|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **JAVA** | | | | | | | | | | | | | | | |
| 클래스변수, 인스턴스변수, 지역변수, 각각의 선언되는 위치, SCOPE 설명(코드 예시) | | | | | | | | | | | | | | | |
| public class test {  int iv; // 인스턴스 변수  static int cv; // 클래스 변수  void method() {  int lv; // 지역 변수  }  } | | | | | | | | | | | | | | | |
| **종류** | | | | | | **선언 위치** | | | | | | **생성시기** | | | |
| 인스턴스 변수 | | | | | | 클래스 영역 | | | | | | 클래스 메모리에 올라갈 때 | | | |
| 클래스 변수 | | | | | | 인스턴스 생성 | | | |
| 지역 변수 | | | | | | 클래스 이외  (메서드, 생성자, 초기화 블럭) | | | | | | 변수 선언문 수행 | | | |
| 배열과 List 차이점 | | | | | | | | | | | | | | | |
| 배열 | | | | | | | | | | List | | | | | |
| 연속적인 메모리공간에 할당되어 시작 주소에서  인덱스 값을 더하면 해당 인덱스의 값을 가져  올 수 있음.  마지막을 제외한 인덱스에 추가 시, 기존  데이터들 뒤로 밀어야 된다. | | | | | | | | | | 노드가 순차적으로 있지 않는 경우도 있기 때문에 처음 노드부터 순차적으로 탐색을 해야됨. 데이터와 이전 노드가 가리키는 다음 노드의 주소값을 갖는 노드 생성 후 이전 노드가 새로 생성된 노드를 가리키게 하면 된다. | | | | | |
| 메서드 오버로딩과 오버라이딩의 차이점 | | | | | | | | | | | | | | | |
|  | | | 오버로딩(Overloading) | | | | | | | | 오버라이딩(Overriding) | | | | |
| 차이점 | | | 같은 이름의 메소드를 여러 개  가지면서 매개변수의 유형과 개수가  다르도록 함 | | | | | | | | 상위 클래스가 가지고 있는 메소드를 하위 클래스가 재정의 해서 사용 | | | | |
| 추상화, 상속, 다형성에 대해 설명과 코드 작성 | | | | | | | | | | | | | | | |
| **추상화** | | | 데이터나 프로세스 등을 의미가 비슷한 개념이나 표현으로 정의해 나가는 과정이면서 동시에 각 개별 개체의 구현에 대한 상세함은 감추는 것, 예를 들면, [프로그래머](http://www.terms.co.kr/programmer.htm)가 거의 같은 작업을 수행하는 두 개의 [함수](http://www.terms.co.kr/function.htm)를 하나의 함수로 합병할 수 있음을 나타내기 위해 추상화를 사용할 수 있다. 추상화는 [소프트웨어 공학](http://www.terms.co.kr/softwareengineering.htm)에서 가장 중요한 기법 중 하나로서, 두 개의 다른 기법인 [캡슐화](http://www.terms.co.kr/encapsulation.htm) 및 [데이터 은폐](http://www.terms.co.kr/datahiding.htm)와 매우 밀접한 관련이 있다. 이 세 가지 기법은 복잡성을 줄이기 위해 사용 | | | | | | | | | | | | |
| **상속** | | | 자식 클래스가 부모 클래스의 기능(자료, 함수)을 받아 쓰는 것  이미 정의된 클래스를 바탕으로 필요한 기능을 추가하여 정의  소프트웨어 재사용 지원  상속을 통해서 기존의 클래스를 상속받은 하위 클래스를 이용해 프로그램의 요구에 맞추어 클래스를 수정할 수 있고 클래스 간의 종속 관계를 형성함으로써 객체를 조직화할 수 있다. | | | | | | | | | | | | |
| **다형성** | | | 어떤 한 요소에 여러 개념을 넣어 놓는 것으로 일반적으로 [오버라이딩](https://ko.wikipedia.org/w/index.php?title=%EC%98%A4%EB%B2%84%EB%9D%BC%EC%9D%B4%EB%94%A9&action=edit&redlink=1)(같은 이름의 메소드가  여러 클래스에서 다른 기능을 하는 것)이나 [오버로딩](https://ko.wikipedia.org/w/index.php?title=%EC%98%A4%EB%B2%84%EB%A1%9C%EB%94%A9&action=edit&redlink=1)(같은 이름의 메소드가 인자의 갯수나 자료형에 따라서 다른 기능을 하는 것)을 의미한다.  다형 개념을 통해서 프로그램 안의 객체 간의 관계를 조직적으로 나타낼 수 있다. | | | | | | | | | | | | |
| 추상클래스와 인터페이스의 공통점과 차이점 | | | | | | | | | | | | | | | |
|  | | | 추상클래스 | | | | | | | | 인터페이스 | | | | |
| 공통점 | | | 클래스  하위 클래스에서 모든 추상메서드를 구현해야 함 | | | | | | | | | | | | |
| 차이점 | | | 추상메서드 외 일반 멤버 변수와  메서드를 가질 수 있다.  Extends를 사용  단일 상속만 가능  작업의 레벨 분할을 위해서 사용 | | | | | | | | 추상메서드와 static final변수만 사용  Implement 를 사용  중복구현 가능  공동 작업을 위한 상호간의 인터페이스를 위해 사용  서로 다른 클래스를 연결시켜줌 | | | | |
| 내부 클래스의 필요성 | | | | | | | | | | | | | | | |
| 코드를 더 읽기 쉽고, 더 오래 유지하기 위해 논리적인 그룹과 인터페이스들에서 이너클래스 사용 개인적인  데이터 멤버와 메소드를 포함하는 외부 클래스의 모든 멤버들에게 접근 할 수 있음  1) 개인적인 것을 포함하는 외부 클래스의 모든 멤버에 접근 할 수 있다는 관련성의 특별한 타입  2) 논리 그룹 클래스와 인터페이스 내부에 있기 때문에 읽기 쉽고 유지 가능한 코드 개발에 사용  3) 코드 최적화 : 작성하는데 더 적은 코드가 요구 | | | | | | | | | | | | | | | |
