Spring Framework

One stop shop for building enterprise applications

Team Emertxe

Spring Core

Objectives of Spring Framework

- Introduction to Spring 3.0
- Steps to use Spring Framework in applications
- Understanding Dependency Injection and IOC
- Understanding the bean life-cycle – Auto wiring and bean scopes
- Annotation-based dependency injection
- Aspect Oriented Programming
- Introducing data access with Spring – JDBC through spring
- Transactions in a Spring environment

Spring

Java Application Framework

What is Spring

What is Spring

- It is a light weight solution and a potential one-step-shop for building your enterprise -ready application.
- It is a open source Java platform which is developed by Rod Johnson and first released under Apache 2.0 licence in June 2003.
- Spring framework targets to make J2EE development easier to use and promote good programming practice by enabling a POJO-based programming model.
- It supports declarative transaction management, remote access to your logic through RMI or web services.
- It offers full featured MVC framework and enables you to integrate AOP(Aspect Oriented Programming) transparently in to your software.

Features of the Spring Framework

- IoC container : It contains the assembler code that handles the configuration management of application objects.
- Data access framework : Allows the developer to use persistence APIs, such as JDBC and Hibernate for storing persistence data in database.
- Spring MVC Framework : Allows you to build web application based on MVC architecture.

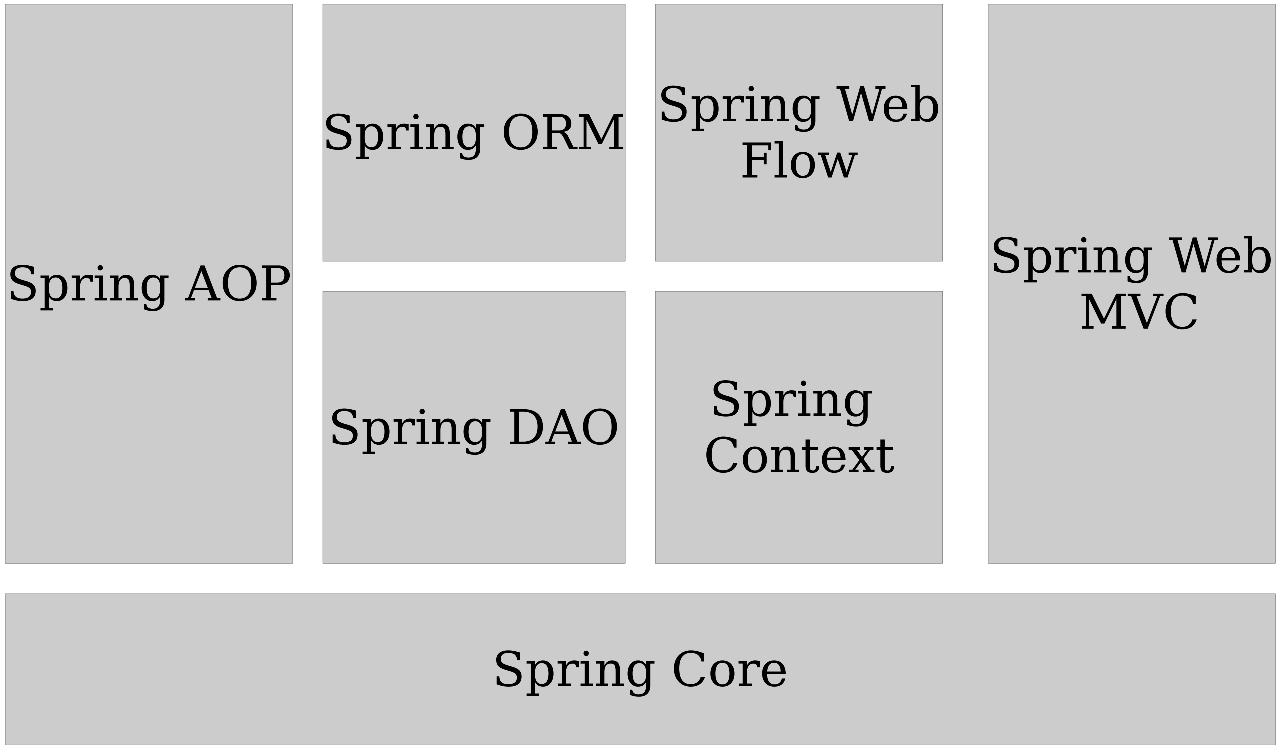
Features of the Spring Framework

- Transaction management : It helps in handling transaction management of an application without affecting its code. This framework uses Java Transaction API (JTA) for global transactions managed by an application server.

- Spring TestContext framework : It provides the facility of unit and integration testing for the spring application.

Spring Modules

Spring Modules



IOC or Dependency Injection

- The basic concept of IOC (Dependency of Injection) is that you do not create your objects but describe how they should be created.
- You don’t directly connect your component and services together in code but describe which services are needed by which component in configuration file.
- You just need to describe the dependency, the Spring container is then responsible for hooking it all up.

Environmental set up Of Spring Framework

- ✓ Download and install jdk1.6 .
- ✓ Download and Install Apache Common logging API from http://commons.apache.org/logging
- ✓ Download spring-framework-3.1.0.m2.tz(for unix) , antlr-runtime -3.0.1.jar from http://WWW Springsource.org

