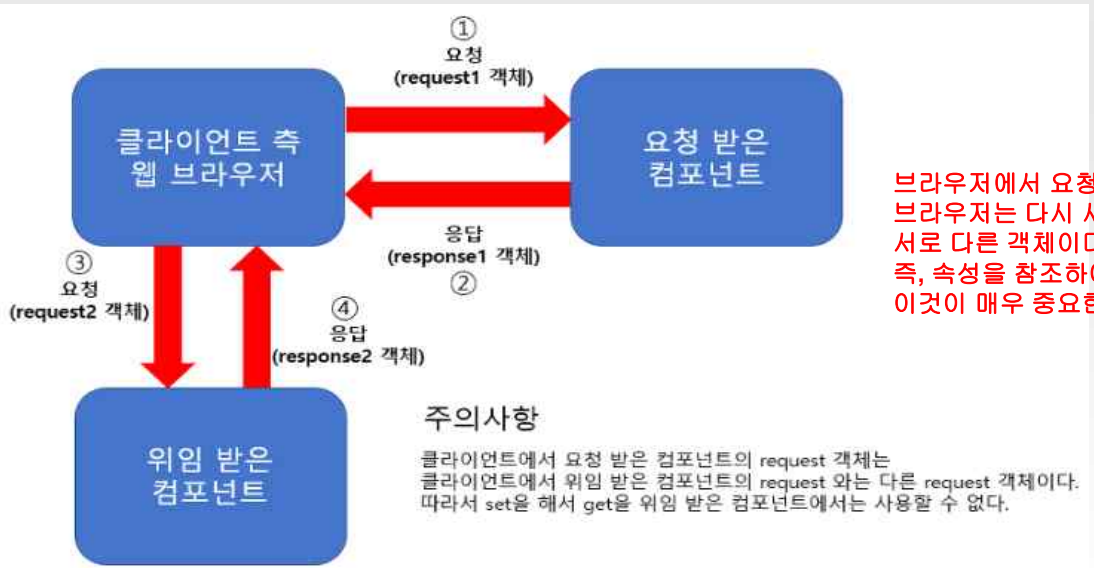
18장. 유용한 패턴

1. RequestDispatcher 클래스

1. 서블릿 또는 **JSP**에서 요청을 받은 후 다른 컴포넌트로 요청을 동일하게 위임할 수 있다.
2. **RequestDispatcher**클래스의 경우 요청받은 요청객체**(request)**를 위임하는 컴포넌트에 동일하게 전달할 수 있다.
3. 웹 브라우저에서 사용자가 전달한 정보를 **request.getParameter()**로 받을 수 있었는데 이것을 다른 컴포넌트로 전달하면 그 컴포넌트에서 똑같이 속성을 받을 수 있다.

2. HttpServletResponse 클래스

1. **RequestDispatcher** 클래스와 동일하게 요청을 위임하는 클래스 이다.
2. **RequestDispatcher** 클래스와 차이점은 요청 받은 요청객체를 위임 받은 컴포넌트에 전달하는 것이 아닌, 새로운 요청객체를 생성한다.

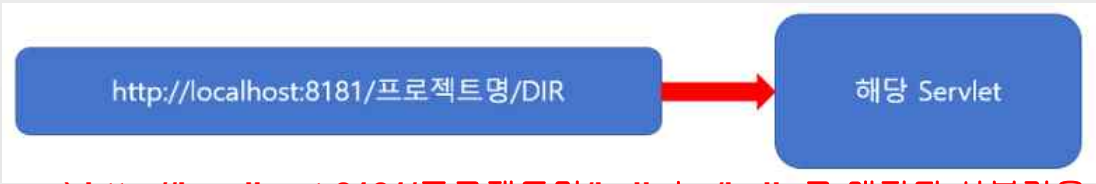


브라우저에서 요청이 날라오면 서블릿에서 응답을 하고 다시 응답 받은 브라우저는 다시 새로운 **request**로 요청하기 때문에 이 두 **request**는 서로 다른 객체이다.
즉, 속성을 참조하여 사용할 수 없다.
이것이 매우 중요한 점이다.

3. url-pattern(디렉토리 패턴, 확장자 패턴)

1 .디렉토리 패턴

- 디렉토리 형태로 서버의 해당 컴포넌트(서블릿을 의미)를 찾아서 실행하는 구조를 말한다.



ex) http://localhost:8181/프로젝트명/hello는 /hello로 매핑된 서블릿을 찾아가서 실행된다.

2. 확장자 패턴

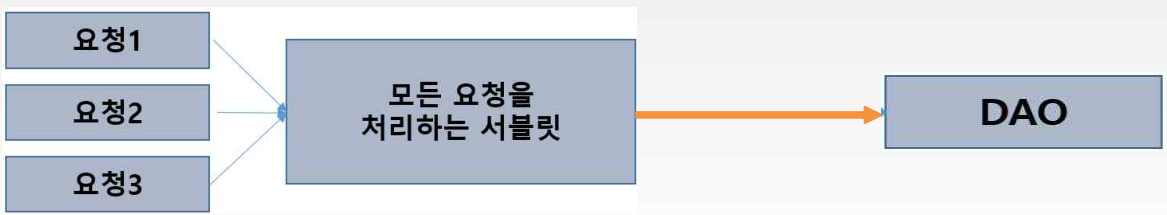
- 확장자 형태로 서버의 해당 컴포넌트(서블릿)를 찾아서 실행하는 구조를 말한다.

ex) http://localhost:8181/프로젝트명/*.do로 끝나는 요청을 동일한 do서블릿으로 찾아가게 맵핑한다.

