters.

This class is intended for passing in a simple Map of parameter values to the methods of the [NamedParameterJdbcTemplate](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/NamedParameterJdbcTemplate.html) class.

The addValue methods on this class will make adding several values easier. The methods return a reference to the [MapSqlParameterSource](http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/jdbc/core/namedparam/MapSqlParameterSource.html) itself, so you can chain several method calls together within a single statement.

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

Also, JdbcTemplate objects have very little state, and are thread-safe. There is very little implact to creating multiple objects.

If you go for **JdbcTemplate**then you need to assign parameters in the code you have to pass in arguments in an array and they get used in the order in which they appear in the array, like this:

**Object[] args = new Object[] {"x", "y"};  
String sql = "select \* from foo where a = ? and b = ?";  
jdbcTemplate.query(sql, args, resultSetExtractor);**

And if you go for **NamedParameterJdbcTemplate**then you need to assign names to the parameter placeholders and pass in a map so the template can match the map names to the placeholders, like this:

**String sql = "select \* from foo where a = :mya and b = :myb";  
Map<String, Object> argMap = new HashMap<String, Object>();  
argMap.put("mya", "x");  
argMap.put("myb", "y");  
namedParameterJdbcTemplate.query(sql, argMap, resultSetExtractor);**

The idea is that matching the arguments by name is less error-prone than having to specify them in a particular order. In a real-life application typically SQL is stored in a separate file from the DAO code, and getting the parameters in the right order can be painful.

When you use JdbcTemplate you give it SQL that has a ? placeholder for each parameter you want substituted into the SQL. When you assign parameters in the code you have to pass in arguments in an array and they get used in the order in which they appear in the array, like this:

Object[] args = new Object[] {"x", "y"};

String sql = "select \* from foo where a = ? and b = ?";

jdbcTemplate.query(sql, args, resultSetExtractor);

so the SQL that gets run is select \* from foo where a = 'x' and b = 'y'.

NamedParameterJdbcTemplate allows you to assign names to the parameter placeholders and pass in a map so the template can match the map names to the placeholders. So your code would look like:

String sql = "select \* from foo where a = :mya and b = :myb";

Map<String, Object> argMap = new HashMap<String, Object>();

argMap.put("mya", "x");

argMap.put("myb", "y");

namedParameterJdbcTemplate.query(sql, argMap, resultSetExtractor);

generating the same SQL as the first example (and running the query is the time-intensive part, the performance of the argument insertion is a non-issue.).

The idea is that matching the arguments by name is less error-prone than having to specify them in a particular order. In a real-life application typically SQL is stored in a separate file from the DAO code, and it may be easy to accidentally get the parameters in the wrong order

<context:annotation-config> is used to activate annotations in beans already registered in the application context (no matter if they were defined with XML or by package scanning).

<context:component-scan> can also do what <context:annotation-config> does but <context:component-scan> also scans packages to find and register beans within the application context.

Bên trong mỗi controller sẽ chứa các phương thức, các phương thức sẽ đảm nhiệm việc nhận yêu cầu từ các URL. Để biết phương thức nào sẽ nhận yêu cầu từ URL nào, ta đặt annotation @RequestMapping với tham số là url tương ứng bên trên phương thức mỗi phương thức.

Specifically, @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET)

@GetMapping is the newer annotaion. @GetMapping does not support the **consumes** attribute of @RequestMapping.

Consume are :

**consumes = "text/plain"  
consumes = {"text/plain", "application/\*"}**

Indicates that an annotated class is a "component". Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

| @Component | generic stereotype for any Spring-managed component |

| @Repository| stereotype for persistence layer |

| @Service | stereotype for service layer |

| @Controller| stereotype for presentation layer (spring-mvc)

hey are almost the same - all of them mean that the class is a Spring bean. @Service, @Repository and @Controller are specialized @Components. You can choose to perform specific actions with them. For example:

* @Controller beans are used by spring-mvc
* @Repository beans are eligible for persistence exception translation

Another thing is that you designate the components semantically to different layers.

One thing that @Component offers is that you can annotate other annotations with it, and then use them the same way as @Service.

* <context:annotation-config /> is used in order to be able to use *@Autowired* annotation
* <context:component-scan /> is used to determine the search of specific beans and attempt of autowiring.

<context:annotation-config> is used to activate annotations in beans already registered in the application context (no matter if they were defined with XML or by package scanning).

<context:component-scan> can also do what <context:annotation-config> does but <context:component-scan> also scans packages to find and register beans within the application context.

the mvc:annotationDriven tag essentially sets you your Spring context to allow for dispatching requests to Controllers.

The tag will configure two beans DefaultAnnotationHandlerMapping and AnnotationMethodHandlerAdapter.

Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors

The @Component annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context.

The @Service annotation is also a specialization of the component annotation. It doesn’t currently provide any additional behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer classes because **it specifies intent better**. Additionally, tool support and additional behavior might rely on it in the future.

@Controller annotation marks a class as a Spring Web MVC controller. It too is a @Component specialization, so beans marked with it are automatically imported into the DI container. When you add the @Controller annotation to a class, you can use another annotation i.e. @RequestMapping; to map URLs to instance methods of a class.

