Servlet Method

# 수업차시	2

01. Servlet Method

01-01. HTTP 데이터 전송 방식

01-01-01. 데이터 전송 구조

1. 브라우저에서 요청 정보를 HTTP 객체에 담아 전송한다.
    - 요청과 응답을 주고받는 패킷은 요청 정보와 응답 정보를 직렬화한 Byte단위의 문자열 데이터로, 인코딩과 디코딩이 필요하다.
2. 전달받은 HTTP객체를 서버(=Tomcat)이 해석하여 요청을 처리할 서블릿을 호출한다. 서블릿의 service() 메소드에서는 request, response 요청 정보를 가지고 처리 로직을 거쳐 응답한다.
    - HttpServletRequest는 ServletRequest 타입을 상속 받아 구현하였으며, HTTP 프로토콜의 정보를 담고 있기 때문에 실제 사용 시에는 HttpServletRequest 타입으로 다운캐스팅해서 사용해야 한다.

01-01-02. 데이터 전송 방식

1. get 방식
    - URL창에 “ **?** “ 뒤에 데이터를 입력하는 방법(= 쿼리스트링)으로 전송한다.
        - URL 쿼리스트링 예시

        ```
        http://localhost:8080/sendMessage?message=abc&code=20
        ```

    - 데이터를 Header에 포함하여 전송한다.
    - 데이터 크기 한계가 있으며 보안에 취약하다.
    - 데이터 조회/검색 시 사용된다.
2. post 방식
    - 데이터를 body에 포함하여 전송한다.
    - 데이터가 URL에 노출 되지 않으므로 GET 방식보다 상대적으로 보안적이다.
    - 전송 데이터 크기에 제한이 없다. 단, 요청 시간의 제한은 있다.
    - 데이터 생성/수정 시 사용 된다.

01-02. 데이터 전송 방식에 따른 Servlet Method

01-02-01. Servlet Method

- 서블릿이 get/post의 두 방식 중 하나로 요청 정보를 전달 받으면, request와 response를 전달하면서 해당하는 처리 메소드(doGet() 메소드 또는 doPost() 메소드)를 호출한다.

  → 즉, 톰캣 서블릿 컨테이너가 요청 url로 매핑된 서블릿 클래스의 인스턴스를 생성하여 service method를 호출하고 HttpServlet을 상속 받아 오버라이딩한 현재 클래스의 doGet() 또는 doPost() 메소드가 동적 바인딩에 의해 호출된다.

  ⇒ 이때, 서블릿 메소드에 대하여 반드시 ServletException 처리를 해야 한다.

- HTML에서 method 속성을 이용해 방식을 결정하며, default는 get 방식이다.

01-02-02. doGet() 메소드

- 클라이언트에서 데이터 전송 방식을 get으로 전송하면 호출되는 메소드이다.
- GET 방식의 데이터는 HTML charset에 기술된 인코딩 방식으로 브라우저가 한글을 이해하고, '%' 문자로 URLEncoder를 이용해 변환 후 url 요청으로 전송한다.

  이때 header의 내용은 ascii 코드로 전송되므로 어떤 언어든 서버의 설정 인코딩 방식과 맞기만 하면 해석하는데 문제가 없으므로 한글이 깨지지 않는다.

- GET 요청은 보통 서버의 리소스를 가져오는 행위를 요청하는 http 요청 방식이기에 별도의 데이터가 필요 없어 요청 본문(= 페이로드)는 해석하지 않는다.
- 쿼리스트링으로 보내는 데이터가 노출되므로 보안에 취약하다는 단점이 있으나, 속도 면에서 더 빠르므로 검색 기능에 적합하다.

