**SendingEmailWithSpring**

**By zmzizi**\ 2005년 06월 09일 현재 Spring 1.2.1 레퍼런스와 변동없음\ \

출처 : <http://openframework.or.kr/JSPWiki/Wiki.jsp?page=SendingEmailWithSpring>

... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... no changes ... [mp3 ringtones](http://www.la-ringtones.com) : http://www.la-ringtones.com real ringtoneshttp://openframework.or.kr/JSPWiki/images/out.png - No InterWiki reference defined in properties for Wiki called 'HTTP'!) : [motorola ringtones](http://www.la-ringtones.com)http://openframework.or.kr/JSPWiki/images/out.png - No InterWiki reference defined in properties for Wiki called 'HTTP'!) : http://www.la-ringtones.com/tones/ : [http://www.la-ringtones.com tracfone ringtones]] : [http://www.la-ringtones.com | qwest ringtones]] : "nextel ringtones" http://www.la-ringtones.com : [http://www.la-ringtones.com](http://openframework.or.kr/JSPWiki/Wiki.jsp?page=FunnyRingtones) **By zmzizi**\ 2005년 06월 09일 현재 Spring 1.2.1 레퍼런스와 변동없음\ \

### Chapter 18. Spring 메일 abstraction 계층을 사용한 이메일 보내기\

\

#### 18.1 소개\

Spring 은 전자메일을 보내기 위한 높은 수준의 abstraction 을 제공하는데, 이것은 사용자들이 기반하고 있는 메일링 시스템에 대한 상세한 기술을 할 필요가 없게 하고, 고객을 대신하여 낮은 레벨의 리소스 핸들링에 대한 책임을 진다.\ \

#### 18.2 Spring 메일 abstraction 구조\

Spring 메일 abstraction 계층의 메인 패키지는 org.springframework.mail 패키지 이다. 이것은 메일을 보내기 위한 주된 인터페이스인 MailSender와, from, to, cc, subject, text 와 같은 간단한 메일의 속성들을 캡슐화하는 값객체(value object) 인 SimpleMailMessage를 포함하고 있다. 이 패키지는 MailException을 루트로 하는 체크된 예외들의 계층을 포함하고 있는데, 이 예외들은 낮은 레벨의 메일 시스템 예외들에 대해 보다 상위의 abstraction을 제공한다. 메일 예외계층에 대한 더 많은 정보는 JavaDocs을 참조하길 바란다.\ Spring은 또한 MIME 메시지와 같이 JavaMail의 특징에 특화된 MailSender의 서브 인터페이스인 org.springframework.mail.javamail.JavaMailSender를 제공한다. 이것은 또한 JavaMail MIME 메시지들의 준비를 위한 callback 인터페이스인 org.springframework.mail.javamail.MimeMessagePreparator를 제공한다.\ \

|  |
| --- |
| MailSender : **public interface**MailSender {      /\*\*      \* Send the given simple mail message.      \* @param simpleMessage message to send      \* @throws MailException in case of message, authentication, or send errors      \*/     **public void**send(SimpleMailMessage simpleMessage) **throws**MailException;      /\*\*      \* Send the given array of simple mail messages in batch.      \* @param simpleMessages messages to send      \* @throws MailException in case of message, authentication, or send errors      \*/     **public void**send(SimpleMailMessage[] simpleMessages) **throws**MailException;  }  JavaMailSender : **public interface**JavaMailSender **extends**MailSender {      /\*\*      \* Create a new JavaMail MimeMessage for the underlying JavaMail Session      \* of this sender. Needs to be called to create MimeMessage instances      \* that can be prepared by the client and passed to send(MimeMessage).      \* @return the new MimeMessage instance      \* @see #send(MimeMessage)      \* @see #send(MimeMessage[])      \*/     **public**MimeMessage createMimeMessage();      /\*\*      \* Send the given JavaMail MIME message.      \* The message needs to have been created with createMimeMessage.      \* @param mimeMessage message to send      \* @throws MailException in case of message, authentication, or send errors      \* @see #createMimeMessage      \*/     **public void**send(MimeMessage mimeMessage) **throws**MailException;      /\*\*      \* Send the given array of JavaMail MIME messages in batch.      \* The messages need to have been created with createMimeMessage.      \* @param mimeMessages messages to send      \* @throws MailException in case of message, authentication, or send errors      \* @see #createMimeMessage      \*/     **public void**send(MimeMessage[] mimeMessages) **throws**MailException;      /\*\*      \* Send the JavaMail MIME message prepared by the given MimeMessagePreparator.      \* Alternative way to prepare MimeMessage instances, instead of createMimeMessage      \* and send(MimeMessage) calls. Takes care of proper exception conversion.      \* @param mimeMessagePreparator the preparator to use      \* @throws MailException in case of message, authentication, or send errors      \*/     **public void**send(MimeMessagePreparator mimeMessagePreparator) **throws**MailException;      /\*\*      \* Send the JavaMail MIME messages prepared by the given MimeMessagePreparators.      \* Alternative way to prepare MimeMessage instances, instead of createMimeMessage      \* and send(MimeMessage[]) calls. Takes care of proper exception conversion.      \* @param mimeMessagePreparators the preparator to use      \* @throws MailException in case of message, authentication, or send errors      \*/     **public void**send(MimeMessagePreparator[] mimeMessagePreparators) **throws**MailException;  }  MimeMessagePreparator : **public interface**MimeMessagePreparator {      /\*\*      \* Prepare the given new MimeMessage instance.      \* @param mimeMessage the message to prepare      \* @throws MessagingException passing any exceptions thrown by MimeMessage      \* methods through for automatic conversion to the MailException hierarchy      \*/     **void**prepare(MimeMessage mimeMessage) **throws**MessagingException;  } |