Spring provides four different types of auto component scan annotations, they are @Component, @Service, @Repository and @Controller. Technically, there is no difference between them, but every auto component scan annotation should be used for a special purpose and with in the defined layer. Here is the purpose of each annotation

***@Component***: It is a basic auto component scan annotation, it indicates annotated class is a auto scan component.  
  
***@Repository***: You need to use this annotation with in the persistance layer, which acts like database repository.  
  
***@Service***: It indicates annotated class is a Service component in the business layer.  
  
***@Controller***: Annotated class indicates that it is a controller components, and mainly used at presentation layer.

As above said, technically there is no difference between them, all of the 3 annotations @Service, @Repository and @Controller implementations are annotated by @Controller annotation. So technically you can use @Controller for any auto component scan, but for better readability and better standards use respective annotations.

Convenient super class for JDBC-based data access objects.

Requires a DataSource to be set, providing a JdbcTemplate based on it to subclasses through the getJdbcTemplate() method.

This base class is mainly intended for JdbcTemplate usage but can also be used when working with a Connection directly or when using org.springframework.jdbc.object operation objects.

Implicit Objects

The JSP expression language defines a set of implicit objects:

* pageContext: The context for the JSP page. Provides access to various objects including:
  + servletContext: The context for the JSP page’s servlet and any web components contained in the same application. See Accessing the Web Context.
  + session: The session object for the client. See Maintaining Client State.
  + **request: The request triggering the execution of the JSP page. See**[**Getting Information from Requests**](http://java.sun.com/javaee/5/docs/tutorial/doc/bnafv.html#bnafw)**.**
  + response: The response returned by the JSP page. See Constructing Responses.
* In addition, several implicit objects are available that allow easy access to the following objects:
  + **param: Maps a request parameter name to a single value**
  + paramValues: Maps a request parameter name to an array of values
  + header: Maps a request header name to a single value
  + headerValues: Maps a request header name to an array of values
  + cookie: Maps a cookie name to a single cookie
  + initParam: Maps a context initialization parameter name to a single value
* Finally, there are objects that allow access to the various scoped variables described in Using Scope Objects.
  + pageScope: Maps page-scoped variable names to their values
  + requestScope: Maps request-scoped variable names to their values
  + sessionScope: Maps session-scoped variable names to their values
  + applicationScope: Maps application-scoped variable names to their values
  + If your data are confidential, it's better to use the POST method because it will not reflect on the url, unlike GET.

commandName = name of a variable in the request scope or session scope that contains the information about this form,or this is model for this view.

In xml based config, we will use command class to pass an object between controller and views. Now in annotation we are using modelattribute.

<bean id=*"jspViewResolver"*

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

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

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

</bean>

<bean id=*"messageSource"*

class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"basename"*>

<value>com.caveofprogramming.spring.web.messages</value>

</property>

</bean>

@Entity annotation to the Employee class which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

rước tiên là nói về JPA, JPA là viết tắt của Java Persistence API, nó là đặc tả hay còn có thể gọi là chuẩn cho cách chúng ta truy cập, lưu trữ và quản lý dữ liệu giữa các đối tượng trong Java và các loại database (Object Relational Mapping). Như các bạn thấy, như tên gọi của mình, JPA chỉ là một API định nghĩa các đặc tả cần thiết và không có code hiện thực từ những đặc tả đó. Nó chỉ chứa những hướng dẫn để hiện thực Object Relational Mapping.

Trong khi đó, Hibernate là một framework hiện thực các đặc tả của JPA. Dựa vào các đặc tả của JPA, Hibernate sẽ hiện thực tất cả các tính năng cần có của một Object Relational Mapping framework. Tất cả những thay đổi trong đặc tả của JPA, Hibernate cũng phải có những cập nhập trong cách hiện thực theo những thay đổi đó.

Nói tóm lại, chúng ta có thể xem JPA là một interface và Hibernate là người hiện thực cái interface đó.

Validated was added to support "validation groups", i.e. group of fields in the validated bean. This can be used in multi step forms where you may validate name, email, etc.. in first step and then other fields in following step(s).

The reason why this wasn't added into @Valid annotation is because that it is standardized using the java community process (JSR-303), which takes time and Spring developers wanted to allow people to use this functionality sooner.

Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime,

Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions. The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context.

annotation-config : Annotation config main job is to activate all the annotations that are present in java beans and those are already registered either by defining in your application context file or being registered while component scanning. Important point is they need to be registered.

component-scan : Component scan can do everything that annotation config does , in addition to it it also registers the java classes as spring bean those are annotated with @Component , @Service ,@Repository etc.

<context:annotation-config> is used to activate annotations in beans already registered in the application context (no matter if they were defined with XML or by package scanning).

<context:component-scan> can also do what <context:annotation-config> does but <context:component-scan> also scans packages to find and register beans within the application context.

There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of @Qualifier annotation.

**SAI: ComponentScan(basePackages = {"spring.java"}) can not be used with Core Spring (Java Standard Edition) programs. It is only used in Spring MVC applications.**

* **Liệt kê những module quan trọng của Spring framework**

Một vài module quan trọng của Spring framework sau:  
Spring Context – hỗ trợ dependency injection.  
Spring AOP – hỗ trợ aspect oriented programming.  
Spring DAO – hỗ trợ database operations sử dụng DAO pattern  
Spring JDBC – hỗ trợ JDBC and DataSource.  
Spring ORM – hỗ trợ ORM  
Spring Web Module – hỗ trợ phát triển ứng dụng web.  
Spring MVC – Model-View-Controller hỗ trợ phát triển web application, web services vv.