01-02-03. doPost() 메소드

- 클라이언트에서 데이터 전송 방식을 post로 전송하면 호출되는 메소드이다.
- POST 요청은 서버의 리소스에 내용을 추가하는 요청이기 때문에 요청하는 데이터가 필요한 경우가 대부분이다. 서버의 리소스에 추가해야 하는 정보를 페이로드에 key&value 방식으로 담아 전송하는데, 헤더와는 별개로 URLEncoder를 이용하지 않고 페이지 meta에 기술된 charset에 따라 UTF-8로 해석된 데이터를 서버로 전송한다.
- 기본적으로 서버 단에서 페이로드를 디코딩 하는 방식은 지정되어 있지 않고, 인코딩된 방식을 명시하지 않으면 기본 ISO-8859-1로 해석하므로 값을 꺼내오면 한글인 글자가 깨지는 현상이 발생한다.
  - request 객체의 getCharacterEncoding() 메소드를 호출해보면 null을 반환한다.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {

  System.out.println(request.getCharacterEncoding());

}
```

    - 따라서 parameter를 꺼내오기 전에 이를 해석할 인코딩 방식을 UTF-8로 지정해야 브라우저에서 요청한 인코딩 방식과 일치하여 한글 깨짐 현상을 막을 수 있다.

- request의 body 영역에 데이터를 담아 보내므로 데이터를 많이 보낼 수 있으나 비교적 속도가 느리다. 또한 개발자 도구를 통해 담은 데이터를 볼 수 있으므로 보안이 엄청 뛰어난 수준은 아니다.

02. Servlet Method Parameter

02-01. HttpServletRequest

02-01-01. HttpServletRequest의 역할

- HTTP Servlet을 위한 요청 정보(request information)를 제공하는 메소드를 지정한다.
- Interface 구현은 컨테이너가 알아서 설정하므로 메소드만 이용한다.

  (상속 받는 인터페이스 : javax.servlet.ServletRequest (interface))
- HttpServletRequest 객체의 주요 메소드

method 명	내용
getParameter(String)	client가 전송한 값의 명칭이 매개변수와 같은 값 가져옴
getParameterNames()	client가 전송한 값의 명칭 가져옴
getParameterValues(String)	client가 전송한 값이 여러 개이면 배열로 가져옴
getParameterMap()	client가 전송한 값 전체를 Map방식으로 가져옴
setAttribute(String, object)	request 객체로 전달할 값을 String 이름-Object 값으로 설정
getAttribute(String)	매개변수와 동일한 객체 속성 값 가져옴
removeAttribute(String)	request객체에 저장된 매개변수와 동일한 속성 값 삭제
setCharacterEncoding(String)	전송 받은 request객체 값들의 CharaterSet 설정
getRequestDispatcher(String)	컨테이너 내에서 request, response객체를 전송하여 처리한 컴포넌트(jsp파일 등)를 가져옴 (forward() method와 같이 사용)

02-01-02. Request Header

1. General header
   - 요청 및 응답 모두에 적용되지만 최종적으로는 body에 전송되는 것과 관련이 없는 헤더이다.
2. Request header
   - Fetch될 리소스나 클라이언트 자체에 대한 상세 정보를 포함하는 헤더이다.
3. Response header
   - 위치나 서버 자체와 같은 응답에 대한 부가적인 정보를 갖는 헤더이다.
4. Entity header
   - 컨텐츠 길이나 MIME타입과 같은 엔티티 body에 대한 상세 정보를 포함하는 헤더이다.

     (요청 응답 모두 사용되며, 메시지 바디의 컨텐츠를 나타내기에 GET요청은 해당하지 않는다.)
5. getHeader() 메소드로 확인 가능한 값

header 속성	값
accept	요청을 보낼 때 서버에게 요청할 응답 타입
accept-encoding	응답 시 원하는 인코딩 방식
accept-language	웅답 시 원하는 언어
connection	HTTP 통신이 완료된 후에 네트워크 접속을 유지할지 결정 (기본값: keep-alive = 연결을 열린 상태로 유지)
