9.8 서블릿 핵심 클래스

1) ServletConfig API를 활용한 초기화 파라미터

서블릿이 초기화될 때, 공통적으로 적용해야 되는 작업들이 필요 경우가 있다.(예;외부 파일 및 디렉터리 경로, JDBC에서 사용하기 위한 데이터베이스 경로, 계정 및 비밀번호와 같은 정보들) 이런 정보들을 서블릿에서 설정하지 않고 web.xml에서 설정한 후 서블릿에서 접근해서 사용한다.

서블릿에서 설정하는 경우에는 정보가 변경되면 반드시 서블릿을 재컴파일 시켜야 한다. 하지만 web.xml에서 설정하면 재컴파일 없이 변경된 정보를 참조할 수 있기 때문에 유지보수가 쉬워진다.

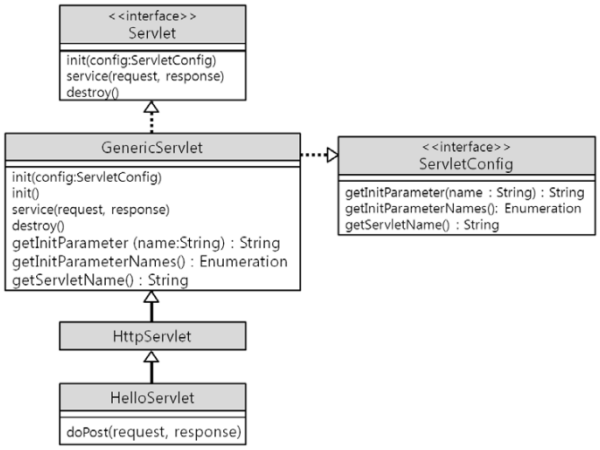
web.xml에 설정된 설정 값을 '초기화 파라미터(Initialization Parameter)라고 하며 ServletConfig API를 이용해서 접근할 수 있다.

여러 서블릿에서 공유해서 사용하지 못하고 <init-param>으로 등록된 서블릿에서만 사용 가능하다.

서블릿 코드 내에서 @WebInitParam 어노테이션을 이용하여 초기화 파라미터를 등록할 수도 있다.

- ServletConfig API의 계층 구조

ServletConfig는 인터페이스로서 GenericServlet에서 구현했기 때문에 사용자가 작성한 서블릿에서 ServletConfig의 메서드를 제약없이 사용 가능하다.



- web.xml에 초기화 파라미터 등록

<servlet>태그 안에서 <init-param>태그를 사용하여 지정한다. 여러 개의 태그 등록이 가능하고 name, value 쌍으로 설정한다.

```
<servlet>
    <servlet-name>서블릿 별칭</servlet-name>
    <servlet-class>패키지를 포함한 서블릿명</servlet-class>
    <init-param>
      <param-name>초기화 파라미터 이름</param-name>
      <param-value>초기화 파라미터 값</param-value>
    </init-param>
    <init-param>
      <param-name>초기화 파라미터 이름</param-name>
      <param-value>초기화 파라미터 값</param-value>
    </init-param>
```

```
</servlet>
[예]
<servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <init-param>
      <param-name>dirPath</param-name>
      <param-value>c:₩₩test</param-value>
    </init-param>
    <init-param>
      <param-name>userid</param-name>
      <param-value>admin</param-value>
    </init-param>
</servlet>
```

ServletConfig의 핵심 메소드

리턴 타입	메서드명	내용
String	String getInitParameter(name)	name에 해당되는 파라미터 값을 리턴한다. 만약 지정된 name의 파라미터 값이 없으면 null을 리턴한다.
Enumeration	getInitParameterNames( )	모든 초기화 파라미터 name 값을 Enumeration 타입으로 리턴한다. 얻은 name 값을 이용하여 초기화 파라미터 값을 얻는다.
String	getServletName( )	요청한 서블릿의 이름을 리턴한다.

