**Spring:**

Spring container takes two inputs like Spring Bean and Configuration File. After Reading them, Spring container instantiates and assigns the values.

1. By default it creates singleton Objects in Container.
2. To read the Object use getBean(String beanName):Object method.

Step 1) Create New Java Project and Add build path (Spring JARS and +Commons-logging JAR)

Spring Download Location:

<https://repo.spring.io/release/org/springframework/spring/3.2.5.RELEASE/spring-framework-3.2.5.RELEASE-dist.zip>

(Extract and Find JARS in dist folder)

Commons-Logging JAR:

<http://central.maven.org/maven2/commons-logging/commons-logging/1.1.1/commons-logging-1.1.1.jar>

>> Click on File->New -> Java Project.

>> Provide Any Sample Name (ex: SpringCore) click on Finish Button

>> Right Click on Project -> Choose Build Path -> Configure buildPath-> Libraries Tab->Add External JARS (Select above downloaded JARS). Even add Commons-logging jar.

Step 2) Right Click on src folder -> New Class-> Enter package and Class name:

Ex:

**package** com.app;

**public** **class** Employee {

**private** **int** empId;

**private** String empName;

**private** **double** empSal;

**public** Employee() {

**super**();

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** **double** getEmpSal() {

**return** empSal;

}

**public** **void** setEmpSal(**double** empSal) {

**this**.empSal = empSal;

}

@Override

**public** String toString() {

**return** "Employee [empId=" + empId + ", empName=" + empName

+ ", empSal=" + empSal + "]";

}

}

Step 3) Create A Spring configuration file to configure the Spring Bean Class.

Right Click on src->new->other->type XML-> select XML File-> Next-> Enter any name.xml (ex: config.xml)->Finish

Double click on file to open and goto source tab.

And provide configuration for Employee.java

Ex:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

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"*>

<bean class=*"com.app.Employee"* name=*"emp"*>

<property name=*"empId"*>

<value>10</value>

</property>

<property name=*"empName"*>

<value>ABCD</value>

</property>

<property name=*"empSal"*>

<value>732.36</value>

</property>

</bean>

</beans>

Step 4) Create a new Test Class to Access the above Object.

Right Click on src->new->class Enter any name (Ex:Main.java)

package com.app;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.app.module.Employee;

**public** **class** Main {

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

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

Employee emp=(Employee)context.getBean("emp");

System.*out*.println(emp);

}

}

Note: Here ApplicationContext is an interface. It represents Spring Container. And It’s Implementation class is ClassPathXMLApplicationContext.

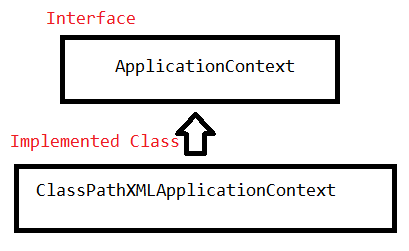
>> getBean(String beanName):Object return required Objects in up-casted format. Again we need to downcast them.

**Some Core Java Concepts used in the above Coding:**

Up-casting: Referring Sub class object using Super Type(Class or interface). To do up casting classes must have inheritance relation.

Down-casting: Already Up casted Object again converting to Sub type is known as Down-casting.

Here



Note: Eclipse must be under Java Perspective.

Eclipse Shortcuts : Alt+Shift+S R (Setters and Getters), Alt+Shift+S S (toString method),Alt+Shift+S O (Constructors).