host	서버의 도메인 네임과 서버가 현재 Listening 중인 TCP포트 지정 (반드시 하나가 존재. 없거나 둘 이상이면 404)
referer	이 페이지 이전에 대한 주소
sec-fetch-dest	요청 대상
sec-fetch-mode	요청 모드
sec-fetch-site	출처(origin)와 요청된 resource 사이의 관계
sec-fetch-user	사용자가 시작한 요청일 때만 보내짐 (항상 ?1 값 가짐)
cache-control	캐시 설정
upgrade-insecure-requests	HTTP 메시지 전송 시 보안 적용
user-agent	현재 사용자가 어떤 클라이언트(OS, browser 포함)을 이용해 보낸 요청인지 명시

▼ request header 중 sec-fetch 속성값 및 활용 참고

- sec-fetch-site
  - `cross-site` : 요청 개시자와 resource를 호스팅하는 서버가 다른 사이트일 경우
  - `same-origin` : 요청 개시자와 resource를 호스팅하는 서버가 동일한 출처(origin)를 가질 경우

    - `same-site` : 요청 개시자와 resource를 호스팅하는 서버가 동일한 scheme, 도메인/서브도메인을 가지지만 port가 다른 경우
    - `none` : 요청이 사용자로부터 시작되었을 경우. (ex. 주소창에 URL 입력, 브라우저 창에 파일 끌어다 놓기 등.)
- sec-fetch-mode
    - `cors` : CORS protocol 요청
    - `navigate` : HTML document 사이 이동 시
    - `no-cors` : no-cors 요청
    - `same-origin` : 요청 중인 resource와 동일한 출처
    - `websocket` : websocket 연결을 설정하기 위한 요청
- sec-fetch-dest
    - 서버는 요청이 사용되는 방식이 적절한지에 따라 요청의 허용 여부를 결정
- sec-fetch-user
    - 서버는 해당 속성값을 통해 document, iframe 등의 탐색 요청이 사용자로부터 시작된 것인지 식별 가능

02-02. HttpServletResponse

02-02-01. 요청에 대한 Servlet의 역할

1. 요청을 받는다.
    - HTTP method GET/POST 요청에 따라 parameter로 전달받은 데이터를 꺼내어 활용할 수 있다.
2. 비지니스 로직을 처리(DB접속과 CRUD에 대한 로직 처리)한다.
    - 서비스 메소드를 호출하여 처리 로직을 수행하도록 한다.
3. 응답한다.
    - 문자열로 동적인 웹(HTML 태그)페이지를 만들고 스트림을 이용해 내보내거나 응답 데이터를 담아 결과 페이지를 호출한다.

02-02-02. HttpServletResponse의 역할

- 요청에 대한 처리 결과를 작성하기 위해 사용하는 객체이다.
- Interface 구현은 컨테이너가 알아서 설정하므로 메소드만 이용한다.

  (상속 받는 인터페이스 : javax.servlet.ServletResponse (interface))

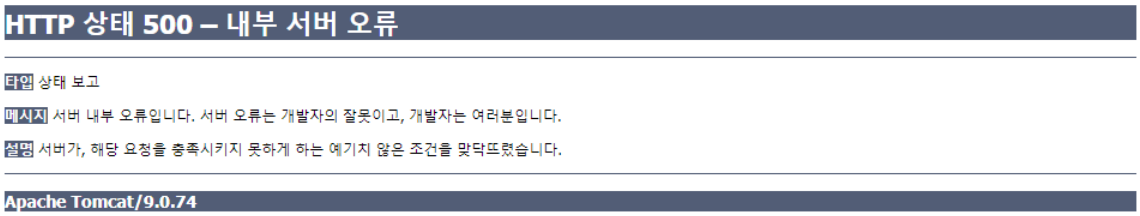
- HttpServletResponse 객체의 주요 메소드

method 명	내용
setContentType(String)	응답으로 작성하는 페이지의 MIME type을 설정
setCharacterEncoding(String)	응답하는 데이터의 CharacterSet을 지정
getWriter()	페이지에 문자 전송을 위한 Stream을 가져옴
getOutputStream()	페이지에 byte단위의 전송을 위한 Stream을 가져옴
sendRedirect(String)	client가 매개변수의 페이지를 다시 서버에 요청함

02-02-03. Exception Handler

1. sendError() 메소드를 사용하여 에러를 발생시킬 수 있다.

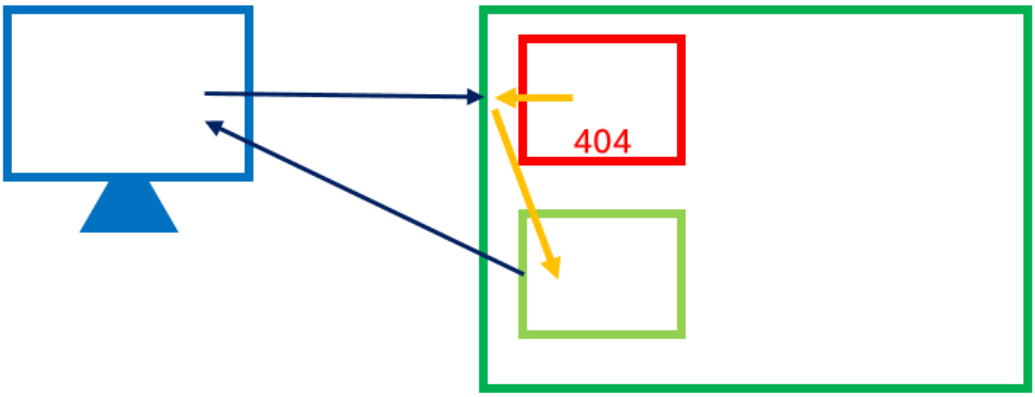
```
// response.sendError(에러상태 코드, "현출할 에러 메시지");
response.sendError(500, "서버 내부 오류입니다. 서버 오류는 개발자의 잘못이고, 개발자는 여러분입니다.");
```



2. web.xml 파일에 <error-page>를 등록하여 에러 상태 코드에 해당하는 에러가 발생하면 서블릿에서 가로채도록 설정할 수 있다.