| 기본 타입의 형변환과 참조타입의 형변환 코드 작성 | | | | | | | | | | | | | | | |
| 자동 형변환(Up-Casting) | | | | | | | | | | 명시적 형변환(Down-Casting) | | | | | |
| 작은 자료형에서 큰 자료형 방향으로 변환 | | | | | | | | | | 변수에 값을 대입할 때 값의 자료형이 변수의 자료형보다 크면 컴파일 에러 발생 | | | | | |
| long money = 300;  float x = 10L | | | | | | | | | | float f = (float) 1.1; //컴파일 에러  byte b = (byte) 258; //b에 2가 할당 | | | | | |
| **Servlet / JSP** | | | | | | | | | | | | | | |
| HttpServlet init(), service(), destroy() 메서드의 호출 시점, HttpServlet 객체의 생명주기 서술 | | | | | | | | | | | | | | |
|  | | **init()** | | | | | | **service()** | | | | | **doGet()/ doPost()** | |
| 호출 시점 | | 컨테이너는 서블릿 인스턴스를 생성한 다음 init() 메소드를 호출  service() 메소드 전에 실행 | | | | | | 최초 클라이언트의 요청을 받았을 때, 컨테이너는 새로운 스레드를 생성하거나, 아니면 스레드 풀로부터 하나를 가지고 와서 서블릿의 service() 메소드를 호출 | | | | | service() 메소드가 클라이언트의 HTTP 메소드(GET, POST등)를 참조하여 doGet() 또는 doPost()를 호출  여기서 doGet(), doPost()만 언급하는 이유는 이것 말고 나머지는 사용할 경우가 거의 없음 | |
| 목적 | | 클라이언트의 요청을 처리하기 전에 서블릿을 초기화할 기회를 주는 것 | | | | | | 클라이언트의 HTTP 메소드(GET, POST 등)를 참조하여 doGet()을 호출할지, doPost()를 호출할지 판단 | | | | | 여기에다가 코딩을 하면 됨어떤 웹 애플리케이션이든지, 무엇을 하든지 관계없이 이곳이 바로 그 작업을 하는 곳  물론 다른 객체에 있는 다른 메소드를 호출할 수 있음  하지만 모두 여기에서부터 시작 | |
| 재정의 | | 가능 초기화할 코드가 있다면 init() 메소드를 재정의  (데이터베이스에 대한 접속, 다른 객체에 서블릿을 등록하는 등). | | | | | | 거의 하지 않습니다. service() 메소드를 재정의할 필요는 없음  doPost()나 doGet()을 재정의하여 HttpServlet의 service()가 이를 실행하도록 하면 됨 | | | | | 항상, 둘 중 하나는 반드시(doGet()이든 doPost()든) 재정의 서비스 하고자 하는 메소드를 재정의. 만약 doPost()를 재정의하지 않았다면, 컨테이너는 "이 서블릿은 HTTP POST를 지원하지 않는구나"라고 생각 | |
| 컨테이너가  관리하는  서블릿 | | 1. 사용자가 서블릿에 대한 링크(URL)를 클릭 2. 컨테이너는 요청된 Request가 서블릿이라는 것을 간파하고 객체를 생성   1. HttpServletResponse / 2. HttpServletRequest   1. 접수한 요청의 URL을 분석, 어떤 서블릿을 요청했는지 파악(DD 참조) 그 다음 해당 서블릿 스레드를 생성하여 Request, Response 객체 참조를 넘김 2. 컨테이너는 서블릿 service() 메소드를 호출. 브라우저에서 지정한 방식에 따라 doGet()을 호출할지, doPost()를 호출할지 결정. 클라이언트가 HTTP GET 메소드를 날렸다면, service() 메소드는 서블릿의 doGet() 메소드를 호출. 호출할 때 Request와Response객체를 인자로 넘김 3. 서블릿은 클라이언트에게 응답을 작성하기 위하여 Response객체를 사용. 이 작업을 완료하면, Response에 대한 제어는 컨테이너에게 넘어감. 4. service() 메소드가 끝나면, 스레드를 소멸하거나 아니면 컨테이너가 관리하는 스레드 풀(Pool)로 돌려보냄. 그 다음 Request와 Response 객체는 가비지 컬렉션이 될 준비, 이 객체에 대한 참조는 이제 범위를 벗어나기에 사라짐 마지막으로 클라이언트는 서버로부터 응답을 받게 됨 | | | | | | | | | | | | |
| 서블릿  일생 | | 1. 웹 컨테이너가 클래스 로딩 후 서블릿 인스턴스화(생성자 실행) 디폴트 생성자(인자가 없는) 실행 (생성자를 코딩 하지 말 것, 컴파일러가 자동으로 만들어주는 것을 사용) 2. init() 서블릿 일생 중 한번만 호출. init() 메소드는service()메소드 전에 실행되야 함. 데이터베이스에 대한 접속, 다른 객체에 서블릿을 등록하는 것과 같은 초기화 코드가 있다면 init() 메소드를 재정의. 재정의하지 않으면 GenericServlet()의 init()이 실행. 그 다음 요청이 들어오고, 컨테이너가 새로운 스레드를 만들거나 아니면 하나를 할당. 그리고 service() 메소드를 호출. 3. service() : 여기서 일생의 대부분을 보냄. 재정의하지 않았다면, 당연히 HttpServlet의 service() 메소드가 실행. 클라이언트 요청을 핸들링함. doGet(), doPost() 등 요청이 들어올 때마다 새로운 스레드에서 실행됨. 4. destroy() 서블릿이 죽기 전에 가지고 있는 자원들을 깨끗이 정리할 기회를 줌. init() 메소드처럼 오직 한번만 호출. | | | | | | | | | | | | |
| DD  (배포서술자, Deployment Descriptor) = web.xml | | * Servlet, Error Page, Listener, Fillter, 보안 설정등 Web Application의 설정 파일 * URL과 실제 서블릿의 매핑 정보도 담고 있다. * 하나의 웹 어플리케이션에 하나만 존재한다. * 보통 Web Document Root 디렉토리에 WEB-INF 폴더 아래 web.xml 파일로 존재한다. | | | | | | | | | | | | |