*@Autowired*: Tự động liên kết các *Bean* lại với nhau

*@Transactional*: dùng đánh dấu các class có sử dụng đến Transaction do Spring quản lý

*@Scope*: mặc định trong Spring, hầu hết **scope** phổ biến cho các *autodetected components* là **singleton** (singleton: Với mỗi *bean container* chỉ tao duy nhất một đối tượng), để thay đổi phạm vi ta sử dụng *@scope*

*@Component*: dùng để scan các components

*@Repository*: dùng để đánh dấu các lớp *DAO/Repository* của Spring

*@Service*: dùng để đánh dấu các *Service* của lớp business

In Spring MVC, **InternalResourceViewResolver** is used to resolve “internal resource view” (in simple, it’s final output, jsp or htmp page) based on a predefined URL pattern. In additional, it allow you to add some predefined prefix or suffix to the view name (prefix + view name + suffix), and generate the final view page URL.

# [Initialize database without XML configuration, but using @Configuration](http://stackoverflow.com/questions/16038360/initialize-database-without-xml-configuration-but-using-configuration)

# Spring embedded database examples

-🡪>Config data by @configcation

## lass EmbeddedDatabaseBuilder

* [java.lang.Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html?is-external=true)
  + org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder

public class **EmbeddedDatabaseBuilder**

extends [Object](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html?is-external=true)

A builder that provides a convenient API for constructing an embedded database.

### *Usage Example*

Annotation providing a convenient and declarative mechanism for adding a PropertySource to Spring's Environment. To be used in conjunction with @Configuration classes.

@Configuration

@PropertySources(value = @PropertySource("classpath:/datasource.properties")})

    @Autowired

    private Environment env;

* Đặc điểm của String là immutable, tức là String đã tạo ra thì không bao giờ thay đổi giá trị.  
  VD: **String str1 = "ABC";** lệnh này tương đương với **String str1 = new String("ABC");**  
  -> Sẽ tạo ra 1 String Object với gia trị là ABC, **Object này sẽ không bao giờ thay đổi giá trị bằng bất cứ cách nào**.  
  Đó là tại sao tất cả các hàm của String như subString(), toLowerCase()... đều không thay đổi giá trị của nó, mà nó return 1 String mới.  
  Vậy chuyện gì sẽ xảy ra nếu chúng ta thêm đoạn code: **String str1 = str1 + "D";**  
  Đầu tiên Java sẽ tạo ra 1 String Object mới với giá trị là **D**, sau đó với toán tử +, một String mới nữa sẽ được tạo ra với giá trị là tổng của chúng (ABCD).  
  -> biến str1 sẽ trỏ đến String Object mới này. String Object với giá trị "ABC" ban đầu vẫn còn tồn tại ở đó
* StringBuilder, StringBuffer: Theo tính chất của String ở trên, vậy cứ khi nào bạn + 2 hay nhiều String, nó lại tạo ra rất nhiều Object mới. StringBuilder và StringBuffer sinh ra để giải quyết vấn đề này. Bạn sẽ chỉ tạo ra duy nhất 1 Object StirngBuilder hay StringBuffer, Java sẽ mặc định cấp phát 32 hay 64 bit cho nó (mình không nhớ rõ lắm). Mỗi khi bạn thao tác trên Object này (append, remove...) thì giá trị của Object thay đổi mà không tạo ra Object khác. Nếu vượt quá 32 bit, nó sẽ tự động extend vùng cấp phát theo sỗ mũ của 2 (32=2^5, 64=2^6, 128=2^7...).
* Sự khác nhau duy nhất giữa StringBuilder và StringBuffer là tất cả các hàm của StringBuffer là Synchronize(thích hợp với xử lý đa luồng – multi thread), StringBuilder thì không.

Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh. Hibernate cho phép bạn truy vẫn dữ liệu bằng ngôn ngữ SQL mở rộng của Hibernate (HQL) hoặc bằng SQL thuần.

Peristence layer: một ứng dụng có thể được chia làm 3 phần như sau: giao diện người dùng (presentation layer), phần xử lý nghiệp vụ (business layer) và phần chứa dữ liệu (data layer). Cụ thể ra, business layer có thể được chia nhỏ thành 2 layer con là business logic layer (các tính toán nhằm thỏa mãn yêu cầu người dùng) và persistence layer. Persistence layer chịu trách nhiệm giao tiếp với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ – Relational DBMS). Persistence sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.

@Entity annotation defines that a class can be mapped to a table. And that is it, it is just a marker, like for example Serializable interface

Okay, so we talked about the fact that FetchType.LAZY is the default fetch type for all Hibernate annotation relationships. We also talked about the fact that when you use the Lazy fetch type, Hibernate won’t load the relationships for that particular object instance. So then, what’s the difference between Eager and Lazy?

Fetch type Eager is essentially the opposite of Lazy, Eager will by default load ALL of the relationships related to a particular object loaded by Hibernate. This means that if you change the relationship to be this

The way I like to put it is like this:

* + **EAGER**: Convenient, but slow
  + **LAZY**: More coding, but much more efficient

Overriding xuất hiện khi trong class con tồn tại một method được định nghĩa với cùng tên và cùng số lượng tham số với method của class cha.

Overloadin xuất hiện khi trong một class tồn tại một method được định nghĩa với cùng tên và cùng số lượng tham số đầu vào với method của class cha.