```
<error-page>
    <error-code>404</error-code>
    <location>/showErrorPage</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/showErrorPage</location>
</error-page>
```

- 위 설정은 아래 그림처럼 원래라면 일반적인 오류 페이지가 반환 될 시점에 오류 송출을 가로채 지정된 페이지로 사용자에게 반환하는 구조이다.



02-02-03. 스트림을 활용한 동적 페이지 응답

1. 응답 헤더의 타입을 설정한다.

```
response.setContentType("text/html");
```

   - 기본값은 text/plain이다.
   - HTML태그를 사용하면 요청 시 text/html도 응답으로 수락 가능하도록 헤더 설정이 되어있어 자동으로 text/html로 인식한다.

2. 응답할 내용의 인코딩 방식을 설정한다.

```
response.setCharacterEncoding("UTF-8");
```

   - 응답 시 별도 인코딩을 지정하지 않으면 기본으로 설정된 인코딩 방식(ISO-8859-1)을 따르므로 한글 데이터가 깨져서 현출된다.
   - 따라서 응답할 인코딩 방식이 UTF-8임을 응답 헤더에 설정하면 브라우저가 이를 해석할 때 UTF-8로 인식하고 해석하여 한글 데이터가 정상적으로 현출된다.
   - 응답 헤더와 인코딩 설정을 동시에 할 수도 있다.

```
response.setContentType("text/html; charset=UTF-8");
```

   ※ 반드시 getWriter()로 스트림을 얻어오기 전에 설정해야 한다.

3. 응답을 위한 스트림을 가져와 응답을 내보낸다.