4. FrontController 패턴

1. FrontController 패턴 적용 전
   - 클라이언트의 다양한 요청이 개별적인 서블릿으로 들어오면, 파일도 많아지고 관리하기도 힘들어진다. 하여, 모든 요청을 처리하는 하나의 서블릿을 만들어서 그에 맞게끔 분기를 하면 될 것이다.

2. FrontController 패턴 적용 후
   - 클라이언트의 다양한 요청을 한곳으로 집중시켜, 개발 및 유지보수에 효율성을 극대화한다.



4. FrontController 패턴

```java
@WebServlet("*.do")
public class DoFrontController extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public DoFrontController() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        actionDo(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        actionDo(request, response);
    }

    private void actionDo(HttpServletRequest request, HttpServletResponse response) {
        // uri = contextPath + 요청한 파일 이름
        String uri = request.getRequestURI();
        System.out.println("uri : " + uri);

        String contextPath = request.getContextPath();
        System.out.println("contextPath : " + contextPath);
        // command = uri - contextPath 길이만큼 빼면 요청한 파일 이름만 남는다.
        String command = uri.substring(contextPath.length());
        System.out.println("command : " + command);

        // command에 있는 파일이름 만으로 관리 가능해진다.
        if(command.equals("/select.do")) {
            System.out.println("--------");
            System.out.println("select");
            System.out.println("--------");
        }else if(command.equals("/insert.do")) {
            System.out.println("--------");
            System.out.println("insert");
            System.out.println("--------");
        }else if(command.equals("/update.do")) {
            System.out.println("--------");
            System.out.println("update");
            System.out.println("--------");
        }else if(command.equals("/delete.do")) {
            System.out.println("--------");
            System.out.println("delete");
            System.out.println("--------");
        }
```

@WebServlet(“*.do”)로 함으로써 *.do로 오는 모든 요청을 좌측 서블릿이 다 받아서 처리하면서 요청 uri와 contextPath를 구한 후 이것을 contextPath만큼 substring하여 파일이름만 구해서 좌측과 같이 해당 if문에서 처리해주는 것이다.

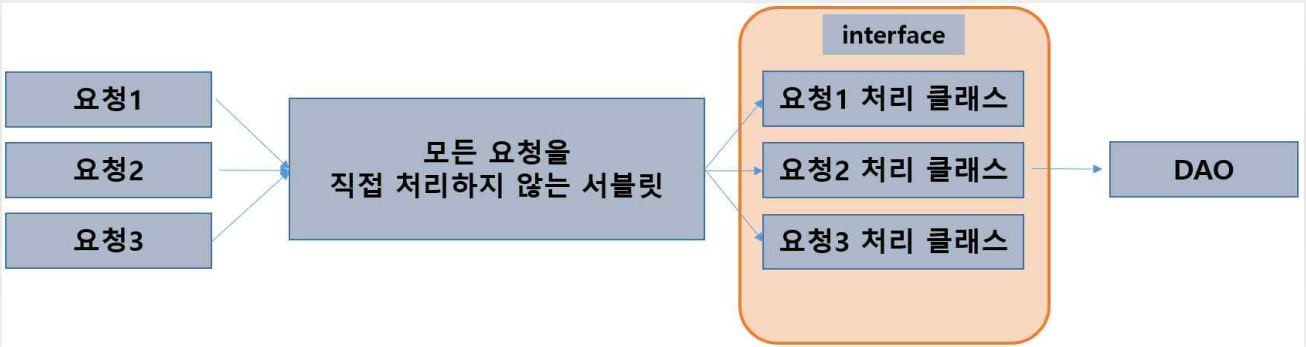
모든 요청이 한 쪽으로 오면서 집중도는 높아지지만 그에 대한 처리(if 구문안)를 하면 해당 소스가 길어지고 역시 관리하기 힘들어지는게 단점이다.

하여, 이에 대한 방안으로 다시 다른 서블릿으로 보내서 분산해서 관리해준다.
이것이 Command패턴인 것이다.

5. Command 패턴

1. Command 패턴 적용

- 클라이언트로부터 받은 요청들에 대해서, 서블릿이 작업을 직접 처리하지 않고, 해당 클래스가 처리하도록 한다.



중요함

클라이언트로부터 받은 요청들에 대해서, FrontController역할을 하는 서블릿이 작업을 직접 처리하지 않고,
해당 담당 서블릿으로 분산을 시켜서 직접 처리하도록 한다.
즉, FrontController패턴과 Command패턴을 합쳐 놓으면 프로그램이 상당히 가독성이 좋을것이다.
(MVC패턴의 핵심)

감사합니다.