| HttpServletRequest, HttpServletResponse 역할 설명 | | | | | | | | | | | | | | |
| **HttpServletRequest** | | | | | | | | | | **HttpServletResponse** | | | | |
| 클라이언트의 모든 요청 정보 보유  Request header  From data, query parameter  InputStream(클라이언트로부터 전송되어지는 데이터)  기타 클라이언트 정보얻기 (세선정보, 쿠기, path…) | | | | | | | | | | 클라이언트에게 보내지는 모든 정보 보유  Response header  OutputStream  (클라이언트에게 보내지는 데이터)  쿠키 설정 / 세션 설정 | | | | |
| 웹 브라우저내에서 웹어플리케이션 서버를 통해 JSP파일을 호출,  Servlet Container(예를 들어 Tomcat)내에서의 처리 과정을 설명 | | | | | | | | | | | | | | |
| Servlet Container | | | | * HTTP 요청을 받아서 Servlet을 실행시키고, 그 결과를 사용자 브라우저에게 전달해주는 기능을 제공하는 컴포넌트 이다. * Servlet을 실행하고 생명주기를 관리하는 역할을 한다. * Servlet과 웹 서버(Apache, nginx..)가 서버 통신 할 수 있는 방법을 제공한다. * 멀티 스레딩을 지원하여 클라이언트의 다중 요청을 알아서 처리해준다. * 대표적인 Conatainer에는 Tomcat, jetty, jboss 등이 있다. | | | | | | | | | | |
| ① Client가 브라우저를 통해 페이지 요청(Http Request)  ② 컨테이너는 웹 서버가 위임한 요청을 파악  ③ 컨테이너에 basic 서블릿이 적재되어 있는지 확인, 없으면 객체 생성  ④ 디스패처 서블릿에게 응답을 보냄  ⑤ 요청을 처리하여 VIEW를 컨테이너에게 반환 | | | | | | | | | | | | | | |
| JSP에 내장되어 있는 속성변수(pageContext, request, session, application)들의 참조 범위 서술 | | | | | | | | | | | | | | |
| **내장객체** | | | | | **Type** | | | | | | | | | **scope** |
| Request | | | | | javax.servlet.ServletRequest  javax.servlet.HttpServletReqeust | | | | | | | | | request |
| response | | | | | javax.servlet.ServletResponse  javax.servlet.HttpServletRequest | | | | | | | | | page |
| pageContext | | | | | javax.servlet.jsp.PageContext | | | | | | | | | page |
| session | | | | | javax.servlet.http.HttpSession | | | | | | | | | session |
| application | | | | | javax.serlvet.ServletContext | | | | | | | | | application |
| out | | | | | javax.servlet.jsp.JspWriter | | | | | | | | | page |
| config | | | | | javax.servlet.ServletConfig | | | | | | | | | page |
| page | | | | | java.lang.Object | | | | | | | | | page |
| **Scope** | | | | | **적용 범위** | | | | | | | | | |
| page | | | | | 말 그대로 현재의 페이지에서만 적용 | | | | | | | | | |
| request | | | | | 요청(request)의 범위가 미치는 곳에서 적용  page를 포함, include, forward와 같이 request를 넘겨주는 페이지까지 적용 | | | | | | | | | |
| session | | | | | 사용자 세션이 유지되는 동안에는 모두 적용 | | | | | | | | | |
| application | | | | | 웹 어플리케이션이 기동되어 서버를 끄기 전까지는 모든 영역  (전체 웹 어플리케이션)에 적용 | | | | | | | | | |
| JSP:include 액션과 include 디렉티브의 사용상 차이점 설명 | | | | | | | | | | | | | | |
| - include 액션 태그는 include 디렉티브(<%@include>) 와 함께 다른 페이지를 현재 페이지에 포함시킬 수 있는 기능을 가지고 있다.  - include 디렉티브는 단순하게 소스의 내용이 텍스트로 포함이 되지만  include 액션 태그는 포함시킬 페이지의 처리 결과를 포함시킨다는 점이 다르다.  - 포함되는 페이지는 HTML, JSP, Servlet 페이지 등 모두 가능하다.  - include 디렉티브는 주로 조각 코드를 삽입할 때 사용되고,  include 액션 태그는 페이지를 모듈화할 때 사용된다. 즉 템플릿 페이지를 작성할 때 사용된다. | | | | | | | | | | | | | | |
| Cookie 객체와 Session 객체의 사용용도 설명 | | | | | | | | | | | | | | |
| **Cookie** | HTTP 프로토콜은 상태가 없는 즉, 이전에 무엇을 했고, 지금 무엇을 했는지에 대한 정보를 갖고 있지 않는 특성을 가짐  쿠키는 상태가 없는 프로토콜을 위해 상태를 지속시키기 위한 방법  쿠키는 웹 브라우저(클라이언트)의 정보를 웹 브라우저(클라이언트) 자신들에게 저장 | | | | | | | | | | | | | |
| **Session** | 세션은 웹 서버쪽의 웹 컨테이너에 상태를 유지하기 위한 정보를 저장  javax.servlet.http 패키지의 HttpSession 인터페이스를 통해서 세션을 사용  서버와 관련된 정보를 노출시키지 않기 위해서 쿠키를 사용하는 것 보다  HttpSession 인터페이스를 통한 세션을 통한 상태관리가 더욱 효율적  세션은 웹 브라우저당 1개씩 생성되어 웹 컨테이너에 저장 | | | | | | | | | | | | | |