SQL - Structured Query Language. It is declarative computer language aimed at querying relational databases.

MySQL is a relational database - a piece of software optimized for data storage and retrieval. There are many such databases - Oracle, Microsoft SQL Server, SQLite and many others are examples of such.

Hashtable is basically a data structure to retain values of key-value pair.

* It does not allow null for both key and value. It will throw NullPointerException.
* Hashtable does not maintain insertion order. The order is defined by the Hash function. So only use this if you do not need data in order.
* It is synchronized. It is slow. Only one thread can access in one time.
* HashTable rea thread safe.
* HashTable uses Enumerator to iterate through elements.

Luu file

FileoutputSttream

OutputStreamWriter

BufferedWriter

Ghi file: out 🡪 int

HashMap makes absolutely no guarantees about the iteration order. It can (and will) even change completely when new elements are added.

LinkedHashMap will iterate in the order in which the entries were put into the map

HashTable

* same as hash map
* it does not allows null keys and null values
* Hashtable sẽ sử dụng hashCode() để xác định được “vị trí” của object và equals() để xác định chính xác object cần tìm.
* **Lưu ý**
* Nếu 2 object equals() là true thì sẽ có cùng hashcode() tuy nhiên nếu 2 object có cùng hash code sẽ không đảm bảo là kết quả equals sẽ là true.
* When the equals() method is overridden, the hashCode() method must be overridden as well.
* If two objects are equal, their hash codes must be equal as well.
* If two objects are not equal, there’s no constraint on their hash codes (their hash codes can be equal or not).
* If two objects have identical hash codes, there’s no constraint on their equality (they can be equal or not).
* If two objects have different hash codes, they must not be equal.
* **equals()** checks if two objects are the same or not and returns a boolean.
* **compareTo()** (from interface Comparable) returns an integer. It checks which of the two objects is "less than", "equal to" or "greater than" the other. Not all objects can be logically ordered, so a compareTo() method doesn't always make sense.
* Note that equals() doesn't define the ordering between objects, which compareTo() does.

Spring bean có hai scope quan trọng là singleton và prototype. Singleton bean là những bean được tạo ra vào thời điểm khởi tạo spring context. Bean này được tạo ra đúng một lần trong suốt thời gian hoạt động của application. Bất cứ khi nào có yêu cầu inject thì instance đã có của bean này sẽ được lôi ra sử dụng. Prototype bean là những bean được tạo ra bất cứ khi nào có yêu cầu inject. Khác với singleton bean, bạn sẽ có nhiều instance của prototype bean trong spring context.

Có hai phương thức quan trọng nhất trong Spring bean lifecycle. Đầu tiên đó là setup(), phương thức nay được gọi khi bean được load vào container. Phương thức thứ hai đó là teardown(), phương thức này được gọi khi bean được unload khỏi container.

Tag <bean> có hai thuộc tính quan trọng là **init-method** và **destroy-method**với thuộc tính này, chúng ta có thể khai báo tùy biến phương thức khởi tạo và destroy cho mỗi bean. Ngoài ra nếu bạn sử dụng Annotation-based thì Spring framework cũng cung cấp hai annotation là @PostConstruct và @PreDestroy

* <list> dùng để inject list, có thể có các phần tử trùng nhau.
* <set> dùng để inject set, các phần tử không trùng nhau.
* <map> dùng để inject collection dạng key-value, trong đó key và value là loại tùy ý.
* <props> dùng để inject collection dạng key-value, trong đó key và value đều là String.

There are several reasons for using nested classes, among them:

It is a way of logically grouping classes that are only used in one place.

It increases encapsulation.

Nested classes can lead to more readable and maintainable code.

Child to parent class connection is simpler as it visually illustrates the variables and methods of each class.

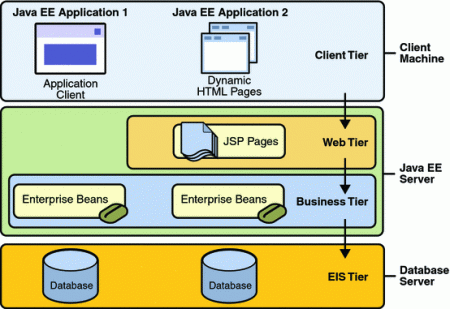
INNER JOIN gets all records that are common between both tables

LEFT JOIN gets all records from the LEFT linked table but if you have selected some columns from the RIGHT table, if there is no related records, these columns will contain NULL

RIGHT JOIN is like the above but gets all records in the RIGHT table

FULL JOIN gets all records from both tables and puts NULL in the columns where related records do not exist in the opposite table

– EJB là 1 thành phần nằm ở phía server -side của 1 ứng dụng web hoặc có thể hiểu là 1 thành phần phần trong kiến trúc Java EE .



– Nhiệm vụ chính là sử lý các nghiệp vụ logic và truy suất dữ liệu, có thể hiểu EJB là Model trong mô hình MVC2 tôi đã đề cập ở các bài viết trước.

– EJB là 1 nền tảng cho phép xây dựng phần mềm có tính di động , có khả năng tái sử dụng cao, tính bảo mật cũng là 1 đặc điểm quan trọng của EJB.

– Đặc điểm quan trọng nữa là EJB là 1 thành phần hướng tới các ứng dụng có quy mô lớn, thích hợp với mô hình phân tán.

