# 第一部分 知识准备篇

## 第一章 开发环境准备

1、准备源代码阅读环境

2、获取spring以及源码

## 第二章 spring初体验

### 2.1、利用官方示例

进行一场quick start。

### 2.2、如何学习开源框架

### 2.3、示例

#### 2.3.1 基于配置的依赖注入

|  |
| --- |
| **public class BraveKnight implements Knight {**  **private Quest quest;**  **public BraveKnight(Quest quest) {**  **this.quest = quest;**  **}**  **public void embarkOnQuest() {**  **quest.embark();**  **}**  **}** |

|  |
| --- |
| **<?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.xsd"*>**  **<bean id=*"knight"* class=*"com.glxt.study.spring.springinaction.chapter1.knight.BraveKnight"*>**  **<constructor-arg ref=*"quest"* />**  **</bean>**  **<bean id=*"quest"* class=*"com.glxt.study.spring.springinaction.chapter1.knight.SlayDragonQuest"*>**  **<constructor-arg value=*"#{T(System).out}"* />**  **</bean>**  **</beans>** |

#### 2.3.2 基于配置的AOP编程

|  |
| --- |
| **public class Minstrel {**  **private PrintStream stream;**    **public Minstrel(PrintStream stream) {**  **this.stream = stream;**  **}**  **public void singBeforeQuest() {**  **stream.println("Fa la la, the knight is so brave!");**  **}**  **public void singAfterQuest() {**  **stream.println("Tee hee hee, the brave knight " +**  **"did embark on a quest!");**  **}**  **}** |

|  |
| --- |
| **<?xml version=*"1.0"* encoding=*"UTF-8"*?>**  **<beans xmlns=*"http://www.springframework.org/schema/beans"***  **xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"***  **xmlns:aop=*"http://www.springframework.org/schema/aop"***  **xsi:schemaLocation=*"http://www.springframework.org/schema/aop***  ***http://www.springframework.org/schema/aop/spring-aop-3.2.xsd***  ***http://www.springframework.org/schema/beans***  ***http://www.springframework.org/schema/beans/spring-beans.xsd"*>**  **<bean id=*"knight"* class=*"com.glxt.study.spring.springinaction.chapter1.knight.BraveKnight"*>**  **<constructor-arg ref=*"quest"* />**  **</bean>**  **<bean id=*"quest"* class=*"com.glxt.study.spring.springinaction.chapter1.knight.SlayDragonQuest"*>**  **<constructor-arg value=*"#{T(System).out}"* />**  **</bean>**  **<bean id=*"minstrel"* class=*"com.glxt.study.spring.springinaction.chapter1.knight.Minstrel"*>**  **<constructor-arg value=*"#{T(System).out}"* />**  **</bean>**    **<aop:config>**  **<aop:aspect ref=*"minstrel"*>**  **<aop:pointcut id=*"embark"***  **expression=*"execution(\* \*.embarkOnQuest(..))"*/>**  **<aop:before pointcut-ref=*"embark"* method=*"singBeforeQuest"*/>**  **<aop:after pointcut-ref=*"embark"* method=*"singAfterQuest"*/>**  **</aop:aspect>**  **</aop:config>**    **</beans>** |

### 2.4 容器

spring容器分为两种类型：

1、BeanFactory

2、ApplicationContext

ApplicationContext是BeanFactory的子接口，在实际开发中，经常优先选用ApplicationContext。

#### 2.4.1 application contexts概述

方式一：**ClassPathXmlApplicationContext**

|  |
| --- |
| **public class Test**  **{**  **public void say()**  **{**  **System.*out*.println("hello world!");**  **}**  **public static void main(String[] args)**  **{**  **// 方式一 ：ClassPathXmlApplicationContext**  **ApplicationContext context = new ClassPathXmlApplicationContext("com/glxt/study/spring/springinaction/chapter1/applicationcontext/application.xml");**  **Test test = (Test) context.getBean("test");**  **test.say();**  **}**  **}** |

方式二：**FileSystemXmlApplicationContext**

|  |
| --- |
| **public static void main(String[] args)**  **{**  ***method2*();**  **}**  **public static void method2()**  **{**  **// 方式二：****FileSystemXmlApplicationContext**  **ApplicationContext context = new FileSystemXmlApplicationContext(**  **"E:/study/spring-study/spring/src/main/java/com/glxt/study/spring/springinaction/chapter1/applicationcontext/application.xml");**  **Test test = (Test) context.getBean("test");**  **test.say();**  **}** |

