기술면접 스크립트

객체

-동일한 성질의 데이터와 메서드를 모아두고 필요할 때 언제든 이용할 수 있게 만들어 놓은 것.

-OOP관점에서 클래스 타입으로 선언되었을 때 객체라고 부름. 클래스의 인스턴스

객체지향

-객체들 간 메시지를 주고 받는 구조

OOP

-프로그래밍에서 필요한 데이터를 추상화시켜 상태와 행위를 가진 객체를 만들고, 그 객체들 간의 유기적인 상호작용을 통해 로직을 구성하는 프로그래밍 방법

****OOP 장점**

-코드 재사용 용이 : 타인이 만든 클래스를 가져다 사용할 수도 있고 상속을 통해 확장 사용가능.

-유지보수 쉬움 : 절차지향프로그래밍에서는 코드를 수정해야할 때 일일이 찾아서 수정. 반면 OOP에서는 수정해야할 부분이 클래스 내부에 멤버변수나 메서드로 있기 때문에 해당 부분만 수정하면 됨.

-대형프로젝트에 적합 : 클래스단위로 모듈화시켜서 개발할 수 있기 때문에 업무분담 시 용이

****OOP 단점**

-객체가 많으면 용량이 커질 수 있음

-설계 시 많은 노력과 시간이 필요

클래스

-객체를 만들어 내기 위한 설계도/틀

-연관된 변수와 메서드의 집합

인스턴스

-OOP관점에서 설계도(클래스)를 바탕으로 객체가 메모리에 할당되어 실제 사용될 때 인스턴스라고 부름

**클래스 vs 객체 : 클래스는 '설계도', 객체는 '설계도로 구현한 모든 대상'

**객체 vs 인스턴스 : 클래스의 타입으로 선언되었을 때 '객체', 그 객체가 메모리에 할당되어 실제

사용될 때 '인스턴스'

생성자

-객체 초기화 함수. 객체 생성시에만 호출되어 메모리 생성과 동시에 객체의 데이터를 초기화함

오버라이딩/오버로딩

추상화

-공통의 속성이나 기능을 묶는 작업

상속

-부모클래스의 속성과 기능을 그대로 이어받아 사용할 수 있게 하고, 그 기능의 일부분을 변경해야 할 경우 상속받은 자식클래스에서 해당 기능만 다시 정의하여 사용할 수 있게 하는 것.

다형성

-하나의 변수명, 함수명 등이 상황에 따라 다른 의미로 해석될 수 있는 것>>오버로딩/오버라이딩

추상클래스와 인터페이스의 차이

-둘 다 새로운 객체를 생성할 수 없음

-추상클래스 : 미완성된 클래스. 공통적인 기능을 하는 객체들의 추상화. 상속을 시켜서 기능을 확장시키는 목적. 다중상속 불가. extends

-인터페이스 : 공통적인 요소를 미리 정의함으로써 표준화 사용 가능. 다형성을 통해 구현한 객체가 같은 동작을 한다는 것을 보장하기 위한 목적

캡슐화

-목적 : 코드를 재수정 없이 재활용하기 위함

-기능과 특성의 모음을 클래스라는 캡슐에 분류해서 넣는 것

Getter, setter 사용 이유

-private로 접근이 제한된 멤버 변수에 접근할 수 있게 함. 메서드를 통해 접근하기 때문에 그 메서드 안에서 올바르지 않은 입력에 대해 사전 처리를 통해서 접근을 조절할 수 있어서 사용.

컬렉션

-List : 순서 있음. 중복 허용

-Vector : 스레드의 개수와 상관없이 동기화처리를 하기 때문에 시간이 많이 걸려 잘 사용x

-ArrayList : Vector와 같은 추가/삭제 기능을 가지고 있으며, 자동 동기화처리가 되지 않기 때문에 검색속도가 빠름. 대신 내부적으로 배열 구조를 이용(배열을 복사하는 방법으로 처리)하기 때문에 추가/제거가 많을 경우 오버헤드가 발생할 수 있음. 중간에 데이터 삽입 시 데이터들이 뒤로 밀리기 때문에 성능저하가 큼. 대신 인덱스를 가지고 어서 조회할 때 한번에 접근이 가능하기 때문에 대용량 데이터를 한번에 가져와서 여러 번 참조해 사용할 때 최상의 성능.