- web.xml 초기화 파라미터 설정과 서블릿에서 ServletConfig를 이용하여 참조하는 예제

InitParamServlet.java

```
package com.test;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class InitParamServlet extends HttpServlet {
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // 초기화 파라미터 얻기
                String dirPath = getInitParameter("dirPath");
                String userid = getInitParameter("userid");
```

```
                response.setContentType("text/html; charset=UTF-8");
                PrintWriter out = response.getWriter();
                out.print("<html><body>");
                out.print("디렉터리 경로 : " +  dirPath +"<br>");
                out.print("아이디 값 : " +  userid +"<br>");
                out.print("</body></html>");
        }
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                doPost(request, response);
        }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <!-- 초기화 파라미터 설정 -->
    <init-param>
      <param-name>dirPath</param-name>
      <param-value>c:₩₩test</param-value>
    </init-param>
    <init-param>
      <param-name>userid</param-name>
      <param-value>admin</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>InitParam</servlet-name>
    <url-pattern>/InitParam</url-pattern>
  </servlet-mapping>
</web-app>
```

@WebInitParam 애노테이션을 이용한 초기화 파라미터 등록

서블릿 코드내에서 @WebInitParam 어노테이션을 사용하여 초기화 파라미터를 등록할 수 있다.

ServletConfig의 getInitParameter(name) 메서드를 사용하여 초기화 파라미터 값을 얻는다.

InitParamAnnoServlet.java

```
package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet( name ="InitParamAnnoServlet" ,
                urlPatterns={"/initParamAnno"} ,
                initParams = { @WebInitParam ( name="dirPath", value = "c:₩₩test") ,
                                        @WebInitParam ( name="userid", value = "system") })
public class InitParamAnnoServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                System.out.println(getInitParameter("dirPath"));
                System.out.println(getInitParameter("userid"));
        }
}
```

3) ServletContext API

웹 어플리케이션에는 여러 가지 자원을 포함할 수 있다. html 파일, 미디어 파일, 이미지 파일, 다수의 JSP 파일과 서블릿 등이 유기적으로 동작된다.

ServletContext는 웹 어플리케이션(Context)마다 하나씩 생성되는 객체로서, 다수의 JSP 파일과 서블릿에서 공유해서 사용할 수 있다. ServletContext 객체는 웹 어플리케이션의 LifeCycle과 일치하기 때문에, 웹 어플리케이션이 Tomcat 컨테이너에 존재한다면 계속 사용 가능하다. 이것을 'application scope'라고 한다.

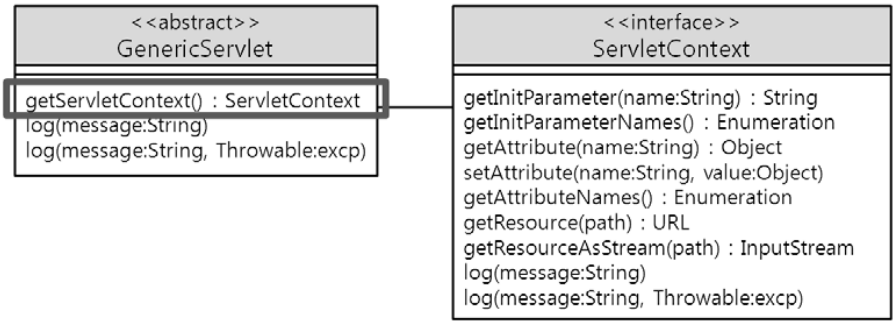
- ServletContext 객체를 이용한 핵심 기능

여러 서블릿에서 사용 가능한 초기화 파라미터 사용할 수 있고 일반적으로 '컨텍스트 파라미터(Context Parameter)'라고 한다. 서블릿에서 파일 접근 가능(읽기 모드만 가능)하다.

application scope에 해당되는 속성(Attribute)을 저장하고 조회할 수 있다.

- ServletContext API의 계층 구조

GenericServlet을 HttpServlet이 상속받았기 때문에 ServletContext 객체를 얻기 위해서 사용자가 생성한 서블릿에서는 getServletContext() 메소드를 사용한다.