| Filter 인터페이스와 ServletContextListener 인터페이스의 사용용도 설명 | | | | | | | | | | | | | | |
| **Filter 인터페이스** | | | | | 필터는 'HTTP 요청과 응답을 변경할 수 있는 재 사용 가능한 코드'이다.  필터는 객체의 형태로 존재하며 클라이언트로부터 오는 요청(request)과 최종 자원(서블릿/JSP/기타 문서) 사이에 위치하여 클라이언트의 요청 정보를 알맞게 변경할 수 있으며, 또한 필터는 최종 자원과 클라이언트로 가는 응답(response) 사이에 위치하여 최종 자원의 요청 결과를 알맞게 변경할 수 있다. | | | | | | | | | |
| **Filter 객체** | | | | | | | | | **설명** | | | | | |
| public void init(FilterConfig filterConfig)  throws ServletException | | | | | | | | | 필터를 웹 콘테이너내에 생성한 후 초기화할 때 호출 | | | | | |
| public void doFilter(ServletRequest request,  ServletResponse response, FilterChain chain) throws java.io.IOException, ServletException | | | | | | | | | 체인을 따라 다음에 존재하는 필터로 이동한다. 체인의 가장 마지막에는 클라이언트가 요청한 최종 자원이 위치한다. | | | | | |
| public void destroy() | | | | | | | | | 필터가 웹 콘테이너에서 삭제될 때 호출 | | | | | |
| **ServletContextListener 인터페이스** | | | | | | | 웹 어플리케이션의 시작과 종료시 자동으로 발생되는 이벤트를 수행하기 위한 메소드를 정의한 인터페이스이다. | | | | | | | |
| contextInitialized(ServletContextEvent sce) : void | | | | | | | | | 웹 컨테이너가 처음 구동될 때 실행되는 메소드 | | | | | |
| contextDestoryed(ServletContextEvent sce) : void | | | | | | | | | 웹 컨테이너가 종료될 때 실행되는 메소드 | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DB** | | | | | |
| 정보, 데이터, 데이터베이스, 데이터베이스관리시스템의 개념 설명 | | | | | |
| 정보 | | 데이터를 모아 둔 것이 자료라면 자료를 특정한 목적의 의사결정을 위해 가공한 형태  훌륭한 정보는 목적적합성과 신뢰성 및 적시성을 유지. 적시성이 매우 중요한 정보를 첩보로  따로 분류하기도 하며 가공을 통해 비교적 장기간 활용이 가능한 정보를 [지식](https://ko.wikipedia.org/wiki/%EC%A7%80%EC%8B%9D)으로 정의 | | | |
| 데이터 | | 이론을 세우는 데 기초가 되는 사실. 또는 바탕이 되는 자료. 관찰이나 실험, 조사로 얻은 사실이나 정보. 컴퓨터가 처리할 수 있는 문자, 숫자, 소리, 그림 따위의 형태로 된 정보. | | | |
| 데이터베이스 | | 데이터 베이스는 자료의 중복을 없애고, 효율적인 데이터를 모으기 위한 기술이다.  여러 응용 시스템들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 묶음  데이터를 효율적으로 활용하기 위해서 자료를 저장하고, 이에 대한 정보를 사용할 수 있게 끔  도와주는 데이터 저장 방식 중에 하나 | | | |
| 데이터베이스관리시스템 | | 다수의 사용자들이 [데이터베이스](https://ko.wikipedia.org/wiki/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4) 내의 데이터를 접근할 수 있도록 해주는 소프트웨어 도구의 집합이다. DBMS은 사용자 또는 다른 프로그램의 요구를 처리하고 적절히 응답하여 데이터를 사용할 수 있도록 해준다.  종류 : IMS, CODASYL DB, DB2, ORACLE, INFORMIX, SYBASE, INGRES, MS-SQL, Objectivity, O2, Versanat, Ontos, Gemstone, Unisql, Object Store, Starburst, Postgres, Tibero | | | |
| 데이터베이스 내부에서 select, from, where, group by, having, order by 처리 순서 나열 | | | | | |
| From > where > group by > having > select > order by | | | | | |
| 1) Self Join 2) Inner Join 3) left outer Join 4) right outer Join 5) full outer Join 설명 | | | | | |
| Self Join | | | - 1개의 테이블을 사용해서 조인 | | |
| Inner Join | | | - 각 테이블에 조건으로 들어간 컬럼값을 조합해서 결과 셋으로 반환  - 부모테이블(PK) - 자식테이블(FK)이 일치하는 행만 유효한 행으로 판단  - 단순 조인 -> (유효한 행 추출) -> 내부조인 | | |
| left outer Join | | | - 왼쪽 테이블의 것은 조건에 부합하지 않더라도 모두 결합되어야 한다. | | |
| right outer Join | | | -오른쪽 테이블의 것은 조건에 부합하지 않더라도 모두 결합되어야 한다. | | |
| full outer Join | | | -양쪽 모두에 조건이 일치하지 않는 것들까지 모두 결합하는 개념 | | |
| DML, DDL, DCL 각각 설명, 각각 3개 이상 명령어 작성 | | | | | |
| DDL | 데이터 정의 언어 | | | | CREATE, ALTER, DROP, TRUNCATE |
| DML | 데이터 조작 언어 | | | | SELECT, INSERT, DELETE, UPDATE |