```
PrintWriter out = response.getWriter();
out.print(responseBuilder.toString());
```

4. 응답에 사용한 스트림을 닫는다.

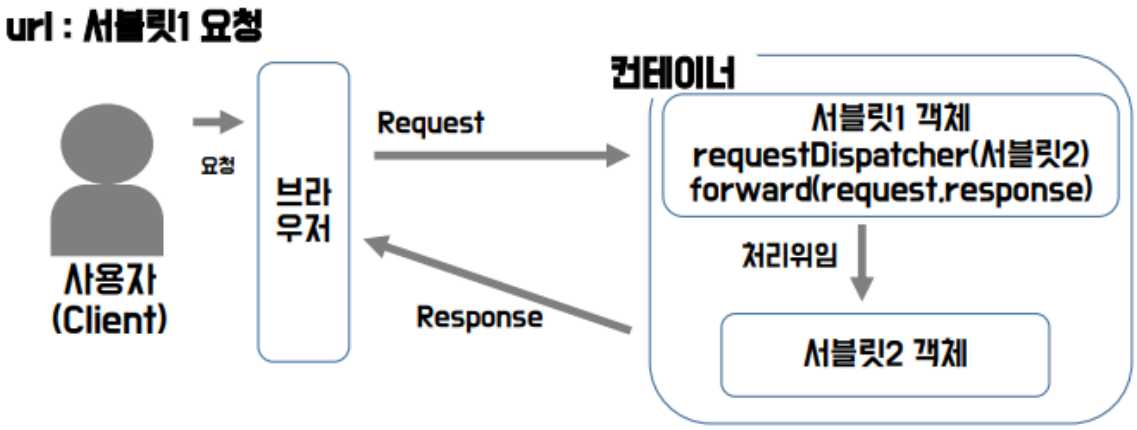
```
out.close();
```

03. Forward & Redirect

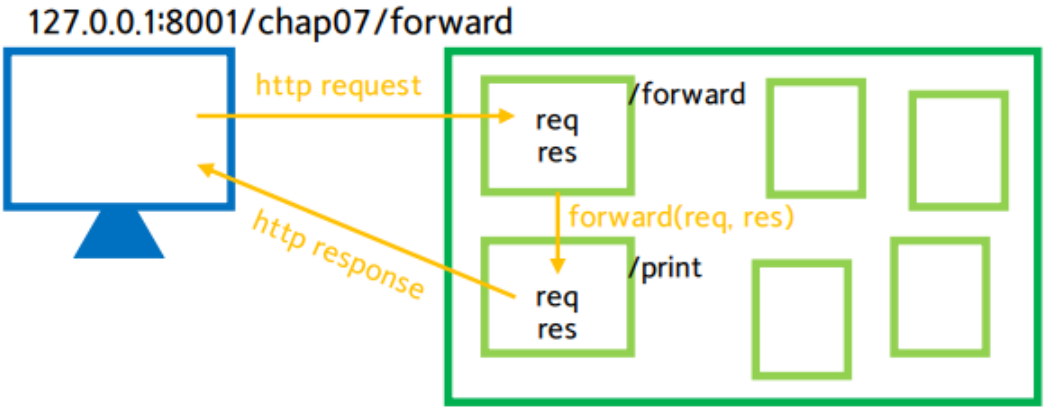
03-01. forward()

03-01-01. forward() 역할

- 컨테이너 내에서 처음 요청 받은 페이지가 요청 데이터 (HttpServletRequest, HttpServletResponse)를 다른 페이지에 전송하여 처리를 요청하고, 자신이 처리한 것처럼 응답한다.
- 클라이언트가 요청한 url주소(페이지)가 변경되지 않는다.



03-01-02. forward() 구조



1. 서버(= Tomcat)를 실행하면 내부 서블릿 컨테이너에 작성한 서블릿이 다 올라가고 doGet, doPost 등 매핑된 url로 연결된다.
2. HTTP에 의한 요청을 전달하면 헤더의 문자열을 파싱해서 헤더, 데이터, 응답 대상 브라우저 등을 request와 response 객체로 쪼개어 doGet() 또는 doPost() 메소드로 보낸다.
3. 이때 요청 받은 서블릿에서 다른 서블릿으로 request, response 객체를 담아 forward하면 동일한 속성을 가지고 처리 권한을 위임한다.
4. 서버 내부에서 다른 서블릿에 위임했으나 요청받은 서블릿이 응답하는 것처럼 처리하므로 위임한 경로를 노출하지 않는다. 즉, 처리하는 서블릿이 변경되었어도 url이 변경되지 않는다.

03-01-03. forward() 활용 예시

1. 요청받은 서블릿에서 forward하는 코드 예시

```
request.setAttribute("userId", userId);

RequestDispatcher rd = request.getRequestDispatcher("print");
rd.forward(request, response);
```

   - 다른 서블릿으로 요청하기 위한 데이터는 request에 setAttribute()로 담아 전달한다.
   - attribute도 일종의 Map 형식으로, key-value 방식으로 값을 저장할 수 있다.
   - RequestDispatcher는 '배차관리자'라는 뜻으로, 서블릿 위임 시 어디로 보낼지 결정하는 역할을 한다.

2. 위임(forward)받은 서블릿에서 속성 값을 꺼내는 코드 예시