方式三：**AnnotationConfigApplicationContext**

|  |
| --- |
| **public static void main(String[] args)**  **{**  ***method3*();**  **}**    **public static void method3()**  **{**  **// 方式三：AnnotationConfigApplicationContext**  **ApplicationContext context = new AnnotationConfigApplicationContext(**  **com.glxt.study.spring.springinaction.chapter1.applicationcontext.Test.class);**  **Test test = (Test) context.getBean(com.glxt.study.spring.springinaction.chapter1.applicationcontext.Test.class);**  **test.say();**  **}** |

方式四： **XmlWebApplicationContext**

方式五： **AnnotationConfigWebApplicationContext**

#### 2.4.2 bean生命周期

### 2.5 实例化bean

实例化bean，分为两种方式：

1、反射模式

2、工厂方法模式

工厂静态方法模式：配置依赖factory-method

工厂方法模式：配置依赖factory-bean、factory-method

待实例化的bean，如下：

|  |
| --- |
| **public class Car**  **{**  **private int id;**  **private String name;**  **private int price;**  **public int getId()**  **{**  **return id;**  **}**  **public void setId(int id)**  **{**  **this.id = id;**  **}**  **public String getName()**  **{**  **return name;**  **}**  **public void setName(String name)**  **{**  **this.name = name;**  **}**  **public int getPrice()**  **{**  **return price;**  **}**  **public void setPrice(int price)**  **{**  **this.price = price;**  **}**  **@Override**  **public String toString()**  **{**  **return "Car [id=" + id + ", name=" + name + ", price=" + price + "]";**  **}**  **public Car()**  **{**  **}**  **public Car(int id, String name, int price)**  **{**  **super();**  **this.id = id;**  **this.name = name;**  **this.price = price;**  **}**  **}** |

#### 2.5.1 反射模式

反射模式实例化bean，可以通过调用无参或者有参构造方法实例化，一般是通过反射调用无参构造函数实例化bean。

1、无参构造函数，配置spring xml

|  |
| --- |
| **<?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.xsd"*>**  **<bean id=*"car"* class=*"com.glxt.study.spring.instantiatingbeans.reflection.nonargument.Car"*>**  **</bean>**  **</beans>** |

这种调用无参构造的方式也是经常使用。

2、有参构造函数，配置spring xml

|  |
| --- |
| **<?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.xsd"*>**  **<bean id=*"car"* class=*"com.glxt.study.spring.instantiatingbeans.reflection.argument.Car"*>**  **<constructor-arg type=*"int"* value=*"1"*></constructor-arg>**  **<constructor-arg type=*"String"* value=*"BMW"*></constructor-arg>**  **<constructor-arg type=*"int"* value=*"540000"*></constructor-arg>**  **</bean>**  **</beans>** |

#### 2.5.2 工厂方法模式

工厂方法实例化bean，分为静态工厂方法和非静态工厂方法。

1、静态工厂方法

|  |
| --- |
| **public class CarStaticFactory**  **{**  **private static Map<Integer, Car> *map* = new HashMap<Integer, Car>();**  **static**  **{**  ***map*.put(1, new Car(1, "Honda", 300000));**  ***map*.put(2, new Car(2, "Audi", 440000));**  ***map*.put(3, new Car(3, "BMW", 540000));**  **}**  **public static Car getCar(int id)**  **{**  **return *map*.get(id);**  **}**  **}** |

|  |
| --- |
| **<?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.xsd"*>**  **<bean id=*"bmwCar"* class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.staticmethod.CarStaticFactory"* factory-method=*"getCar"*>**  **<constructor-arg value=*"3"*></constructor-arg>**  **</bean>**    **<bean id=*"audiCar"* class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.staticmethod.CarStaticFactory"* factory-method=*"getCar"*>**  **<constructor-arg value=*"2"*></constructor-arg>**  **</bean>**    **</beans>** |

|  |
| --- |
| **public class Test**  **{**  **public static void main(String[] args)**  **{**  **ApplicationContext context = new ClassPathXmlApplicationContext("com/glxt/study/spring/instantiatingbeans/factorymethod/staticmethod/car.xml");**  **Car car = context.getBean("bmwCar", Car.class);**  **System.*out*.println(car);**    **Car car2 = context.getBean("audiCar", Car.class);**  **System.*out*.println(car2);**  **}**  **}** |