– Nơi đặt các EJB là trong 1 EJB container. Container này sẽ lưu trữ , lập kế hoạch và, và quản lý các EJB.

**huận lợi của EJB:**

* Transaction Management (Quản lý transaction)
* Declarative Transaction support
* Persistence ( CMP & BMP)
* Declarative Security
* Distributed Computing (Container managed RPC)

ột vài cách thức sử dụng ApplicationContext thường dùng là:

* AnnotationConfigApplicationContex: Cho standalone java applications sử dụng annotations để configuration.
* ClassPathXmlApplicationContext: Cho standalone java applications sử dụng XML configuration.
* FileSystemXmlApplicationContext: Tương tự như ClassPathXmlApplicationContext, ngoại trừ việc file XML có thể được load ở bất kì đâu trong hệ thống.
* AnnotationConfigWebApplicationContext và XmlWebApplicationContext được sử dụng cho web applications.

ORM (Object Relational Mapping) framework là một cơ chế cho phép người lập trình thao tác với database một cách hoàn toàn tự nhiên thông qua các đối tượng. Lập trình viên hoàn toàn không quan tâm đến loại database sử dụng, SQL…  
  
Peristence layer: một ứng dụng có thể được chia làm 3 phần như sau: giao diện người dùng (presentation layer), phần xử lý nghiệp vụ (business layer) và phần chứa dữ liệu (data layer). Cụ thể ra, business layer có thể được chia nhỏ thành 2 layer con là business logic layer (các tính toán nhằm thỏa mãn yêu cầu người dùng) và persistence layer. Persistence layer chịu trách nhiệm giao tiếp với data layer (thường là một hệ quản trị cơ sở dữ liệu quan hệ – Relational DBMS). Persistence sẽ đảm nhiệm các nhiệm vụ mở kết nối, truy xuất và lưu trữ dữ liệu vào các Relational DBMS.  
  
**3. Giới thiệu Hibernate Framework.**  
  
Hibernate là một trong những ORM Framework. Hibernate framework là một framework cho persistence layer. Như vậy, nhờ có Hibernate framework mà giờ đây khi bạn phát triển ứng dụng bạn chỉ còn chú tâm vào những layer khác mà không phải bận tâm nhiều về persistence layer nữa. Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh. Hibernate cho phép bạn truy vẫn dữ liệu bằng ngôn ngữ SQL mở rộng của Hibernate (HQL) hoặc bằng SQL thuần.

**1. Các khái niệm trong JPA**

Khái niệm JPA bao gồm ba thành phần chính là Entity, EntityManager, và EntityManagerFactory

1. Entity

Entity là các đối tượng persistence thể hiện một mẫu tin trong cơ sở dữ liệu. Entity chỉ là các lớp POJO đơn giản, dễ dàng tạo. Dưới đây là một số đặc điểm của một Entity:

* Entity có thể tương tác với cơ sở dữ liệu quan hệ.
* Entity được xác định thông qua một định danh( tương đương với khóa chình trong table của cơ sở dữ liệu quan hệ)
* Entity hỗ trợ giao tác (transaction)
* Entity hỗ trợ kế thừa giống như những lớp thường khác.

b.  EntityManager

EntityManager là một giao diện (interface) cung cấp các API cho việc tương tác với các Entity. Một số chức năng cơ bản của EntityManager như:

* Persist: phương thức này dùng để lưu một thực thể mới tạo vào cơ sở dữ liệu
* Merge: dùng để cập nhật trạng thái của entity vào cơ sở dữ liệu.
* Remove: xóa một thể hiện của entity.

c.  EntityManagerFactory:

EntityManagerFactory được dùng để tạo ra một thể hiện của EntityManager

**2. Một số lợi ích của JPA.**

Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu)

Không phụ thuộc vào các framework ORM

Có nhiều nhà cung cấp hỗ trợ cài đặt JPA

Dữ liệu có thể được lưu trữ thông qua việc ORM

**- Tạo session**  
Persistence layer có nhiệm vụ tạo các phiên kết nối từ ứng dụng đến Cơ sở dữ liệu, mà qua các phiên này, ứng dụng sẽ tương tác được với DB, các phiên kết nối được thực hiện khi các connections được thiết lập.  
**- Thực thi query**  
Khi các connections được thiết lập, tầng persistence có nhiệm vụ thực thi các truy vấn của ứng dụng lên cơ sở dữ liệu như update, insert và select data, call store procedure... thông qua các phương thức được xây dựng và cung cấp trong persistence layer  
**- Mapping**  
Persistence layer sẽ ánh xạ cấu trúc DB, tables, columns, rows với các class, object của java, và nhờ dó, các java object có nhiệm vụ lưu trữ thông tin của các thực thể trong tables khi truy vấn (SELECT) hoặc lưu giữ thông tin từ người dùng và cập nhật vào DB(INSERT,UPDATE). ví dụ một mapping như sau