```
String userId = (String) request.getAttribute("userId");
```

- forward 받은 서블릿에서도 요청 방식이 get이면 doGet메소드를, 요청 방식이 post이면 doPost메소드를 호출한다.
- request에 전달 정보를 담았으므로, 위임받은 서블릿에서 사용하기 위해 request에서 getAttribute()로 담아 전달한다.
- forward할 때 전달한 request와 response의 모든 정보를 이용해 새로운 request, response를 만들고 그 정보를 이용해 다시 http 메소드에 맞는 서블릿의 doGet 혹은 doPost를 요청하는 방식이다.

  → 깊은 복사를 이용해 값을 그대로 복사했기 때문에 내부에 존재하는 헤더 정보나 인스턴스는 그대로 유지한다.

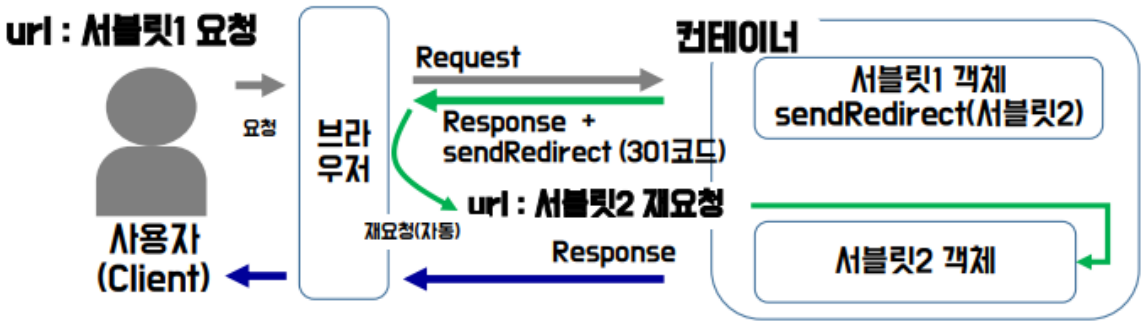
03-02. redirect()

03-02-01. redirect() 역할

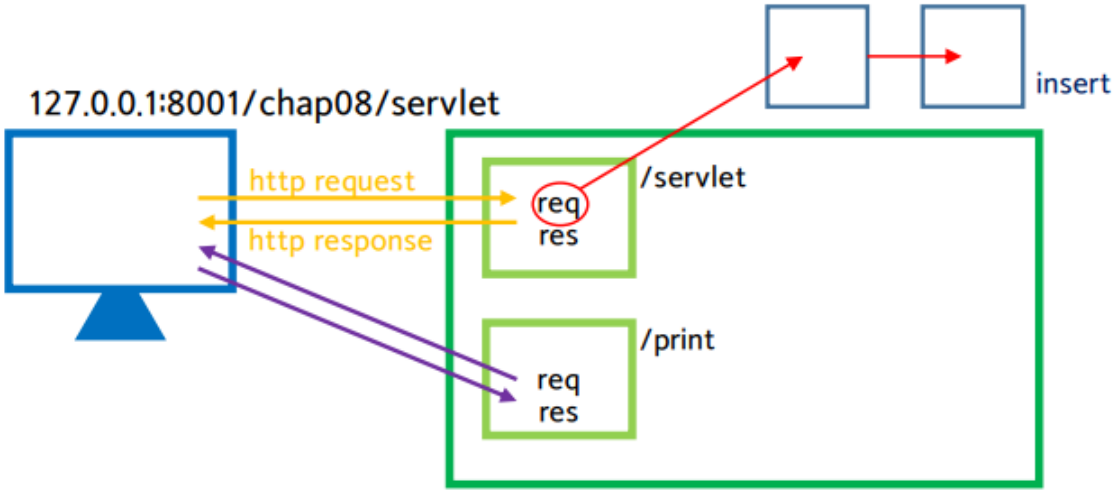
- 클라이언트 브라우저에게 “(매개변수로 등록한) 페이지를 재요청하라”고 응답한다.

  (응답 상태 코드 : 301, 302)
- encodeRedirectURL은 매개변수(URL)에 Session ID 정보를 추가하여 재요청 처리한다.
- 클라이언트가 별도로 다른 페이지 요청을 하지 않아도 url주소(페이지)가 변경된다.

  (브라우저 요청에 따라 서버가 알아서 해당 페이지를 요청하며, 쿼리스트링으로 별도의 데이터를 전송하지 않으면 요청 데이터가 없다.)



03-02-02. redirect() 구조



1. 서버(= Tomcat)를 실행하면 내부 서블릿 컨테이너에 작성한 서블릿이 다 올라가고 doGet, doPost 등 매핑된 url로 연결된다.
2. HTTP에 의한 요청을 전달하면 헤더의 문자열을 파싱해서 헤더, 데이터, 응답 대상 브라우저 등을 request와 response 객체로 쪼개어 doGet() 또는 doPost() 메소드로 보낸다.
3. 이때 요청 받은 서블릿에서 재요청할 url을 담아 sendRedirect로 응답하면 처리하는 서블릿으로 브라우저가 재요청을 보내도록 한다.
    - 302 status code를 보냄으로써 요청 url을 바꿔 다시 요청하라는 의미를 전달한다.
        - 사용자 url 재작성이라고 불리는 redirect 방식은 302번 응답 코드인 경우 요청에 대한 처리를 완료하였고, 사용자의 url을 강제로 redirect 경로로 이동시키라는 의미이다.

03-02-03. redirect() 활용 예시

1. 요청받은 서블릿에서 sendRedirect하는 코드 예시
    a. 타 사이트로 이동하는 경우