-LinkedList : C언어의 연결리스트처럼 다음 자료의 정보만 가지고 있고, 내부적으로 인덱스는 없는 컬렉션. 연결된 자료의 정보만 담고 있기 때문에 중간 노드에 추가/삭제 시 다른 데이터의 위치를 변경시킬 필요가 없어 간단하게 추가/삭제 가능. >> 추가/삭제 빈번한 대용량 데이터처리 시 유용. 대신 어디에 어떤 데이터가 있는지 첫 노드부터 정보를 타고 들어가서 찾아야하기 때문에 검색에 있어서 성능이 좋은 편은 아님. 인덱스가 없기 때문에 iterator(반복자)를 사용해야 함.

**랜덤 액세스 시 ArrayList가 유리. ArrayList는 인덱스가 있는 반면, Linked는 노드를 head나 tail부터 이동하며 접근하기 때문에 오래걸림.

**만약 추가/삭제도 많고 조회도 많을 때

ArrayList. Linked가 삽입/삭제에 있어서 데이터 상의 이점을 가지고 있긴 하지만 결국 어떤 값을 삭제할 때 존재하는지 확인을 해야하는데, 그 확인 작업을 위해 결국 head/tail부터 접근해야함

-Set : 순서x. 중복x

-set은 인덱스가 없기 때문에 iterator(반복자)를 사용해야 함.

-HashSet : 빠른 접근 속도. 단, 순서를 알 수 없음

-LinkedHashSet : 추가된 순서대로 접근 가능

-TreeSet : 정렬방법을 지정할 수 있음

-Map : 키,밸류로 구성된 데이터 집합

-HashMap : 중복x, 순서x, null 허용

-HashTable : 해시맵보다는 느리지만 동기화 지원, null 불가

-TreeMap : 정렬된 순서대로 저장되어 검색은 빠르지만, 요소 추가/삭제시 성능x

프로세스

- 운영체제로부터 작업을 할당 받는 작업의 단위. 실행중인 하나의 어플리케이션.

스레드

-프로세스가 할당받은 자원을 이용하는 실행의 단위.

****멀티프로세스로 처리하면 될 걸 다시 스레드로 쪼개서 처리하는 이유**

-프로세스는 호출할 때마다 문맥교환이라는 오버헤드가 발생하는데 스레드로 처리를 하면 프로세스끼리 통신하는 비용보다 통신 비용이 적고, 문맥교환이 적게 발생하기 때문에 보다 효율적인 작업이 가능하기 때문이다.

멀티스레드

-하나의 프로세스가 두가지 이상의 작업을 처리할 수 있도록 하는 것. CPU 사용률 향상. 자원 효율적 사용 등

****멀티스레드 장점**

- 두가지 이상의 작업을 동시에 실행할 수 있음. 단, dead lock 및 동기화에 대한 검증 철저해야.

****자바에서 멀티스레드 구현방법 :** Thread/Runnable

****자바 스레드**

-JVM이 운영체제의 역할을 하기 때문에 프로세스가 존재하지 않고 스레드만 존재. JVM에 의해 스케줄되는 실행 단위 코드 블록.

스택, 큐, 덱

-스택 : 자료의 입출력을 한 곳으로 제한한 자료구조. 문자열 역순 출력, 연산자 후위표기법 등

-큐 : 자료의 입출력을 한쪽 끝으로 제한한 자료구조. 컴퓨터 버퍼에서 주로 사용. 마구 입력되었으나 처리하지 못할 때 버퍼(큐)를 만들어 대기시킴. 배열로 구현되어 있기 때문에 큐의 크기가 제한되는 단점 > 링크드리스트 등장 : 링크드리스트로 구현한 큐는 큐의 크기제한x, 삽입/삭제 편리

-덱 : 자료의 입출력을 양쪽 끝에서 가능하게 하는 자료구조.