Building Spring Application

Example using Eclipse

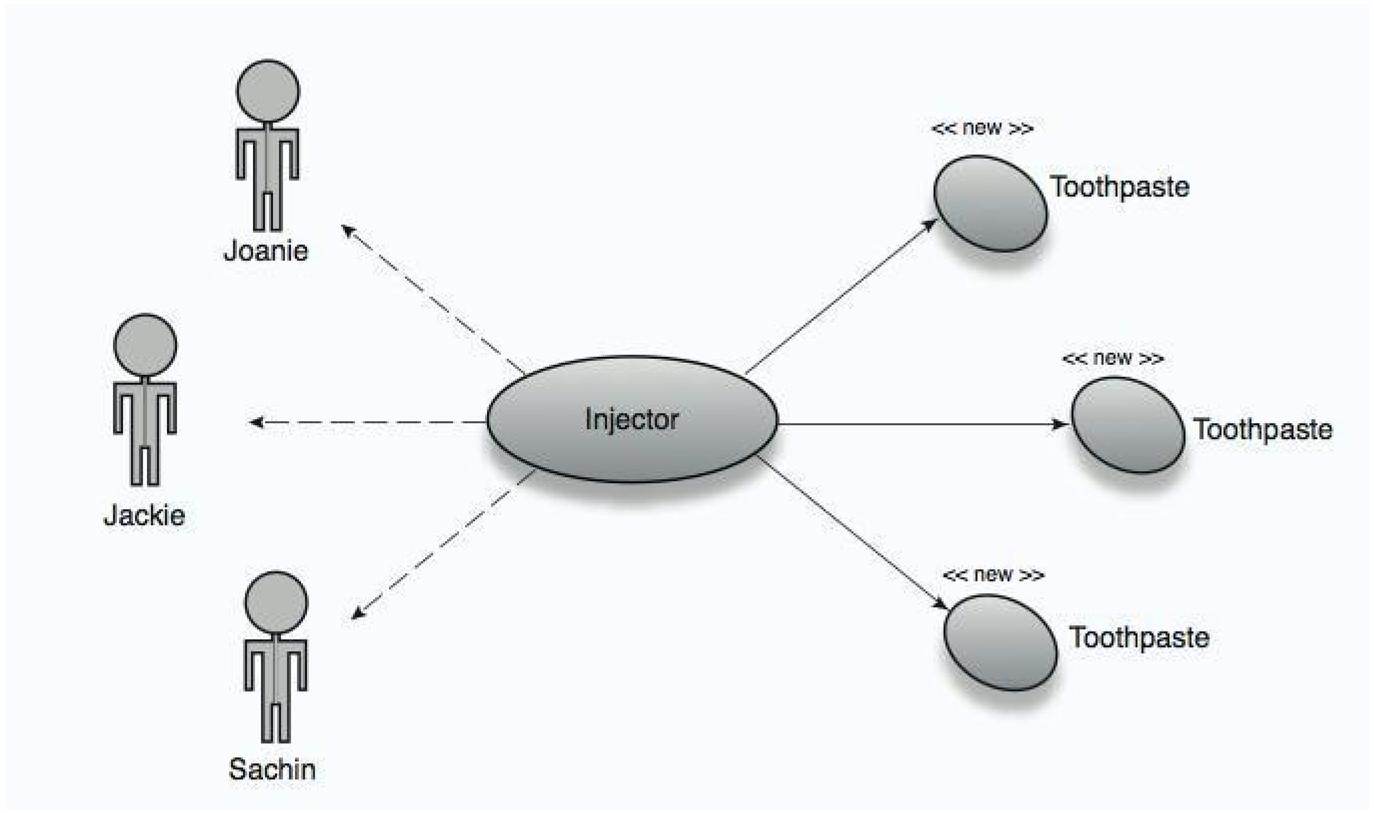
- ✓ Create java Project and give any name
- ✓ Right click on jre of project and select Build path->Configure Build Path->Libraries->Add External Jars-> and select spring -framework-3.1.0 and antlr-runtime -3.0.1.jar files.
- ✓ Next right click on src and create new java classes
- ✓ Right click on src and create Bean Configuration file
- ✓ Finally run the project by ctrl+f11.

Questions

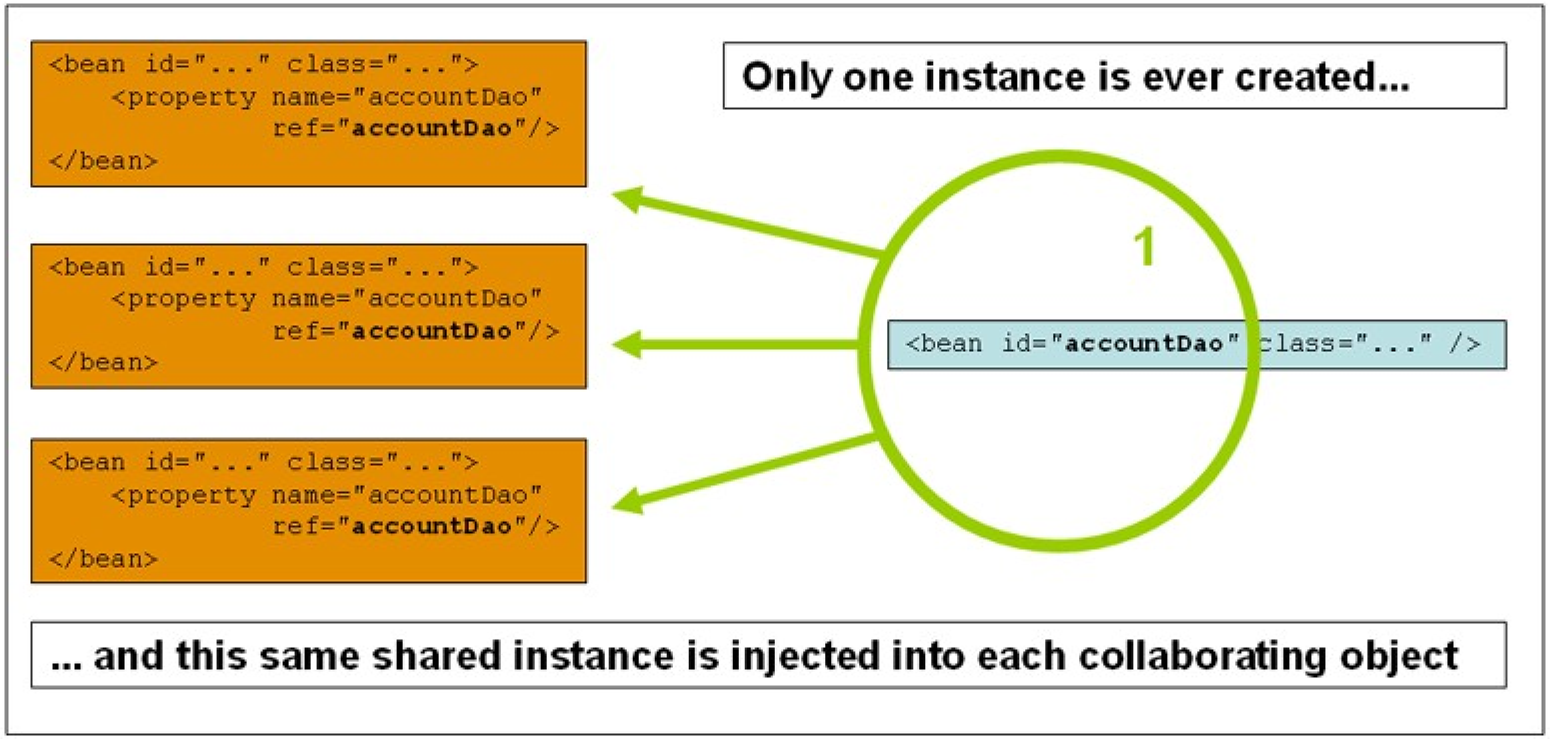
- What is Spring ?
- What are features of spring?
- How many modules are there in spring? What are they?

