Springboot 동작원리

스프링부트 동작원리

내장 톰켓을 가진다.

톰켓을 따로 설치할 필요 없이 바로 실행이 가능하다.

Socket : 운영체제가 가지고 있는 것이다.

소켓이 새로 열릴때마다 쓰레드도 열려 동시작업을 가능하게 해준다.

서로 연결이 계속되어 있어 지속적으로 통신이 가능하지만, 부하가 걸린다.

http 통신은 stateless 방식으로 작동한다.

http는 문서를 전달하는 통신이다.

http는 소켓에 비해 부하는 적지만 작업이 끝나면 연결이 끊겨 다시 연결을 할때는 새로운 연결을 해야 한다.

위의 단점을 보안한게 웹서버이다.

http의 목적은 html(확장자)로 생성된 문서를 요청자에게 전달해주는 것이다.

http : 소켓을 기반으로 함

톰켓 | 웹서버의 차이를 알아야 한다.

웹서버 : 갑/을 구조이다. 필요한 사람이 을이고 웹서버가 갑이다.

웹서버 통신은 항상 을이 갑에게 request(요청)를 한다.

request할때는 위치를 알아야하기 때문에 IP주소가 필요하다.

request할때는 어떤 것이 필요한지 알 수 있게 URL을 통해 전달한다.

URL은 자원을 요청하는 주소이다.

또한 요청을 받고 response(응답)할때는 위의 정보를 토대로 보내기때문에 위의 정보들을 다시 찾지 않아도 응답할 수 있다.

즉 http는 응답자는 요청자의 정보를 알 필요가 없다.

대신 요청자가 요청을 하지 않았으면 , 응답자는 요청자랑 연결이 되지 않기 때문에 불가능하다.

을의 주소를 알기 위해서는 소켓을 사용해야 한다.

소켓은 을이 연결을 하는 순간 연결이 계속 지속되기 때문에 한번 연결하면 갑이 원하는 시점에 데이터 전달이 가능하다.

http통신은 단순히 요청시에 응답하는 구조이고, 응답은 문서나 자원을 요청자에게 응답을 해준다.

이 자원들은 전부 static 자원이라고 한다.

즉 정적인 데이터를 응답해주는 것을 웹서버라 한다.

웹서버는 자바코드등 인식못하는것이 있다.

대신 여기다 톰켓을 달면 톰켓이 컴파일을 해줘서 html파일로 만들어준다.

즉 웹서버가 다시 인식할 수 있게 해주는 것이다.

즉 정상적으로 웹브라우저가 파일을 읽을 수 있게 도와주는 것이다.

서블릿 컨테이너

만약 정적 자원들을 요청을 하면 톰켓대신 아파치(웹서버)가 일을 한다.

만약 자바 파일을 요청하게 되면 톰켓이 일하게 된다.

스프링에서는 정적자원들은 요청이 불가하다.

스프링은 URL접근 방식을 막아놔서 URI(식별자 접근) 방식을 사용한다 .

즉 특정한 파일을 요청할 수 없고, 요청시에는 무조건 자바를 거쳐야 한다.

자바 파일을 받으면 아파치는 톰켓에게 제어권을 넘기게 된다.

서블릿 : 자바 코드로 웹을 다루는 것

클라이언트가 자바와 관련된 자원을 요청하게 되면 서블릿 컨테이너(톰캣)를 거치게 된다.

요청이 들어오면 톰켓은 스레드를 생성하게 된다 .

스레드가 서블릿 객체를 생성하게 된다.

만약 요청이 스레드 설정 개수보다 많아지면 대기하게 된다.

스레드가 하던 일이 끝나고 response를 하면 스레드는 할 일이 끝나게 된다.

스레드가 할 일이 끝나면 사라지는 것이 아니고 대기 중인 요청을 다시 처리하게 된다.

즉 , 재사용하여 불필요한 연산을 없애 속도를 증가 시킨다.

결국 톰캣은 HttpServeletRequest, HttpServletResponse 객체를 가지고 있는다.

web.xml가 하는 일

ServletContext의 초기 파라미터 설정

Session의 유효시간 설정

Servlet/JSP에 대한 정의 및 매핑을 한다.

Mime Type 매핑을 한다.

Welcome File List를 설정한다.

Error Pages를 처리한다.

리스너와 필터를 설정한다.

보안을 설정한다.

초기 파라미터는 한번 설정을 해놓으면 어디서든지 동작이 가능하다.

세션은 인증의 개념이랑 비슷하다. 인증의 유효기간은 web.xml이 설정이 가능하다.

web.xml의 정의된 정보를 토대로 알맞은 곳으로 매핑시킨다.

Mime Type은 정보가 들고온 데이터의 타입이다.

정보를 들고오지 않은 것들은 http의 get방식으로 전달된다.