- ServletContext의 핵심 메소드

리턴 타입	메소드명	내용
String	getInitParameter(name)	name에 해당되는 컨텍스트 파라미터 값을 리턴한다. 만약 지정된 name의 파라미터 값이 없으면 null을 리턴한다.
InputStream	getResourceAsStream(path)	웹 어플리케이션의 path 경로에 해당되는 파일을 읽기 모드로 접근 가능하다.
void	setAttribute(name,value)	application scope 해당되는 속성 값을 저장할 때 사용한다. 브라우저를 종료해도 속성 값을 사용 가능하다.
Object	getAttribute(name)	name에 해당되는 속성 값을 리턴한다.

Context Parameter 설정

다수의 서블릿이 공통적으로 사용되는 특정 데이터가 필요하다면, 컨텍스트 파라미터를 사용하는 것이 바람직하다. 초기화 파라미터는 특정 서블릿만이 파라미터 값을 사용 가능할 수 있지만 컨텍스트 파라미터는 application scope이기 때문에 어플리케이션의 모든 서블릿이 공유해서 사용 가능하다.

초기화 파라미터와 마찬가지로 web.xml에 등록하여 사용하고 ServletConfig 대신에 ServletContext 객체의 getInitParameter(name) 메서드를 사용해서 컨텍스트 파라미터 값을 얻는다.

- 예제

ContextParamServlet.java

```
package com.test2;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextParam")
public class ContextParamServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
```

```
ServletException, IOException {
                    // 컨텍스트 파라미터 얻기
                    String driver = getServletContext().getInitParameter("driver");
                    String savePath = getServletContext().getInitParameter("savePath");
                    response.setContentType("text/html; charset=UTF-8");
                    PrintWriter out = response.getWriter();
                    out.print("<html><body>");
                    out.print("드라이버명: " + driver +"<br>");
                    out.print("저장 경로: " + savePath +"<br>");
                    out.print("</body></html>");
          }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <!-- 컨텍스트 파라미터 설정 -->
  <context-param>
    <param-name>driver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </context-param>
  <context-param>
    <param-name>savePath</param-name>
    <param-value>c:₩₩save</param-value>
  </context-param>
  <servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <!-- 초기화 파라미터 설정 -->
    <init-param>
      <param-name>dirPath</param-name>
      <param-value>c:₩₩test</param-value>
    </init-param>
    <init-param>
      <param-name>userid</param-name>
      <param-value>admin</param-value>
    </init-param>
```

```
  </servlet>
  <servlet-mapping>
    <servlet-name>InitParam</servlet-name>
    <url-pattern>/InitParam</url-pattern>
  </servlet-mapping>
</web-app>
```

서블릿에서 파일 접근(읽기 모드)

서블릿에서 웹 어플리케이션내의 특정 파일을 접근하기 위해서 ServletContext 객체를 사용 가능하다.
읽기 모드만 가능하고 쓰기는 불가능 하다.

- 파일 접근 예제

testFile.txt 파일을 WEB-INF 폴더에 저장한다.

ContextFileServlet.java

```
package com.test2;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextFile")
public class ContextFileServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                String readFile = "/WEB-INF/testFile.txt";
                InputStream is = getServletContext().getResourceAsStream(readFile);
                BufferedReader reader = new  BufferedReader(new InputStreamReader(is));
                response.setContentType("text/html; charset=UTF-8");
                PrintWriter out = response.getWriter();
                out.print("<html><body>");
                String str = reader.readLine();
                while(str != null){
                        out.print(str +"<br>");
                        str = reader.readLine();
                }
```

```
                    reader.close();
                    out.print("</body></html>");
        }
}
```

4) 서블릿에서 attribute 설정 및 참조

웹 어플리케이션에서 브라우저를 종료해도 지속적으로 사용해야 되는 데이터가 필요하다면, application scope에 해당되는 속성(attribute)을 사용할 수 있다.

대표적인 예로 '웹 사이트의 방문자수 조회' 형태로서, 웹 어플리케이션을 종료할 때까지 속성 값은 유지된다.

- 속성 설정 및 참조 예제

브라우저를 종료하고 다시 접속해도 계속 유지되는 데이터를 ServletContext 객체의 setAttribute(name, value) 메소드로 저장하고, getAttribute(name) 메소드로 참조한다.