2、非静态工厂方法

|  |
| --- |
| **public class CarFactory**  **{**  **private Map<Integer, Car> map = new HashMap<Integer, Car>();**  **public void setMap(Map<Integer, Car> map)**  **{**  **this.map = map;**  **}**  **public CarFactory()**  **{**  **}**  **public Car getCar(int id)**  **{**  **return map.get(id);**  **}**  **}** |

|  |
| --- |
| **<?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.xsd"*>**    **<!-- Instance Factory Method:**  **1.must create a bean for the Instance Factroy First**  **-->**  **<bean id=*"carFactory"* class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.nonstaticmethod.CarFactory"*>**  **<property name=*"map"*>**  **<map>**  **<entry key=*"1"*>**  **<bean class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.nonstaticmethod.Car"*>**  **<property name=*"id"* value=*"1"*></property>**  **<property name=*"name"* value=*"Honda"*></property>**  **<property name=*"price"* value=*"300000"*></property>**  **</bean>**  **</entry>**  **<entry key=*"2"*>**  **<bean class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.nonstaticmethod.Car"*>**  **<property name=*"id"* value=*"2"*></property>**  **<property name=*"name"* value=*"Audi"*></property>**  **<property name=*"price"* value=*"440000"*></property>**  **</bean>**  **</entry>**  **<entry key=*"3"*>**  **<bean class=*"com.glxt.study.spring.instantiatingbeans.factorymethod.nonstaticmethod.Car"*>**  **<property name=*"id"* value=*"3"*></property>**  **<property name=*"name"* value=*"BMW"*></property>**  **<property name=*"price"* value=*"540000"*></property>**  **</bean>**  **</entry>**  **</map>**  **</property>**  **</bean>**    **<!-- 2.use Factory bean to get bean object**  **factory-bean : the bean define above**  **factory-method: method of get Bean Object**  **constructor-arg: parameters of factory-method**  **-->**  **<bean id=*"bmwCar"* factory-bean=*"carFactory"* factory-method=*"getCar"*>**  **<constructor-arg value=*"3"*></constructor-arg>**  **</bean>**    **<bean id=*"audiCar"* factory-bean=*"carFactory"* factory-method=*"getCar"*>**  **<constructor-arg value=*"2"*></constructor-arg>**  **</bean>**    **</beans>** |

### 2.6 基于注解的容器配置

#### @Required

[Spring依赖检查](http://www.yiibai.com/spring/spring-properties-dependency-checking.html) bean 配置文件用于确定的特定类型(基本，集合或对象)的所有属性被设置。在大多数情况下，你只需要确保特定属性已经设置但不是所有属性..

对于这种情况，你需要 @Required 注解。

#### @Autowired

@Autowired注解提供更细粒度地控制在何处以及如何使用自动装配时应完成。这个注解就是spring可以自动帮你把bean里面引用的对象的setter/getter方法省略，它会自动帮你set/get。

<http://www.yiibai.com/spring/spring_autowired_annotation.html>

#### @Qualifier

可能会有这样一种情况，当你创建多个具有相同类型的 bean 时，并且想要用一个属性只为它们其中的一个进行装配，在这种情况下，你可以使用 **@Qualifier** 注释和 **@Autowired** 注释通过指定哪一个真正的 bean 将会被装配来消除混乱。