-스크롤 : 입력이 한쪽 끝으로만 가능하도록 제한한 덱

-셀프 : 출력이 한쪽 끝으로만 가능하도록 제한한 댁

접근제어자

동기화

-하나의 자원을 여러 태스크가 사용하려 할 때 한 시점에서 하나의 태스크만 사용할 수 있도록 하는 것

데드락

-두개의 스레드가 있을 때 서로에게 걸려있는 락이 풀릴 때까지 기다리게 되어 영원히 락이 풀리지 않는 것.

데드락 예방

-락 정렬, 락 타임아웃, 데드락 감지

가비지콜렉터

-동적 할당된 메모리 영역 가운데 더 이상 사용할 수 없게 된 영역을 탐지하여 자동으로 해제하는 기법.

해시(Hash)와 암호화(Encryption)의 차이점이 무엇인가요?

-Hash는 '단방향' 암호화기법, Encryption은 '양방향' 암호화 기법

-Hash는 평문을 암호화된 텍스트로 만들어주고, Encryption은 평문 암호화 + 암호화된 문장을 다시 평문화(복호화)

JVM의 동작방식

내부클래스

- 코드의 간략화 목적
즉, 내부 클래스의 코드가 매우 간단하거나 내부 클래스가 외부 클래스에 의해서 독점적으로 사용된다면 두 클래스는 내부 클래스 형태로 합치는 것이다.
그러나 내부 클래스의 코드가 복잡해지거나 다른 클래스에 의해서 사용될 상황이 생긴다면 내부 클래스를 다른 Java 소스 파일로 생성하는 것을 추천한다.
- 외부 클래스의 이름을 파일의 이름으로 저장하기 때문에,
하나의 자바 파일로 2 개의 클래스를 사용할 수 있기 때문에 코드 관리에 편리하다.
또한 하나의 외부 클래스 내부에 여러 개의 내부 클래스를 선언할 수도 있다.

싱글톤 패턴

-애플리케이션이 시작될 때 어떤 클래스가 최초 한번만 메모리를 할당하게 해서 하나의 인스턴스만 생성되게 하고, 그 인스턴스가 필요할 때마다 똑같은 인스턴스를 계속 만들어내는 게 아니라

이미 만들어진 인스턴스를 사용하게 함

****싱글톤 패턴을 쓰는 이유**

-고정된 메모리 영역을 얻으면서 한번의 new로 인스턴스를 사용하기 때문에 메모리낭비 방지

-싱글톤으로 만들어진 인스턴스는 전역인스턴스이기 때문에 다른 인스턴스들의 데이터 공유 용이

-DBCP처럼 공통된 객체를 여러 개 생성해서 사용해야 하는 상황에서 많이 사용

-처음 한번만 생성하기 때문에 두번째 호출때부터는 객체로딩시간이 현저하게 줄어 성능에 도움

****싱글톤 패턴의 문제점**

-싱글톤 인스턴스가 너무 많은 일을 하거나 많은 데이터를 공유시킬 경우, 다른 클래스의 인스턴스들 간에 결합도가 높아져 객체지향설계원칙에 어긋나게 됨 >> 수정/테스트 어려워짐

-멀티스레드환경에서 동기화처리 안하면 인스턴스가 두개가 생성된다는지 하는 경우 발생 가능

>>holder에 의한 초기화(클래스 안에 클래스를 두어 JVM의 클래스 로더 매커니즘과 클래스가 로드되는 시점을 이용한 방법. JVM의 클래스 초기화 과정에서 보장되는 원자적 특성을 이용해 싱글턴의 초기화 문제에 대한 책임을 JVM이 지게 됨.

