|  |
| --- |
| (주) SOC SOFT |
| JSP 표준 코딩 가이드 |
| JSP 표준 코딩 규칙 정의서 |
|  |
| **공공사업 2팀 우병은** |
| **2014-11-18** |

|  |
| --- |
| JSP에 관련된 코딩 규칙을 정의하고 사용하기 위한 문서입니다. |

**문서 정보**

|  |  |  |
| --- | --- | --- |
| **버전** | 1.0 | |
| **발행일** | 2014. 11. 18 | |
| **상태** | 수정중 | |
| **참여자** | **작성자** | **검토자** |
| 공공사업 2팀 우병은 |  |
|  |  |

**개정이력**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **개정일자** | **버전** | **개정내역** | **작성자** | **확인자** |
| 2014.11.18 | 1.0 | 초안작성 | 우병은 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

목차

[**문서 정보** 1](#_Toc404355504)

[**개정이력** 1](#_Toc404355505)

[1. 소개 5](#_Toc404355506)

[1.1. 목표 5](#_Toc404355507)

[1.2. 대상 5](#_Toc404355508)

[1.3. 참고 문서 5](#_Toc404355509)

[2. 기본 규칙 6](#_Toc404355510)

[2.1. 포맷 규칙(Formatting rule) 6](#_Toc404355511)

[2.1.1. 들여쓰기 및 탭 문자 사용 6](#_Toc404355512)

[2.1.2. 블록의 시작 및 끝 6](#_Toc404355513)

[2.1.3. 라인의 작성 기준 7](#_Toc404355514)

[2.1.4. 줄바꿈 및 여백 활용 기준 7](#_Toc404355515)

[2.2. 클래스 이름 부여 규칙 7](#_Toc404355516)

[2.2.1. 클래스 명칭의 접두 문자 7](#_Toc404355517)

[2.2.2. 클래스 명칭의 첫번째 단어 8](#_Toc404355518)

[2.2.3. 클래스의접미 단어 8](#_Toc404355519)

[2.3. 변수, 상수,메소드 이름 부여 규칙 8](#_Toc404355520)

[2.3.1. 변수명 8](#_Toc404355521)

[2.3.2. 메소드명 10](#_Toc404355522)

[2.4. 상수명 부여 규칙 12](#_Toc404355523)

[2.4.1. 대문자 사용 원칙 12](#_Toc404355524)

[2.5. 소스 파일 작성 규칙 12](#_Toc404355525)

[2.5.1. 소스 파일 설명 12](#_Toc404355526)

[2.6. Java 코딩 스타일 13](#_Toc404355527)

[2.6.1. 공통 13](#_Toc404355528)

[2.6.2. String 13](#_Toc404355529)

[2.6.3. 블록문 13](#_Toc404355530)

[2.6.4. 반복문 13](#_Toc404355531)

[2.6.5. 조건문 14](#_Toc404355532)

[2.7. 주석 14](#_Toc404355533)

[2.7.1. 항목 뒤에 붙이는 주석 14](#_Toc404355534)

[2.7.2. 항목 위에 붙이는 주석 14](#_Toc404355535)

[2.8. 영역 (area) 15](#_Toc404355536)

[2.8.1. 패키지 선언 15](#_Toc404355537)

[2.8.2. 임포트(import) 선언 15](#_Toc404355538)

[2.8.3. 클래스 선언 15](#_Toc404355539)

[2.8.4. 상수 선언 16](#_Toc404355540)

[2.8.5. 변수 선언 16](#_Toc404355541)

[2.8.6. 생성자(constructor) 선언 16](#_Toc404355542)

[2.8.7. property 선언 16](#_Toc404355543)

[2.8.8. public,protected, private 메소드 선언 16](#_Toc404355544)

[3. JSP 프로그램 코딩 규칙 18](#_Toc404355545)

[3.1. 머리말(header) 18](#_Toc404355546)

[3.2. 주석(comment) 18](#_Toc404355547)

[3.2.1. 영역에 대한 주석 18](#_Toc404355548)

[3.2.2. 한 줄에 대한 주석 18](#_Toc404355549)

[3.2.3. 여러 줄에 대한 주석 18](#_Toc404355550)

[3.2.4. 항목 뒤에 붙이는 주석 18](#_Toc404355551)

[3.3. 블록 설정 시 코딩 형식 18](#_Toc404355552)

[3.3.1. 명령문이한줄이 경우 18](#_Toc404355553)

[3.3.2. 명령문이한줄이고 else가 있을 경우 19](#_Toc404355554)

[3.3.3. 명령문이 2줄이상이고 else가 있을 경우 19](#_Toc404355555)

[3.4. 영역 (area) 19](#_Toc404355556)

[3.4.1. 설명문(description) 서술 19](#_Toc404355557)

[3.4.2. 지시어(directive) 서술 19](#_Toc404355558)

[3.4.3. 각종 빈(bean) 선언 20](#_Toc404355559)

[3.4.4. 빈(bean)의 속성(property) 설정 20](#_Toc404355560)

[3.4.5. 선택적 JSP 선언문 21](#_Toc404355561)

[3.4.6. 각종 상태 점검 및 초기화 코드 작성 21](#_Toc404355562)

