EMAIL 전송

강병준

CommonsEmail

CommonsEmail 컴포넌트 : 이메일에 관련된 컴포넌트

자바를 사용해서 이메일을 보낼 수 있는 각종 API를 제공

CommonsEmail 컴포넌트는 서버 프로그래머들이 많이 사용

다음과 같은 몇몇 목적으로 사용

- 서비스 사용자들에게 이벤트 혹은 정보를 발송하기 위한 광고 메일
- 암호를 잊어버린 사용자에게 암호를 전달해주기 위한 메일
- 서버에 문제가 발생했을 때 관리자에게 알려주기 위한 알람성 메일

CommonsEmail 라이브러리는 이메일을 전송하는 기능만 제공할 뿐 받는 기능은 제공하지 않음

CommonsEmail

SMTP 프로토콜(Simple Mail Transfer Protocol)

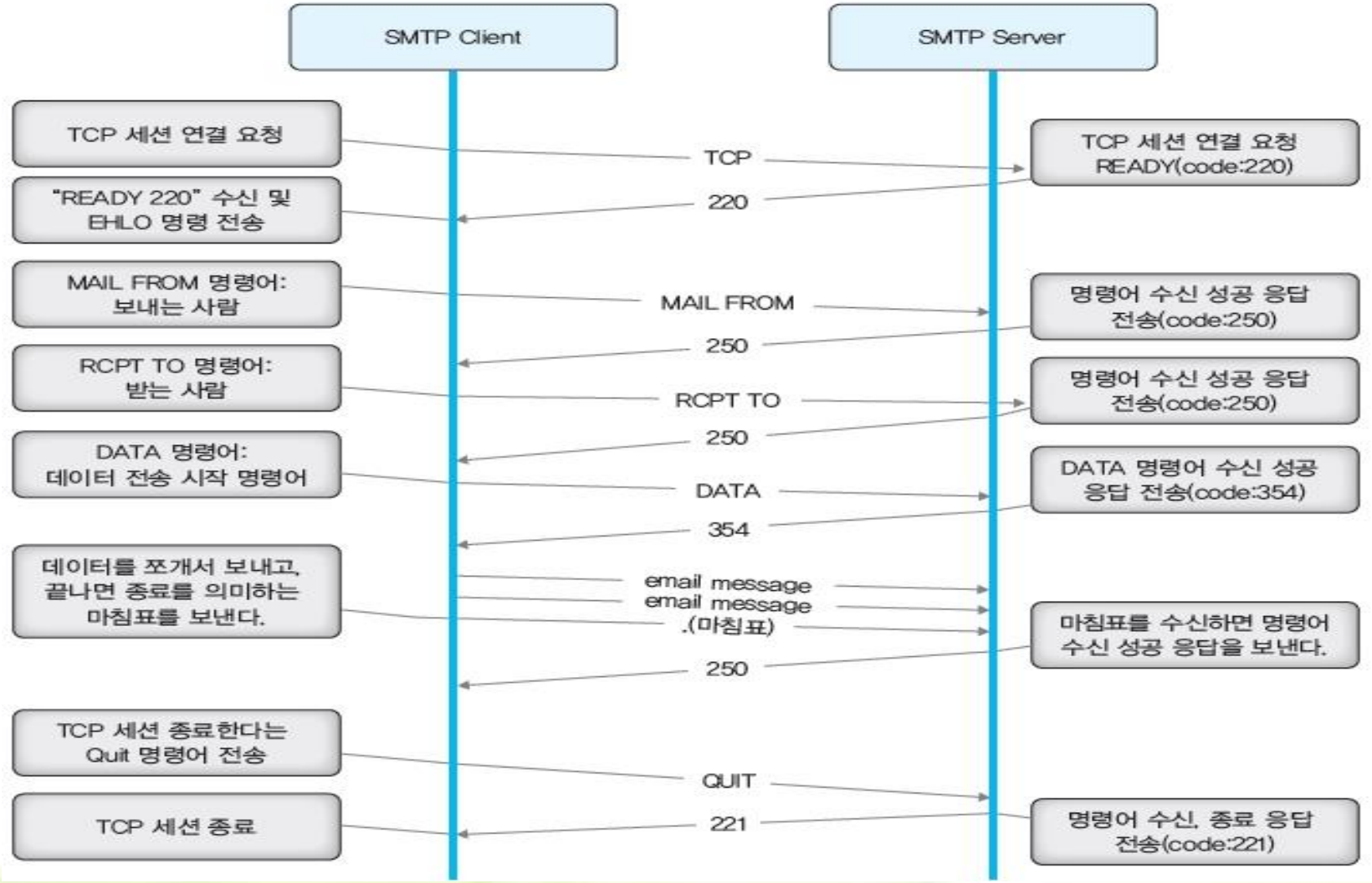
인터넷상에서 이메일을 전송하기 위해서 사용되는 통신 규약 중에 하나

CommonsEmail 라이브러리를 이용해서는 이메일을 보내는 기능만 구현

클라이언트는 SMTP 프로토콜에 의해서 미리 정의된 절차에 따라서 명령어를 전송