Dependency Injection

Dependency Injection



Dependency Injection



Only one instance is ever created...

```
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

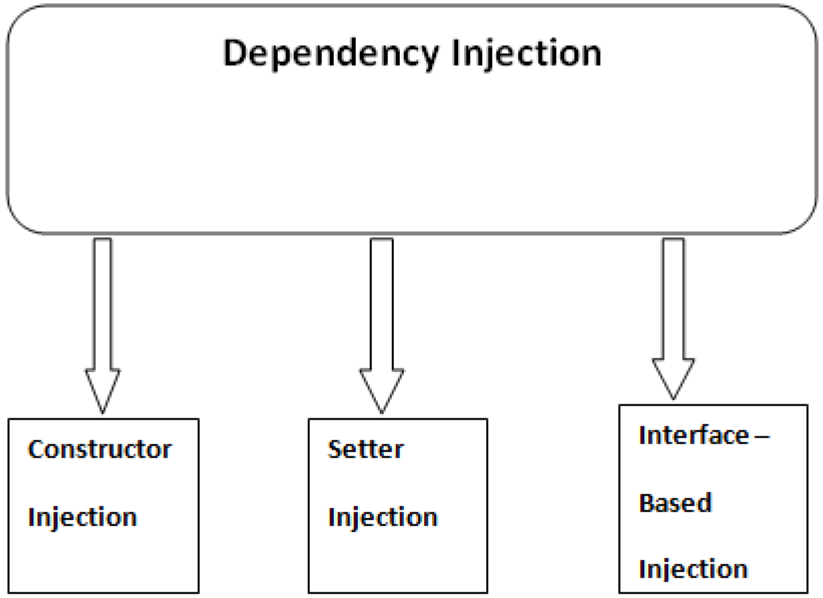
```
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

```
<bean id="..." class="...">
    <property name="accountDao"
              ref="accountDao"/>
</bean>
```

```
<bean id="accountDao" class="..." />
```

... and this same shared instance is injected into each collaborating object

Dependency Injection



Dependency Injection

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

- There are two ways to perform dependency injection in Spring Framework.

    - By Constructor : The <constructor-arg> subelement of <bean> is used for constructor injection.

```
<bean id="e" class="com.Emertxe.Employee">
<constructor-arg value="10" type="int"></constructor-arg>
<constructor-arg value="park"></constructor-arg>
</bean>
```

Dependency Injection

Constructor injection in Collection

```
<bean id="S" class="com.javatpoint.Study">

<constructor-arg value="1"></constructor-arg>
<constructor-arg value="What is spring?"></constructor-arg>
<constructor-arg>

<list>
<value>Spring is a frame work</value>
<value>Spring is a platform</value>
<value>Spring is an Island</value>
</list>

</constructor-arg>
</bean>
```

Dependency Injection

- By Setter methods : The <property> subelement of <bean> is used for setter injection.

```
<bean id="ex" class="com.emertxe.Employee">
<property name="id">
<value>1</value>
</property>
<property name="name">
<value>Lee</value>
</property>
<property name="city">
<value>Korea</value>
</property>
 </bean>
```

Constructor Injection with Dependent Object

If there is HAS-A relationship between the classes, we create the instance of dependent object (contained object) first then pass it as an argument of the main class constructor. Here, our scenario is Employee HAS-A Address. The Address class object will be termed as the dependent object.

Dependency Injection

Setter injection with collection

```
<bean id="ex" class="com.emertxe.Study">

<property name="id" value="1"></property>
<property name="name" value="What is Spring?"></property>
<property name="answers">

<list>
<value>Spring is a frame work</value>
<value>Spring is a platform</value>
<value>Spring is an Island</value>
</list>

</property>
</bean>
```

Annotation Based Dependency Injection

Annotation Based DI

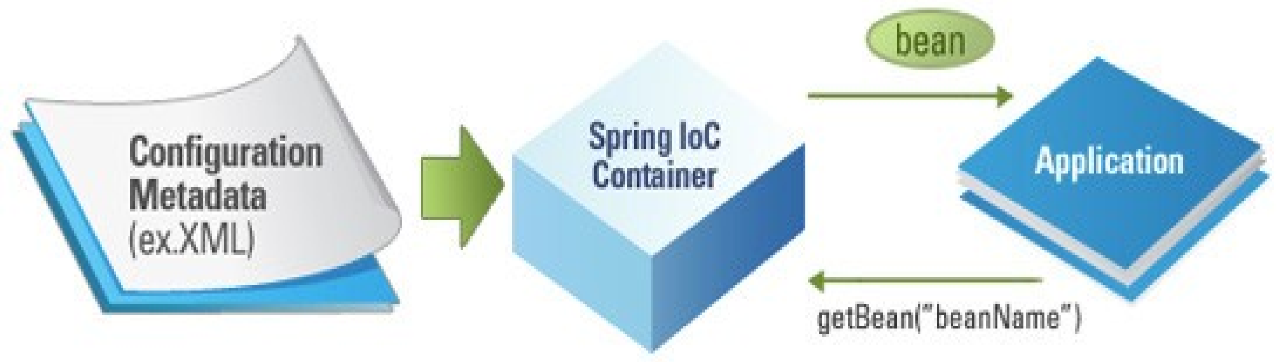
- Instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

- We need to enable annotation auto wiring in our spring configuration file to use annotation wiring.