Spring Bean Primitive type setter injection code can be written in 3 ways. They are

i)Value as Tag(<value></value>)

ii)Value as Attribute(<property name=”” value=””/>)

iii)p-Name space(<bean name=”” class=”” p:variable-name=”value”/>)

ex:

Spring Bean:

**package** com.app.module;

**public** **class** Employee {

**private** **int** empId;

**private** String empName;

**private** **double** empSal;

**public** Employee() {

**super**();

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** **double** getEmpSal() {

**return** empSal;

}

**public** **void** setEmpSal(**double** empSal) {

**this**.empSal = empSal;

}

@Override

**public** String toString() {

**return** "Employee [empId=" + empId + ", empName=" + empName

+ ", empSal=" + empSal + "]";

}

}

For the Above Spring bean Configuration code can be written as:

i)Value as tag: (Spring Configuration XML Code)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

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"*>

<bean class=*"com.app.module.Employee"* name=*"emp"*>

<property name=*"empId"*>

<value>10</value>

</property>

<property name=*"empName"*>

<value>ABCD</value>

</property>

<property name=*"empSal"*>

<value>732.36</value>

</property>

</bean>

</beans>

ii)Value as attribute:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

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"*>

<bean class=*"com.app.module.Employee"* name=*"emp"*>

<property name=*"empId"* value=*"10"*/>

<property name=*"empName"* value=*"abcd"*/>

<property name=*"empSal"* value=*"23.36"*/>

</bean>

</beans>

iii)p-Name Space(p-Schema):

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

**xmlns:p=*"http://www.springframework.org/schema/p"***

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"*>

<bean class=*"com.app.module.Employee"* name=*"emp"* p:empId=*"101"* p:empName=*"ABCD"* p:empSal=*"20.36"*/>

</beans>

**(Observe the red colour line. That is need to be added while using p-schema)**

**Note: a) For a dependency , we cannot configure a property more than one time**

**Ex:**

<bean class=*"com.app.module.Employee"* name=*"emp"* p:empId=*"101"*>

<property name=*"empId"*>

<value>101</value>

</property>

</bean>

Main Class:

Public class Main{

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

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

Employee emp=(Employee)context.getBean("emp");

System.*out*.println(emp);

}

}

Output:

xception in thread "main" org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Property 'empId' is already defined using both <property> and inline syntax. Only one approach may be used per property.

<bean class=*"com.app.module.Employee"* name=*"emp"* >

<property name=*"empId"*>

<value>101</value>

</property>

<property name=*"empId"*>

<value>102</value>

</property>

</bean>

Exception:

org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: Multiple 'property' definitions for property 'empId'

**Note: b) By default <value> tag takes input as String type and later it will be parsed to specific type. If Data can’t be parsed (or any mismatch) then conversion problem will occur, and appropriate exception will be thrown.**

**Like Number format Exception.**

**Ex: empId is int type and sending data as “ABCD”**

<bean class=*"com.app.module.Employee"* name=*"emp"* >

<property name=*"empId"*>

<value>ABCD</value>

</property>

</bean>

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'emp' defined in class path resource [config.xml]: Initialization of bean failed; nested exception is org.springframework.beans.TypeMismatchException: Failed to convert property value of type 'java.lang.String' to required type 'int' for property 'empId'; nested exception is java.lang.NumberFormatException: For input string: "ABCD"

**Note: c) getBean(String beanName,Class classtype) is a overloaded method, which is used to read a bean from spring container with any down casting.**

**Ex:**

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

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

Employee emp=(Employee)context.getBean("emp",Employee.**class**);

System.*out*.println(emp);

}

Spring Collections:

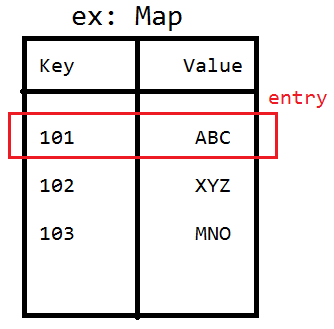
In Spring basically collections are configured as

1. List
2. Set
3. Map
4. Properties

To specify above collections , tag are <list>,<set>,<map> and <props>

Here <map> contains internally entries (<entry>). Every entry contains <key> and <value> , these even can be represents as attributes.

Ex:



Properties also stores data in the key value pair only. But by default key and values are type String in Properties. It contains child tag <prop key=””></prop>. It doesn’t contain any <value> tag to represent the value; we can directly specify the value in between the <prop> tag.