Spring MVC 처리 순서

1. 클라이언트(Client)가 서버에 어떤 요청(Request)을 한다면 스프링에서 제공하는 DispatcherServlet 이라는 클래스(일종의 front controller)가 요청을 가로챈다. (web.xml 에 살펴보면 모든 url (/)에 서블릿 매핑을하여 모든 요청을 DispatcherServlet 이 가로채게 해둠(변경 가능))
2. 요청을 가로챈 DispatcherServlet 은 HandlerMapping(URL 분석등..)에게 어떤 컨트롤러에게 요청을 위임하면 좋을지 물어본다. (servlet-context.xml 에서 @Controller 로 등록한 것들을 스캔해서 찾아준다.)
3. 요청에 매핑된 컨트롤러가 있다면 @RequestMapping 을 통하여 요청을 처리할 메서드에 도달한다.
4. 컨트롤러에서는 해당 요청을 처리할 Service 를 주입(DI)받아 비즈니스로직을 Service 에게 위임한다.
5. Service 에서는 요청에 필요한 작업 대부분(코딩)을 담당하며 데이터베이스에 접근이 필요하면 DAO 를 주입받아 DB 처리는 DAO 에게 위임한다.
6. DAO 는 mybatis(또는 hibernate 등) 설정을 이용해서 SQL 쿼리를 날려 DB 의 정보를 받아 서비스에게 다시 돌려준다.
(이 때 보통 VO(dto)를 컨트롤러에서 부터 내려받아 쿼리의 결과를 VO 에 담는다. (mybatis 의 resultType)
7. 모든 로직을 끝낸 서비스가 결과를 컨트롤러에게 넘긴다.

8. 결과를 받은 컨트롤러는 Model 객체에 결과물 어떤 view(jsp)파일을 보여줄 것인지등의 정보를 담아 DispatcherServlet 에게 보낸다.
9. DispatcherServlet 은 ViewResolver 에게 받은 뷰의 대한 정보를 넘긴다.
10. ViewResolver 는 해당 JSP 를 찾아서(응답할 View 를 찾음) DispatcherServlet 에게 알려준다. (servlet-context.xml 에서 suffix, prefix 를 통해 /WEB-INF/views/index.jsp 이렇게 만들어주는 것도 ViewResolver)
11. DispatcherServlet 은 응답할 View 에게 Render 를 지시하고 View 는 응답 로직을 처리한다.
12. 결과적으로 DispatcherServlet 이 클라이언트에게 렌더링된 View 를 응답한다.

쿠키

클라이언트 로컬에 저장되는 키와 값이 들어있는 데이터파일. 만료시간 O. 파일로 저장되기 때문에 브라우저 종료해도 정보가 남아있을 수 있고, 만료기간을 길게 잡아두면 쿠키를 삭제할 때까지 유지될 수도 있음.

세션

일정시간동안 같은 브라우저로부터 들어오는 일련의 요구를 하나의 상태로 보고 그 상태를 유지하는 기술. 만료시간 정할 수 있지만 브라우저 종료하면 만료시간 상관없이 종료됨

캐시

이미지나 css, js 파일 등이 사용자의 브라우저에 저장되는 것.

var, let, const

-var, let 은 변수 선언 키워드이며 리터럴 값 재할당 가능, const 는 상수 선언 키워드이며 선언과 동시에 값 할당. 리터럴값 재할당 불가능

-var 가 가진 고질적이고 치명적인 문제점을 해결하기 위해 let 과 const 가 등장.

-var 는 변수명을 재선언해도 아무런 문제가 발생하지 않고

- 변수를 호출하는 코드가 변수를 선언하기 전에 사용돼도 자바스크립트 해석기가 호이스팅을 통해 var 키워드를 재해석하기 때문에 실행 시 오류가 나지 않는데 여기서 선언만 끌어올려지고 할당은 끌어올리지 않는다는 문제점이 발생. > 이를 위해 use strict 를 사용

-var 키워드 변수는 스코프에 가두려면 반드시 함수가 필요해서 function-scoped 라고 부름.