Annotation Based DI

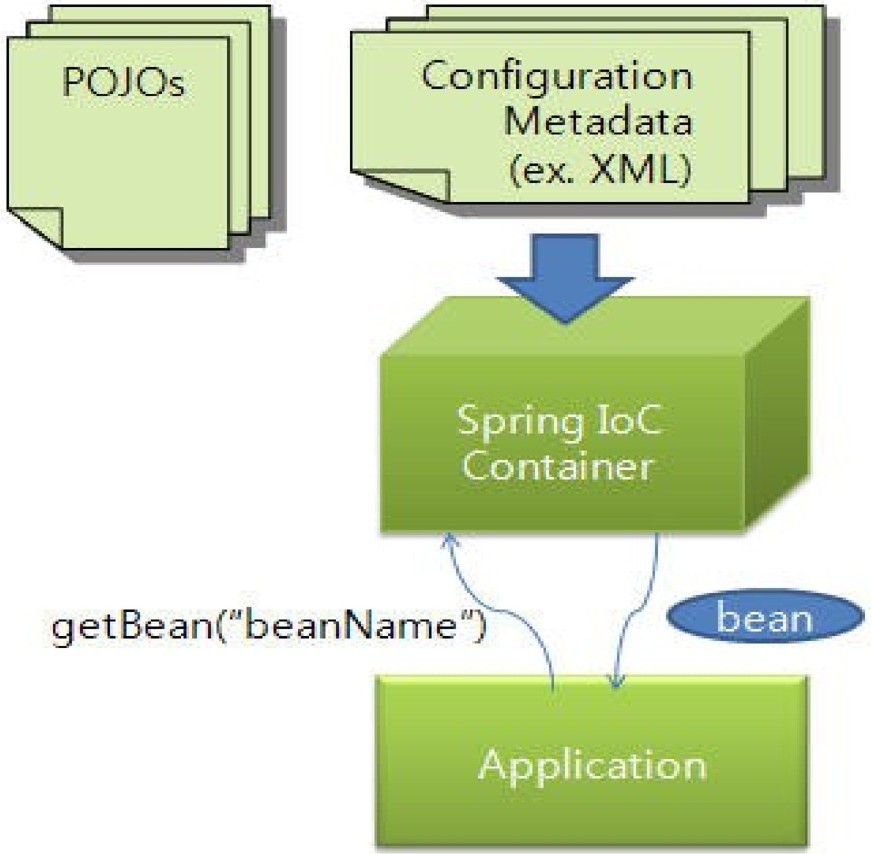
Annotations :

- @Required : This applies to bean property setter methods.
- @Autowiried :This applies to bean property setter methods, non-setter methods, constructors and properties.
- @Qualifier : The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
- JSR-250 Annotations : It includes @Resource, @PostConstruct and @PreDestroy annotations.



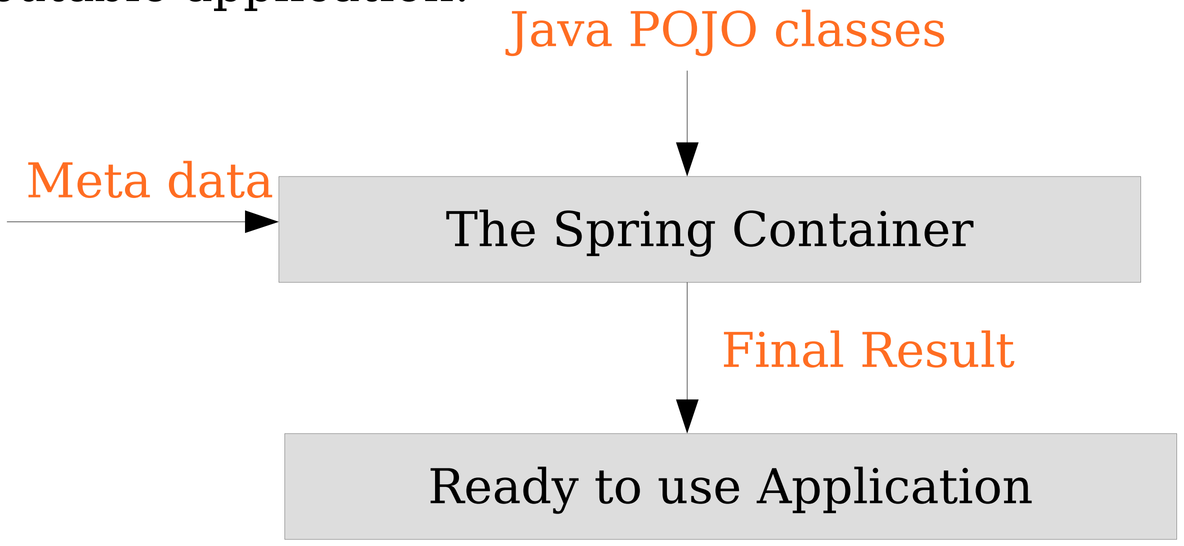
IOC Container

IOC Containers



IOC Containers

- It will create objects, wire them together, configure them, and manage their complete life cycle from creation till destruction.
- It uses DI to manage the components that will make up an application.
- The Spring IoC Container makes use of Java POJO classes and configuration meta data to produce fully configured and executable application.



Types of IOC Containers

1. BeanFactory
2. ApplicationContext

The BeanFactory and the ApplicationContext interfaces acts as the IoC container.

Application Context and BeanFactory

The ApplicationContext interface is subinterface of BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling.

BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality.

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.

Questions

- What are the types of Dependency Injection Spring supports?
- What are benifits of IoC?
- Explain DI with setter methods?

Bean Life Cycle

Bean Life Cycle

- A bean is an object that is instantiated, assembled and otherwise managed by Spring IoC Containers.

- The two important methods of spring bean lifecycle are

  - init-method : It specifies a method that is to be called on the bean immediately upon instantiation.

```
<bean id="sample"
        class="samples.SampleBean" init-method="init"/>
```

  - destroy-method : It specifies a method that is called just before a bean is removed from the container.

```
bean id="sample"  class="samples.SampleBean"
                  destroy-method="destroy"/>
```

AUTOWIRING

Auto Wiring

Spring Beans Auto Wiring

- The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements.
- It helps cut down on the amount of XML configuration you write for a big Spring based application.

```
<bean id="stu" class="com.emertxe.Student"
               autowire="byName" />
```

Autowiring modes

- no : this means there is no auto-wiring and you should use explicit bean reference for wiring.
- byName : If the name of a bean is same as the name of other bean property, auto wire it.

Spring Beans Auto Wiring

- byType : Auto wiring by property data type. If data type of a bean is compatible with the data type of other bean property, auto wire it.

- constructor : It is similar to byType mode,but type applies to constructor argument.

- autodetect : If a default constructor is found, use “autowired by constructor”. Otherwise, use “autowire by type”.