示例：package com.glxt.study.spring.annotation.qualifier

### 2.7 Classpath scanning and managed components

#### @ComponentScan

它一般搭配@Configuration，扫描@Component组件。

<http://javarticles.com/2016/01/spring-componentscan-annotation-example.html>

**基本使用**

|  |
| --- |
| **@Configuration**  **@ComponentScan(basePackages={"com.glxt.study.spring.annotation.componentscan.packageA",**  **"package com.glxt.study.spring.annotation.componentscan.packageB"})**  **public class ComponentScanAnnotationExample**  **{**  **public ComponentScanAnnotationExample()**  **{**  **System.*out*.println("ComponentScanAnnotationExample Constructor");**  **}**  **public static void main(String[] args)**  **{**  **AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(ComponentScanAnnotationExample.class);**  **System.*out*.println(ctx.getBean("beanA"));**  **System.*out*.println(ctx.getBean("componentScanAnnotationExample"));**  **System.*out*.println(ctx.getBean("beanB"));**  **}**    **@Override**  **public String toString()**  **{**  **return "ComponentScanAnnotationExample";**  **}**  **}** |

**黑名单过滤**

|  |
| --- |
| **@Configuration**  **@ComponentScan(basePackageClasses=BeanA.class,**  **excludeFilters = @ComponentScan.Filter(type = FilterType.*ASSIGNABLE\_TYPE*,value=BeanA1.class))**  **public class ComponentScanExcludeAnnotationExample**  **{**  **public static void main(String[] args)**  **{**  **AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(ComponentScanExcludeAnnotationExample.class);**  **System.*out*.println("Contains BeanA? : " + ctx.containsBean("beanA"));**  **System.*out*.println("Contains BeanA1? : " + ctx.containsBean("beanA1"));**  **}**  **}** |

**白名单过滤**

|  |
| --- |
| **@Configuration**  **@ComponentScan(basePackages={"com.glxt.study.spring.annotation.componentscan.packageA",**  **"com.glxt.study.spring.annotation.componentscan.packageB"},**  **includeFilters = @ComponentScan.Filter(type = FilterType.*ASSIGNABLE\_TYPE*, value = BeanB.class),**  **useDefaultFilters = false)**  **public class ComponentScanIncludeAnnotationExample**  **{**  **public static void main(String[] args)**  **{**  **AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(ComponentScanIncludeAnnotationExample.class);**  **System.*out*.println("Contains BeanA? : " + ctx.containsBean("beanA"));**  **System.*out*.println("Contains BeanA1? : " + ctx.containsBean("beanA1"));**  **System.*out*.println("Contains BeanB? : " + ctx.containsBean("beanB"));**  **}**  **}** |

注意：在使用白名单过滤时，如果想选出指定的类，需要使用useDefaultFilters = false。

**XML方式触发**

|  |
| --- |
| **@Configuration**  **@ComponentScan(basePackages={"com.glxt.study.spring.annotation.componentscan.packageA",**  **"com.glxt.study.spring.annotation.componentscan.packageB"})**  **public class ComponentScanAnnotationViaXMLExample**  **{**  **public ComponentScanAnnotationViaXMLExample()**  **{**  **System.*out*.println("Constructor ComponentScanAnnotationViaXMLExample");**  **}**  **public static void main(String[] args)**  **{**  **ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("com/glxt/study/spring/annotation/componentscan/beans.xml");**  **System.*out*.println(ctx.getBean("beanA"));**  **System.*out*.println(ctx.getBean("beanA1"));**  **System.*out*.println(ctx.getBean("beanB"));**  **}**  **}** |

|  |
| --- |
| **<?xml version=*"1.0"* encoding=*"UTF-8"*?>**  **<beans xmlns=*"http://www.springframework.org/schema/beans"***  **xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"***  **xmlns:context=*"http://www.springframework.org/schema/context"***  **xsi:schemaLocation=*"http://www.springframework.org/schema/beans***  ***http://www.springframework.org/schema/beans/spring-beans.xsd***  ***http://www.springframework.org/schema/context***  ***http://www.springframework.org/schema/context/spring-context.xsd"*>**    **<context:annotation-config/>**  **<bean class=*"com.glxt.study.spring.annotation.componentscan.ComponentScanAnnotationViaXMLExample"*></bean>**    **</beans>** |

### 2.8 基于java的容器配置

#### @Configuration

@Configuration是一个类级别的注解，它是定义bean的源头。简单地说，它相当于在xml中定义了bean。

|  |
| --- |
| **@Configuration**  **public class HelloWorldConfig**  **{**  **@Bean**  **public HelloWorld helloWorld()**  **{**  **return new HelloWorld();**  **}**  **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(**  **HelloWorldConfig.class);**  **HelloWorld helloWorld = ctx.getBean(HelloWorld.class);**  **helloWorld.setMessage("Hello World!");**  **helloWorld.getMessage();**  **}**  **}** |