서버는 해당 명령어가 성공적으로 처리되었을 때
사전에 정의된 응답 코드에 따라서 다음 절차를 수행

CommonsEmail

SMTP 프로토콜



CommonsEmail

SMTP 프로토콜

SMTP 프로토콜을 지원하는 Email 서버가 필요

리눅스나 유닉스 환경의 서버를 운영하고 있다면 SendMail과 같은 데몬을 설치하여 Mail 서버 구축

윈도우 환경의 서버를 운영하고 있다면 윈도우에서 제공하는 메일 서버를 설치해서 구성

Gmail 계정이 있으면 접근 가능 한 SMTP 서버의 주소

SMTP Server address : smtp.gmail.com

SMTP Server port : 587

SMTP Server account : 여러분의 google 가입 계정(아이디와 패스워드)

CommonsEmail

환경 설정

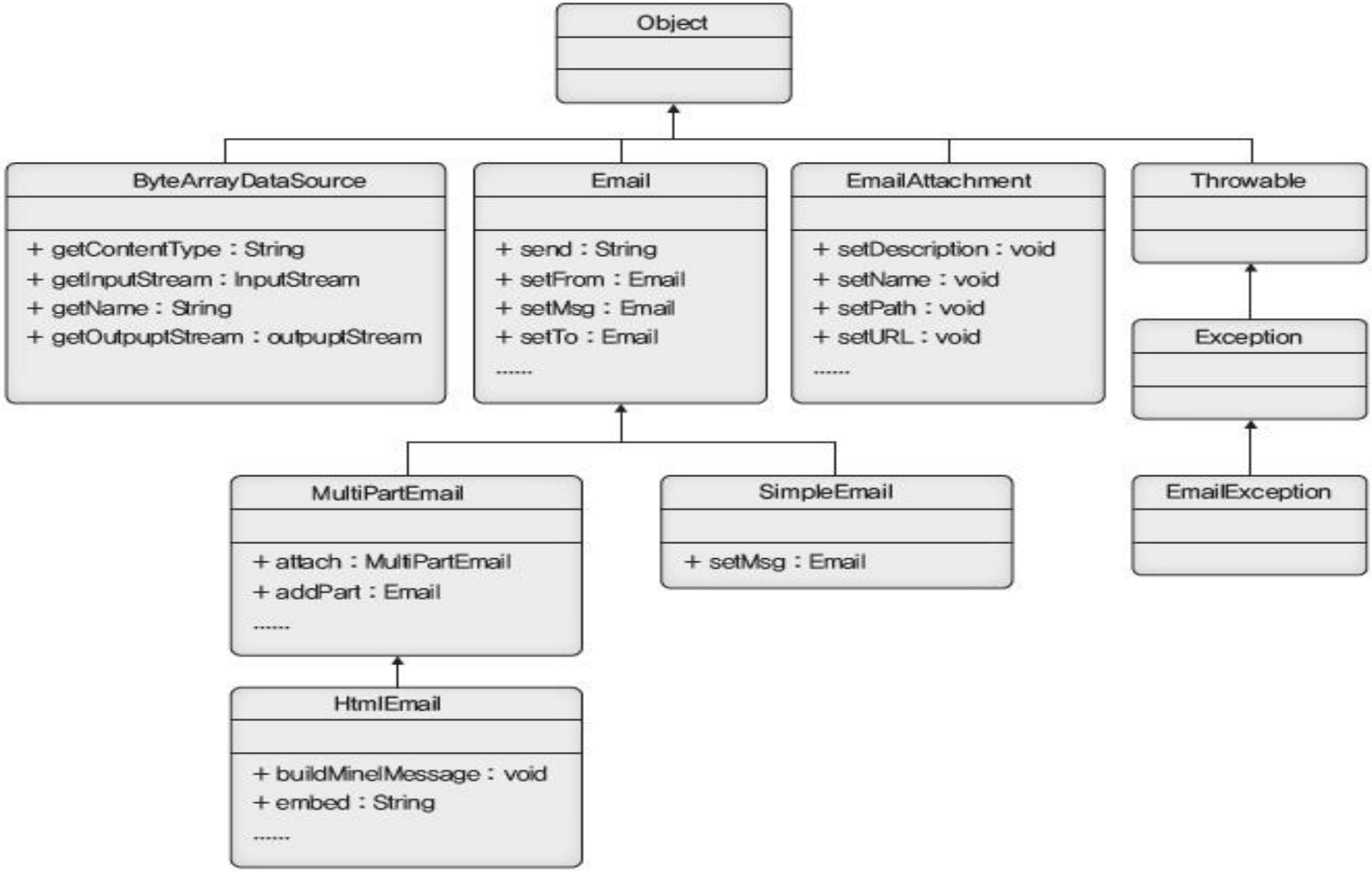
1. CommonEmail 컴포넌트를 이용하여 이메일을 보내기 위해서는 3개의 라이브러리 파일 필요

