# 一、使用外部属性

1.使用PropertyPlaceholderConfigurer引用属性文件

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
   <property name="location" value="classpath:com/smart/place/jdbc.properties"></property>  
   <property name="fileEncoding" value="UTF-8"></property>  
</bean>  
  
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">  
   <property name="driverClassName" value="${driverClassName}"></property>  
   <property name="url" value="${url}"></property>  
   <property name="username" value="${userName}"></property>  
   <property name="password" value="${password}"></property>  
</bean>

先引入属性文件，再通过${KEY}来使用

2.使用context:property-placeholder来引入

<context:property-placeholder location="classpath:hibernate.properties" />

3.以下附录一个DES加密的程序

import java.security.Key;  
import java.security.SecureRandom;  
  
import javax.crypto.Cipher;  
import javax.crypto.KeyGenerator;  
  
import sun.misc.BASE64Decoder;  
import sun.misc.BASE64Encoder;  
  
public class DESUtils {  
   private static Key key;  
   private static String KEY\_STR = "myKey";  
   static {  
      try {  
         KeyGenerator generator = KeyGenerator.getInstance("DES");  
         generator.init(new SecureRandom(KEY\_STR.getBytes()));  
         key = generator.generateKey();  
         generator = null;  
      } catch (Exception e) {  
         throw new RuntimeException(e);  
      }  
   }  
  
   /\*\*  
    \* 对str进行DES加密  
    \*   
    \* @param str  
    \* @return  
    \*/  
   public static String getEncryptString(String str) {  
      BASE64Encoder base64en = new BASE64Encoder();  
      try {  
         byte[] strBytes = str.getBytes("UTF8");  
         Cipher cipher = Cipher.getInstance("DES");  
         cipher.init(Cipher.ENCRYPT\_MODE, key);  
         byte[] encryptStrBytes = cipher.doFinal(strBytes);  
         return base64en.encode(encryptStrBytes);  
      } catch (Exception e) {  
         throw new RuntimeException(e);  
      }  
   }  
  
   /\*\*  
    \* 对str进行DES解密  
    \*   
    \* @param str  
    \* @return  
    \*/  
   public static String getDecryptString(String str) {  
      BASE64Decoder base64De = new BASE64Decoder();  
      try {  
         byte[] strBytes = base64De.decodeBuffer(str);  
         Cipher cipher = Cipher.getInstance("DES");  
         cipher.init(Cipher.DECRYPT\_MODE, key);  
         byte[] decryptStrBytes = cipher.doFinal(strBytes);  
         return new String(decryptStrBytes, "UTF8");  
      } catch (Exception e) {  
         throw new RuntimeException(e);  
      }  
   }  
}

# 二、容器事件

1. 事件类：ApplicationEvent的唯一构造函数ApplicationEvent(Object Source)通过Source指定事件源。它有两个子类ApplicationContextEvent：容器事件。RequestHandleEvent：与web相关的事件，当http请求处理后，产生该事件，只有在web.xml中定义了DispatcherServlet时才会产生该事件。
2. 事件监听器接口：ApplicationListener接口，该接口只有一个方法onApplicationEvent(E event)该方法接受ApplicationEvent事件对象，进行事件处理。

public class MailSender implements ApplicationContextAware {  
  
   private ApplicationContext ctx ;  
    //ApplicationContextAware的接口方法，以便容器启动时，注入容器实例。  
   public void setApplicationContext(ApplicationContext ctx)  
         throws BeansException {  
      this.ctx = ctx;  
   }  
  
   public void sendMail(String to){  
      System.out.println("MailSender:模拟发送邮件...");  
      MailSendEvent mse = new MailSendEvent(this.ctx,to);  
      //向容器中所有事件监听器发送事件  
      ctx.publishEvent(mse);  
   }  
}

public class MailSendEvent extends ApplicationContextEvent {  
   private String to;  
     
   public MailSendEvent(ApplicationContext source, String to) {  
      super(source);  
      this.to = to;  
   }  
   public String getTo() {  
        
      return this.to;  
   }  
}

public class MailSendListener implements ApplicationListener<MailSendEvent>{  
  
   public void onApplicationEvent(MailSendEvent event) {  
         MailSendEvent mse = (MailSendEvent) event;  
         System.out.println("MailSendListener:向" + mse.getTo() + "发送完一封邮件");  
   }  
}

public class ApplicationEventTest{

public static void main(String[] args){

String resourceFile = “com/smart/event/beans.xml”;

ApplicationContext ctx = new ClassPathXmlApplicationContext(resourceFile);

MailSender mailSender = ctx.getBean(MailSender.class);

mailSender.sendMail(“test mail.”);

}

}