Let

-let, const 는 블록(중괄호) 내부에 let, const 키워드로 선언된 변수는 외부 스코프에 영향을 주지 않아서 block-scope 라고 부름

IOC

개발자는 JAVA 코딩시 New 연산자, 인터페이스 호출,팩토리 호출방식으로 객체를 생성하고 소멸시킨다. IOC 란 인스턴스의 생성부터 소멸까지의 객체 생명주기 관리를 개발자가하는 대신 스프링(컨테이너)가 관리한다

DI

IoC 를 실제로 구현하는 방법으로서 의존성있는 컴포넌트들 간의 관계를 개발자가 직접 코드로 명시하지 않고 컨테이너인 Spring 이 런타임에 찾아서 연결해주게 하는 것이다.

Spring Container

Spring Container 는 Bean 들의 생명주기를 관리한다. Spring Container 는 어플리케이션을 구성하는 Bean 들을 관리하기위해 IOC 를 사용한다. Spring 컨테이너 종류에는 BeanFactory 와 이를 상속한 ApplicationContext 가 존재한다. 이 두개의 컨테이너로 의존성 주입된 빈들을 제어하고 관리할 수 있다. 아래는 스프링 웹어플리케이션 동작원리이다.

웹어플리케이션 동작원리

1. 웹 애플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml 이 loading 된다.
2. web.xml 에 등록되어 있는 ContextLoaderListener(Java Class)가 생성된다. ContextLoaderListener 클래스는 ServletContextListener 인터페이스를 구현하고 있으며, ApplicationContext 를 생성하는 역할을 수행한다.
3. 생성된 ContextLoaderListener 는 root-context.xml 을 loading 한다.
4. root-context.xml 에 등록되어 있는 Spring Container 가 구동된다. 이 때 개발자가 작성한 비즈니스 로직에 대한 부분과 DAO, VO 객체들이 생성된다.
5. 클라이언트로부터 웹 애플리케이션이 요청이 온다.
6. DispatcherServlet(Servlet)이 생성된다. DispatcherServlet 은 FrontController 의 역할을 수행한다. 클라이언트로부터 요청 온 메시지를 분석하여 알맞은 PageController 에게 전달하고 응답을 받아 요청에 따른 응답을 어떻게 할 지 결정만한다. 실질적은 작업은 PageController 에서 이루어지기 때문이다. 이러한 클래스들을 HandlerMapping, ViewResolver 클래스라고 한다.
7. DispatcherServlet 은 servlet-context.xml 을 loading 한다.
8. 두번째 Spring Container 가 구동되며 응답에 맞는 PageController 들이 동작한다. 이 때 첫번째 Spring Container 가 구동되면서 생성된 DAO, VO, ServiceImpl 클래스들과 협업하여 알맞은 작업을 처리하게 된다.

Filter, Interceptor, AOP 차이

서버실행 > 서블릿 올라오는동안 init() 실행 > doFilter 실행 > 컨트롤러 들어가기 전 preHandler 실행 > AOP 실행 > 컨트롤러 탈출 > postHandler, after Completion, doFilter 실행 > destroy()실행

-Interceptor 와 Filter 는 서블릿 단위에서 실행, AOP 는 Proxy(대행자)패턴 이용해서 실행

-Filter 가장 밖, 그 안에 Interceptor, 그 안에 AOP.

-서블릿필터 : DispatcherServlet 이전에 실행. Web.xml 에 등록. 스프링과 무관하게 필터가 동작하도록 지정된 자원의 앞단에서 요청내용을 변경하거나 여러가지 체크를 수행할 수 있음. Init(), doFilter(), destroy()

-Interceptor : 스프링의 DispatcherServlet 이 컨트롤러를 호출하기 전,후로 끼어들어 스프링 컨텍스트 내에 존재. 스프링 내의 모든 객체에 접근 가능. preHandler() 컨트롤러 메서드 실행 전, postHandler() 컨트롤러 메서드 실행 직후 view 페이지 렌더링 전, afterComlpetion() view 페이지 렌더링 후

-AOP : 포인트컷으로 메소드 전후의 지점에 @after, @before, @around 로 자유롭게 위치 지정

MVC

-모델-뷰-컨트롤러는 소프트웨어 아키텍처 패턴.