만약 정보를 들고오면 검사 및 가공을 하여 정보를 사용한다.

즉 Mime Type을 틀리게 되면 에러가 발생하게 된다.

Welcome File List는 정보를 들고오지 않은 것들이 처음으로 가는 곳이다. 이곳은 관리자가 직접 설정이 가능하다.

Error Pages는 정보가 잘못들어왔거나 등 에러가 발생하였을 때 보내는 곳이다.

필터는 잘못된 정보가 들어왔을 때 걸러내는 방식이다.

리스너는 필요한 정보만 얻어오는 대리인의 느낌이다.

즉 웹서버에 진입을 하게 되면 최초로 도는게 web.xml 이다.

FrontController 패턴

최초 앞단에서 request 요청을 받아서 필요한 클래스에 넘겨준다.

왜냐하면 web.xml에서 다 정의하기 힘들기 때문이다.

내부에서 request가 발생하였을 때 기존 메모리에 존재하던 request가 없어질 수 있기 때문에 이를 방지하기 위한 requestDispatcher 기법을 사용한다.

즉, 새로운 요청이 생기기 때문에 request와 response가 새롭게 new 될 수 있다. 그래서 RequsetDispatcher가 필요하다.

RequestDispatcher : 필요한 클래스 요청이 도달했을 때 FrontController에 도착한 requset와 response를 그대로 유지시켜준다.

즉 데이터를 들고 다른 페이지로 이동할 수 있다.

위의 두 기술을 가지고 있는게 Spring의 DispatchServlet이다.

FrontController 패턴을 직접 짜거나 RequestDispatcher를 직접 구현할 필요가 없다.

DispatchServelt은 FrontController 패턴 + RequestDispatcher이다.

DispatchServelt이 자동생성되어 질 때 수 많은 객체가 생성(IoC)된다. 보통 필터들이다. 해당필터들은 내가 직접 등록 할 수도 있고, 기본적으로 필요한 필터들은 자동 등록 되어진다.

스프링 컨테이너

DispatchServlet에 의해 생성되어지는 수 많은 객체들은 어디에서 관리될까?

첫째, ApplicationContext

수 많은 객체들이 ApplicationContext에 등록된다. 이것을 IoC라고 한다. IoC란 제어의 역전을 의미한다. 개발자가 직접 new를 통해 객체를 생성하게 된다면 해당 객체를 가르키는 레퍼런스 변수를 관리하기 어렵다. 그래서 스프링이 직접 해당 객체를 관리한다. 이때 우리는 주소를 몰라도 된다. 왜냐하면 필요할 때 DI하면 되기 때문이다.

DI를 의존성 주입이라고 한다. 필요한 곳에서 ApplicationContext에 접근하여 필요한 객체를 가져올 수 있다. ApplicationContext는 싱글톤으로 관리되기 때문에 어디에서 접근하든 동일한 객체라는 것을 보장해 준다.

ApplicationContext의 종류에는 두 가지 종류가 존재한다.(root-applicationContext와 servlet-applicationContext) 이다.

servlet-applicationContext는 ViewResolver, Interceptor, MultipartResolver 객체를 생성하고 웹과 관련된 어노테이션 Controller, RestController를 스캔한다.

—>해당 파일은 DispatchServelt에 의해 실행된다.

root-applicationContext는 해당 어노테이션을 제외한 어노테이션 Service, Repository등을 스캔하고 DB관련 객체를 생성한다. (스캔 : 메모리에 로딩)

—> 해당 파일은 ContextLoaderListener에 의해 실행된다. ContextLoaderListener를 실행해주는 것은 web.xml이기 때문에 root-applicationContext는 servlet-applicationContext보다 먼저 로드된다.

당연히 servlet-applicationContext에서는 root-applicationContext가 로드한 객체를 참조할 수 있지만 그 반대는 불 가능하다. 생성 시점이 다르기 때문이다.

둘째, Bean Factory

필요한 객체를 Bean Factory에 등록할 수도 있다. 여기에 등록하면 초기에 메모리에 로드되지 않고 필요할 때 getBean()이라는 메서드를 호출하여 메모리에 로드할 수 있다. 이것 또한 IoC이다. 그리고 필요할 때 DI하여 사용할 수 있다. ApplicationContext와 다른 점은 Bean Factory에 로드되는 객체들은 미리 로드되지 않고 필요할 때 호출하여 로드하기 때문에 lazy-loading이 된다는 점이다.

요청 주소에 따른 적절한 컨트롤러 요청 (Handler Mapping)

Get 요청시 ⇒ http://~~~/~~/~

해당 주소 요청이 오면 적절한 컨트롤러의 함수를 찾아서 실행한다.

응답

html을 응답할 지 Data를 응답할지 결정해야 하는데 html파일을 응답하게 되면 ViewResolver가 관여하게 된다.