Limitations of Autowiring :

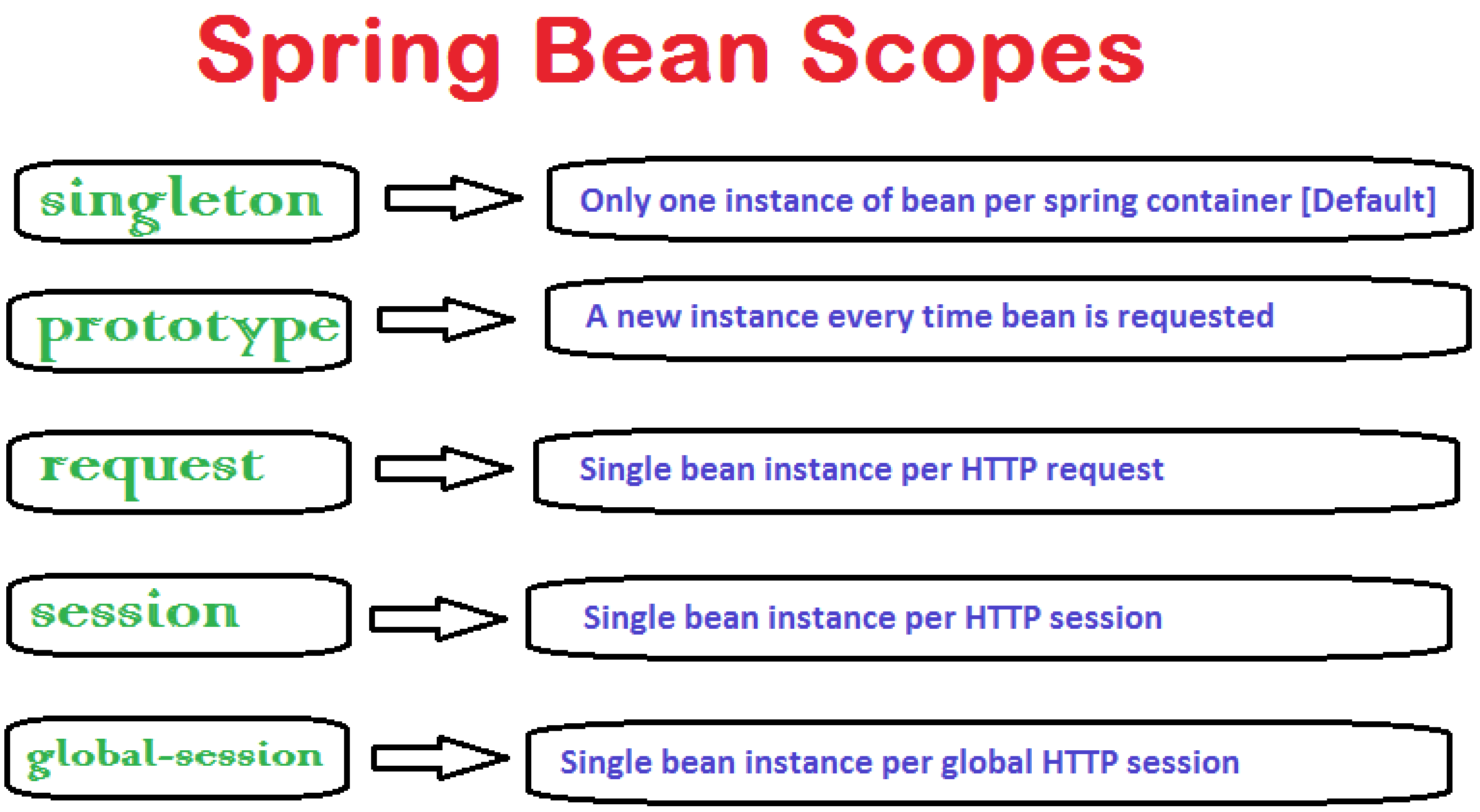
- There is a possibility to override autowiring.

- We cannot use primitive data types.

- It has confusing nature.