-모델 : 데이터와 비즈니스 로직 관리.
-뷰 : 사용자에게 보여지는 화면 구성
-컨트롤러 : 사용자에게 받은 요청을 모델에게 전달 후 처리된 요청을 뷰에게 응답.
-사용자 인터페이스로부터 비즈니스 로직을 분리해 애플리케이션의 시작적 요소나 그 이면에서 실행되는 비즈니스 로직을 서로 영향 없이 고칠 수 있게 됨.

Index

-데이터베이스의 검색성능을 향상시키기 위한 것으로, 그 정보의 소재를 표시해 주고, 원하는 자료의 유무를 확인시켜 주며 자료의 신속한 이용을 가능하게 함.
-인덱스 동작원리

NDEX 를 해당 컬럼에 주게 되면 초기 TABLE 생성시 만들어진 MYD,MYI,FRM 3 개의 파일중에서 MYI 에 해당 컬럼을 색인화 하여 저장합니다. 물론 INDEX 를 사용안할시에는 MYI 파일은 비어 있습니다. 그래서 INDEX 를 해당컬럼에 만들게 되면 해당컬럼을 따로 인덱싱하여 MYI 파일에 입력합니다. 그래서 사용자가 SELECT 쿼리로 INDEX 가 사용하는 쿼리를 사용시 해당 TABLE 을 검색하는것이 아니라 빠른 TREE 로 정리해둔 MYI 파일의 내용을 검색합니다.
만약 INDEX 를 사용하지 않은 SEELCT 쿼리라면 해당 TABLE full scan 하여 모두 검색합니다. 이는 책의 뒷부분에 찾아보기와 같은 의미로 정리해둔 단어중에서 원하는 단어를 찾아서 페이지수를 보고 쉽게 찾을수 있는 개념과 같습니다. 만약 이 찾아보기 없다면 처음부터 끝까지 모든 페이지를 보고 찾아야 할것입니다.

트랜잭션

-데이터베이스의 상태를 변화시키기 위해 수행하는 작업의 단위.
-원자성: 모두 반영되든지, 전혀 반영 안되든지
-일관성: 작업처리결과의 일관성
-독립성: 둘 이상의 트랜잭션이 동시에 수행 중인 경우, 서로 간섭불가
-지속성: 트랜잭션이 성공적으로 완료됐을 시, 결과는 영구적으로 반영되어야 함

락

혹은 그 이상의 사용자가 같은 시간에 같은 데이터를 업데이트하는 것을 방지

Exclusive lock과 Shared lock의 차이

-Exclusive lock : 배타적(읽기) 잠금 >> 트랜잭션이 수행 중 데이터를 변경하고자 할 때 해당 트랜잭션이 완료할 때까지 해당 테이블/레코드를 다른 트랜잭션에서 읽거나 쓰지 못하게 함

-Shared lock : 공유(읽기) 잠금 >> 어떤 트랜잭션이 데이터를 읽고자 할 때 다른 트랜잭션이 동시에 읽을 수는 있게하되, 변경은 불가하게 함. 어떤 자원에 shared가 걸려있으면 exclusive 못걺.

트리거

특정 테이블에 INSERT, DELETE, UPDATE 같은 DML 문이 수행되었을 때, 데이터베이스에서 자동으로 동작하도록 작성된 프로그램

무결성

개체무결성 : 모든 테이블이 기본 키 (primary key)로 선택된 필드 (column)를 가져야 한다. 기본 키로 선택된 필드는 고유한 값을 가져야 하며, 빈 값은 허용하지 않는다.
참조무결성 : 외래키 값은 Null 이거나 참조 릴레이션의 기본키 값과 동일해야 한다는 규정
도메인무결성 : 테이블에 존재하는 필드의 무결성을 보장하기 위한 것으로 필드의 타입, NULL 값의 허용 등에 대한 사항을 정의하고, 올바른 데이터의 입력 되었는지를 확인하는 것. 특정 속성의 값이, 그 속성이 정의된 도메인에 속한 값이어야 한다는 규정
Null 무결성 : 릴레이션의 특정속성 값이 Null 이 될 수 없도록 하는 규정
키무결성 : 하나의 테이블에는 적어도 하나의 키가 존재해야 한다는 규정
고유 무결성 : 릴레이션의 특정 속성에 대해서 각 튜플이 갖는 값들이 서로 달라야 한다는 규정

서블릿

HTTP protocol 서비스를 지원하는 javax.servlet.http.HttpServlet 클래스를 상속하여 개발하며, Servlet 은 컨테이너에 의해서 실행되고 관리된다.