\

#### 18.3 Spring 메일 abstraction 사용하기\

OrderManager라는 비즈니스 인터페이스가 있다고 가정해보자.\

|  |
| --- |
| **public interface**OrderManager {      **void**placeOrder(Order order); } |

\ 그리고, 주문번호를 가진 이메일 메시지가 생성되어 그 주문을 한 고객에게 보내져야만 한다는 유스케이스가 있다고도 가정해보자. 그렇다면 이 목적을 달성하기 위해 우리는 MailSender 와 SimpleMailMessage를 사용할 것이다.\ *일반적으로, 우리는 비즈니스 코드에서 인터페이스들을 사용하게 될 것이고, Spring IoC 컨테이너로 하여금 우리에게 필요한 모든 협력자(구현 클래스)들을 다루도록 할 것임을 전제하고 있다.*\ 아래에 OrderManager의 구현클래스가 있다.\

|  |
| --- |
| **import**org.springframework.mail.MailException; **import**org.springframework.mail.MailSender; **import**org.springframework.mail.SimpleMailMessage;  **public class**OrderManagerImpl **implements**OrderManager {      **private**MailSender mailSender;     **private**SimpleMailMessage message;      **public void**setMailSender(MailSender mailSender) {         **this**.mailSender = mailSender;     }      **public void**setMessage(SimpleMailMessage message) {         **this**.message = message;     }      **public void**placeOrder(Order order) {          //... \* Do the business calculations....         //... \* Call the collaborators to persist the order          //Create a thread safe "sandbox" of the message         SimpleMailMessage msg = **new**SimpleMailMessage(**this**.message);         msg.setTo(order.getCustomer().getEmailAddress());         msg.setText(             "Dear "                 + order.getCustomer().getFirstName()                 + order.getCustomer().getLastName()                 + ", thank you for placing order. Your order number is "                 + order.getOrderNumber());         **try**{             mailSender.send(msg);         }         **catch**(MailException ex) {             //log it and go on             System.err.println(ex.getMessage());                     }     } } |

\ 그리고 위의 코드에 대한 bean 정의는 다음과 같을 것이다.\

|  |
| --- |
| <bean id="mailSender"       **class**="org.springframework.mail.javamail.JavaMailSenderImpl">     <property name="host"><value>mail.mycompany.com</value></property> </bean>  <bean id="mailMessage"       **class**="org.springframework.mail.SimpleMailMessage">     <property name="from"><value>customerservice@mycompany.com</value></property>     <property name="subject"><value>Your order</value></property> </bean>  <bean id="orderManager"       **class**="com.mycompany.businessapp.support.OrderManagerImpl">     <property name="mailSender"><ref bean="mailSender"/></property>     <property name="message"><ref bean="mailMessage"/></property> </bean> |

\ 이제 MimeMessagePreparator callback 인터페이스를 사용한 OrderManager의 구현클래스를 보자. 아래의 경우 JavaMail MimeMessage를 사용할 수 있도록 하기 위해 mailSender 속성이 JavaMailSender 타입이라는 점에 주의해야 한다.\