| DCL | 데이터 제어 언어 | | | | COMMIT, ROLLBACK, GRANT, REVOKE |
| 데이터 무결성을 지키기 위해 사용되는 제약조건 나열과 설명 | | | | | |
| NOT NULL | | | | - 반드시 컬럼 값을 기재 (필수 값들은 NOT NULL로 정의하는 것이 좋음)  - 컬럼 레벨로만 정의 가능 | |
| PRIMARY KEY | | | | : 행을 구분하기 위한 유일성을 보장 받음  - 기본 키, 모든 테이블에 1개 이상이 있어야 함  - 키(KEY) : 속성, 컬럼  - 엔티티 객체들을 구분하는 식별자 -> 행과 행을 구분하는 유일한 식별자(\*\*\*)  - 자동으로 NOT NULL(NULL값을 가질 수 없음)  - 유일한 값을 가진다.(UNIQUE)  - 일반적으로 테이블에는 PK 1개가 있다.(1개 이상의 PK도 존재 가능- 복합키) | |
| Foreign Key | | | | - 두 테이블 사이의 관계를 선언함으로써, 데이터의 무결성을 보장해 주는 역할  - 하나의 테이블(외래 키 테이블)이 다른 테이블(기준 테이블)에 의존  -'외래 키 테이블'이 참조하는 '기준 테이블'의 열은 반드시 PK이거나 UNIQUE 제약 조건이 설정되어 있어야 한다. | |
| UNIQUE | | | | - 해당 컬럼 값은 테이블 내에서 유일해야 한다.(PK와 거의 유사함)  - 데이터의 유일성을 보장 받기 위해 사용  - 행을 구분하는 용도가 아니다!  - NULL을 가질 수 있다. (ENIQUE + NOT NULL -> PRIMARY KEY) ,  NULL은 데이터로 중복이 가능 | |
| CHECK | | | | - 열거형 제약 (미리 정해진 값 외에는 사용 불가)  - CHECK (컬럼명 IN (제약사항 열거)  - CHECK (컬럼명 BETWEEN 최소값 AND 최대값) | |
| DEFAULT | | | | -값을 입력하지 않았을 때, 자동으로 입력되는 기본 값을 정의 | |
| 데이터베이스 트랜잭션과 락(Lock)에 대해 설명 | | | | | |
|  | | | | | |
| 트랜잭션 | | - 오라클에서 발생하는 여러 개의 SQL 명령문들을 하나의 논리적인 작업  단위로 묶어 처리하는 기술  - 모두 성공하면 정상 종료를 하고 하나라도 실패하면 전체를 취소시킨다.  - 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구한다.  - 명령을 실행했던 구간, 단위  1. COMMIT / 2. ROLLBACK / 3. SAVEPOINT  - 글을 삭제하는데 오류가 나서 댓글만 삭제했을 경우  기존에 삭제했던 댓글을 복구 해야 하기 때문에  문제가 생김 -> 이런 문제를 해결하기 위해 트랜잭션으로 묶어줘야 한다.  - 쿼리 조작의 일련의 과정 | | | |
| 락 | |  | | | |
| 트리거 개념과 사용용도 설명 | | | | | |
| - 특정 사건이 발생하면 미리 준비해 놓은 일을 자동으로 실행하는 역할  - 데이터베이스가 미리 정해놓은 조건을 만족하면 자동으로 실행되는 업무  - 직접 호출은 불가능하고 미리 정해놓은 조건이 발생할 때만 실행된다.  - 프로시저의 일종  - 테이블 감시를 통한 업무를 진행(과하면 시스템이 느려질 수 있음)  - 테이블의 변경 사항이 생겼을 때, 레코드의 추가, 수정, 삭제 시 사용 | | | | | |
| 인덱스 사용시 장점 설명 | | | | | |
| 인덱스 | | - 검색을 빠른 속도로 하기 위해서 제공  - SQL 명령문의 처리 속도를 빠르게 하기 위해서 컬럼에 대해서 생성되는 객체  - B\* 트리 구조(자바 TreeSet과 유사)  - 오라클은 인덱스를 따로 생성하지 않아도 자동으로 생성이 된다.  (해당 테이블의 PK, UNIQUE 컬럼들이 자동으로 인덱스가 생성), 유일한 값  - 명시적으로 인덱스를 생성하는 경우  (주로 WHERE절의 대상이 되는 컬럼을 사용 권장) 총 관리자 | | | |
| 사용하는  경우 | | 1. 테이블의 행 개수가 많은 경우  2. WFERE 절에 해당 컬럼이 많이 사용이 되는 경우  3. 검색 결과가 전체 데이터의 2%~4% 정도 일때  - 100건의 테이블 중 2~4개의결과를 가져올 때  4. JOIN에 사용되는 컬럼(PK, FK)  5. NULL을 포함하는 컬럼이 많은 경우(NULL값은 색인에 제외된다.),  색인은 NULL값은 찾을 수 없음 | | | |
| 사용하지  않는 경우 | | 1. 테이블의 행 개수가 적은 경우  2, 검색 결과가 전체 데이터의 많은 비중을 차지하는 경우 | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **HTML/CSS** | | | | | | | |
| DTD, XML Schema의 장, 단점 설명 | | | | | | | |
|  | | | | **DTD** | | | **XML Schema** |
| 문법 | | | | XML 문서와 다른 문법을 쓰기 때문에 별도의 처리 시스템이 필요  예를 들어 DOM을 이용한 브라우징이나  트리 처리 등을 할 수가 없음 | | | XML을 위해 개발된 시스템을 이용  예를 들어 XPath, XLink와 같은 다른 표준들은 스키마와 함께 사용할 수 있도록 함.  또 DOM을 이용 한 검색이나 문서트리를 처리할 수가 있다. |
| 데이터 타입들 | | | | 최근 XML의 응용분야가 확산되면서 기존의 프로그래밍 수준에 가까운 다양한 데이터 타입,  예를 들면, number, date/time, structure 등을 요구하게 되었다.  그러나 DTD는 지원 가능한 데이터 타입이 제한적이다. (string, name tokens, ID 등) | | | 많은 데이터 타입을 제공  데이터 타입과 구조의 객체지향적 상속을 지원하는 데이터 타입들을 사용할 수 있게 됨. 유지 보수를 쉽게 할 수 있음, 객체지향의 개념에서의 장점을 포함할 수도 있음  장점 : XML을 이용하여 사용자 또는 설계자가 직접 자신이 필요한 문서의 구조를 기술할 수 있음 |