   'activation. jar', 'common_email_1.2.jar' 그리고 'mail.jar' 파일

   - CommonEmail : http://commons.apache.org/email/download_email.cgi
   - Javamail : http://www.oracle.com/technetwork/java/index-138643.html
   - JAF : http://www.oracle.com/technetwork/java/jaf11-139815.html

2. 다운로드한 zip 파일의 압축을 해제한 뒤에 lib 폴더에 Jar 파일들 확인 (activation.jar/commons-email-1.2.jar/mail.jar)
3. 압축을 푼 jar 파일들을 빌드 패스에 추가

CommonsEmail



CommonsEmail

Email 클래스

Email 클래스 : 이메일을 보내기 위한 여러 정보들
즉, 받는 사람, 보내는 사람, 제목 그리고 메시지 등을
설정할 수 있는 API와 Email 전송 API를 제공

다른 Email 클래스들의 상위 클래스이므로 Email 클래스에서 제공되는 메소드들은 다른 하위 클래스들에게 상속

Email 클래스는 abstract 키워드로 선언된 클래스이므로 Email 클래스를 사용하여 직접 인스턴스 생성 불가

Email 추상 클래스를 구현한 SimpleEmail이나 MultiPartEmail 클래스의 객체를 생성해서 사용

CommonsEmail

Email 클래스

메소드	설명
addBcc	• 선언부 : public Email addBcc(String email) public Email addBcc(String email, String name) public Email addBcc(String email, String name, String charset) • 설명 : BCC란 숨은 참조를 의미한다. 숨은 참조에 포함된 사람도 이메일을 전송받으며 그 사람들은 단지 이메일을 참고할 뿐이다. 위의 메소드들을 사용하면 숨은 참조를 추가할 수 있다. 매개변수 email은 이메일 주소를 의미하며 name은 그 사람의 이름을 의미한다. charset은 문자열 인코딩 셋을 의미한다.
addCc	• 선언부 : public Email addCc(String email) public Email addCc(String email, String name) public Email addCc(String email, String name, String charset) • 설명 : Cc란 참조를 의미한다. Cc에 포함된 사람은 이메일을 단지 참고하면 된다. 참조될 사람의 이메일을 addCc( ) 메소드를 사용하여 추가한다.
addHeader	• 선언부 : public void addHeader(String name, String value) • 설명 : SMTP 프로토콜의 헤더에 특정 값을 추가할 때 사용한다. 매개변수 name은 헤더 이름이며 value는 그 헤더의 값을 의미한다.