```
response.sendRedirect("http://www.naver.com");
```

    - 브라우저의 개발자도구 network 탭을 보면 302번 코드와 함께 naver 사이트로 이동하는 것을 확인할 수 있다.
    - 응답 헤더 작성은 General Header의 302번 코드와 Response header의 Location 헤더값에 redirect할 경로를 포함하여 응답한다.

    b. 타 서블릿으로 이동하는 경우 (타 서블릿의 url pattern 작성)

```
response.sendRedirect("redirect");
```

- redirect하면 url이 재작성되어 새로고침할 때 redirect된 페이지에 대한 요청을 반복한다.

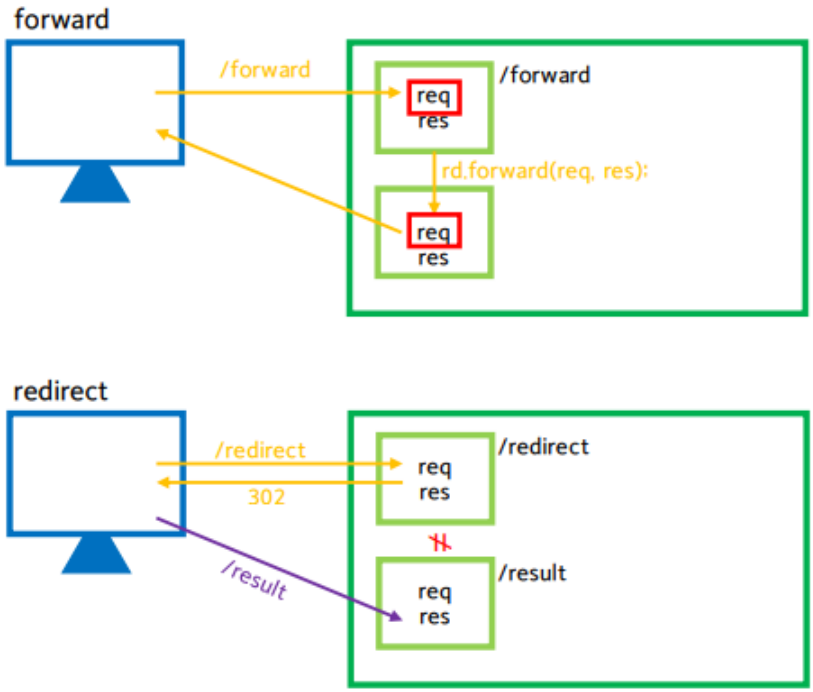
  즉, 이전 요청에 포함된 정보는 남아있지 않고 url이 변경되는 것이 redirect의 특징이다.

- HTTP 요청은 요청 시 connection을 맺었다 끊고, 응답 시에도 connection을 맺었다 끊으므로 요청 단위당 request객체는 한 개만 생성된다.\

  따라서 첫 요청 시의 request와 redirect된 페이지의 request는 서로 다른 객체이므로, redirect를 쓰면 이전 서블릿의 request 객체 속성 값을 공유해서 사용할 수 없다. (이를 해결하기 위해 쿠키 및 세션 객체를 활용한다.)

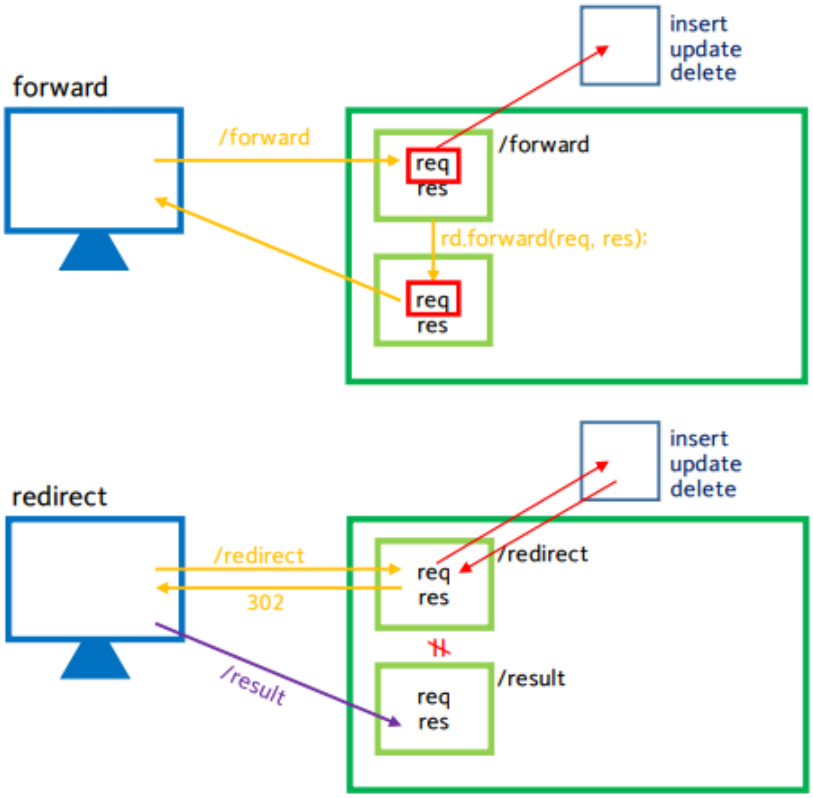
03-03. forward()와 redirect() 비교

03-03-01. 구조 비교



- forward는 서버 내부에서 요청을 동일한 파라미터 객체로 위임하여 응답하고, 마치 요청받은 서블릿이 응답한 것처럼 보여준다.