| 내용 모델(content model) | | | | 요소의 순서(order)나  출현 횟수(occurrence)의 지정이 제한적 | | | 출현 횟수를 명시적인 값으로 줄 수 있음.  그룹 개념의 도입과 현대에 사용되는 거의 모든 데이터 타입을 지원하여 보다 강력한 내용 모델을 지원할 수 있고 요소와 그것의  내용을 표현하기 위한 융통성 있는 내용 모델을 제공 |
| 확장성(Extensibility) | | | | 한 문서에 오직 하나의 DTD만이 적용  XML 네임스페이스(Namespace)를 이용하여 한 문서가 여러 DTD로부터 태그를 가져올 수 없다.  또한 상속(Inheritance) 기능을 지원하지 못한다. 확장 가능하도록 설계된 XML 언어가 확장 가능한 XML 문서 구조나 데이터  를 기술하는 언어를 가지고 있지 않다는 것은 XML에서 많은 제약이 된다. | | | DTD의 확장성 문제를 제거  XML 데이터를 기술하고 유효성을 검증할 때 SGML을 기반으로 하였기 때문에 가지고 있던 제약 요소들을 제거할 수 있음.  XML 스키마의 ‘사용자에 의해 파생된’ 데이터 타입, ‘복잡한’ 데이터 타입 또는 상속 같은 것들은 모두 실제 애플리케이션들의 요구에 맞추어 스키마를 확장할 수 있는 방법을 제공  XML 안에 네임스페이스를 기반으로 필요한 만큼 여러 문서구조들을 이용  (공유되고 표준화된 XML 어휘집들의 사용을 증가) |
| 동적인 스키마 | | | | 일단 작성되면 런타임 상에서 DTD를 변경할 수 없음 | | | 런타임에 선택될 수도 있고 추가, 삭제, 변경 등이 가능 |
| 장점 | | | |  | | |  |
| 단점 | | | |  | | |  |
| 위와 같은 DTD의 문제점들을 개선하기 위해 XML 스키마가 출현 | | | | | | | |
| HTML Block Element, Inline Element 차이점, 각각 3개 이상 Element 작성 | | | | | | | |
| Block Element | | | | | | Inline Element | |
| - 자신의 내용만을 한 라인에 출력  - 앞뒤의 내용이 있어도 절대로 같은 라인에 출력하지 않음  - 내용물의 크기와 상관없이 항상 너비 100%를  자기 영역이라고 판단 | | | | | | - 자신의 내용을 앞, 뒤의 요소의 내용과 같은 라인에 출력  - 내용물의 크기가 태그의 영역 | |
| P / h1~h6 / ul / ol / pre / dl / div / noscript / blockquote  Form / hr / table / fieldset / address | | | | | | Samp / kbd / var / cite / abbr / acronym / a / img / object  Br / script / map / q / sub / sup / span / bdo / input  Select / textarea / label / button | |
| HTML과 비교하여 XHTML 문법규칙 설명 | | | | | | | |
|  | | | | | | | |
| CSS Box Model 설명 | | | | | | | |
|  | | | | | | margin : border를 기준으로 박스의 여백을 지정  border : 박스의 테두리  padding : 테두리와 content간의 간격  content : 엘리먼트 안에 포함되는 컨텐츠 | |
| CSS fixed, relative, absolute, static, positioning 차이점 설명 | | | | | | | |
| static | | | | SS에서 position 속성을 선언하지 않았을 때의 기본 값입니다. | | | |
| relative | | | | 상대 위치라고도 하며, 본래 자신의 위치를 기준으로 좌표 속성을 통해 이동할 수 있습니다. | | | |
| absolute | | | | 절대 위치라고도 하며, 본래 자신의 위치 혹은, relative 속성이 선언된 객체를 기준으로 좌표 속성을 통해 이동할 수 있습니다. | | | |
| fixed | | | | 고정 위치라고도 하며, 웹 브라우저의 윈도우를 기준으로 좌표 속성을 통해 이동할 수 있으며, 스크롤이 생겨도 화면에서 사라지지 않는 붙박이 성질을 가지고 있습니다. | | | |
| JavaScript 전역변수와 지역변수의 선언위치, 사용 가능한 범위 설명 | | | | | | | |
| 전역변수 | | | | | | 지역변수 | |
| 함수 정의 외부에 선언된 변수로, 프로그램 전체에서 접근할 수 있는 변수 | | | | | | 함수 정의 내부에 선언된 변수로, 함수가 실행되면 만들어지고 함수가 종료되면 소멸하는 변수입니다. 함수 외부에서는 접근할 수 없습니다. | |
| JavaScript에서 false로 판별되는 것 서술 | | | | | | | |
|  | | | | | | | |
| JavaScript Function 객체가 가지고 있는 Property 역할 서술 | | | | | | | |
| Function 객체는 함수가 생성될 때 구문 해석(parse)되는 Function 생성자로 생성. 이는 그러한 함수가 나머지 코드와 함께 구문 해석되기 때문에, [function 식](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/function) 또는 [function 문](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/function)으로 함수를 선언하고 코드 내에서 그것을 호출하는 것보다 효율이 떨어집니다.  함수에 건네지는 모든 인수는 전달된 순서대로 생성되는 함수에 매개변수의 식별자 명으로 취급됩니다. | | | | | | | |
| Java 배열과 JavaScript 배열의 특징 설명 | | | | | | | |
| Java 배열 | | | | | | JavaScript 배열 | |
| 크기가 정해져 있음  배열 수정 시 기존의 배열을 버리고 새로 생성 | | | | | | 크기가 정해져 있지 않음  동적으로 생성 | |
| JavaScript call 메서드와 apply 메서드 사용법, 예시 코드 작성 | | | | | | | |
| **call()** | | | | | | **Apply()** | |
| 함수를 실행시키는 컨텍스트를 직접 지정,  This 변수에 연결될 객체를 선책 할 수 있음 | | | | | | | |