CommonsEmail

Email 클래스

addReplyTo	• 선언부 : public Email addReplyTo (String email) public Email addReplyTo (String email, String name) public Email addReplyTo (String email, String name, String charset) • 설명 : 이메일을 읽은 뒤 응답을 위해 받는 사람들을 추가할 때 사용하는 메소드다.
addTo	• 선언부 : public Email addTo (String email) public Email addTo (String email, String name) public Email addTo (String email, String name, String charset) • 설명 : 이메일을 받는 사람의 이메일 주소를 넣는다. 여러 사람에게 보낼 수 있으므로 계속 메소드를 호출해도 무방하다.
send	• 선언부 : public String send( ) • 설명 : Email 객체에 여러 정보를 설정하고 그 데이터들을 전송할 때 호출하는 메소드다. 이 메소드가 호출되면 실제 이메일이 전송된다.

CommonsEmail

Email 클래스

setAuthentication	• 선언부 : public void setAuthentication (String username, String password) • 설명 : SMTP 서버와 연결할 때 SMTP 서버에 접속하기 위한 인증을 받아야 한다. 이때 setAuthentication( ) 메소드를 사용하여 SMTP 계정 정보인 사용자 아이디와 암호를 설정하면 된다. setAuthentication( ) 메소드에 의해서 설정된 계정은 send( ) 메소드에 의해서 SMTP 서버 인증을 받을 때 사용된다.
setFrom	• 선언부 : public Email setFrom(String email) public Email setFrom(String email, String name) public Email setFrom(String email, String name, String charset) • 설명 : 보내는 사람의 이메일 주소를 넣는다. 보내는 사람은 항상 한 명뿐이므로 addTo( ) 메소드와 다르게 setFrom( ) 메소드는 계속 호출하더라도 마지막에 호출된 값에 의해서 보낸 사람의 정보가 설정된다.
setHostName	• 선언부 : public void setHostName(String aHostName) • 설명 : STMP 서버의 Host 이름이나 IP 주소를 입력하도록 한다.

CommonsEmail

Email 클래스

setSmtpPort	• 선언부 : public void setSmtpPort(int aPortNumber) • 설명 : SMTP 기본 포트는 465번이지만 SMTP 서버의 설정에 따라서 port 번호가 다를 수 있다. 이 메소드를 사용해서 포트를 변경하지 않으면 Email 클래스는 기본 포트 465번을 이용하여 SMTP 서버에 연결하려고 한다.
setSSL	• 선언부 : public void setSSL(boolean ssl) • 설명 : 메일 클라이언트와 메일 서버 사이에 데이터를 암호화하기 위해서 사용한다.
setSubject	• 선언부 : public Email setSubject(String subject) • 설명 : 보내고자 하는 메일의 제목을 입력한다.
setMsg	• 선언부 : public static Email set msg(String msg) • 설명 : 메일 본문을 사용한다. 개행 문자(₩n)를 사용해서 줄바꿈을 한다.
setDebug	• 선언부 : public void setDebug(boolena d) • 설명 : 디버깅 관련 정보를 출력하는 메소드다. true인 경우 디버깅 정보가 출력된다.