Bean Scopes

Bean Scope

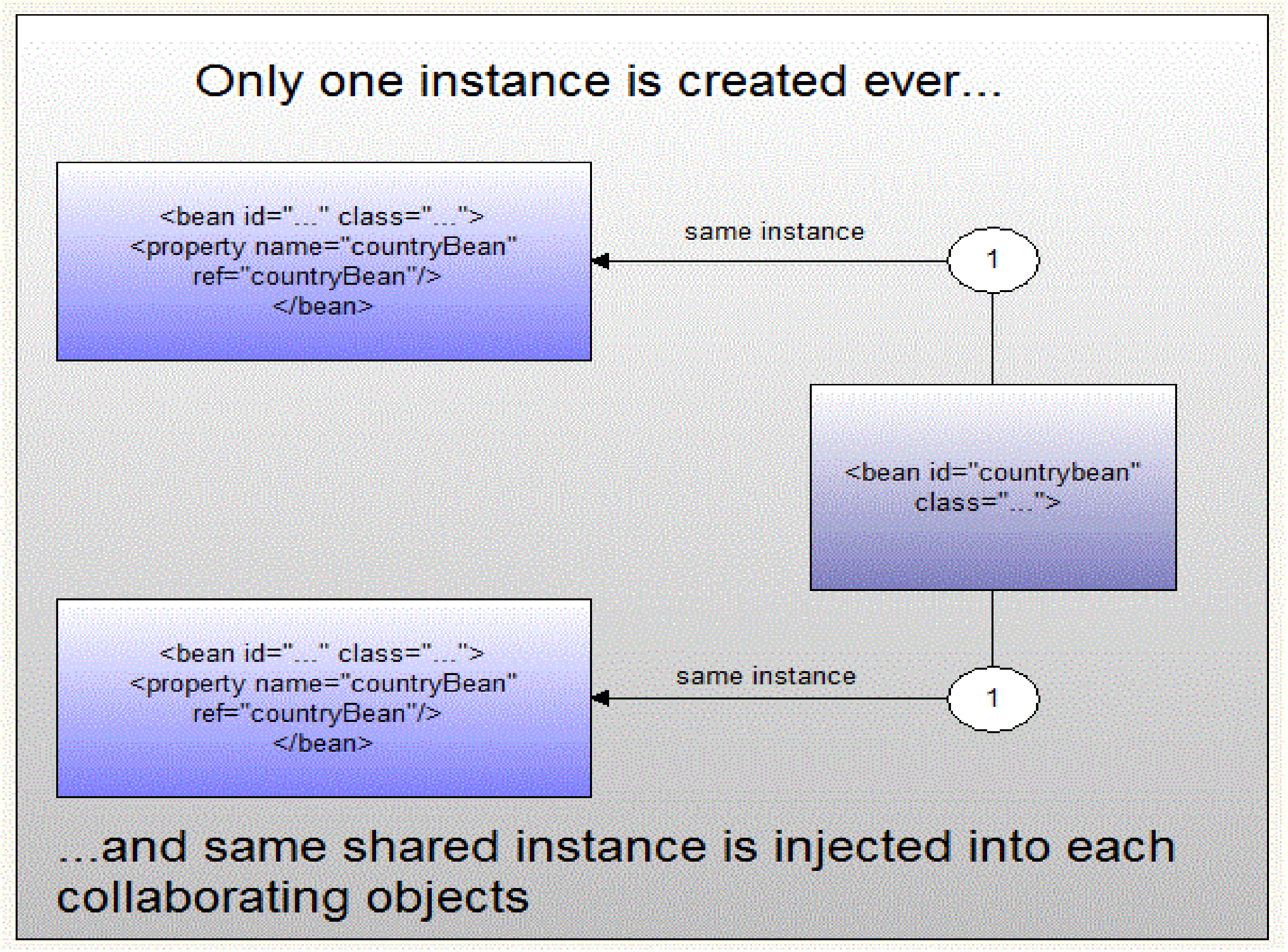


Spring Bean Scopes

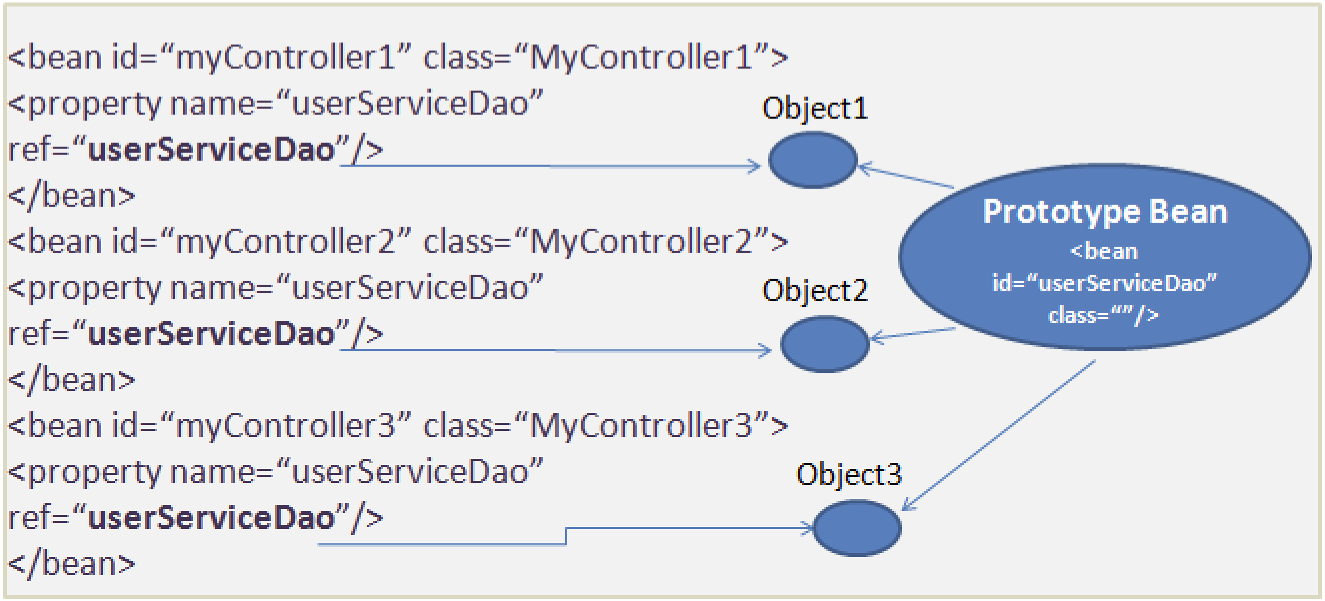
singleton	Only one instance of bean per spring container [Default]
prototype	A new instance every time bean is requested
request	Single bean instance per HTTP request
session	Single bean instance per HTTP session
global-session	Single bean instance per global HTTP session

Note: request, session, global-session are only valid in the context of a web-aware Spring ApplicationContext.

Bean Scope: Singleton


Only one instance is created ever...
<bean id="..." class="...">
<property name="countryBean"
ref="countryBean"/>
</bean>
same instance
1
<bean id="countrybean"
class="...">
<bean id="..." class="...">
<property name="countryBean"
ref="countryBean"/>
</bean>
same instance
1
...and same shared instance is injected into each collaborating objects

Bean Scope: Prototype



```
<bean id="myController1" class="MyController1">
<property name="userServiceDao"
ref="userServiceDao"/>
</bean>
<bean id="myController2" class="MyController2">
<property name="userServiceDao"
ref="userServiceDao"/>
</bean>
<bean id="myController3" class="MyController3">
<property name="userServiceDao"
ref="userServiceDao"/>
</bean>
```

Bean Scope

- Bean definition with singleton scope.

```
<bean id="..." class="..." scope="singleton">
  ------------------------------------------------------------
</bean>
```

- Bean Definition with prototype scope.

```
<bean id="..." class="..." scope="prototype">
  ------------------------------------------------------------
</bean>
```

Questions

- What is Bean Factory?
- What are the limitaions of auto-wiring?
- What is default scope of bean in Spring framework?
- Are Singleton beans thread safe in Spring Framework?
- Can you inject null and empty string values in Spring?

Spring AOP

Aspect Oriented Programming

Aspect oriented programming complements object oriented programming by providing another way of programming model structure.

Aspect Oriented Programming(AOP)

- AOP is one of the key component of Spring framework.
- It breaks down the program logic into distinct parts called concerns. It is used to increase modularity by cross-cutting concerns.
- The functions that span multiple points of an application are called cross-cutting concerns.
- AOP helps you decouple cross-cutting concerns from the objects that they affect.
- AOP is like triggers in programming languages such as Java, .NET and others.
- The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.