| call(컨텍스트 [, 인자1, 인자2....])  인자를 하나하나 나열하는 형식 | | | | | | apply(컨텍스트 [, 인자의 배열 ])  함수를 실행할 인자를 배열 형식으로 전달 | |
| XML형식, JSON형식 장, 단점 설명 | | | | | | | |
| XML형식 | | | | | | JSON형식 | |
| 구조화된 문서를 전송 가능하게 만든 텍스트 포멧 형식  모두 특정한 의미를 가진 데이터를 담는 포멧을 정의 | | | | | | | |
| 작성하기가 간편하다(tag구조) XML 사람이 읽기가 쉽다.  (즉 각 장보들이 의미하는 바를 한눈에 보기가 좋다. DTD 등 XML자체의 기능을 확장할 여지가 많이 있다. | | | | | | 내용이 함축적으로 최소한의 정보만을 가지고 있다. (XML대비 용량이 줄어들고 속도는 빨라짐) 객체구조와 {} 배열구조[]로 효율적인 데이터 구성  파싱이 매우 간편하고, 때로는 일반적인 변수처럼 사용도 가능하다. 즉, 사용하기 쉽다. | |
| 문서의 양이 필요이상으로 많다.  (실 데이터가 아닌 tag글자) 배열형식이나 반복구조의 경우 불필요한 데이터가  계속 해서 나타난다. 결국 이로 인해 파싱이 힘들어지고 속도는 느려진다. | | | | | | 내용이 함축적이다 보니 내용의 의미파악은 힘듦.  아무래도 적은 규격의 데이터 전송에 적합한 방식이기 때문에 XML보다는 빠르지만 대용량 급의 데이터 송수신엔 부적합 모습도 있다. | |
| DOM기술 효용성 설명 | | | | | | | |
| 문서 객체 모델(The Document Object Model, 이하 DOM)은 HTML, XML 문서의 프로그래밍 interface 이다.  문서의 구조화된 표현(structured representation)을 제공하며 프로그래밍 언어가 DOM 구조에 접근할 수 있는 방법을 제공하여 그들이 문서 구조, 스타일, 내용 등을 변경할 수 있게 돕는다.  구조화된 nodes와 property 와 method 를 갖고 있는 objects로 문서를 표현 | | | | | | | |
| XML 효용성 설명 | | | | | | | |
|  | | | | | | | |
| well Formed XML Document, valid XML Document 차이점 | | | | | | | |
| well Formed XML Document | | | | | | valid XML Document | |
| XML .0 스펙을 만족하는 XML 문서 | | | | | | DTD에 정의된 규칙을 따르는 Well-Formed XML Document | |
| 동기식, 비동기식 통신방식의 차이점 설명 | | | | | | | |
|  | | | 동기식 | | | | 비동기식 |
| 통신 속도 | | | 고속 | | | | 저속 |
| 회로 복잡도 | | | 복잡 | | | | 단순 |
| 구축 비용 | | | 고가 | | | | 저가 |
| 동기 제어 방식 | | | 클럭 동기 | | | | Start bit, Stop bit |
| 전송 단위 | | | 블록 단위 전송 | | | | 문자 단위 전송 |
| 적용 예 | | | 전화 교환망, ATM, 데이터 통신망 | | | | RS-232C |
| AJAX 기술 효용성 설명 | | | | | | | |
| 장점 | | | | | | 단점 | |
| 페이지 이동 없이 고속으로 화면 전환  (실시간 Interactive 성능 향상)  비동기 통신으로 인한 클라이언트에서 작업 처리 (클라이언트 분산처리)  수신 데이터의 양을 줄임 | | | | | | 크로스 브라우저 처리가 필요  AJAX를 쓸 수 없는 브라우저의 처리  HTTP 클라이언트로 기능이 한정  보안에 취할 수 있다.  코딩 양이 늘어나며, 디버깅이 어렵다. | |
| AJAX XML HttpRequest 객체의 역할 서술 | | | | | | | |
| 브라우저는 XMLHttpRequest 객체를 이용하여 Ajax 요청을 생성하고,  서버가 브라우저의 요청에 대해 응답을 전달하면 같은 XMLHttpRequest 객체가 그 결과를 처리한다. | | | | | | | |
| jQuery 메서드 체이닝 설명과 예시 코드 작성 | | | | | | | |
| 메서드 체이닝 : 메서드를 연속적으로 사용하는 방법  Object.function().function().function() | | | | | | | |
| **Framework** | | | | | | | | | |
| Framework 장점 서술 | | | | | | | | | |
| 장점 | | | | | | | 단점 | | |
| 1. 통일화된 코드규칙 적용에 따른 향후 유지보수성 향상 2. 개발 생산성 향상 (프레임워크는 Library가 제공되고 있고 여기에 MVC패턴이 적용된 반제품이다.) 3. 코드 재활용성 향상 | | | | | | | 프레임워크에 대한 학습시간 요구  한 프로젝트 내의 모든 개발자들에게 동일한 스킬(프레임워크에 대한 이해 및 활용도)이 요구 | | |
| ORM의 개념과 Framework 나열 | | | | | | | | | |
| 객체형 데이터(Java의 Object)와 관계형 데이터(관계형 데이터베이스의 테이블) 사이에서 개념적으로 일치하지  않는 부분을 해결하기 위하여 이 둘 사이의 데이터를 매핑(Mapping)하는 것.  객체형 데이터와 관계형 데이터의 각 속성들을 매핑할 경우 관계형 데이터를 객체형 데이터처럼 사용하는 것이 가능  종류 : Hibernate. mybatis (iBatis), JPA, Spring JPA … | | | | | | | | | |
| MVC 개념과 Framework 나열 | | | | | | | | | |
| **M**odel **V**iew **C**ontroller의 약자로 어플리케이션을 세 가지의 역할로 구분한 개발 방법론  사용자가 Controller를 조작하면 Controller는 Model을 통해서 데이터를 가져오고 그 정보를 바탕으로 시각적인  표현을 담당하는 View를 제어해서 사용자에게 전달하게 된다.  모델 1 구조  - 정의 : 모든 클라이언트 요청과 응답을 JSP가 담당하는 구조이다.  - 장점 : 단순한 페이지 작성으로 쉽게 구현 가능하다. 중소형 프로젝트에 적합하다.  - 단점 : 웹 애플리케이션이 복잡해지면 유지보수 문제가 발생된다.  모델 2 구조  - 정의 : 정의 : 클라이언트의 요청처리와 응답처리, 비즈니스 로직 처리하는 부분을 모듈화시킨 구조이다.  - 장점 : 처리작업의 분리로 인해 유지보수와 확장이 용이하다.  - 단점 : 구조 설계를 위한 시간이 많이 소요되므로 개발 기간이 증가한다.  종류 : | | | | | | | | | |
