**MVC1 과 MVC2**

Model1은 view 와 Controller를 하나의 JSP로 구현한다.

따라서 클라이언트로부터 요청이 들어왔을 때 JSP가 요청을 받아 처리하면서 동시에 view의 역할도 같이 하도록 한다.

view와 Controller를 하나의 JSP에 구현하는 만큼 Model2와 비교해서는 구현이 쉽지만

기능이 복잡해질수록 유지 보수가 어려워지는 단점이 있다.

Model2 는 view는 JSP, Controller는 servlet으로 작성되어 클라이언트의 요청을 servlet으로 구현된 Controller가 받아 Parameter 를 Service객체로 전달하여 처리하여

처리 결과를 받아 view에 전달한다.

Model2 는 역할 구분이 확실하기 때문에 설계가 Model1 에 비해 어렵지만, 유지보수가 수월해지는 장점이 있다.

**FrontController 패턴**

: 클라이언트의 다양한 요청을 한 곳으로 집중시켜, 개발 및 유지보수 효율을 극대화

다양한 요청에 일일히 Servlet을 만든다면 유지보수 및 개발 효율이 떨어질 것이다.

따라서 각각의 분기를 처리해주는 어떤 Controller 를 앞에 ( 그래서 Front Controller ) 두고 각 요청에 맞게 분기하여 처리해주는 패턴을 발전시켰다.

String type = null;

// 1. getRequestURI() 메소드를 통해서 요청된 전체 uri를 가져온다.

String command = request.getRequestURI();

// 2. getContextPath() 메소드를 통해서 프로젝트의 Context Path 명을 가져온다.

if( command.indexOf(request.getContextPath()) == 0 ) {

// 3. subString() 메소드를 통해서 ContextPath 문장의 길이만큼을 전체 uri에서 지운다.

// 그럼 요청한 파일 이름만 남게 된다.

type = command.substring(request.getContextPath().length());

}

// 4. subString 결과값 ( command) 을 통해서 요청을 분기한다.

// subString 결과에 따라

if(type == null || type.equals("/greeting") ) {

resultObj = "안녕하세요";

page="/WEB-INF/views/view01.jsp";

} else if(type.equals("/date")) {

resultObj = new Date();

page="/WEB-INF/views/view02.jsp";

} else if(type.equals("/")) {

page = "/WEB-INF/views/index.jsp";

} else {

resultObj = "Invalid Type";

}

만약에 클라이언트가 요청할 수 있는 것들이 점점 많아질수록 요청에 대한 분기를 else if로 계속 나눠줘야 한다. 이 때문에 코드가 난잡해지고 유지보수가 어려워지니 이를 보안하여 나온 방법이 모든 요청을 직접 처리하지 않고, 담당 class에 처리를 보내는 것이다.

**Command 패턴**

Front Controller 패턴과 Command 패턴의 주요 차이점은 요청에 대한 처리방법이다.

Front Controller 패턴은 \*.do 로 오는 모든 요청을 한 서블릿으로 받고 요청 uri 와 contextPath를 구한 후 이것을 contextPath 만큼 subString 하여 파일 이름만을 구한 후 이것으로 관리한다.

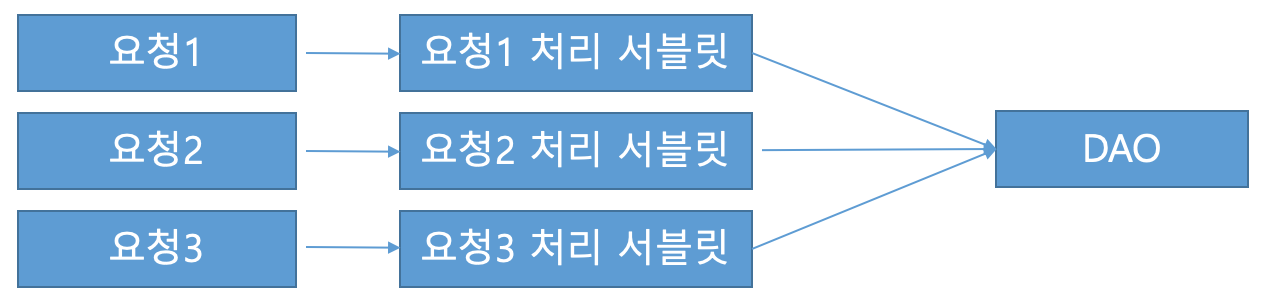
하지만 이렇게 Front Controller에서 모든 요청에 대한 처리를 하면 if 구문의 소스코드가 길어지고 관리하기 힘들어진다. 이에 대한 대처가 처리를 담당하는 class(servlet) 를 분산 관리하는 Command 패턴이다.