서블릿 컨테이너

HTTP 요청을 받아서 Servlet 을 실행하고 생명주기를 관리하는 역할을 한다. 서블릿 컨테이너는 요청이 들어올 때마다 새로운 자바 스레드를 만든다.
servlet 과 웹서버가 통신할 수 있는 방법을 제공
멀티 스레딩을 지원하여 클라이언트의 다중 요청을 알아서 처리

서블릿동작과정

1. 사용자가 URL 을 클릭하면 HTTP Request 를 Servlet Container 에 보낸다.
2. Servlet Container 는 HttpServletRequest, HttpServletResponse 두 객체를 생성한다.
3. 사용자가 요청한 URL 을 분석하여 어느 서블릿에 대한 요청인지 찾는다.
4. 컨테이너는 서블릿 service() 메소드를 호출하며, POST, GET 여부에 따라 doGet() 또는 doPost()가 호출된다.
5. doGet() or doPost() 메소드는 동적인 페이지를 생성한 후 HttpServletResponse 객체에 응답을 보낸다.
6. 응답이 완료되면 HttpServletRequest, HttpServletResponse 두 객체를 소멸시킨다.

ISO7 계층

A-물리적인 연결 상태와 기능 조정
P-노드 간의 프레임 전달.
S-패킷을 목적지로 전달. IP 프로토콜이 대표적
T-패킷들의 전송이 유효한지 확인하는 등 발신지 대 목적지 간의 제어 및 에러 관리. TCP
N-포트 연결. 통신 장치간의 상호작용 설정.
D-입출력되는 데이터를 하나의 표현 형태로 변환. 암호화도 여기서.
P-사용자 인터페이스 제공
소프트웨어 개발 단계

프로토콜의 종류

SMTP-전자메일 주고받을 때 다시 한번 정렬

POP3-인터넷서버가 메일을 수신, 저장. 메일 열람 시 서버에서 삭제됨
HTTP-인터넷 상에서 요청-응답 구조
TCP-IP 위에서 연결형 서비스를 지원하는 전송 계층 프로토콜.
UDP-일방향 정보전달 프로토콜
ICMP-IP 가 정보오류를 수정하거나 보고하지 못하는 단점을 보완하기 위한 프로토콜
ARP-IP 주소는 알지만 MAC 주소는 모를 때 서버로 IP 주소 요청하기 위한 프로토콜
RARP-반대.

웹 브라우저의 HTML 문서 렌더링 과정

1. 불러오기
   로더(Loader)가 서버로부터 전달 받는 리소스 스트림을 읽는 과정. 읽으면서 어떤 파일인지, 데이터인지 파일을 다운로드할 것인지 등을 결정한다.
2. 파싱 (Phasing)
   웹 엔진이 가지고 있는 HTML/XML 파서가 문서를 파싱해서 DOM Tree 를 만든다.
3. 렌더링 트리 만들기
   DOM Tree 는 내용을 저장하는 트리로 자바스크립트에서 접근하는 DOM 객체를 쓸 때 이용하는 것이고 별도로 그리기 위한 트리가 만들어져야 하는데 그것이 렌더링 트리다. (그릴 때 필요없는 head, title, body 태그등이 없음 + display:none 처럼 DOM 에는 있지만 화면에서는 걸러내야할 것들을 걸러냄)
4. CSS 결정
   CSS 는 선택자에 따라서 적용되는 태그가 다르기 때문에 모든 CSS 스타일을 분석해 태그에 스타일 규칙이 적용되게 결정한다.
5. 레이아웃
   렌더링 트리에서 위치나 크기를 가지고 있지 않기 때문에 객체들에게 위치와 크기를 정해주는 과정을 레이아웃이라고 함.
6. 그리기
   렌더링 트리를 탐색하면서 그리기.
   * 참고로 렌더링 엔진은 가능하면 HTML 문서가 파싱될 때까지 기다리지 않고, 배치와 그리기 과정을 시작한다.