|  |
| --- |
| **import**javax.mail.Message; **import**javax.mail.MessagingException; **import**javax.mail.internet.InternetAddress; **import**javax.mail.internet.MimeMessage;  **import**javax.mail.internet.MimeMessage; **import**org.springframework.mail.MailException; **import**org.springframework.mail.javamail.JavaMailSender; **import**org.springframework.mail.javamail.MimeMessagePreparator;  **public class**OrderManagerImpl **implements**OrderManager {     **private**JavaMailSender mailSender;          **public void**setMailSender(JavaMailSender mailSender) {         **this**.mailSender = mailSender;     }      **public void**placeOrder(**final**Order order) {          //... \* Do the business calculations....         //... \* Call the collaborators to persist the order                           MimeMessagePreparator preparator = **new**MimeMessagePreparator() {             **public void**prepare(MimeMessage mimeMessage) **throws**MessagingException {                 mimeMessage.setRecipient(Message.RecipientType.TO,                          **new**InternetAddress(order.getCustomer().getEmailAddress()));                 mimeMessage.setFrom(**new**InternetAddress("mail@mycompany.com"));                 mimeMessage.setText(                     "Dear "                         + order.getCustomer().getFirstName()                         + order.getCustomer().getLastName()                         + ", thank you for placing order. Your order number is "                         + order.getOrderNumber());             }         };         **try**{             mailSender.send(preparator);         }         **catch**(MailException ex) {             //log it and go on             System.err.println(ex.getMessage());                     }     } } |

\ 만약 당신이 JavaMail MimeMessage의 모든 것을 사용하려 한다면, MimeMessagePreparator를 이용할 수 있다.\ 위의 메일코드는 단지 대조적인 하나의 방법이며, (Spring 메일 abstraction을 사용한 메일코드는) 임의의 Spring AOP를 통한 리팩토링을 완벽하게 지원하고 있기 때문에, OrderManager 타겟에 쉽게 적용될 수 있다. 이 문제에 대해서는 AOP 챕터를 보도록 하라.\ \ **18.3.1. 플러그인할 수 있는 MailSender 구현클래스들**\ Spring 은 2개의 MailSender 구현클래스를 부수적으로 가지는데, 하나는 JavaMail 구현체이고 다른 하나는 http://servlets.com/cos (com.oreilly.servlet)에 포함된 Jason Hunter 의 MailMessage 클래스의 구현체이다. 더 많은 정보는 JavaDocs을 참조하도록 하라.\ \

#### 18.4. JavaMail MimeMessageHelper 사용하기\

JavaMail message를 다룰 때 가장 간편한 컴포넌트들 가운데 하나는 org.springframework.mail.javamail.MimeMessageHelper이다. 이것은 당신으로 하여금 귀찮은 javax.mail.internet 클래스들의 API들을 사용하지 않도록 도와준다. 가능한 2개의 시나리오는 다음과 같다.\ \ **18.4.1. 간단한 MimeMessage 를 생성하고 보내기**\ MimeMessageHelper를 사용하면, MimeMessage를 셋업하고 보내는 것이 매우 간단해진다.\

|  |
| --- |
| // of course you would setup the mail sender using  // DI in any real-world cases JavaMailSenderImpl sender = **new**JavaMailSenderImpl(); sender.setHost("mail.host.com");  MimeMessage message = sender.createMimeMesage(); MimeMessageHelper helper = **new**MimeMessageHelper(message); helper.setTo("test@host.com"); helper.setText("Thank you for ordering!");  sender.send(message); |

\ **18.4.2. 첨부파일들과 inline 리소스들을 보내기**\ 이메일은 첨부파일과 멀티파트 메시지 속의 inline 리소스들을 허용한다. inline 리소스들은 이미지, 스타일시트와 같이 당신이 메시지 속에서 사용하려고 하지만, 첨부파일로 명시되는 것은 원하지 않는 리소스들을 말한다. 다음의 코드는 당신이 어떻게 MimeMessageHelper를 사용하여 inline 이미지를 이메일에 딸려 보낼 수 있는지를 보여준다.\

|  |
| --- |
| JavaMailSenderImpl sender = **new**JavaMailSenderImpl(); sender.setHost("mail.host.com");  MimeMessage message = sender.createMimeMesage();  // use the true flag to indicate you need a multipart message MimeMessageHelper helper = **new**MimeMessageHelper(message, **true**); helper.setTo("test@host.com");  // use the true flag to indicate the text included is HTML helper.setText(   "<html><body><img src='cid:identifier1234'></body></html>"   **true**);  // let's include the infamous windows Sample file (this time copied to c:/) FileSystemResource res = **new**FileSystemResource(**new**File("c:/Sample.jpg")); helper.addInline("identifier1234", res);  // if you would need to include the file as an attachment, use // addAttachment() methods on the MimeMessageHelper  sender.send(message); |

\ *inline 리소스들은 위에서 본 바와 같이 (위의 경우 identifier1234) Content-ID를 사용하여 mime message에 첨부된다. 당신이 텍스트와 리소스를 추가하는 순서는 굉장히 중요하다. 먼저 텍스트를 추가하고 이후에 리소스를 추가해야 한다. 만약 당신이 다른 방식으로 한다면, 결코 동작하지 않을 것이다.*\