ContextSetServlet.java 서블릿을 먼저 실행하고 브라우저를 닫는다.

```
package com.test2;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextSet")
public class ContextSetServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                    // 속성값 설정
                    String name = "홍길동";
                    int age = 20;
                    getServletContext().setAttribute("name", name );
                    getServletContext().setAttribute("age", age );
        }
}
```

ContextGetServlet.java

```
package com.test2;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ContextGet")
public class ContextGetServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		//속성값 얻기
		String name = (String)getServletContext().getAttribute("name");
		int age = (Integer)getServletContext().getAttribute("age");
		response.setContentType("text/html; charset=UTF-8");
		PrintWriter out = response.getWriter();
		out.print("<html><body>");
		out.print("이름 : " + name +"<br>");
		out.print("나이 : " + age +"<br>");
		out.print("</body></html>");
	}
}
```

5) ServletContextListener API

서블릿이 LifeCycle를 가지고 있는 것처럼, 웹 어플리케이션도 LifeCycle를 갖는다. Tomcat 컨테이너가 시작될 때 웹 어플리케이션도 초기화되고, Tomcat 컨테이너가 종료될 때 웹 어플리케 이션도 제거된다.
웹 어플리케이션이 초기화되고 제거되는 이벤트를 감지하는 ServletContextListener API를 사용하면, 언제 초기화되고 제거 되었는지를 쉽게 알 수 있다.
이 이벤트는 JDBC의 Pooling기법에 적용 가능하다. 웹 어플리케이션이 초기화될 때 Pooling을 활성화하고 제거될 때 Pooling을 비활성화 시키면 효율적으로 Connection을 관리할 수 있다.(JDBC에서 자세히)

- ServletContextListener API 계층 구조

- 예제 실습 순서

① ServletContextListener 인터페이스를 구현하는 클래스를 작성한다.
② web.xml에 구현한 클래스를 <listener> 태그로 등록 또는 @WebListener 어노테이션을 이용한다.
③ Tomcat 컨테이너를 시작하고 종료하는 작업을 실행하여 이벤트 감지를 확인한다.

ContextListenerImpl.java

```
package com.test2;
import javax.servlet.ServletContextEvent;
```

```
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
public class ContextListenerImpl implements ServletContextListener {
	@Override
	public void contextDestroyed(ServletContextEvent event) {
		System.out.println("웹 어플리케이션 제거");
	}
	@Override
	public void contextInitialized(ServletContextEvent event) {
		System.out.println("웹 어플리케이션 초기화");
	}
}
```

web.xml에 <listener> 등록

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <!-- 컨텍스트 파라미터 설정 -->
  <context-param>
    <param-name>driver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </context-param>
  <context-param>
    <param-name>savePath</param-name>
    <param-value>c:₩₩save</param-value>
  </context-param>
  <!-- Listener 설정 -->
  <listener>
    <listener-class>com.test2.ContextListenerImpl</listener-class>
  </listener>
  <servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <!-- 초기화 파라미터 설정 -->
    <init-param>
      <param-name>dirPath</param-name>
      <param-value>c:₩₩test</param-value>
    </init-param>
```

```
    <init-param>
      <param-name>userid</param-name>
      <param-value>admin</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>InitParam</servlet-name>
    <url-pattern>/InitParam</url-pattern>
  </servlet-mapping>
</web-app>
```

console 창에서 이벤트 감지 확인하고 서버를 종료하고 이벤트 감지 확인한다.

@WebListener 애노테이션으로 등록하는 방법

web.xml에 등록한 <listener> 태그를 제거하거나 주석 처리하고 Context ListenerImpl 클래스에 @WebListener 어노테이션을 추가하고 테스트한다.

ContextListenerImpl.java