[3.4.7. HTML 태그 서술 21](#_Toc404355563)

[3.4.8. <HEAD></HEAD> 블록 서술 22](#_Toc404355564)

[3.4.9. <BODY></BODY> 블록 기술 23](#_Toc404355565)

1. 소개

## 목표

프로그램소스의 표준화를 통해서 코딩의 규격화와 효율성 및 가독성을 증대시키고, 유지보수를 원활히 하는데 목적이 있다.소프트웨어 제품 개발에 사용되는 언어가 다양하므로 특정 언어 및 환경에 의존적이지 않는 일반적인 표준 코드 서술 규범을 먼저 기술하고 특정 언어 및 환경을 위한 서술 규범은 별도로 기술한다. 또한, 이 문서에 기술되지 않았거나 문서의 기준을 적용하기 곤란한 부분에 대해서는 Java 진영에서 추천하는 Camel Case 및 관습적인 코딩 규약을 사용할 것을 권장한다.

## 대상

자바기반의 웹 개발자

## 참고 문서

* Oracle 에서 제공하는 자바 코딩 관습 문서  
  <http://www.oracle.com/technetwork/java/codeconv-138413.html>
* Javadoc 작성 방법  
  <http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>
* Doxygen 관련 메뉴얼  
  <http://www.stack.nl/~dimitri/doxygen/manual/index.html>

1. 기본 규칙

## 포맷 규칙(Formatting rule)

### 들여쓰기 및 탭 문자 사용

들여쓰기는 코드에서 블록의 시작과 끝을 쉽게 알아볼 수 있게 해준다. 들여쓰기 되었을 때, 더욱 읽기 좋고, 더 많은 단계의 블록이 다른 블록 내부에 포함되어 있을 때, 들여쓰기의 필요성은 더욱 강조된다. 들여쓰기 및 탭 크기는 기본적으로 4 글자로 한다. 탭 문자는 사용하지 않는 것을 원칙으로 한다. 다음은 들여쓰기 예이다.

|  |
| --- |
| <myTagLib:forEachvar="client" items="${clients}">  <myTagLib:mail value="${client}" />  </myTagLib:forEach> |

연속들여쓰기는 이전 줄의 적당한 지점에서 시작한다. 연속 들여쓰기의 공백은 기본 들여쓰기 공백의 배수이다. (4개의 공백문자의 배수)

|  |
| --- |
| <%@ page attribute1="value1"  attribute2="value2"  ...  attributeN="valueN"  %> |

### 블록의 시작 및 끝

중괄호”{“ 와 중괄호”}” 는 항상 수직선상에 위치하도록 한다 또한 항상 블록의 시작에서는 4칸 들여쓰기를 한다. 자바 진영의 주된 경향(흐름)은 시작하는 괄호를 코드의 끝에 위치시키는 것이지만, 시각적 효과를 위해 수직 정렬을 표준으로 채택하였다.

|  |
| --- |
| main() {  for( ; ; ) {  }  } |

### 라인의 작성 기준

* 1024\*768 화면 기준으로 각 라인은 한 화면에 가급적 전부 보이도록 100 컬럼이내로 짧게 작성하며, 코드의 길이가 긴 라인은 두 줄에 걸쳐서 작성한다.
* 한 줄에 꼭 하나의 명령문만을 기술하여야 한다.
* 복잡한 수식을 가진 긴 라인이라면 여러 개의 작은 수식으로 분할하여 작성한다.

double length = Math.sqrt( Math.pow(Math.random(),2.0) + Math.pow(Math.randown(),2,0) ) ; 은 아래와 같이 분리한다.

|  |
| --- |
| double xSquared = Math.pow(Math.random(), 2.0 ) ;  double ySquared = Math.pow(Math.random(), 2, 0 ) ;  double length = Math.sqrt(xSquared + ySquared) ; |

### 줄바꿈 및 여백 활용 기준

* 각 코드 블록(code block)간의 여백은 한 줄을 띄우는 것을 원칙으로 하며, 메소드 혹은 각 영역(area)간의 간격은 두 줄을 원칙으로 한다.
* 각각의생성자 그리고 각각의 메소드와 같은 클래스의 부분들(parts) 사이에 적어도 하나 이상의 빈줄을삽입하라.
* 메소드 내부에서 지역 변수의 선언 이후에, 그리고 코드의 다른 영역들 사이에 빈 줄을 삽입하라.
* 연산자( =, /, \* 등)의 양 옆에 공백을 삽입하라. 그러나 증가(++), 감소(--), 부정(!) 같은 복합연산자에는 공백을 이용해서는 안된다
* for 구문의 부분들을 공백으로 구분하여라. 초기화, 검사, 증가 부분을 구분하는 세미콜론(;) 다음에 공백을 준다.
* 메소드의 인자(parameter) 목록의 쉼표(comma)와 뒤의 공백은 각각의 인자의 시작과 끝을 명확하게 만들어준다.

## 클래스 이름 부여 규칙

### 클래스 명칭의 접두 문자

클래스 명칭의 첫번째 문자는 반드시 알파벳 대문자로 사용한다. 클래스의 명칭은 일반적으로 명사로 이루어진다.