AOP Concepts

- Join point
- Advice
- Pointcut
- Introduction
- Target Object
- Aspect
- AOP Proxy
- Weaving

AOP Concepts

- Join point : It is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

- Advice : It represents an action taken by an aspect at a particular join point. The different types of advices are given below.

AOP Concepts

- Before Advice: it executes before a join point.
- After Returning Advice: it executes after a joint point completes normally.
- After Throwing Advice: it executes if method exits by throwing an exception.
- After (finally) Advice: it executes after a join point regardless of join point exit whether normally or exceptional return.
- Around Advice: It executes before and after a join point.

AOP Concepts

- Pointcut : This is set of one or more join points where an advice should be executed.
- Introduction : It allows us to add new methods or attributes to existing classes.
- Target Object : It is advised by one or more aspects It is also known as proxied object in spring because Spring AOP is implemented using runtime proxies.
- Aspect : It is a class that contain advices and joinpoints.
- AOP Proxy : It is used to implement aspect contracts, created by AOP framework.
- Weaving : It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime.

Spring AOP

Spring AOP is implemented in 3 ways.

- Spring 1.2 old-style(DTD).
- AspectJ annotation-style.
- Spring XML configuration-style.

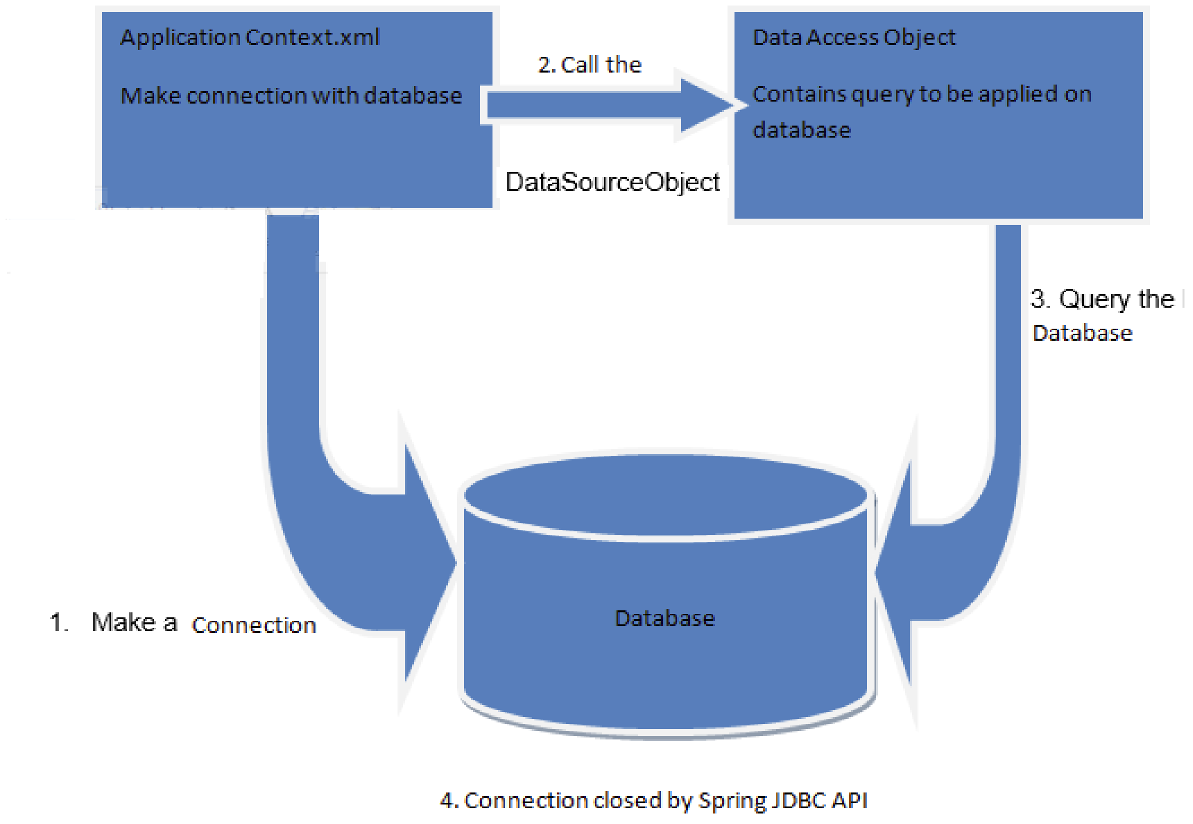
AspectJ annotation-style approach is most widely used.

Questions

- How do you turn on annotation wiring?
- What does @Required annotation mean?
- What are the JSR-250 Annotations?
- What is the difference between concern and cross-cutting concern in Spring AOP?
- What are the different points where weaving can be applied?

Spring JDBC

Data Access Using Spring



Data Access Using Spring

- When we access data from database using JDBC, we have to write code to handle exceptions, opening the connection and closing the connection.

- But with Spring Framework it takes care of all the low-level details starting from opening the connection, prepare and execute the SQL statement etc.

Data Access Using Spring

Spring framework provides following approaches for JDBC database access :

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert and SimpleJdbcCall

JDBC Template Class

- This class is the central class of Spring JDBC support classes which takes the responsibilty of creating, closing connection objects.
- It handles the exceptions by the help of exception classes defined in the org.springframework.dao package.
- Jdbc Template class can perform all Jdbc operations such as insertion, deletion, updation and retrieval of data from database.
- It internally uses JDBC api, but eliminates a lot of problems of JDBC API.
- It provides you methods to write the queries directly, so it saves a lot of work and time.

JDBC Template Class Operations

```
// Create one property for Jdbc template

public void setJdbcTemplate(JdbcTemplate jdbcTemplate)
{
    this.jdbcTemplate = jdbcTemplate;
}



//inserting data in to student table

public int saveStudent(Student s)
{
    String query="insert into student values(
                        '"+e.getId()+"','"+e.getName()"')";

    return jdbcTemplate.update(query);
}
```

JDBC Template Class Operations

```
//updating data to student table

public int updateStudent(Student e)
{
    String query="update student set
    name='"+e.getName()+"' where id='"+e.getId()+"' ";

    return jdbcTemplate.update(query);
}

//deleting data from student table

public int deleteStudent(Student e)
{
    String query="delete from Student where id='"+e.getId()+"' ";

    return jdbcTemplate.update(query);
}
```

Prepared Statement in Spring JDBC Template

- The execute() method of Jdbc template class executes the parameterized queries.