**Transaction and Exception**  
Khi truy vấn dữ liệu sẽ xảy ra các trường hợp các truy vấn được thực thi không thành công( truy vấn theo batch, ngắt kết nối....) những rủi ro xảy ra cho ứng dụng và dữ liệu trong DB khi truy vấn thất bại, hoặc connection bị ngắt kết nối thì peristence phải có nhiệm vụ phục hồi tình trạng trước đó của dữ liệu, hoặc đưa ra các ngoại lệ, các hướng dẫn, log để thông báo cho ứng dụng.  
**- Cache**  
Để phục vụ tốt hơn cho việc truy vấn ( hiệu suất, tốc độ, tính tương thích..) thông thường persistence layer sẽ load một metadata của Database vào ứng dụng, các mapping giữa các đối tượng trong ứng dụng với các tables thông qua các config, xml file, hoặc tạo ra các đối tượng, được lưu trữ như cache để có thể truy vấn lại mà không phải kết nối đến DB... Diều này sẽ làm cải thiện tốc độ cho chương trình nhưng đôi khi lại gây ra một sự cồng kềnh, phức tạp.  
Java cung cấp JDBC để xây dựng tầng persistence, các  JDBC API sẽ sử dụng các SQL query được viết chung với code java và thực thi nó thông qua các command, statement... Tuy nhiên những ORM fw sau này được xây dựng đã lược bỏ sự phức tạp của JDBC, thay vào đó, bản thân các ORM sẽ định nghĩa các API cho riêng mình, giúp cho developer khỏi bận tâm hơn việc thao tác DB, mà tập trung vào xây dựng bussiness cho ứng dụng.

[Home](http://javabeat.net/) >> [Hibernate](http://javabeat.net/tutorials/hibernate/) >> JPA EntityManagerFactory Vs Hibernate’s SessionFactory

# JPA EntityManagerFactory Vs Hibernate’s SessionFactory 🡪 open sesion

Để thực hiện được DI, chúng ta có 2 cách:

* Sử dụng Constructor inject
* Sử dụng Setter method..

pring Beans chính là những Java Object mà từ đó tạo nên khung sườn của một Spring application.Chúng được cài đặt, lắp ráp và quản lý bởi Spring IoC container. Những bean này được tạo ra bởi configuration metadata được cung cấp từ container, ví dụ, trong tag <bean> nằm trong file XML.

Các bean được define trong spring framework là singleton bean. Có một thuộc trính trong bean với tên là “singleton” nếu được gán giá trị là true thì bean đó sẽ trở thành singleton, nếu là false thì bean đó sẽ trở thành prototype bean. Mặc định nếu không được định nghĩa giá trị của nó sẽ là true. Vì thế tất cả các bean trong spring framework mặc định sẽ là singleton bean.

**22. Định nghĩa Spring bean gồm những gì?**

Một Spring bean definition chứa tất cả các configuration metadata cái mà cần cung cấp cho container biết làm sao để tạo ra bean, cũng như chi tiết về lifecycle và những depedencies của nó.

**23. Làm sao để cung cấp configuration metadata cho Spring Container?**

Có ba cách để cũng cấp configuration metadata cho Spring container.

* Thông qua XML configuration file
* Thông qua Annotation-based configuration
* Thông qua Java-based configuration

### ****15.Implement Application context phổ biến****

**FileSystemXmlApplicationContext,** chúng ta phải truyền đường dẫn đầy đủ của file XML vào constructor.

**ClassPathXmlApplicationContext,**container sẽ tìm file XML từ trong CLASSPATH.

**WebXmlApplicationContext,**container sẽ tìm file XML từ web application.

**JDBC API (các gói java.sql & javax.sql )**  
· Các kiểu JDBC Driver JDBC API, JDBC API có sẵn trong các gói java.sql và javax.sql. Sau đây là các lớp JDBC, các giao diện và các lỗi ngoại lệ quan trọng trong gói java. sql:

**1. DriverManager** – Nạp các JDBC driver vào trong bộ nhớ. Có thể sử dụng nó để mở các kết nối tới một nguồn dữ liệu.

**2. Connection** – Biểu thị một kết nối đến một nguồn dữ liệu. Được dùng để tạo ra các đối tượng Statement, PreparedStatement và CallableStatement.  
Statement – Biểu diễn một lệnh SQL tĩnh. Có thể sử dụng nó để thu về đối tượng ResultSet.  
**3. PreparedStatement** – Một giải pháp thay thế hoạt động tốt hơn đối tượng Statement, thực thi một câu lệnh SQL đã được biên dịch trước.  
**4. CallableStatement** – biểu diễn một thủ tục được lưu trữ. Có thể được sử dụng để thực thi các thủ tục được lưu trữ trong một RDBMS có hỗ trợ chúng.  
**5. ResultSet** – biểu diễn một tập kết quả trong cơ sở dữ liệu tạo ra bởi việc sử dụng một câu lệnh SQL là SELECT.  
**6. SQLException** – một lớp xử lý lỗi ngoại lệ chứa các lỗi truy cập cơ sở dữ liệu.  
Gói thứ hai, javax.sql là một bộ phận của J2SE 1.4 và J2EE 1.3. Nó bổ sung các tính năng sau đây vào JDBC để hỗ trợ thêm cho các tính năng đã có trong gói java.sql:  
**7. DataSource** – Trừu tượng hóa một nguồn dữ liệu. Đối tượng này có thể sử dụng thế cho 8. DriverManager để tạo ra một cách có hiệu quả các kết nối cơ sở dữ liệu (có khả năng sử dụng việc chứa/phân chia các đường kết nối ngầm).  
Tạo sẵn cơ chế phân chia đường kết nối (built-in connection pooling).  
**9. XADataSource, XAConnection** – Cho phép/Hỗ trợ các giao dịch phân phối.  
RowSet – Nó mở rộng giao diện ResultSet để tăng thêm sự hỗ trợ đối với các tập kết nối bị ngắt.