CommonsEmail

텍스트 전송을 위한 SimpleEmail 클래스

SimpleEmail 클래스 : 특별한 추가 기능 없이 순수한 텍스트 기반의 이메일을 전송하고자 할 때 사용

SimpleEmail 객체를 사용하는 방법

```
사용법 : SimpleEmail [객체 이름] = new SimpleEmail();
사용예 : SimpleEmail email = new SimpleEmail();
```

SimpleEmail 클래스는 public으로 선언된 생성자들이 존재하므로
new 키워드를 사용하여 객체를 인스턴스
매개변수가 없는 기본 생성자만 제공하고 있으므로 매우 간단

일반 메일 보내기

네이버 메일 서버

HostName: smtp.naver.com

SmtpPort:465, 587

일반 메일 보내기

mail.jsp 파일을 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html><html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
<link rel="stylesheet" type="text/css" href="comm.css"></head><body>
  <form action="mail.do">
  <table><caption>메일보내기</caption>
      <tr><th>받는 사람</th><td><input type="email" name="to"
            required="required"></td></tr>
      <tr><th>제목</th><td><input type="text" name="subject"
            required="required"></td></tr>
      <tr><th>내용</th><td><textarea rows="5" cols="30" name="msg"
            required="required"></textarea>
       <tr><th colspan="2"><input type="submit" value="확인"></th></tr>
</table>
</form>
</body>
</html>
```

일반 메일 보내기

CommandProcess 인터페이스 작성

```
package service;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public interface CommandProcess {
        void execute(HttpServletRequest request,
                        HttpServletResponse response);
}
```

일반 메일 보내기

메일을 보내주는 EmailAction 클래스 작성

```
import java.util.Date;
import java.util.Properties;
import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MailAction implements CommandProcess {
   public String requestPro(HttpServletRequest request, HttpServletResponse
            response) {
            Properties p = System.getProperties();
            p.put("mail.smtp.starttls.enable", "true");
            p.put("mail.smtp.host","smtp.naver.com");
            p.put("mail.smtp.auth","true");
            p.put("mail.smtp.port","587");
            p.put("mail.smtp.ssl.protocols","TLSv1.2");
            String to2 = request.getParameter("to");
```

일반 메일 보내기

```
String subject = request.getParameter("subject");
String msg = request.getParameter("msg");
String myEmail = "kbj010@naver.com";  // 본인 이메일
Authenticator auth = new MyAuth();
Session session = Session.getDefaultInstance(p, auth);
MimeMessage mm = new MimeMessage(session);
try {
        mm.setSentDate(new Date());
        InternetAddress from = new InternetAddress();
        from = new InternetAddress(myEmail);
        mm.setFrom(from);

        InternetAddress to = new InternetAddress(to2);
        mm.setRecipient(Message.RecipientType.TO, to);

        mm.setSubject(subject,"utf-8");
        mm.setText(msg,"utf-8");
        mm.setHeader("content-Type", "text/html");

        javax.mail.Transport.send(mm);
        request.setAttribute("msg", "보내기 성공 !!");
```

일반 메일 보내기

```
            }catch (Exception e) {
                    e.printStackTrace();
                    request.setAttribute("msg", "보내기 실패 ㅠㅠ");
            }
            return "mailto";
        }
}
class MyAuth extends Authenticator{
        PasswordAuthentication account;
        public MyAuth() {
                String id = "본인 네이버 아이디"; // kbj010
                String pw = "본인 네이버 암호";
                account = new PasswordAuthentication(id, pw);
         }
         protected PasswordAuthentication getPasswordAuthentication() {
                return account;
         }
}
```

일반 메일 보내기

확장자가 do로 끝나는 요청을 처리하는 Servlet(FrontController)을 작성하고 doGet 메소드에 추가

```
protected void doGet(HttpServletRequest request, HttpServletResponse
                response) throws ServletException, IOException {
        String uri  = request.getRequestURI();
        String path = request.getContextPath();
        String command = uri.substring(path.length()+1);
        CommandProcess action = null;
        if (command.equals("mail.do")) {
                action = new MailAction();
                action.execute(request,response);
                RequestDispatcher rd =request.getRequestDispatcher("result.jsp");
                rd.forward(request, response);
        }
```

일반 메일 보내기

결과를 출력하는 result.jsp 파일을 생성하고 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>결과 페이지</title>
</head>
<body>
${message}
</body>
</html>
```