하지만 Data를 응답하게 되면 MessageConverter가 작동하게 되는데 메시지를 컨버팅할 때 기본전략은 json이다.

톰켓시작 → 필터 메모리 적재 → 디스패처 메모리 적재(/admin /user) →Controller, Service, JPA Repository, 영속성 컨텍스트 메모리 적재 → Datasource → View Resolver , 인터셉터

디스패처는 중간에 어떤 주소가 들어오는지 확인하고 그 주소에 맞는 컨트롤러를 요청해준다.

Controller, Service, JPA Repository, 영속성 컨텍스트 메모리 적재 : 요청시마다 메모리에 적재된다.

즉 사용자가 요청할때마다 쓰레드가 새로 생기면서 동시작업을 한다. 즉 하나로 유지되지 않고 지속적으로 뜬다.

시작할때 뜨는게 아니고, 요청 전까지 대기한다. 하지만 Datasource는 뜬다.

DataSource : DB 랑 직접적인 연결이 되어있다.

만약 사용자가 로그인 요청을 한다고 가정

POST 방식으로 Body에 데이터가 담겨 들어온다.

후에 필터를 거쳐 디스패처로 간다.

디스패처는 알맞는 컨트롤러를 뜰수 있게 한다.

login 으로 요청하였으니 login PostMapping을 가지고있는 컨트롤러에 요청을 한다.

컨트롤러는 알맞는 메서드를 메모리에 띄어준다.

그 컨트롤러는 Body에 있는 정보 username, password를 받는다.

컨트롤러는 정보를 받기만 하면 역할이 끝난다.

컨트롤러는 어떤 주소 요청이 왔을때 그 주소 요청에대한 함수를 하나 만들어서 그 함수에 body 데이터를 받는 역할이다.

이제 역할 이 종료됬으니 받은 Body 데이터를 Service에게 넘긴다.

Service에게 login 요청을 한다.

Service에서는 login 시작이 된다.

login을 하기 위해선 DB에 SELECT 질의를 해야한다.

(이 username과 password가 있는 사용자 정보를 SELECT 요청한다)

이 SELECT 요청을 JPA Repository에게 위임한다.

JPA Repository는 자기가 들고 있는 함수를 호출한다.

(select * from user where username=? and password=?) 의 쿼리문이 생긴다.

이제 영속성 컨텍스트에 알맞는 값이 있는지 확인한다.

username과 password로 user테이블을 확인했으니 영속성컨텍스트에 user 오브젝트가 존재하는지 확인을 한다. 하지만 최초실행이면 영속성 컨텍스트는 비어있다.

영속성 컨텍스트가 비어있으니 Datasource에게 DB에게 물어봐달라고 요청을 한다.

Datasource는 메모리에 한개만 적재되어 있다.

이제 Datasource가 DB에게 질의를 한다. 이제전달된 쿼리에 맞게 DB에게 요청을 해 알맞은 user 오브젝트를 Datasource에게 반환을해주고, 다시 Datasource는 영속성 컨텍스트

에 user 오브젝트를 전달한다. (만약 존재하지 않으면 null 반환)

영속성 컨텍스트에는 이제 User 오브젝트가 새로 생긴다.(유지)

이제 user오브젝트가 생겼으니 다시 JPA Repository에게 user오브젝트를 전달해준다.

Repository는 다시 service에게 user오브젝트를 돌려준다.

service에서는 user오브젝트가 알맞은 값인지 null인지 확인을 한다.

Service에서 널이 아니면 세션에 user오브젝트를 등록하는 것을 짜줘야 한다.

Service에서 확인을해 user가 맞으면 Controller에게 user 오브젝트를 전달하고 , 세션에 User 정보를 등록한다.

Service는 세션에 user 정보를 등록했다는 것과 user 오브젝트를 Controller에 넘기면

Controller는 이제 인증을 받은 사용자이므로 다음 후속 조치를 한다.(메인페이지이동 등)

이때 Controller의 종류가 중요한데

만약 RestController이면 데이터를 응답하는 컨트롤러이다.

1)

일반적인 Controller이면 html 페이지를 만들어서 사용자에게 응답하는 컨트롤러이다.

만약 메인페이지 이동이라면 html 페이지 이동이므로 ViewResolver가 작동하게 된다.

ViewResolver는 페이지를 만들어서 응답을 한다.

ViewResolver가 작동하는 시기는 Controller가 일반 Controller이면 항상 작동을 한다.

Controller에서 마지막에 return 되는 값은 "home"; 이 된다라고 가정하면

ViewResolver는 프로젝트에서 home이라는 페이지를 찾는다. home은 JSP 파일이다.

후에 JSP 페이지를 html로 만들어서 사용자에게 응답을 해준다.

2)

만약 RestController이면 ViewResolver가 작동되지 않는다.