**EJB viết tắt của Enterprise JavaBeans**  
+ Stateless Session Beans: chứa các nghiệp vụ logic mà public ra bên ngoài cho phép các hệ thống khác có thể truy nhập.  
+ Stateful Session Beans : giống với loại trên nhưng 2 loại bean này khác nhau đúng 1 điểm: nếu giữa các người dùng sử  
dụng đến thuộc tính của Session Beans thì với Stateful Session Beans, lần sử dụng sau vẫn lưu giá trị của lần trước đối  
với người dùng đó còn Stateless Session Bean thì không phân biệt

Tóm lại: nều dùng biến Session như viết web asp thì nó chính là Stateful Session Beans, còn nếu bạn muốn biến global thì nó sẽ là Session Beans.

Nhà**tuyển dụng**: Bạn hãy cho biết ưu điểm của việc sử dụng Hibernate so với JDBC khi tương tác với CSDL quan hệ?  
Ứng viên: Năm ưu điểm nổi bật của việc sử dụng Hibernate so với JDBCkhi tương tác với CSDL quan hệ là  
– Hibernate thao tác trên các đối tượng (object), cấu trúc dữ liệu (data structure, như List, Set, v.v..) nên phù hợp hoàn toàn với lập trình hướng đối tượng trong Java, chứ không thao tác trực tiếp trên các table/bảng của CSDL quan hệ.  
– Hiệu suất cao: Hibernate sử dụng cơ chế cache (đệm) giúp hệ thống hoạt động nhanh hơn. Hai loại cache trong Hibernate là: First level cache (đệm mức thứ nhất), Second level cache (đệm mức thứ hai). First level cache là mặc định, luôn sẵn có. Second level cache cần bật tùy chọn này lên thì sẽ tận dụng được.  
– Truy vấn độc lập với Hệ quản trị cơ sở dữ liệu (Database management system). HQL (Hibernate Query Language: Ngôn ngữ truy vấn Hibernate) cho phép tạo câu truy vấn 1 lần và thực thi trên mọi Hệ quản trị cơ sở dữ liệu quan hệ. Điều đó có nghĩa là viết truy vấn HQL 1 lần, có thể thực thi trên Oracle, MySQL, Microsoft SQLServer hay PostgreSQL v.v..  
– Cho phép tạo bảng tự động: Có thể tạo table dữ liệu tự động, không cần tạo bảng dữ liệu thủ công.  
– Đơn giản các lệnh join phức tạp: Lấy dữ liệu từ nhiều bảng dễ hơn.

**Prototype Scope:** For prototype scope, for each request for the bean, a new instance of the bean will be created and returned. This is similar to calling **new** operator in java for a class.

Prototype scope = A new object is created each time it is injected/looked up. It will use new SomeClass() each time.

Singleton scope = (Default) The same object is returned each time it is injected/looked up. Here it will instantiate one instance of SomeClass and then return it each time.

**28. Phương thức nào là quan trọng nhất trong Spring Bean lifecycle**

Có hai phương thức quan trọng nhất trong Spring bean lifecycle. Đầu tiên đó là setup(), phương thức nay được gọi khi bean được load vào container. Phương thức thứ hai đó là teardown(), phương thức này được gọi khi bean được unload khỏi container.

Tag <bean> có hai thuộc tính quan trọng là **init-method** và **destroy-method**với thuộc tính này, chúng ta có thể khai báo tùy biến phương thức khởi tạo và destroy cho mỗi bean. Ngoài ra nếu bạn sử dụng Annotation-based thì Spring framework cũng cung cấp hai annotation là @PostConstruct và @PreDestroy

**29. Inner bean trong Spring**

Khi mà bean chỉ được sử dụng như một thuộc tính của một bean khác thì nó được gọi là inner bean. XML-based configuration metadata cung cấp cho chúng ta sử dụng tag <bean> bên trong tag <property> hoặc <constructor-arg> để khai báo gọi inner bean. Inner bean luôn luôn là anonymous và scope của chúng luôn là prototype.

**30. Làm sao để inject Java Collection trong Spring?**

Spring đưa ra cho chúng ta các loại collection như sau:

* <list> dùng để inject list, có thể có các phần tử trùng nhau.
* <set> dùng để inject set, các phần tử không trùng nhau.
* <map> dùng để inject collection dạng key-value, trong đó key và value là loại tùy ý.
* <props> dùng để inject collection dạng key-value, trong đó key và value đều là String

Để bạn đọc hiểu được các trình tự của các sự kiện xảy ra khi một yêu cầu nào đó của người dùng gửi đến thì nó sẽ xử lý như thế nào,mời các bạn làm theo trình tự bên dưới :

1. Các DispatcherServlet cấu hình trong file web.xml nhận được yêu cầu.
2. DispatcherServlet sẽ tìm đến controller thích hợp dựa vào sự trợ giúp của HandlerMapping và sau đó nó sẽ gọi đến controller liên kết.(**Handler Mapping** nhằm xác định đúng **Controller** nào xử lý yêu cầu này).
3. Sau đó controller này sẽ thực thi business logic (việc xử lý trong controller này là do chúng ta viết code) và sau đó sẽ trả về ModelAndView object cho DispatcherServlet.
4. DispatcherServlet sẽ xác định view từ ModelAndView object
5. Sau đó DispatcherServlet thông qua mô hình object để gửi cho View
6. View đã nhận được kết quả thì Dispatcher Servlet sẽ gửi đầu ra đến cho Servlet container và cuối cùng Servlet container sẽ gửi kết quả cho người dùng.