### 클래스 명칭의 첫번째 단어

클래스 명칭의접두 문자 이후의 첫번째 단어는 클래스의 성격을 잘 나타낼 수 있는 명사 혹은 형용사를 사용하며, 단어의 첫 글자는 대문자로 시작하며, 나머지는 소문자로 기술하여야 한다.

### 클래스의접미 단어

클래스의 종류를 나타내는 접미사를 붙여주는 것을 권장한다.

|  |
| --- |
| → 다이얼로그 : Dialog  → 윈도우 : Window  → 문서 : Doc  → 자바 빈 : Bean  → 커스텀태그 : Tag |

## 변수, 상수,메소드 이름 부여 규칙

모든 이름은 영어로 작성한다.

### 변수명

* 변수의 이름은 대소문자를 혼용할 수 있지만 반드시 소문자로 시작한다.

|  |
| --- |
| ex) account, eventHandler |

* 일반적인 변수의 이름은 타입의 이름과 동일하게 지정한다.

|  |
| --- |
| ex) void setTopic (Topic topic) // void setTopic (Topic value) 이 아님 // void setTopic (Topic aTopic) 이 아님 // void setTopic (Topic x) 이 아님 void connect (Database database) // void connect (Database db) 가 아님 // void connect (Database oracleDB) 가 아님 |

* 넓은 범위에 영향을 미치는 변수는 긴 이름을 부여하고, 좁은 범위의 변수는 짧은 이름을 부여한다.
* 임시 저장공간이나 인덱스로 사용되는 Scratch variable(loop문내에서의 i, j, k 등과 같은)들은 매우 짧은 이름을 부여하도록한다.
* 프로그래머가 그러한 변수들을 읽음과 동시에, 이 변수는 몇 라인 뒤에 그 값이 유효하지 않을 것임을 짐작할 수 있게 해야 한다.
* 컬렉션의 이름은 반드시 복수형으로 사용한다.  
  여기서의 컬렉션은 java.util.Collection 변수와 단순 배열과 같은 Collection의 자손들을 의미한다.
* n 접두사는 객체의 개수를 나타내는 변수에 사용한다.

|  |
| --- |
| * ex) nPoints, nLines |

* No 접미사는 엔터티 번호를 나타내는 변수에 사용한다.

|  |
| --- |
| * ex) tableNo, employeeNo |

* Iterator 변수들은 i, j, k 등 과 같은 이름을 사용한다.

|  |
| --- |
| ex) while (Iterator i = pointList.iterator(); i.hasNext();) {  ; } for (inti = 0; i<nTables; i++) {  ; } |

* 불리언 변수 이름은 절대로 부정적인(거짓인) 이름을 사용하지 않는다.

|  |
| --- |
| ex) booleanisError; // isNotError 가 아님 booleanisFound; // isNotFound 가 아님 |

* 관련있는 상수(final 변수)들은 공통 타입의 이름을 접두사로 사용하여 그룹핑한다.

|  |
| --- |
| ex) final int COLOR\_RED = 1; final int COLOR\_GREEN = 2; final int COLOR\_BLUE = 3; |

* 예외(Exception) 클래스들은 Exception 이라는 접미사를 사용한다.

|  |
| --- |
| ex) DataAccessException |

* 디폴트 인터페이스 구현은 Default 라는 접두사를 사용할 수 있다.

|  |
| --- |
| ex) class DefaultTableCellRenderer implements TableCellRenderer {  ... } |

### 메소드명

* 메소드의 이름은 대소문자를 혼용할 수 있지만 반드시 동사를 사용하며 소문자로 시작한다.

|  |
| --- |
| * ex) getName(), computeTotalWidth() |

* 축약형(Abbreviation) 과 두문자어형(頭文字語: Acronym)을 이름에 사용할 경우에는 전부 대문자로 지정하지 않는다.

|  |
| --- |
| ex) exportHtmlSource(); // exportHTMLSource(); 가 아님  openDvdPlayer(); // openDVDPlayer(); 가 아님 |

* 호출하려는 객체의 이름을 통해 의미를 짐작할 수 있다면, 메소드의 이름을 간략화할 수 있다.

|  |
| --- |
| ex) line.getLength(); // line.getLineLength(); 가 아님 |

* get/set 이라는 용어는 반드시 애트리뷰트에 직접 접근하는 메소드에 사용한다.

|  |
| --- |
| ex) employee.getName(); matrix.getElement(2, 4); employee.setName(name); matrix.setElement(2, 4, value); |

* is 접두사를 불리언 변수와 메소드에 사용한다.

|  |
| --- |
| * ex) isSet, isVisible, isFinished, isFound, isOpen |

* is 접두사를 사용함으로써 일반적으로 status 와 flag 와 같은 좋지 않은 불리언 이름을 선택하는 문제를 해결할 수 있다. 일부 상황에서는 is 접두사가 아닌 보다 더 적합한 접두사를 사용할 수도 있습니다.has, can, should 접두사들을 그런 대안으로 활용가능 하다.

|  |
| --- |
| ex) booleanhasLicense(); booleancanEvaluate();  booleanshouldAbort = false; |