#### @Bean

|  |
| --- |
| **@Configuration**  **public class AppConfig {**  **@Bean(initMethod = "init")**  **public Foo foo() {**  **return new Foo();**  **}**  **@Bean(destroyMethod = "cleanup")**  **public Bar bar() {**  **return new Bar();**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(**  **AppConfig.class);**  **Foo foo = ctx.getBean(Foo.class);**  **}**  **}** |

@Bean可以管理对象生命周期的调用。

#### @Scope

@Scope是用来指定Bean的作用域，因而它的先决条件就是存在bean。

|  |
| --- |
| **@Configuration**  **public class AppConfig {**  **@Bean**  **@Scope("prototype")**  **public Bar bar() {**  **return new Bar();**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(**  **AppConfig.class);**    **System.*out*.println("First Bar: ");**  **Bar bar1 = ctx.getBean(Bar.class);**    **System.*out*.println("Second Bar: ");**  **Bar bar2 = ctx.getBean(Bar.class);**  **}**  **}** |

如果不为@Bean显式地配置@Scope，默认情况会为@Bean配置@Scope(“singleton”)。

#### 2.8.1 组装基于java的配置

##### @Import

|  |
| --- |
| **@Configuration**  **@Import(ConfigA.class)**  **public class ConfigB**  **{**  **@Bean**  **public B createB()**  **{**  **return new B();**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(ConfigB.class);**  **A a = ctx.getBean(A.class);**  **B b = ctx.getBean(B.class);**  **System.*out*.println(a);**  **System.*out*.println(b);**  **}**  **}** |

##### @ImportResource

@ImportResource与@Import的区别：

@ImportResource导入xml文件中的bean资源；

@Import导入java注解类中的bean资源。

|  |
| --- |
| **@Configuration**  **@Import(ConfigA.class)**  **@ImportResource("classpath:/com/glxt/study/spring/annotation/importresource/beans.xml")**  **public class ConfigB**  **{**  **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(ConfigB.class);**  **A a = ctx.getBean(A.class);**  **B b = ctx.getBean(B.class);**  **System.*out*.println(a);**  **System.*out*.println(b);**  **}**  **}** |

### 2.9 容器配置小结

简单地理解基于XML、注解(Annotation based)、java三种方式的容器配置：

* 基于XML的配置，就是完全使用XML文件配置spring context。
* 基于注解的配置(Annotation based configuration)，一部分使用XML文件配置spring bean，一部分使用注解关联XML中的bean。
* 基于java的配置(Java based configuration)，就是代替XML文件配置，完全使用类注解配置spring context。

### 2.10 @Profile

一个Profile对象就是一系列为特定领域的集合。类似抽象工厂设计模式，对应着某个产品家族簇。

### 2.11 @PropertySource

|  |
| --- |
| **@Configuration**  **@PropertySource("classpath:/com/glxt/study/spring/annotation/propertysource/app.properties")**  **public class AppConfig**  **{**  **@Autowired**  **Environment environment;**    **@Bean**  **public Car getCar()**  **{**  **Car car = new Car();**  **car.setName(environment.getProperty("car.name"));**  **car.setPrice(Double.*parseDouble*(environment.getProperty("car.price")));**  **return car;**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);**  **Car car = ctx.getBean(Car.class);**  **System.*out*.println(car);**  **}**  **}** |

### 2.12 ApplicationEvent与ApplicationListener

Event handling in the ApplicationContext is provided through the **ApplicationEvent** class and **ApplicationListener** interface。Essentially, this is the standard Observer design pattern