일반 메일 보내기 2

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.mail.DefaultAuthenticator;
import org.apache.commons.mail.SimpleEmail;
public class MailAction implements CommandProcess{
    public String requestPro(HttpServletRequest request, HttpServletResponse
response) {
            String to = request.getParameter("to");
            String subject = request.getParameter("subject");
            String msg = request.getParameter("msg");
            SimpleEmail se = new SimpleEmail();
            se.setHostName("smtp.naver.com");
            se.setSmtpPort(465); //                    naver ID     naver 암호
            se.setAuthenticator(new DefaultAuthenticator("kbj010", "xxxxxx"));
            try {
                        se.setSSLOnConnect(true);
                        se.addTo(to);
```

일반 메일 보내기 2

```
            se.setFrom("kbj010@naver.com");  // 본인 네이버
            se.setSubject(subject);
            se.setMsg(msg);
            se.send();  // 메일 보내기
            request.setAttribute("msg","메일 보내기 성공");
            }catch (Exception e) {
                        System.out.println(e.getMessage());
                        request.setAttribute("msg","메일 모내기 실패 ㅠㅠ");
            }
            return "mailto";
      }
}
```

HTML5 API

Web Push

1. Server-Sent Events

   HTML5가 등장하기 전까지는 HTML에 서버 푸시를 위한 표준화된 기술이 없었기 때문에 웹에서 실시간 정보를 받아와야 할 때 외부 플러그인을 이용하거나 서버 푸시를 흉내 낸 Ajax 폴링(polling) 기법 등을 사용했습니다. 하지만 플러그인 종속적인 웹은 해당 플러그인을 설치해야 한다는 불편함이 있으며 폴링처럼 주기적인 요청을 통한 구현은 쓸모없는 요청의 발생으로 인한 대역폭의 낭비가 불가피하였습니다.
   HTML5의 Server-Sent Events(이하 SSE)는 이러한 문제 없이 서버가 필요할 때마다 클라이언트에게 데이터를 줄 수 있게 해주는 서버 푸시 기술입니다.

2. SSE의 장점

   1) 전통적인 HTTP를 통해 통신하므로 다른 프로토콜이 필요 없음
   2) 재 접속 처리 같은 대부분의 저 수준 처리가 자동으로 됩니다.
   3) 표준 기술답게 IE를 제외한 브라우저 대부분을 지원합니다.

Web Push

- 클라이언트에서는 EventSource 객체를 생성하는 것만으로 주기적인 서버 호출이 일어납니다.

  var eventSource = new EventSource("server.jsp");
- 데이터 수신 이벤트: 서버로 부터 전달받는 데이터를 처리하기 위해 수신 이벤트를 정의합니다.
- 이 이벤트 역시 EventSource 객체를 통해 정의하면 됩니다.
- 이벤트로 전달되는 data 속성으로 수신 데이터를 액세스 할 수 있다

```
eventSource.addEventListener("message",
    function(e){
      alert(e.data);
    }
,false);
```

Web Push

1. 서버측에서 클라이언트로 전달하는 데이터는 일반적인 텍스트 형태이지만 그 포맷이 정해져 있으며 몇 가지 규칙을 따라야 합니다.
2. MIME 타입: 서버 데이터는 text/event-stream 라는 MIME 타입으로 제공
3. 문자 인코딩: 서버 데이터의 문자 인코딩은 UTF-8 형식
4. 빈 줄은 이벤트를 구분하는 역할
5. 주석은 :(콜론)
6. 데이터는 '필드 명: 필드 값' 형식

data : 서버가 전달할 실제 데이터를 정의

retry: 반복 주기를 설정(단위: millisecond)

event: 이벤트 이름을 지정(지정하지 않으면 기본값인 message 가 된다)

id: 이벤트 id를 지정

Web Push

webPush.jsp 파일

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html><html><head><meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Insert title here</title>
<script type="text/javascript">
function pushStart() {
    var es = new EventSource("push.action");
    es.addEventListener("message",function(event) {
            document.getElementById("disp").innerHTML=event.data;
    });
}
</script></head>
<body>
    <button onclick="pushStart()">웹푸시 시작</button><p>
    <div id="disp"></div>
</body>
</html>
```