* compute 라는 용어는 무엇인가를 계산(시간이 소요되는)하는 메소드에 사용할 수 있다.

|  |
| --- |
| ex) valueSet.computeAverage(); matrix.computeInverse(); |

* find 라는 용어는 무엇인가를 찾는 메소드에 사용할 수 있다.

|  |
| --- |
| ex) vertex.findNearestVertex(); matrix.findMinElement(); dao.findList(); |

* initialize 라는 용어는 객체나 개념이 확립되어지는 곳에 사용할 수 있다.

|  |
| --- |
| ex) printer.initializeFontSet(); |

* JFC (Java Swing) 변수들은 각 컴포넌트의 타입을 접미사로 사용한다.

|  |
| --- |
| ex) widthScale, nameTextField, leftScrollbar, mainPanel, fileToggle, minLabel, printerDialog |

* 대응하는 단어가 있는 이름은 반드시 함께 사용한다.

|  |
| --- |
| ex) get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide ... |

* 축약형 이름의 사용은 금한다.

|  |
| --- |
| ex) computeAverage(); // compAvg(); 가 아님 initialize(); // init(); 가 아님 |

* 도메인에서 사용하는 특수한 두문자어나축약어들은축약형을 그대로 사용한다. 이들 구문은 굳이 풀어 나열하지 않고 축약형을 사용하십시오. 다음과 같이 사용하시면 안됩니다.

|  |
| --- |
| ex) html 사용 //HypertextMarkupLanguage (x) cpu사용 //CentralProcessingUnit (x) |

## 상수명 부여 규칙

### 대문자 사용 원칙

* 상수에 대문자를 사용하는 것은 다른 종류의 명칭들과 구별하기 위함이다.
* 상수(final 변수)를 표현하는 이름은 반드시 모두 대문자로 지정하되 '\_' 를 사용하여 단어들을 구분한다.  
   ex) MAX\_ITERATIONS, COLOR\_RED

## 소스 파일 작성 규칙

### 소스 파일 설명

.html, .js, .java 등 소스 파일은 파일의 최상위 부분에 그 파일에 대한 정보를 표시해주어야 한다.

* 다음과 같은 주석(Doxygen지원형식)을 작성한다.

|  |
| --- |
| /\*\*  \* @Project : 프로젝트명  \* @Class Name : 패키지명을포함안 클래스 풀 네임  \* @Autor : 우병은  \* @Description : 소스코드에 대한 설명  \* @Modification Information  \* @  \* @ 수정일 수정자 수정내용  \* @ ------------- --------- -------------------------------  \* @ 2014. 10. 14. 우병은 최초생성  \*/ |
|  |

## Java 코딩 스타일

### 공통

* 코드는 한줄에 하나의 문장만 기술한다.
* java class 내에서 Static 변수는 항상 static final로 선언한다.
* public 변수는 사용하지 않는다.
* 변수선언시 classtype 별로 모은다.
* 변수 선언 시에는 항상 초기화를 한다.(ex. String str = null;)
* 배열 선언은 [class][] [variable-name]로 한다. (ex. EgovString[] sampleArray = {"1","2"};)
* 변수 선언 순서
  + - * constant 변수
      * private 변수
* protected method는 사용하지 않는다.
* method 선언 순서
  + - * constructor(default기술 후 argument가 적은 순으로)
      * main method
      * private method
      * public method
* return 값에는 연산을 수행하지 않는다.
* code상에서 의도적으로 Exception을 발생시키지 않는다.
* code상에서 Exception을 사용해서 비지니스 로직을 처리하지 않는다.

### String

* String의 연결 시에는 StringBuffer를 사용한다.

### 블록문

* 내용이 없는 블록문은 사용하지 않는다.(꼭 사용해야 하는 경우 Log라도 입력한다.)

### 반복문

* 반복문 내는 가급적이면 신규 변수를 생성하지 않는다.
* 반복문 조건절에는 연산이 들어가지 않도록 한다.

### 조건문

* 조건문은 3중을 초과해서 사용하지 않는다.
* 조건절에 not(!)을 사용하지 않는다. (가독성 저하 우려)

## 주석

### 항목 뒤에 붙이는 주석

* 소스 한 라인에 대한 주석문 혹은 변수의 선언 후에 변수에 대한 설명을 짧게 적을 경우에 문자의 오른편에 기술한다.
* Doxygen을 문서 출력을 위한 형식으로 작성한다

|  |
| --- |
| private PageContextpageContext; //< JSP페이지의 모든 객체정보를 가지고 있다.  private long startTime; //< 시스템의 현재의 시간을 가져온다. |

### 항목 위에 붙이는 주석

주석을 항목 위에 붙이는 경우는 두 라인 이상의 명령문에 대한 해석이나 메소드에 대한 설명을 붙이는 경우이다.