Ex:

**Employee.java**

**package** com.app.module;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Properties;

**import** java.util.Set;

**public** **class** Employee {

**private** List<String> empList;

**private** Set<String> empSet;

**private** Map<String,String> empMap;

**private** Properties empProperties;

**public** List<String> getEmpList() {

**return** empList;

}

**public** **void** setEmpList(List<String> empList) {

**this**.empList = empList;

}

**public** Set<String> getEmpSet() {

**return** empSet;

}

**public** **void** setEmpSet(Set<String> empSet) {

**this**.empSet = empSet;

}

**public** Map<String, String> getEmpMap() {

**return** empMap;

}

**public** **void** setEmpMap(Map<String, String> empMap) {

**this**.empMap = empMap;

}

**public** Properties getEmpProperties() {

**return** empProperties;

}

**public** **void** setEmpProperties(Properties empProperties) {

**this**.empProperties = empProperties;

}

@Override

**public** String toString() {

**return** "Employee [empList=" + empList + ", empSet=" + empSet

+ ", empMap=" + empMap + ", empProperties=" + empProperties

+ "]";

}

}

**Spring Configuration File :config.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

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

xmlns:p=*"http://www.springframework.org/schema/p"*

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"*>

<bean class=*"com.app.module.Employee"* name=*"emp"* >

<property name=*"empList"*> <!-- it will maintain duplicates -->

<list>

<value>A</value>

<value>A</value>

<value>A</value>

<value>B</value>

<value>C</value>

<value>D</value>

</list>

</property>

<property name=*"empSet"*> <!-- it will not maintain duplicates -->

<set>

<value>A</value>

<value>A</value>

<value>A</value>

<value>B</value>

<value>C</value>

<value>D</value>

</set>

</property>

<property name=*"empMap"*>

<map>

<entry>

<key>

<value>K1</value>

</key>

<value>V1</value>

</entry>

<entry key=*"K2"* value=*"V2"*/>

<entry key=*"K3"*>

<value>V3</value>

</entry>

<entry value=*"V4"*>

<key>

<value>K4</value>

</key>

</entry>

</map>

</property>

<property name=*"empProperties"*>

<props>

<prop key=*"key1"*>val1</prop>

<prop key=*"key1"*>val3</prop>

<prop key=*"key2"*>val2</prop>

</props>

</property>

</bean>

</beans>

**Main Program:**

**public** **class** Main {

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

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

Employee emp=(Employee)context.getBean("emp");

System.*out*.println(emp);

}

}

Output:

Employee [empList=[A, A, A, B, C, D], empSet=[A, B, C, D], empMap={K1=V1, K2=V2, K3=V3, K4=V4}, empProperties={key2=val2, key1=val3}]

Note 1):

Empty Object Reference and Null Reference:-

If any Object created with no data and referred with a reference is known as empty Object.

For ex: List l1=new ArrayList(); Creating empty Array List.

If a reference not even contains empty object. Or reference, that doesn’t point to Object is known as null Reference.

**Ex:**

**package** com.app.module;

**import** java.util.List;

**public** **class** Employee {

**private** List<String> empList;

**public** List<String> getEmpList() {

**return** empList;

}

**public** **void** setEmpList(List<String> empList) {

**this**.empList = empList;

}

@Override

**public** String toString() {

**return** "Employee [empList=" + empList + "]";

}

}

For the above class, configuration code is given as:

Case 1)

<bean class=*"com.app.module.Employee"* name=*"emp"* >

</bean>

Employee Object Created with empList as null

Case2)

<bean class=*"com.app.module.Employee"* name=*"emp"* >

<property name=*"empList"*>

<list></list>

</property>

</bean>

Employee Object created with empList with zero size (Or empty List)

Case 3)

<bean class=*"com.app.module.Employee"* name=*"emp"* >

<property name=*"empList"*></property>

</bean>

In case of any property tag is defined without any value tag, then Spring Container throws Exception:

org.springframework.beans.factory.parsing.BeanDefinitionParsingException: Configuration problem: <property> element for property 'empList' must specify a ref or value