Web Push

FrontController의 doGet 메소드 수정

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //서블릿에서 직접 출력, 응답할 데이터 타입을 설정, SSE로 응답
            response.setContentType("text/event-stream");
            response.setCharacterEncoding("UTF-8");
            //브라우저에 출력하기 위한 Writer 가져오기
            PrintWriter out = response.getWriter();
            Random r = new Random();
            int su = r.nextInt(45)+1;
            //데이터 전송 - data는 예약어이고 마지막 빈 줄은 이벤트를 구분하기 위해서
            out.write("data:" + su + "₩n₩n");
            //2초 대기 - 2초마다 보내집니다. 기재하지 않으면 1초 정도
            try{          Thread.sleep(2000);
            }catch(Exception e){ System.out.println(e.getMessage());  }
            out.close();
}
```

Web Socket

1. 순수 웹 환경에서 실시간 양방향 통신을 위한 Spec이 바로 '웹 소켓 (Web Socket)'
2. 웹 소켓은 웹 서버와 웹 브라우저가 지속적으로 연결된 TCP 라인을 통해 실시간으로 데이터를 주고 받을 수 있도록 하는 HTML5의 새로운 사양으로 웹 소켓을 이용하면 일반적인 TCP소켓과 같이 연결지향 양방향 전이중 통신이 가능
3. 웹 소켓이 필요한 경우
   1) 실시간 양방향 데이터 통신이 필요한 경우
   2) 많은 수의 동시 접속자를 수용해야 하는 경우
   3) 브라우저에서 TCP 기반의 통신으로 확장해야 하는 경우
   4) 개발자에게 사용하기 쉬운 API가 필요할 경우
   5) 클라우드 환경이나 웹을 넘어 SOA 로 확장해야 하는 경우

Web Socket

1. 웹 소켓 역시 일반적인 TCP 소켓 통신처럼 웹 소켓 역시 서버와 클라이언트간 데이터 교환이 이루어지는 형태로 다른 HTML5 스펙과는 달리 클라이언트 코드만으로 실행 가능한 코드를 만들 수 없습니다.
2. 웹 소켓 클라이언트
3. 웹 소켓이 제공하는 클라이언트 측 API는 매우 심플하다. 기본적인 서버 연결, 데이터 송신, 데이터 수신만 정의하면 나머지는 일반적인 스크립트 로직입니다.
4. 웹 소켓이 동작하기 위해서 제일 처음 서버와 연결이 되어야 하는데 HTML5가 제공하는 WebSocket 객체를 통해 서버 연결을 수행합니다.

```
var wSocket = new WebSocket("ws://yourdomain/demo");
```

1. 서버와 연결이 되면 이제부터 데이터를 주고 받을 수 있게 되는데 WebSocket 객체의 send 함수로 데이터를 서버로 송신할 수 있습니다

```
wSocket.send( " 송신 메시지 " );
```

1. 서버에서 푸시(전송)하는 데이터를 받으려면 message 이벤트를 구현
2. wSocket.onmessage = function(e){ //매개변수 e를 통해 수신된 데이터를 조회할 수 있다
3. open 이벤트: 연결이 설정되면 발생
4. close 이벤트: 연결이 끊어지면 발생

Web Socket

1. **자바 웹 소켓**
2. 웹 소켓 클래스 생성: @ServerEndpoint("/접속할 경로")
3. 메시지 전송 메소드

```
@OnMessage
  public void onMessage(String message, Session session)
    throws IOException, InterruptedException { }
```

위 메소드에서 message는 클라이언트가 전송한 메시지

Session은 클라이언트인데 메시지를 전송하고자 할 때는

session.getBasicRemote().sendText("메시지");

1. 서버가 열릴 때, 닫힐 때, 에러 날 때 호출되는 메소드

```
 @OnOpen
  public void onOpen() {}
  @OnClose
  public void onClose() {}
@OnError
   public void onError(Throwable t) throws Throwable { }
```

Web Socket

웹 소켓 클래스 생성