* 주석 내용의 위 아래에 단일선으로 시작과 끝을 표시하도록 한다.

|  |
| --- |
| /\*\*  \* 설명을 작성한다.  \*  \* Use {@link #doMove(int, int, int, int)} to move a piece.  \*  \* @paramtheFromFile 파라미터에 대한 설명을 작성한다.  \* @paramtheFromRank 파라미터에 대한 설명을 작성한다.  \* @paramtheToFile 파라미터에 대한 설명을 작성한다.  \* @paramtheToRank 파라미터에 대한 설명을 작성한다.  \* @return 반환값에 대한 설명을 작성한다.  \*/  booleanisValidMove(inttheFromFile, inttheFromRank, inttheToFile, inttheToRank){  ...  }  /\*\*  \* Moves a chess piece.  \*  \* @see java.math.RoundingMode  \*/  void doMove(inttheFromFile, inttheFromRank, inttheToFile, inttheToRank){  ...  } |

## 영역 (area)

* 영역이란 소스 파일을 기술하는 내용의 성격에 따라서 분류하는 것을 의미하며 다음과 같은 순서에 따라 서술하도록 한다.

### 패키지 선언

* 자바 클래스가 위치한 패키지 명을 가장 우선 선언한다.

|  |
| --- |
| package gw; |

### 임포트(import) 선언

* 자바 표준 API 패키지를 먼저 기술하고, 커스텀 패키지를 나중에 기술한다.

|  |
| --- |
| importjava.util.\*;  importjavax.servlet.\*;  importjavax.servlet.http.\*;  importjavax.servlet.jsp.\*; |

### 클래스 선언

* 클래스의 명칭과 공개 여부 등을 선언한다.
* Type(형식)에 따른 접두사는 사용하지 않는다.

ex) public class CDebugBean (X), public class DebugBean(O)

|  |
| --- |
| public class DebugBean |

### 상수 선언

* 상수는 전부 대문자로 생성하고 각 어절의 사이는 언더바(\_)로 구분하여 기록한다.

|  |
| --- |
| private static final boolean TRACE\_FLAG = true;  private static final String LINE\_FEED =System.getProperty("line.separator"); |

### 변수 선언

* public, protected, pritvae 순으로 변수를 선언한다.

|  |
| --- |
| private PageContextpageContext; // JSP페이지의 모든 객체를 가지고 있다.  private long startTime; // 시스템의 현재의 시간을 가져온다.  private String debugType;  private ServletContext context; |

### 생성자(constructor) 선언

* 클래스의생성자메소드를 가장 우선적으로 기술한다.

|  |
| --- |
| public DebugBean() {  debugType = "log";  } |

### property 선언

* 클래스의 외부에서 사용할 수 있는 모든 property는 생성자 및 파괴자 다음에 기술해야 한다.

|  |
| --- |
| /\*\*  \* setDebugType  \* debugType값을 설정한다.  \*/  public void setDebugType(String debugType) {  this.debugType = debugType;  } |

### public,protected, private 메소드 선언

* public ->protected-> private 순으로 기술한다.

|  |
| --- |
| public void setDebugType(String \_str){  }  public void getDebugType(){  }  protected void onCreated(){  }  protected void onCreated(){  }  private void doSomething(){  } |

1. JSP 프로그램 코딩 규칙

## 머리말(header)

## 주석(comment)

HTML 스타일의 주석은 클라이언트에게 전달되기 때문에 주의가 필요하다

### 영역에 대한 주석

|  |
| --- |
| <%------------------------------------------------------------------  comment area  ------------------------------------------------------------------%> |

### 한 줄에 대한 주석

|  |
| --- |
| <% // comment area %> |

### 여러 줄에 대한 주석

|  |
| --- |
| <% --  comment area  -- %> |

### 항목 뒤에 붙이는 주석

* 변수 선언에는 반드시 주석을 서술한다.

|  |
| --- |
| String count; // 전체 행수를 취득  String sql; // 행수 취득 sql  intboardPage = 1; // 선택 page  intmaxList = 3; // 모든 열수  inttotalRow = 0; // 모든 행수  inttotalPage = 4; // 보여 줄 전체 page |

## 블록 설정 시 코딩 형식

### 명령문이한줄이 경우

|  |
| --- |
| if (str\_Page != null )  i\_boradPage = Integer.parseInt(str\_Page); |

### 명령문이한줄이고 else가 있을 경우

|  |
| --- |
| if ( (I\_total\_row % i\_MAX\_list ) > 0 )  i\_totalPage = i\_tempPage + 1;  else  i\_totalPage = i\_tempPage + 1; |

### 명령문이 2줄이상이고 else가 있을 경우

|  |
| --- |
| if ( (i\_total\_row % i\_MAX\_list ) > 0 )  {  i\_totalPage = i\_tempPage + 1;  ……  }  else  {  i\_totalPage = i\_tempPage + 1;  ……  } |

## 영역 (area)

영역이란 소스 파일을 기술하는 내용의 성격에 따라서 분류하는 것을 의미한다.

### 설명문(description) 서술

* 소스의 관련 시스템 명, 작성자, 작성 이력, 기능에 대한 설명 등을 서술한다.
* JSP 머리말 작성 방법을 참고하라.

