❑ IController 인터페이스 생성

- 메인 컨트롤러에서 위임을 받아 실행하는 커맨드객체 인터페이스를 생성.
- 일반적으로 커맨드패턴에서는 execute 메서드로 정의 하지만 여기서는 process 로 한다.

```
public interface IController {
  /**
   * Front Controller 에서 작업을 위임 받아 요청에 대한 처리를 한다. <br>
   * 비즈니스객체를 사용하여 구현하고 결과객체(모델)는 request 또는 session의 속성에 저장한다.<br>
   * 뷰에 대한 정보를 문자열로 리턴한다.
   * @param HttpServletRequest req
   * @param HttpServletResponse resp
   * @return ViewName
   * @throws Exception
   */
  public String process(HttpServletRequest req , HttpServletResponse resp) throws Exception;

}
```

❑ FrontController 작성 1/4

- 클라이언트의 요청을 받아서 처리하는 FrontController 생성한다.
- WAS에서 모든 요청이 하나의 객체에 집중하려면 서블릿 객체 중 HttpServlet 또는 Filter 로 구현하면 된다.
- 일반적으로 HttpServlet 을 상속하여 만든다.

```
@SuppressWarnings("serial")
@WebServlet(   name = "frontController"
             , urlPatterns = {"*.do"}
             , description = "사용자의 요청을 처리하는 메인 컨트롤러"
             , initParams  = {@WebInitParam( name  = "configFile"
                                           ,value = "/WEB-INF/classes/config/study_uri.properties")} )
public class StudyController extends HttpServlet {

  // 커멘드객체(요청 담당 컨트롤러)를 담을 맵 선언
  private Map<String, IController > controllerMap = new HashMap<>();

}
```

❑ FrontController 작성 2/4

– 서블릿의 init() 메서드를 재정의 하여 초기화 작업을 한다.

```
// 서블릿의 초기화 메서드(init)에서 프로퍼티를 읽어 객체 생성 후 맵에 저장하기
@Override
public void init() throws ServletException {
  String configFile = getInitParameter("configFile");
  Properties prop = new Properties();
  String configFilePath = getServletContext().getRealPath(configFile);
  try(FileReader fis = new FileReader(configFilePath)){
    prop.load(fis);
  }catch (IOException e) {
    throw new ServletException(e);
  }
  Iterator<?> keyIter = prop.keySet().iterator();
  while(keyIter.hasNext()){
    String command          = (String)keyIter.next();
    String handlerClassName = prop.getProperty(command);
    try {
      // 클래스 로드 -> 인스턴스 생성 -> 맵에 저장
      Class<?> handlerClass = Class.forName(handlerClassName);
      IController handlerInstance = (IController)handlerClass.newInstance();
      controllerMap.put(command, handlerInstance);
    } catch (Exception e) {
      e.printStackTrace();
      throw new ServletException(e);
    }
} } // init
```

❑ FrontController 작성 3/4

– 서블릿의 service() 매서드를 재정의하여 클라이언트의 요청을 처리한다.

```
// 1. 클라이언트의 요청을 처리하려면 (doGet, doPost, service) 재정의 한다.
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException
{
  // 요청 처리 전 공통적인 기능이 필요하면 기술한다.

  // 2. 요청을 분석한다.
  String uri =  req.getRequestURI();
  uri = uri.substring( req.getContextPath().length() );
  System.out.printf("요청 URI =  %s\n" , uri );

  try {
    String viewPage = null;
    IController controller = null;
    controller = controllerMap.get(uri);
    if(controller != null){
      // 3. 요청에 따른 기능 수행
      // 4. 기능 수행에 따른 결과를 속성에 저장한다.
      // 각각의 컨트롤(커맨드) 객체는 3,4 의 기능을 위임 받아 수행하고 뷰정보를 리턴한다.
      viewPage = controller.process(req, resp);
```

❑ FrontController 작성 4/4

```
      // 5. 알맞은 뷰로 이동 (포워드)
      if(viewPage != null){
        if(viewPage.startsWith("redirect:")){
          resp.sendRedirect( req.getContextPath() + viewPage.substring("redirect:".length()) );
        }else{
          // RequestDispatcher 인클루드, 포워드를 전담하는 객체
          RequestDispatcher dispatcher = req.getRequestDispatcher(viewPage);
          dispatcher.forward(req, resp);
        }
      }
    }else{
      // controller가 널이면 요청에 대한 정보가 없는 것이므로
      System.out.printf("uri=[%s] 에 해당하는 컨트롤러가 존재하지 않습니다.",uri);
      resp.sendError(HttpServletResponse.SC_NOT_FOUND ); // 404
    }
  } catch (Exception e) {
    e.printStackTrace();
    // resp.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR );
    throw new ServletException(e);
  }
} // service
```

❑ ResultMessageVO 생성

– 공통으로 메시지 정보를 담을 빈 객체

```
package com.study.common.vo;

public class ResultMessageVO {
  private boolean result;
  private String title;
  private String message;
  private String url;
  private String urlTitle;
  // getter/ setter 생성
  // setter 메서드는 void 가 아닌 해당 객체를 리턴하도록 변경한다.
  // Method chaining은 중간결과를 저장하지 않고도 단일 명령문으로 호출이 가능하도록 한다.
```

❑ message.jsp

– /WEB-INF/views/common/message.jsp

```
<div class="row col-md-8 col-md-offset-2">
  <div class="page-header">
    <h3>${resultMessage.title}</h3>
  </div>
  <div class="panel panel-default">
    <div class="panel-heading">
      <p>${resultMessage.message}</p>
    </div>
    <div class="panel-body">
      <a href="${pageContext.request.contextPath}/index.jsp"  class="btn btn-primary">
        <span class="glyphicon glyphicon-home" aria-hidden="true"></span>  Home
      </a>
      <div class="pull-right">
        <a href="#" onclick="history.back()"  class="btn btn-default">
          <span class="glyphicon glyphicon-arrow-left" aria-hidden="true"></span>  뒤로가기
        </a>   
        <c:if test="${not empty resultMessage.url}">
          <a href="${pageContext.request.contextPath}${resultMessage.url}" class="btn btn-warning">
              <span class="glyphicon glyphicon-new-window aria-hidden="true"></span>
               ${resultMessage.urlTitle}
          </a>
        </c:if>
      </div>
    </div>
</div></div>
```