- redirect는 요청한 서블릿이 302 status로 응답을 보내주고, 브라우저가 응답 받은 방향으로 다시 요청을 보내 결과 페이지를 반환 받는 구조이다.

03-03-02. CURD 로직에서의 활용



- 새로고침을 반복할 때마다 동일한 요청이 반복되는 forward는 대게 조회 기능에 사용한다.
  - select 처리한 조회 값 데이터가 많으면 한번에 많은 값을 전달할 수 있다.
  - 새로고침하면 재조회해서 추가 또는 삭제된 데이터를 반영해서 조회할 수 있다.
- 새로고침을 하면 재요청된 페이지에 대한 요청이 반복되는 redirect는 주로 삽입, 수정, 삭제 기능에 사용한다.
  - 데이터 insert, update, delete 로직을 수행할 때 해당 기능이 중복 수행되지 않도록 하는 것이 바람직하다.
- 이처럼 forward와 redirect는 특징을 알고 특징에 맞게 사용해야 하고, 정해진 방식이 있는 것은 아니다.
  - 예를 들어 로그인 기능은 select 기능이지만, redirect 방식을 활용해 로그인 횟수 제한 등을 설정하기도 한다.

03-04. 객체별 공유 데이터 설정

- 공유 데이터는 Map 형식의 key-value 방식으로 저장된다.
- 이러한 공유 데이터를 포함한 ServletContext, ServletRequest, HttpSession 객체에서 사용하는 method는 다음 표와 같다.

method	내용
setAttribute(String,Object)	공유 데이터 저장
getAttribute(String)	공유 데이터 가져옴
getAttributeNames()	공유 데이터 전체의 명칭 가져옴
removeAttribute(String)	공유 데이터 자체를 삭제