### 지시어(directive) 서술

* JSP 페이지지시자는 JSP 변환 시점에 관련된 속성을 정의한다.
* JSP 스펙은 같은 페이지에 JSP 페이지 지시자의 수량을 제한하지 않는다.
* 페이지에 사용할 스크립트 언어, 세션 트래킹의 사용 여부, 그리고 오류를 기록할 페이지 정보, JAVA import 경로, file include, 커스텀 태그 라이브러리 등을 선언한다.

|  |
| --- |
| ①<%@ page language="java" contentType="text/html" %>  ②<%@ page errorPage="/public/script/errorPage.jsp" %>  ③<%@ tagliburi="/WEB-INF/ufTagLib.tld" prefix="uftag" %>  <%@ tagliburi="http://java.sun.com/jsp/jstl/core" prefix="jstlc" %>  <%@ tagliburi="http://java.sun.com/jsp/jstl/functions" prefix="jstlfn" %>  ④<%@ page import="java.util.List" %>  <%@ page import="jsp.sql.DataSourceManager" %>  <%@ page import="jsp.sql.ConnectionWrapper" %>  <%@ page import="jsp.beans.security.\*" %> |

① JSP에서 사용하는 언어, 컨텐츠 타입을 정의한다.

② 페이지 내부에서 예외가 발생할 경우, 오류를 처리할 페이지 주소를 나타낸다.

③ 태그 라이브러리 지시자는 JSP 에서 사용하는 커스텀 태그 라이브러리를 선언한다. 짧은 지시자는 하나의 줄에 선언할 수 있다. 여러 개의 태그 라이브러리 지시자들은 한 곳에 같이 모아둔다:

④ import하는 외부 클래스들을 선언한다. 가급적 ".\*" 를 이용해 패키지를 import하지 말고, 클래스를 일일이 선언한다.

### 각종 빈(bean) 선언

* JSP 페이지 내에서 참조하거나 생성하는 모든 bean에 대한 선언을 기술한다.

|  |
| --- |
| <%-- 디버깅 빈 적재 --%>  <jsp:useBean id="debug" class="jsp.beans.util.DebugBean" >  <jsp:setProperty name="debug" property="pageContext" value="<%=pageContext %>" />  <jsp:setProperty name="debug" property="debugType" value="log" />  </jsp:useBean>  <%-- 코드 관리 빈(bean) 적재 --%>  <jsp:useBean id="codeRegistry" scope="session" class="jsp.beans.registry.CodeRegistryBean">  <jsp:setProperty name="codeRegistry" property="dataSource" value="admDB" />  </jsp:useBean> |

### 빈(bean)의 속성(property) 설정

* 앞서 선언한 빈(bean)의 속성을 설정하는 코드를 기술한다.

|  |
| --- |
| <jsp:setProperty name="codeRegistry" property="pageContext" value="<%=pageContext %>" />  <jsp:setProperty name="codeRegistry" property="codeId" />  <jsp:setProperty name="codeRegistry" property="codeName" />  <jsp:setProperty name="codeRegistry" property="codeExplain" />  <jsp:setProperty name="codeRegistry" property="codeDate" />  <jsp:setProperty name="debug" property="debugLog" value='<%=request.getParameter("operation") %>' /> |

### 선택적 JSP 선언문

* JSP 선언문은 JSP에서 소속된 메소드와 변수를 선언한다.
* 선언문은 <%! ... %> JSP 선언문 블록 하나에 모두 몰아서 한 곳에 모아 둔다.

|  |
| --- |
| <%!  private inthitCount;  private Date today;  public intgetHitCount() {  return hitCount;  }  %> |

### 각종 상태 점검 및 초기화 코드 작성

* 사용자 로그인 상태 체크, 페이지 파라미터 점검 등의 작업을 수행하는 코드를 기술한다.

|  |
| --- |
| <%-- 사용자 로그인 확인 --%>  <pubtag:validateSession name="userBean" /> |

### HTML 태그 서술

* HTML 소스의 시작을 표시하는 <HTML> 태그를 기술한다.
* 비즈니스 로직(business logic)만을 처리하며, 화면에 내용을 표시하지 않는 페이지일 경우에는 생략한다.

### <HEAD></HEAD> 블록 서술

HTML 페이지에서 머리말(header)에 해당하는 블록이며, 이 부분에는 다음과 같은 내용을 서술한다.

HTML 페이지의 간단한 제목, <TITLE> 태그

CSS(Cascading Stytle Sheet) 기술

CSS를 사용해서 헤딩, 테이블 등의 공통적인 특성을 중앙에서 제어하도록 한다. 이 방법은 사용자들에게 프리젠테이션의 일관성을 향상시키고, 유지보수 노력과 JSP 페이지의 코드 사이즈를 줄여준다.

* 가급적 아래의 예제 중에서 앞선 방식을 권장한다.

|  |
| --- |
| <LINK REL=StyleSheet HREF="../css/common.css"> |

또는,

|  |
| --- |
| <STYLE TYPE="text/css">  .ic {  font-family: webdings;  color: #999900;  }  </STYLE> |

클라이언트 자바스크립트(Client-side Javascript) 메소드 기술

자바스크립트가 제대로 돌아가기 위해서는 자바스크립트는 브라우저 타입의 특성과 관계없어야 된다. 또한 동작을 한다면, JSP에서 자바스크립트 코드를 독립적인 파일로 분리하고, 다음과 같이 JSP에서 자바스크립트를 불러쓰는 방식을 권장한다.