이제 return값이 "home"이면 home 파일을 찾는게 아니라 "home" 이 메세지 자체를 요청자에게 응답해준다.

/user/1 이 1번 유저의 개인정보를 보는 페이지라고 가정할때

함수가 시작되면 1번 정보의 데이터를 만들어서 사용자에게 응답해준다.

이 함수가 실행 직전에 인터셉터가 동작해서 지금 요청한 사용자가 세션이 있는지 확인을 한다.

만약 세션에 user 오브젝트가 존재하면 그 오브젝트가 1번 유저인지 확인을 한다

같은 유저이면 정보를 보여주고 다른 유저이면 정보를 보여주지 않는다.

인터셉터 : 함수 실행전에 권한을 체크한다.
인터셉터는 필터랑 다르다.

필터는 애초에 요청이 들어올때 걸러내는 역할을 한다

인터셉터는 그 함수(개인정보를 주는 함수) 실행되기 직전에 낚아채서 권한이 있는지 확인을 한다.

이제 권한이 있으면 데이터를 응답해준다.

인터셉터는 실행전 이나 실행뒤에 낚아챈다.

정리

톰켓이 시작되면 필터와 디스패처가 메모리에 뜬다.

DataSource와 인터셉터, 세션도 메모리에 뜬다.

이제 뜬 상태에서 사용자가 Request 요청을 하면 알맞은 Controller 메서드가 실행이된다.

이제 Controller는 Body 데이터를 받아서 Service에게 넘긴다. Service도 요청에 맞는 알맞은 서비스를 시작한다.

만약 DB에게 확인을 해야하는 요청이면 JPA Repository에게 전달한다.

이제 JPA Repository는 영속성 컨텍스트에 요청하는 데이터가 존재하는지 확인한다 .

만약 들고있으면 그 오브젝트를 바로 응답하면 된다.

없으면 영속성 컨텍스트에서 DataSource에게 전달을해 알맞는 데이터를 DB에서 뽑아온다.

(만약 영속성 컨텍스트에 값이 존재하면 그 데이터만 변경하고 바로 return 해주고,

영속성 컨텍스트는 DB에 데이터를 flush해서 갱신해준다.)

이제 데이터를 return 받으면 역순으로 다시 return해준다 .

이제 서비스에서 null체크를 한 후 null이면 거절하고

사용자가 맞으면 세션에 오브젝트를 등록한다.

이제 회원가입 상황이라고 가정을 하면

insert 요청이 온다.

톰켓시작, 필터, 디스패처 시작

디스패처에서 insert 요청을 받아서 Controller에게 넘겨

Controller에서 회원가입 Controller 메서드를 찾는다 . Controller에서 JDBC가 연결이 되고,

(JDBC : 데이터베이스에 연결이 됨)

Body데이터에 데이터가 담겨서 오는데 이것을 Service에게 넘긴다 .

Service에서는 트랜잭션이 시작된다. 이제 Service에서 JPA Repository에게 insert 요청을 한다.

최초 insert이면 영속성 컨텍스트에 아무것도 존재하지 않기 때문에

바로 DB에게 질의해 insert해달라고 요청을 한다.

이제 DB에 알맞는 유저정보가 들어간다.

이제 정상적으로 insert가 되면 다시 역순으로 응답을 해준다 .

만약 정상적으로 Service의 회원가입이 끝나면 다시 Controller에게 돌아간다.

이제 Service가 끝나는 순간에 트랜잭션이 종료가 된다.

트랜잭션 종료전에는 DB에 insert한 데이터는 실제로 DB에 들어가는 것이 아니고 메모리에 남아있다. 이제 트랜잭션이 종료되면 Commit 요청이되서 실제로 DB에 데이터가 들어가게 된다.

이때문에 Service에서 트랜잭션 관리가 가능하다. (스프링 부트가 기본적으로 가지고 있는 규칙)

이제 Controller에서 후속 조치를 하게 된다 .

만약 다른 페이지로 이동을 시켜야 한다면 ViewResolver가 작동해 알맞은 홈페이지를 응답해준다.

만약 일반적인 메세지만 전달하게 된다면 RestController가 데이터만 응답을 해준다.

만약 회원가입 후 바로 로그인되게 하고 싶으면 세션에 user 정보를 등록하면 된다.

만약 서비스에서 데이터 요청이 여러개라고 가정하면

여기서 데이터 요청이 한개라도 실패하면 롤백(Rollback)을 해야한다.

즉 이런것을 서비스에서 관리하므로

서비스에서 트랜잭션이 시작되는것이다.

즉 송금과 같이 하나의 기능인데 여러번의 데이터 요청이 필요로 하는 것들이 많이 있다.

그래서 서비스는 여러번의 데이터 요청을 하나의 패키지로 담고있는 역할을 한다.