```
package com.test2;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
@WebListener
public class ContextListenerImpl implements ServletContextListener {
        @Override
        public void contextDestroyed(ServletContextEvent event) {
                System.out.println("웹 어플리케이션 제거");
        }
        @Override
        public void contextInitialized(ServletContextEvent event) {
                System.out.println("웹 어플리케이션 초기화");
        }
}
```

5) Filter API

클라이언트인 웹 브라우저에서 서블릿으로 요청하면, 웹 컴포넌트인 서블릿이 요청을 받아서 작업을 처리하고 결과를 HTML 형식으로 작성하여 웹 브라우저에게 응답 처리한다.

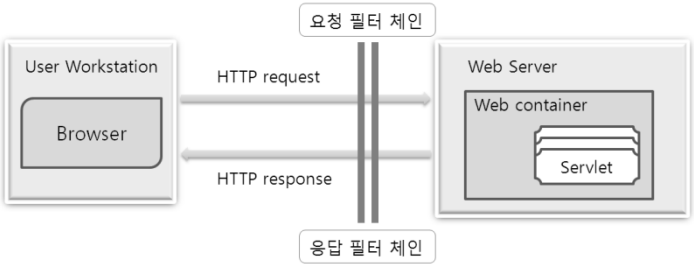
서블릿이 요청 받기 전과 결과를 웹 브라우저에게 응답하기 전에 특정 작업을 수행할 수 있도록 Filter API를 사용할 수 있다. 웹 컴포넌트가 실행되기 전의 선처리(pre-processing) 작업과 응답되기 전의 후처리(post-processing) 작업을 수행하는 API이다.

다수의 Filter를 체인(Chain)처럼 묶어서 적용시킬 수도 있다.

선처리 작업의 필터 → 요청 필터(Request Filter)
후처리 작업의 필터 → 응답 필터(Response Filter)

선처리 작업의 적용 예 : 한글인코딩 및 보안관련 작업 등
후처리 작업의 적용 예 : 압축 및 데이터 변환 작업 등

Filter를 적용한 아키텍처



request 요청이 요청 필터 체인을 거쳐서 서블릿으로 전송된다. 서블릿이 처리하기 전에 실행되어야 하는 선처리 작업 수행할 수 있다. response 응답이 응답 필터 체인을 거쳐서 전송된다. 요청과 마찬가지로 웹 브라우저로 응답되기 전에 실행되어야 하는 후처리 작업 수행할 수 있다.

Filter를 적용하기 위한 실습 순서
① Filter 인터페이스를 구현한 클래스를 작성한다.
② web.xml에 <filter> 태그를 사용하여 등록하거나, @WebFilter 어노테이션을 사용하여 등록한다.
③ 웹 서버에 요청하여 Filter가 적용되었는지를 확인한다.

- web.xml에 <filter> 태그로 등록하는 방법
웹 어플리케이션의 서블릿이 요청을 처리하기 전의 선처리(pre-processing) 작업과 후처리(post-processing) 작업을 처리하는 Filter 예제이다. 코드를 간단하게 하기 위해서 console 창에 문자열을 출력

MyFilter.java

```
package com.test2;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class MyFilter implements Filter {
	public void init(FilterConfig fConfig) throws ServletException {
		System.out.println("MyFilter 초기화");
	}
```

```
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
		System.out.println("MyFilter 요청필터 코드작업");
		request.setCharacterEncoding("UTF-8");
		chain.doFilter(request, response);
		System.out.println("MyFilter 응답필터 코드작업");
	}
	public void destroy() {
		System.out.println("MyFilter 제거");
	}
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <!-- 컨텍스트 파라미터 설정 -->
  <context-param>
    <param-name>driver</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </context-param>
  <context-param>
    <param-name>savePath</param-name>
    <param-value>c:₩₩save</param-value>
  </context-param>
  <!-- Filter 설정 -->
  <filter>
    <filter-name>myFilter</filter-name>
    <filter-class>com.test2.MyFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>myFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <servlet>
    <servlet-name>InitParam</servlet-name>
    <servlet-class>com.test.InitParamServlet</servlet-class>
    <!-- 초기화 파라미터 설정 -->
```

```
    <init-param>
      <param-name>dirPath</param-name>
      <param-value>c:₩₩test</param-value>
    </init-param>
    <init-param>
      <param-name>userid</param-name>
      <param-value>admin</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>InitParam</servlet-name>
    <url-pattern>/InitParam</url-pattern>
  </servlet-mapping>
</web-app>
```

@WebFilter 애노테이션을 사용하면 web.xml에서 <filter>관련 태그를 주석처리한다.

MyFilter.java