이렇게 하면 자바스크립트의 재사용성도 향상되고, 여러 JSP 사이에 일관된 습성의 자바스크립트를 유지할 수 있고, JSP 페이지의 복잡성을 줄여준다.

|  |
| --- |
| <S-RIPT LANGUAGE="JavaS-ript" SRC="./Lib/jsc/gwMenu.jsc"></SCRIPT>  <S-RIPT LANGUAGE="JavaS-ript">  function goSubMenu( url )  {  varthisForm = document.forms[0];  thisForm.action = url;  thisForm.submit();  }  </SCRIPT> |

### <BODY></BODY> 블록 기술

HTML 코드 작성시 HTML 태그 혹은 예약어는 모두 대문자로 서술하는 것을 원칙으로 한다.

JSP 스크립틀릿 JSP Scriptlets

가급적, 태그라이브러리로 동일한 기능을 구현할 수 있으면, JSP 스클립틀릿은 피하기 바란다. 이렇게 하면 페이지 가독성도 높이고, 유지보수도 쉬워진다.다음 예제는 customers 표시 방법이 데이터 형에 따라서 다른 스크립틀릿을 사용하는 것을 보여준다:

customers 는 Customers의 배열

|  |
| --- |
| <table>  <% for ( inti=0; i<customers.length; i++ ) { %>  <tr>  <td><%= customers[i].getLastName() %></td>  <td><%= customers[i].getFirstName() %></td>  </tr>  <% } %>  </table> |

customers 는 Enumeration 형

|  |
| --- |
| <table>  <% for ( Enumeration e = customers.elements();  e.hasMoreElements(); ) {  Customer customer = (Customer)e.nextElement();  %>  <tr>  <td><%= customer.getLastName() %></td>  <td><%= customer.getFirstName() %></td>  </tr>  <% } %>  </table> |

그러나, 공통 태그라이브러리를 사용하면, customers 의 데이터 형이 달라도 보다 높은 유연성이 생긴다. 예를 들어서, JSTL 에서는customers 를 나타내기 위해서 배열과 Enumeration 양쪽 모두 지원하는 다음과 같은 코드로 표시할 수 있다:

|  |
| --- |
| <table>  <c:forEachvar="customer" items="${customers}">  <tr>  <td><c:out value="${customer.lastName}"/></td>  <td><c:out value="${customer.firstName}"/></td>  </tr>  </c:forEach>  </table> |

비지니스로직과프리젠테이션 층과의 결합도을 줄이기 위한 모델-뷰-컨트롤러(MVC) 디자인 패턴을 적용하려고 한다면, JSP 스크립틀릿은비지니스로직을 구현하기 위해서 사용되어서는 안된다. 오히려, JSP스크립틀릿은 클라이언트의 요청을 처리한 뒤에 나오는 데이터("가치 객체 value objects"로

불리는)를 적절한 클라이언트의 준비된 형식에 전송하기 위해 필요하다면 사용된다. 심지어 그럴 경우에도, 프론트 컨트롤러 서블릿이나커스텀 태그로 하는 게 더 낫다.

JSP 표현식 JSP Expressions

JSP 표현식은 JSP scriptlet처럼 가능하면 삼가는 것이 좋다. 동일한 작업을 수행하는 다음 3가지 예제가 잘 설명해준다. 가급적 나중에 기술될 코드 스타일을 권장한다.

|  |
| --- |
| Example 1 (자바 코드 사용):  <%= myBean.getName() %>  Example 2 (JSP 태그 사용):  <jsp:getProperty name="myBean" property="name" />  Example 3 (JSTL 태그 사용):  <c:out value="${myBean.name}" /> |

프로그래밍 습관 Programming Practices

일반적으로, 다음과 같은 이유로 JSP 페이지에 자바 코드(선언문, 스크립틀릿, 표현식) 쓰기를 피한다:

* JSP 페이지의 자바 코드 문법 에러는 페이지가 배치되기 전까지 발견되지 않는다. 반면에, 태그 라이브러리와 서블릿에 있는 문법 에러는 배치 이전에 발견된다.
* JSP 페이지 내의 자바 코드는 디버깅하기 힘들다.
* JSP 페이지 내의 자바 코드는 유지보수하기 힘들다.
* 보통 복잡한 비즈니스 로직과프리젠테이션로직을 섞어놓지 않는 것이 납득이 가는 습관이다. JSP는 우선적으로 프리젠테이션로직을 목적으로 한다.
* 자바 코드가 포함된 코드와 HTML과 다른 스크립트 명령어가 섞여있다면 읽기 어렵다.
* JSP 2.0 은 더 단순한 표현 언어(EL) 덕분에 scriptlet의 중요도를 낮추었다. JSP 에서JSP 2.0 스타일로 쉽게 옮기기 위해서는 자바 코드를 사용하지 않는 것이 좋다.