Front Controller는 요청별로 키값에 저장된 servlet 만 연결해주며 더이상 요청에 대한 실제 처리는 하지 않는다.

요청에 관한 처리는 Interface로 만들어진 각 요청에 맞는 service 클래스에서 해결하며 해당 처리를 위한 DAO DTO 의 처리도 service 클래스에서 인스턴스를 전달받아 처리한 후 request에 Attribute를 set한 후 처리된 데이터를 사용하여 표시 해줄 view 페이지에 리턴해준다.

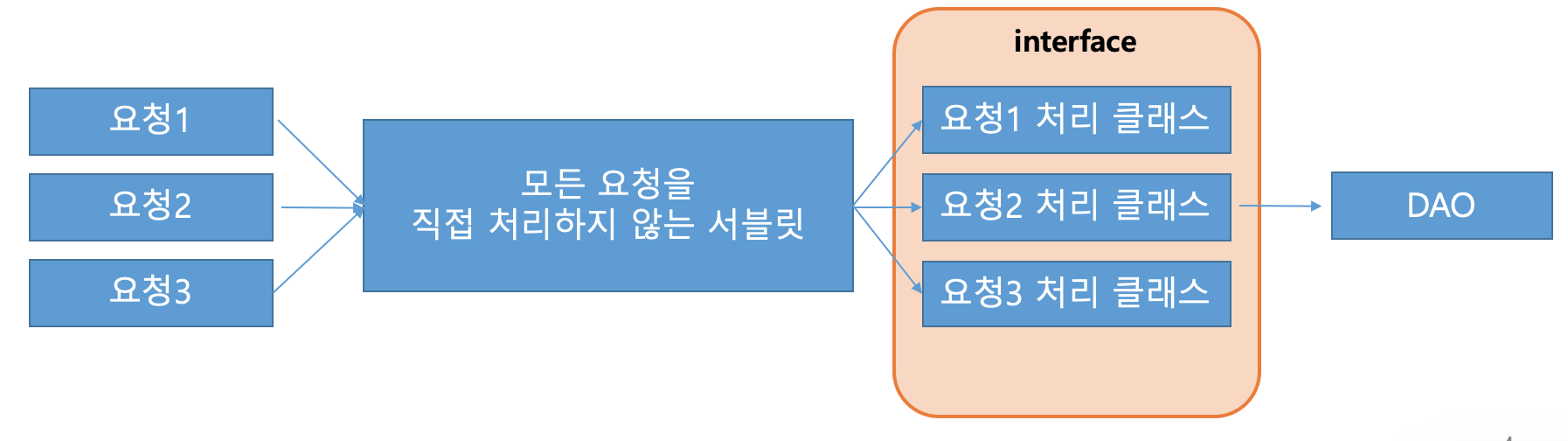
**Review**

아래 이미지처럼 원시적인 MVC 2패턴은 모든 요청마다 해당 요청을 처리하는 서블릿을 하나씩 만들었다.



하지만 위 방식은 클라이언트가 할 수 있는 요청의 수가 증가할 때 마다 그만큼 서블릿 수가 무한정으로 증가한다는 문제가 있었다.

이를 해결하고자 FrontController 패턴이 대두되었다.

하지만 FrontController 패턴도 요청이 많아질수록 처리해야하는 분기가 많아지는 문제점이 발생하였고, URI 분기는 FrontController에서 처리하고 나머지 세부적인 로직들은 인터페이스를 통해 처리하려 했고, 그에 따라 command 패턴이 대두되었다.

이처럼 인터페이스를 활용하여 각 요청에 대한 세부로직은 인터페이스가 처리하며, FrontController가 위치한 Servlet은 URI 분기만 처리해주는 식으로 발전했고,

이 방식이 현재 Servlet MVC 방식에서 가장 많이 쓰이는 방식이다.

**WAR 확장자**

WAR 확장자는 JAR 확장자에서 servlet과 jsp 컨테이너를 배치 할 수 있도록 한 웹 애플리케이션 압축 (Web Application Archive) 파일 포맷이다. JSP/Servlet/JAR/Class/XML/HTML/ JavaScript 등 Servlet Context 관련 파일들을 패키징하고 있으며 웹 응용 프로그램을 위한 포맷이기 때문에 웹 관련 자원만 포함하고 있고 이를 사용하면 웹 애플리케이션을 쉽게 배포하고 테스트 할 수 있다.

원하는 구성을 할 수 있는 JAR 포맷과 달리 WAR는 WEB-INF 디렉토리로 사전 정의된 구조를 사용하고 WAR 확장자 파일을 실행(배포) 하려면 Tomcat, Weblogic, Websphere 등의 웹 서버 (WEB)혹은 웹 컨테이너 (WAS)가 필요하다.