| DI(의존주입) 개념과 장점 설명 | | | | | | | | | |
| 객체 또는 구성 요소간의 종속관계를 소스코드로 설정하지 않고 외부의 설정파일을 통해서 주입하도록 함  객체간의 의존성을 자동으로 주입하는 기술  장점 : 코드가 단순해 진다., 컴포넌트간의 결합도가 제거된다. , 테스트가 유리하다. Mock/Stub 사용 가능 | | | | | | | | | |
| AOP(관점지향프로그래밍) 개념과 장점 설명 | | | | | | | | | |
| 공통 관심사를 유연하게 중복되지 않게 처리하기 위한 OOP의 보완적인 프로그래밍 구조를 지원하기 위해 탄생 | | | | | | | | | |
| 장점 | | 중복코드 제거, 코드의 간결성, 생산성 향상, 클래스 재사용성 증가, 유지보수성 향상 | | | | | | | |
| JoinPoint | | 공통 관심 기능 삽입되어 동작하는 특정 위치, 메서드 호출, 필드 값 변경 등 이에 해당 | | | | | | | |
| Advice | | 언제 공통 관심 기능을 핵심 로직에 적용 할지 정의.  메소드를 호출하기 전/후, 트랜잭션을 시작하기 전 | | | | | | | |
| PointCut | | JoinPoint 의 부분집합.. 실제 Advice 가 적용되는 JoinPoint를 의미 | | | | | | | |
| Weaving | | 포인트컷에 의해 결정된 조인포인트에 지정된 어드바이스를 삽입하는 것  즉, 핵심로직코드에 공통코드를 삽입되는 행위. | | | | | | | |
| Spring Auto-wiring, Auto-weaving 설명 | | | | | | | | | |
| **Spring Auto-wiring**  Spring container는 대규모 Spring기반 프로그램에서 XML설정파일을 줄일 수 있도록 <constructor-arg>와 <property> 요소의 사용없이 관련된 bean들의 관계를 자동으로 묶을 수 있다.(autowire - DI가 wire로 불리기도 하므로 autowire는 자동DI정도로 봐도 무방할 듯 싶다.) | | | | | | | | | |
| Autowiring Modes : DI의 autowiring을 사용하기 위해 Spring container에 지시(명령)하는데 사용 | | | | | | | | | |
| Mode | | | Description | | | | | | |
| no | | | autowiring이 아님을 의미하는 기본설정이고 wiring을 위한 명시적 bean참조를 사용한다.  이 wiring을 위해 특별하게 할 작업은 없다. | | | | | | |
| byName | | | 속성(property)명에 의한 autowiring. Spring container는 XML 설정파일에서 autowire 속성이 'byName'으로 설정된 bean의 속성을 찾는다. 그리고 설정 파일 내에서 속성이 동일 이름으로 bean이 정의된 것을 찾고 묶으려고 시도한다. | | | | | | |
| byType | | | Datatype속성에 의한 autowiring. Spring container는 XML 설정파일에서 autowire 속성이 'byType'으로 설정된 bean의 속성을 찾고, 그것의 type이 정확하게 하나로 일치한다면 속성을 묶는다. 만약 하나이상의 bean이 있다면, 'fatal exception'이 발생한다. | | | | | | |
| constructor | | | byType과 유사하지만 type이 생성자의 인자에 적용된다. 만약 container의 생성자의 인자 타입이 정확하게 일치하는 것이 없다면, 'fatal exception'이 발생한다. | | | | | | |
| autodetect | | | Spring은 우선 'constructor'로 autowiring을 시도한다. 만약 실패하면, Spring은 'byType'으로 autowire를 시도한다. | | | | | | |
| Limitations with autowiring : Autowiring은 프로젝트에 걸쳐 지속적으로 사용되어질 때 최고로 작동  autowiring이 특정 property 또는 생성인자의 필요를 감소시킬 수 있지만,  사용에 앞서 autowiring의 불이익과 제한을 고려해야 함 | | | | | | | | | |
| Limitations | | | | | | Description | | | |
| Overriding possibility | | | | | | Autowiring을 사용해도 여전히 <constructor-arg>와 <property>사정으로 의존관계를  명시할 수 있다. 이는 항상 autowiring을 덮어써버릴 것이다(override). | | | |
| Primitive data types | | | | | | 기본타입, String 그리고 class같은 간단한 속성을 autowire 할 수 없다. | | | |
| Confusing nature | | | | | | Autowiring은 명시적 wiring보다 명확하지 않다.  따라서 가능하면 명시적인 wiring을 사용하는 것이 좋다. | | | |
| **Spring Auto-weaving**  다른 어플리케이션 타입이나 어드바이즈된 객체를 생성하는 객체와 관점을 연결한다.  위빙은 컴파일시점(예를 들면 AspectJ 컴파일러를 사용해서), 로딩시점, 런타임시점에서 수행될 수 있다.  다른 순수한 자바 AOP 프레임워크와 같이 스프링 AOP도 런타임시에 위빙을 수행한다. | | | | | | | | | |
| Spring 사용 효용성 | | | | | | | | | |
| Spring 프레임워크가 테스트하기 쉬운 구조로 코드를 개발하는 것이 가능하도록 지원  인스턴스의 생성과 클래스 간의 의존관계에 대한 고민을 덜어주고 있다는 측면이다. | | | | | | | | | |