JSP 내장 객체 JSP Implicit Objects

API 호출을 통해서 객체를 참조하기 보다는 JSP 내장 객체를 직접 사용하는 것이 좋다. 그래서 ServletContext인스턴스에 의해 제공되는 초기 파라미터에 접근하기 위해 다음을 쓰는 대신에 다음의 코딩 스타일 중 가급적 나중 방식을 권장한다.

|  |
| --- |
| getServletConfig().getServletContext().getInitParameter("param")  application.getInitParameter("param")  <c:out value="${initParam['param']}" /> |

인용부호 (Quoting)

일정한 인용부호의 사용을 채택한다. 인용부호는 작은따옴표 ' 대신 큰따옴표 " 를 사용한다.

인용 부호 표기법

|  |  |
| --- | --- |
| 불규칙한 인용부호 | <%@ page import='javabeans.\*'%> |
| 좋은 인용부호 | <%@ page import="javabeans.\*" %> |

예외적인경우는 작은따옴표가 필요할 때이다. 예를 들어서 스크립트 언어에서 큰따옴표를 사용하는경우를 들 수 있다:

|  |
| --- |
| <jsp:include page='<%= getFoodMenuBar("Monday") %>' /> |

커스텀 태그 사용하기 Using Custom Tags

만일커스텀태그가몸체가 없다면, 컨텐트는 (생략해서 "JSP" 기본값으로 하지 말고) empty라고 명확하게 선언되어야 한다. 태그 라이브러리 설명서에서 다음과 같이 정해준다.

|  |
| --- |
| <tag>  <name>hello</name>  <tag-class>com.mycorp.util.taglib.HelloTagSupport</tag-class>  <body-content>empty</body-content>  ...  </tag> |

이렇게 하면 JSP 컨테이너는 태그 몸체가 무조건 비어야 되며 파싱할 어떤 JSP 문법도 포함하지 않아야 된다고 인식한다. 효과는 빈 몸체를파싱하기 위해서 불필요하게 자원의 할당이 이뤄지는 것을 제거한다.

빈 태그는, 가독성향상을 위해 열고 닫는 XML 엘리먼트 쌍으로 표시하기보다는, 짧은 XML 엘리먼트로 표시한다. 그래서,

|  |
| --- |
| <myTag:hello /> |

라고 표기하는 것이 아래보다 낫다.

|  |
| --- |
| <myTag:hello></myTag:hello> |

스크립트엘리먼트의 들여쓰기 Indentation of Scripting Elements

JSP 스크립트엘리먼트(선언문, 스크립틀릿, 표현식과 같은)가 한 줄에 맞지 않는 경우, 스크립트 언어의 들여쓰기 작성요령이 엘리먼트 내에 적용된다. 몸체는 엘리먼트의 여는 기호 <%=와 같은 줄에서 시작한다. 다음줄의 기준도 여는 기호 <%=가 된다. 몸체의 끝은 다른 줄에 있는 엘리먼트의 닫는 기호(%>)로 마무리된다. 예제이다:

|  |
| --- |
| <%= (Calendar.getInstance().get(Calendar.DAY\_OF\_WEEK)  = Calendar.SUNDAY) ?  "Sleep in" :  "Go to work"  %> |

첫줄과 마지막 줄을 제외한 몸체의 가운데 줄들은 단위 들여쓰기 공백(앞 예제에서 로 보인다)이 앞에 있기 때문에 나머지 JSP 보다 확연히 드러나게 된다.

JSP, HTML, JAVA의 복합 들여쓰기 (Compound Indentation with JSP, HTML and Java)

자바의 스크립트 코드와 템플릿 텍스트(HTML)가 섞여있는 JSP 엘리먼트를 위해서 복합 들여쓰기가 필요한 이유는 JSP 소스를 이해하기 위해 드는 노력을 덜어주기 위해서이다. 기존의 들여쓰기 요령은 JSP 소스파일을 더 알아보기 어렵게 할 수 있기 때문이다.

보편적인 규칙에 따라서, 다른 소스 사이에 끼어 있을 경우마다 보통 들여쓰기에 특별한 단위를 할당한다. 클라이언트에 보여지는 들여쓰기에 영향을 준다. 추가적인 들여쓰기는 (브라우저에 의해) 보통 무시되고, 브라우저에 나타나는 결과는 차이가 없다. 예를 들어, <TABLE> 태그 앞에 공백이 몇개 더 추가된다고 해서, 브라우저에서 보여지는 테이블의 위치가 달라지는 것은 아니다. 그래서 이러한 작성요령을 사용하는 것이 더 좋아 보인다:

|  |
| --- |
| <table>  <% if { tableHeaderRequired ) { %>  <tr>  <th>Last Name</th>  <th>First Name</th>  </tr>  <% } %>  <c:forEachvar="customer" items="${customers}">  <tr>  <td><c:out value="${customer.lastName}"/></td>  <td><c:out value="${customer.firstName}"/></td>  </tr>  </c:forEach>  </table> |

위 방식보다는 아래 방식이 낫다:

|  |
| --- |
| <table>  <% if { tableHeaderRequired ) { %>  <tr>  <th>Last Name</th>  <th>First Name</th>  </tr>  <% } %>  <c:forEachvar="customer" items="${customers}">  <tr>  <td><c:out value="${customer.lastName}"/></td>  <td><c:out value="${customer.firstName}"/></td>  </tr>  </c:forEach>  </table> |