CSS 는 위에 Script 는 아래에?
웹브라우저의 렌더링 순서를 이해하면 이해가 가는 말이다.
문서를 파싱해서 DOM Tree 를 만들어도 스타일 규칙이 없으면 렌더링 할 수가 없다.
즉, 최대한 빨리 스타일 규칙을 알아야 렌더링트리가 완전히 만들어지므로 스타일시트 파일을 모두 다운로드시키기 위해 <head></head>태그 사이에 놓는 것이다. (인터프리터에서 html 파일 위에서 아래로 읽음)
자바스크립트는 왜 아래에 놓아야 성능이 좋아질까?
자바스크립트는 DOM 객체를 이용해서 컴포넌트들을 조작하는데 <head></head>태그 사이 처럼 상단에 놓게 되면 HTML 파서가 파싱을 멈추고 스크립트파일을 읽기 때문이다.
파싱을 멈추고 읽기 때문에 위에서 스크립트파일이 많거나 파일이 크면 읽는 시간이 오래걸려 사용자 입장에서 웹페이지가 느리게 보이게 되므로 느리다고 느낄 수 있다.

심지어 잘 못 코딩했을 경우 HTML 파싱보다 자바스크립트 파일이 먼저 실행되서 적용이 안되는 모습도 볼 수 있다.
따라서 일반적으로 </body>태그 위에 스크립트를 모아둔다.
* 이렇게 하다보면 자바스크립트 애니메이션같은 것이 늦게 적용되서 처음 웹페이지를 들어갔을 때 아주 잠깐 다른 웹페이지가 보이다가 애니메이션이 적용되는 것을 볼 수 있는데, 이런 부분은 감안하는 방법이 있고, 애니메이션 관련된 부분만 <head></head>태그 사이로 올려서 적용하는 방법이 있다.

jQuery Document.ready(), Window.load() 실행 순서
$(document).ready() : 외부 리소스, 이미지와 상관없이 DOM Tree 생성 후 실행됨.
$(window).load() : 화면에 필요한 모든 요소들이 메모리에 모두 올려진 다음에 실행됨. (=화면이 모두 그려진 다음)
결론이 벌써 나왔다.
$(document).ready()가 DOM Tree 가 만들어진 후에 바로 실행되므로 $(window).load()보다 먼저 실행된다.

WAS 와 Web Server 의 차이
-웹서버(아파치) : 클라이언트가 서버에 페이지 요청을 하면 요청을 받아 정적 컨텐츠를 제공하는 서버
-WAS : 동적 컨텐츠를 제공하기 위해 만들어진 애플리케이션 서버(DB 조회 등 로직처리 요구컨텐츠) >> JSP, Servlet 구동 환경 제공. =컨테이너. 톰캣도 와스의 한 종류.

동작 프로세스
1. 웹서버로부터 요청이 오면 컨테이너가 받아서 처리
2. 컨테이너는 web.xml 을 참조하여 해당 서블릿에 대한 쓰레드 생성하고 httpServletRequest 와 httpServletResponse 객체를 생성하여 전달한다.
3. 컨테이너는 서블릿을 호출한다.
4. 호출된 서블릿의 작업을 담당하게 된 쓰레드(2 번에서 만든 쓰레드)는 doPost()또는 doGet()을 호출한다.
5. 호출된 doPost(), doGet() 메소드는 생성된 동적 페이지를 Response 객체에 담아 컨테이너에 전달한다.
6. 컨테이너는 전달받은 Response 객체를 HTTPResponse 형태로 바꿔 웹서버에 전달하고 생성되었던 쓰레드를 종료하고 httpServletRequest, httpServletResponse 객체를 소멸시킨다.