#### 1、通过implement ApplicationListener方式处理事件

|  |
| --- |
| **public class EmailEvent extends ApplicationEvent**  **{**  **private String address;**  **private String text;**    **public EmailEvent(Object source)**  **{**  **super(source);**  **}**    **public EmailEvent(Object source, String address, String text)**  **{**  **super(source);**  **this.address = address;**  **this.text = text;**  **}**    **public void print()**  **{**  **System.*out*.println("hello spring event!");**  **}**  **public String getAddress()**  **{**  **return address;**  **}**  **public void setAddress(String address)**  **{**  **this.address = address;**  **}**  **public String getText()**  **{**  **return text;**  **}**  **public void setText(String text)**  **{**  **this.text = text;**  **}**  **}** |

|  |
| --- |
| **@Configuration**  **public class EmailListener implements ApplicationListener<EmailEvent>**  **{**  **public void onApplicationEvent(EmailEvent emailEvent)**  **{**  **emailEvent.print();**  **System.*out*.println("the source is:"+emailEvent.getSource());**  **System.*out*.println("the address is:"+emailEvent.getAddress());**  **System.*out*.println("the email's context is:"+emailEvent.getText());**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(EmailListener.class);**  **EmailEvent event = new EmailEvent("hello","boylmx@163.com","this is a email text!");**  **EmailListener emailListener = new EmailListener();**  **// 3、发布事件**  **ctx.publishEvent(event);**  **}**  **}** |

#### 2、通过@EventListener注解方式处理事件

|  |
| --- |
| **@Configuration**  **public class EmailNotifier**  **{**  **@EventListener**  **public void processEmailEvent(EmailEvent emailEvent)**  **{**  **emailEvent.print();**  **System.*out*.println("the source is:"+emailEvent.getSource());**  **System.*out*.println("the address is:"+emailEvent.getAddress());**  **System.*out*.println("the email's context is:"+emailEvent.getText());**  **}**    **public static void main(String[] args)**  **{**  **ApplicationContext ctx = new AnnotationConfigApplicationContext(EmailNotifier.class);**  **EmailEvent event = new EmailEvent("hello","boylmx@163.com","this is a email text!");**  **EmailNotifier emailNotifier = new EmailNotifier();**  **// 3、发布事件**  **ctx.publishEvent(event);**  **}**  **}** |

## 第三章 spring概览

3.1 使用场景

3.2 使用技术

依赖注入和控制反转

3.3 模块

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test, as shown in the following diagram.



### 3.3.1 Core Container

The [Core Container](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-introduction) consists of the spring-core, spring-beans, spring-context, spring-context-support, and spring-expression (Spring Expression Language) modules.

The spring-core and spring-beans modules [provide the fundamental parts of the framework](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-introduction), including the IoC and Dependency Injection features.

The [Context](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#context-introduction) (spring-context) module builds on the solid base provided by the [Core and Beans](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-introduction) modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry.

spring-context-support provides support for integrating common third-party libraries into a Spring application context for caching (EhCache, Guava, JCache), mailing (JavaMail), scheduling (CommonJ, Quartz) and template engines (FreeMarker, JasperReports, Velocity).

The spring-expression module provides a powerful [Expression Language](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#expressions) for querying and manipulating an object graph at runtime.

### 3.3.2 AOP and Instrumentation

The spring-aop module provides an [AOP](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#aop-introduction) Alliance-compliant aspect-oriented programming

The separate spring-aspects module provides integration with AspectJ

The spring-instrument module provides class instrumentation support and classloader implementations

The spring-instrument-tomcat module contains Spring’s instrumentation agent for Tomcat

### 3.3.3 Messaging

Spring Framework 4 includes a spring-messaging module with key abstractions from the Spring Integration project such as Message, MessageChannel, MessageHandler, and others to serve as a foundation for messaging-based applications.

### 3.3.4 Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS, and Transaction modules.

The spring-jdbc module provides a [JDBC](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#jdbc-introduction)-abstraction layer

The spring-tx module supports [programmatic and declarative transaction](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#transaction) management for classes

The spring-orm module provides integration layers for popular [object-relational mapping](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-introduction) APIs, including [JPA](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-jpa), [JDO](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-jdo), and [Hibernate](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-hibernate).

The spring-oxm module provides an abstraction layer that supports [Object/XML mapping](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#oxm) implementations such as JAXB, Castor, XMLBeans, JiBX and XStream

The spring-jms module ([Java Messaging Service](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#jms)) contains features for producing and consuming messages.

### 3.3.5 Web

The Web layer consists of the spring-web, spring-webmvc, spring-websocket, and spring-webmvc-portlet modules.

### 3.3.6 Test

The spring-test module supports the [unit testing](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#unit-testing) and [integration testing](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#integration-testing) of Spring components with JUnit or TestNG