```
 //웹 소켓 클래스를 만들고 호출하기 위한 주소를 설정하는 어노테이션
@ServerEndpoint("/websocket")
public class WebSocketTest {
    private static final Set<Session> connections =  new HashSet<>();
  // 클라이언트가 새로 접속할 때마다 하나의 Session 객체가 생성된다.
  // Session객체를 컬렉션에 보관하고 데이터를 전송할 때마다 사용한다
  private Session session;
 @OnMessage
  public void incoming(String message) {
        if  (message==null || message.trim().equals("")) return;
       broadcast(message);
  }
```

Web Socket

```
 //서버가 오픈될 때 호출되는 메소드
@OnOpen
 public void start(Session session) {
 // 접속자마다 한개의 세션이 생성되어 데이터 통신수단으로 사용됨
       this.session = session;
       connections.add(session);
       String message ="connect";
       broadcast(message);
}
//서버가 닫힐 때 호출되는 메소드
@OnClose
public void end() {
     connections.remove(session);
     String message = "disconnect";
     broadcast(message);
}
@OnError
public void onError(Throwable t) throws Throwable {
      System.err.println("Chat Error: " + t.toString());
}
```

Web Socket

```
    // 현재 세션으로부터 도착한 메시지를 모든 접속자에게 전송한다
    private void broadcast(String msg) {
        for (Session client : connections) {
            try {   synchronized (client) {
                        client.getBasicRemote().sendText(msg);
                    }
            } catch (IOException e) {
                // 오류 발생(클라이언트 퇴장)하면 해당 클라이언트를 서버에서 제거한다
                // System.err.println("Chat Error: Failed to send message to client:"+e);
                 connections.remove(client);
                 try {   // 접속 종료
                     client.close();
                 } catch (IOException e1) {             // Ignore
                 }
                 // 한 클라이언트의 퇴장을 모든 이용자에게 알린다
                 String message = "disconnect";
                 broadcast(message);
            }
        }
    }
}
```

chat.jsp시작하기

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html><html><head><meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Insert title here</title>
<style type="text/css">
            table {     height: 450px; border: 2px solid green; width:90%;
                        table-layout: fixed; overflow: hidden; }
            div { height: 400px;  overflow: scroll; }
</style>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
            var websocket = // 웹 소켓을 생성
                        new WebSocket("ws://127.0.0.1:8080/ch18/websocket")
            var disp;
// 웹 소켓이 열릴때 호출되는 메소드 설정
            websocket.onopen = function() {
                        disp = document.getElementById("disp");
                        disp.innerHTML += "연결되었습니다<br>";
                        opener.close();
            }
            websocket.onclose = function(event) {
                        disp.innerHTML += "종료되었습니다<br>";
            }
```

chat.jsp

```
websocket.onmessage = function(event) {
        disp.innerHTML += event.data+"<br>";
}
websocket.onerror = function(event) {
        disp.innerHTML += "어쿠 에러<br>";
}
function webstart() {
        var message = document.getElementById("content").value;
        var name = document.getElementById("name").value;
        websocket.send(name+" > "+message);
        document.getElementById("content").value  = "";
        var objDiv = document.getElementById("disp");
        objDiv.scrollTop = objDiv.scrollHeight;
}
```

chat.jsp

```
        function init() {
                cont = document.getElementById("content");
                cont.addEventListener('keyup',function(event) {
                        // 누른 키보드 값 가져오기
                        var keycode = event.keyCode?event.keyCode:event.which;
                        if (keycode==13) webstart(); // 누른 키가 엔터이면 send()호출
                        event.stopPropagation(); // 이벤트 전달을 하지 않음
                });
        }
</script></head><body onload="init()">
        별명 : <input type="text" name="name" id="name">
        <table border="1">
        <tr><td height="400" id="a"><div id="disp"></div></td></tr>
        <tr><td height="50"><input type="text" id="content"><br>
                        <button onclick="webstart()">웹채팅</button></td></tr>
        </table>
</body>
</html>
```