- To use parameterized query, we pass the instance of PreparedStatementCallback in the execute method.

```
public T execute(String sql,PreparedStatementCallback<T>);
```

- PreparedStatementCallback has only one method doInPreparedStatement.

```
public T doInPreparedStatement(PreparedStatement ps)
throws SQLException, DataAccessException
```

Prepared Statement in Spring JDBC Template

```
public Boolean saveStudentByPreparedStatement(final Student s)
{
  String query="insert into student values(?,?)";

    return jdbcTemplate.execute(query,
    new PreparedStatementCallback<Boolean>()
    {
          @Override
          public Boolean doInPreparedStatement(PreparedStatement ps)
            throws SQLException, DataAccessException
           {
                ps.setInt(1,s.getId());
                ps.setString(2,s.getName());

                return ps.execute();
            }
    });
}
```

Retrieve Records By Spring JDBCTemplate

- The query() method of Jdbc template class is used to fetch the records from database.
- To fetch records, we pass the instance of ResultSetExtractor in the query() method.

```
public T query(String sql,ResultSetExtractor<T> rse)
```

- ResultSetExtractor has only one method extractData that accepts ResultSet instance as a parameter.

```
public T extractData(ResultSet rs)throws
 SQLException,DataAccessException
```

Retrieve Records By Spring JDBC Template

```
public List<Student> getAllStudents()
{
       return template.query("select * from student",
       new ResultSetExtractor<List<Student>>()
    {
      @Override
       public List<Student> extractData(ResultSet rs) throws
       SQLException, DataAccessException
      {
         List<Student> list=new ArrayList<Student>();
         while(rs.next()){
         Student s=new Student();
         s.setId(rs.getInt(1));
         s.setName(rs.getString(2));
         list.add(s);
       }
          return list;
      }
    });
  }
```

Fetch Records By Spring Jdbc Template using RowMapper

- The query() method of Jdbc template class is used to fetch the records from database.

- To fetch records, we pass the instance of RowMapper in the query() method.

```
public T query(String sql,RowMapper<T> rm)
```

- RowMapper interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection.

- Advantage of RowMapper over ResultSetExtractor.

RowMapper reduces the burden of written lot of code becuase it internally adds the data of ResultSet into the collection.

Fetch records by Spring Jdbc Template using RowMapper

```
public List<Student> getAllStudentsRowMapper()
{

    return template.query("select * from student",
   new RowMapper<Student>()
   {
     @Override
      public Student mapRow(ResultSet rs, int rownum) throws
SQLException
      {
            Student s=new Student();
            s.setId(rs.getInt(1));
            s.setName(rs.getString(2));

            return s;
      }
   });
}
```

NamedParameter JDBC Template

- This is another way to insert the data into the database. Here we use name instead of **?**. So, it is better to remember the data for the columns.

```
insert into student values(:id, :name, :cellno);
```

- execute method of NamedParameterJdbcTemplate class

```
pubic T execute(String sql,Map map,PreparedStatementCallback psc)
```

NamedParameter JdbcTemplate

```
public  void save (Stu s)
{
    String query="insert into student values (:id,:name)";

    Map<String,Object> map=new HashMap<String,Object>();
    map.put("id",s.getId());
    map.put("name",s.getName());

    template.execute(query,map,new PreparedStatementCallback()
    {
        @Override
        public Object doInPreparedStatement(PreparedStatement ps)
        throws SQLException, DataAccessException
         {
            return ps.executeUpdate();
        }
    });
}
```

SimpleJdbcTemplate

- It wraps the Jdbc template class and provides update method where we can pass random number of argments.

  Syntax:

```
int update(String sql, object.....parameters)
```

- Create a table with name student(id, name) in database.

- Create a class Student with constructor, setter and getter methods.

Simple JDBC Template

```
public class StuDao
{
     SimpleJdbcTemplate template;

     public StuDao(SimpleJdbcTemplate template)
     {
          this.template = template;
     }
     public int update (Stu s)
     {
          String query="update student set name=? where id=?";
          return template.update(query,s.getName(),s.getId());
     }
}
```

Questions

- How JDBC can be used more efficiently in spring framework?
- What is the advantage of RowMapper over ResultSetExtractor?
- Name the method present in PreparedStatementCallback interface?

Spring Transaction

Why Transaction

- To ensure data integrity and consistency.
- The transaction management can be described by following ACID properties.
  - Atomicity
  - Consistency
  - Isolation
  - Durability

ACID Properties

- Atomicity : The Atomicity of a transaction ensures that either the entire sequence of operations are successfull or unsuccessfull.
- Consistenct : Data and resources will be in consistent state that confirms to bussiness rules.
- Isolation: There may be many transactions processing with the same data set at the same time, each transaction should be isolated from others to prevent data corruption.
- Durability: Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

Types of Transaction Management

According to Spring Framework there are two types of transaction management.

- Programmatic transaction management: Here transactions are managed with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.

- Declarative transaction management: Here transactions are managed with the help of annotaions or XML based configuration.This means you separate transaction management from the business code.

Transaction Management Configuration

- JDBC

```
<bean id="transactionManager"class="org.springframework.jdbc.
datasource.DataSourceTransactionManager">
<propertyname="dataSource"ref="dataSource"/>
 </bean>
```

- Hibernate

```
<bean id="transactionManager" class="org.springframework.
orm.hibernate3.HibernateTransactionManager">
<propertyname="sessionFactory"ref="sessionFactory"/>
```

Transaction Attribute-Propagation

- It tells the transaction manager if a new transaction should be started or can use the transaction which is already existed.

Types of Propagation

- PROPAGATION_MANDATORY : It throw an exception if no current transaction exists.

- PROPAGATION_NESTED : If a current transaction exists, it executes with a nested transactions.

- PROPAGATION_NEVER: It throw an exception if a current transaction exists.

Transaction Attribute-Propagation

- PROPAGATION_REQUIRED : It creates a new Transaction if it does not exist.
- PROPAGATION_REQUIRES_NEW : It suspends the current transaction and creates new one.
- PROPAGATION_SUPPORT : It supports current transaction.
- PROPAGATION_NOT_SUPPORTED : It doesnot support current transaction.

Questions

- What are the types of the transaction management Spring supports?
- Name the ACID properties?
- What is the use of PROPAGATION_REQUIRED ?

Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046
T: +91 80 6562 9666
E: tr

https://www.facebook.com/Emertxe

https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

Thank You