Dưới đây là giải thích về một url được ánh xạ khi một khách hàng truy cập URL trên máy chủ :

1. Spring Font Controller (DispatcherServlet) nhận yêu cầu nó sẽ tìm đến Handler Mapping thích hợp
2. Handler Mapping sẽ ánh xạ yêu cầu của client đến controller thích hợp.
3. Sau đó DispatcherServlet sẽ gửi yêu cầu đến Controller thích hợp.
4. Sau khi thực hiện tiến trình từ yêu cầu của client, nó thực thi các logic được xác định trong controller và cuối cùng trả về đối tượng ModelAndView.
5. Dựa trên các giá trị trong ModelAndView Controller mà nó sẽ view, có thể là JSP, FreeMaker,tiles do file cấu hình của chúng ta config.
6. Controller sẽ gửi đến cho Servlet container và cuối cùng Servlet container sẽ gửi kết quả cho người dùng

**48. Spring hỗ trợ những kiểu quản lý transaction nào?**

Spring hỗ trợ hai kiểu quản lý transaction:

* **Programmatic transaction management:** Có nghĩa là bạn có thể quản lý transaction với việc lập trình. Điều này khiến mọi việc trở nên vô cùng tường minh và linh hoạt, tuy nhiên rất khó để maintain.
* **Declarative transaction management:** Có nghĩa là bạn phải tách biệt phần quản lý transaction với business code. Chỉ sử dụng annotaion hoặc XML config để manage transaction.

**49. Lợi ích của việc quản lý transaction bằng Spring Framework**

* Cung cấp một phương thức lập trình thích hợp cho việc sử dụng các transaction API khác nhau như JDBC, Hibernate, JPA.
* Cung cấp API đơn giản hơn.
* Tiện lợi trong việc khai báo và sử dụng với **Declarative transaction management**.
* Tích hợp dễ dàng với các data access abstraction.

**50. Nên sử dụng kiểu quản lý transaction như thế nào?**

Phần lớn những người sử dụng Spring framework đều chọn sử dụng **Declarative transaction management**bởi vì nó giúp giảm thiểu việc tác động tới application code.

* Nếu chúng ta implements interface Runnable, chúng ta vẫn có thể extends 1 class khác, nếu chúng ta extends Thread class chúng ta sẽ không thể extends thêm bất kì class nào vì Java không hỗ trợ đa kế thừa
* Khi một class implements Runnable, ta nói class có khả năng “Runnable” hay còn gọi là mối quan hệ has – a, class extends Thread ta gọi class đó là “Thread” hay còn gọi mối quan hệ is – a. Trong thiết kế hướng đối tượng, thường dùng kế thừa has – a vì giúp giảm sự phụ thuộc code, code sẽ dễ test và maintain hơn.
* Extends Thread có nghĩa là class chúng ta sẽ được kế thừa tất cả các method của Thread class, đôi khi không cần thiết, gây lãng phí bộ nhớ khi tạo thread.

*Vậy khi nào ta tạo thread bằng cách extends Thread class?*

Chỉ khi nào ta cần override và sử dụng các method của Thread class. Các trường hợp chỉ cần tạo thread để chạy, ta nên implements Runnable.

New − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.  
Runnable − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.  
  
Waiting − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.  
Timed Waiting − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.  
Terminated (Dead) − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Một Thread sẽ có các state (trạng thái dưới đây):

NEW: The thread has been created, but has never been started.  
RUNNABLE: The thread may be run.  
BLOCKED: The thread is blocked and waiting for a lock.  
WAITING: The thread is waiting.  
TIMED\_WAITING: The thread is waiting for a specified amount of time.  
TERMINATED: The thread has been terminat

Như đã đề cập ở trên thì các thread tạo ra sẽ dùng chung một vùng nhớ. Nếu các process này cùng truy cập vào một vùng nhớ nào đó tại cùng một thời điểm thì dẫn đến sai, mất dữ liệu.

Để khắc phục điều này thì Java cung cấp cho chúng ta synchronized để đồng bộ các thread khi sử dụng chung vùng nhớ chia sẻ (shared memory).

Mội khối synchronized đánh dấu một phương thức hay một khối mã được đồng bộ tránh xung đột giữa các thread.

Khi một thread can thiệt vào phương thức hay khối mã được đánh dấu là synchronized thì thread này sẽ khóa (**lock**) không cho các thread khác can thiệp vào cho đến khi thread này thực hiện xong thì mới đánh thức các thread khác. Và như vậy tại một thời điểm chỉ có một thread truy cập vào vùng nhớ được chia sẻ.

Có hai phương thức quan trọng nhất trong Spring bean lifecycle. Đầu tiên đó là setup(), phương thức nay được gọi khi bean được load vào container. Phương thức thứ hai đó là teardown(), phương thức này được gọi khi bean được unload khỏi container.

Tag <bean> có hai thuộc tính quan trọng là **init-method** và **destroy-method**với thuộc tính này, chúng ta có thể khai báo tùy biến phương thức khởi tạo và destroy cho mỗi bean. Ngoài ra nếu bạn sử dụng Annotation-based thì Spring framework cũng cung cấp hai annotation là @PostConstruct và @PreDestroy