```
package com.test2;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
@WebFilter (urlPatterns={ "/*"})
public class MyFilter implements Filter {
        public void init(FilterConfig fConfig) throws ServletException {
                System.out.println("MyFilter 초기화");
        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
                System.out.println("MyFilter 요청필터 코드작업");
                request.setCharacterEncoding("UTF-8");
                chain.doFilter(request, response);
                System.out.println("MyFilter 응답필터 코드작업");
        }
        public void destroy() {
                System.out.println("MyFilter 제거");
        }
```

```
}
```

6) url-pattern의 개요

서블릿 맵핑과 필터 맵핑은 web.xml 파일 및 어노테이션을 사용하여 설정한다. 이때 공통적으로 사용하는 값이 url-pattern이다.

url-pattern은 웹 브라우저에서 클라이언트가 요청하는 URL 값의 패턴에 따라서, 서버의 어떤 웹 컴포넌트가 실행될지를 결정하는 방법이다.

① 디렉터리 패턴

디렉터리 패턴과 일치하는 형태의 url-pattern을 지정한 웹 컴포넌트가 수행

이때, url-pattern 값은 반드시 '/디렉터리 패턴 값' 형식으로 지정해야 되며, 계층 구조를 가질 수 있음

만약에 '/*' 형식으로 지정하면, 클라이언트가 요청하는 URL 값의 패턴과 무관하게 항상 수행

다음은 웹 컨테이너에 4개의 서블릿을 작성하고 각각 url-pattern 값을 지정한 후에, 웹 브라우저에서 특정 패턴을 사용하여 요청한 경우

http://localhost/ServletTest/ATest	ATestServlet 서블릿 /ATest
http://localhost/ServletTest/test/BTest	BTestServlet 서블릿 /test/BTest
http://localhost/ServletTest/test	CTestServlet 서블릿 /test
http://localhost/ServletTest/ http://localhost/ServletTest/xxx	ATestServlet 서블릿 /*

ATestServlet.java

```
package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ATest")
// @WebServlet("/test/BTest")    클래스명 BTestServlet
// @WebServlet("/test")   클래스명 CTestServlet
// @WebServlet("/*")   클래스명 DTestServlet
public class ATestServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		System.out.println("디렉토리 패턴");
	}
}
```

② 확장자 패턴

일치하는 확장자 형태의 url-pattern을 지정한 웹 컴포넌트가 수행

url-pattern 값은 반드시 '*.확장자' 형식으로 지정해야 됨

주의할 점은 '/'를 사용하면 안되며 Spring, Struts, Struts2 프레임워크 같은 다양한 웹 프레임워크에서 많이 사용되는 패턴

다음은 웹 컨테이너에 3개의 서블릿을 작성하고 각각 url-pattern 값을 지정한 후에, 웹 브라우저에서 특정 패턴을 사용하여 요청한 경우이며, /test, /xyz 또는 /my 같은 경로 값 설정과 무관하게 요청

http://localhost/ServletTest/xxx.do http://localhost/ServletTest/xyz/a.do	ETestServlet 서블릿 *.do
http://localhost/ServletTest/my/ss.go http://localhost/ServletTest/kk.go	FTestServlet 서블릿 *.go
http://localhost/ServletTest/test/a.nhn http://localhost/ServletTest/xyz.nhn	GTestServlet 서블릿 *.nhn

ETestServlet.java

```
package com.test;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;e
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("*.do")
// @WebServlet("*.go")      클래스명 FTestServlet
// @WebServlet("*.nhn")     클래스명 GTestServlet
public class ETestServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                System.out.println("ETestServlet");
        }
}
```

필터를 이용한 문자 인코딩 설정

web.xml의 일부분

```
<!-- Filter 문자 인코딩 설정 -->
  <filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.filters.CharacterEncodingFilter</filter-class>
    <init-param>
```

```
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </filter>
    <filter-mapping>
        <filter-name>CharacterEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

위 처럼 설정하면 request.serCharacterEncoding("UTF-8"); 을 사용할 필요가 없다.

애노테이션을 이용한 필터 배치

```
@WebFilter( urlPatterns = "/*",
            initParams={ @WebInitParam(name="encoding", value="UTF-8") })
```