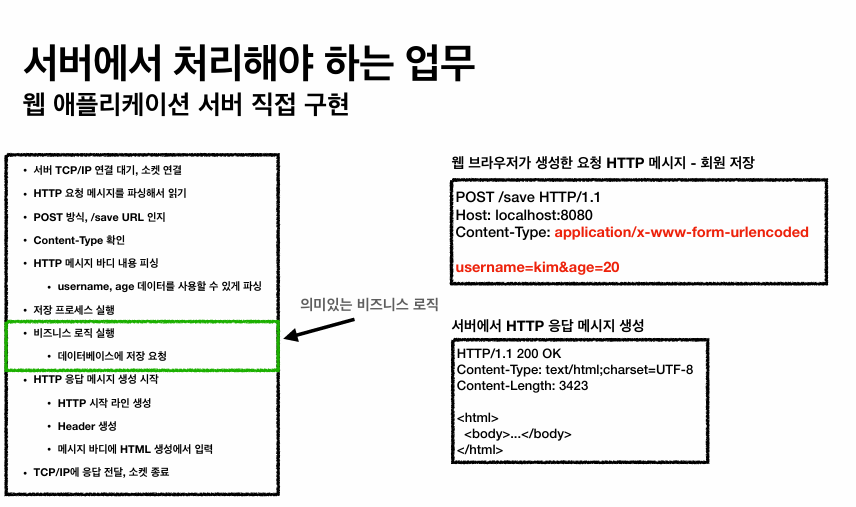


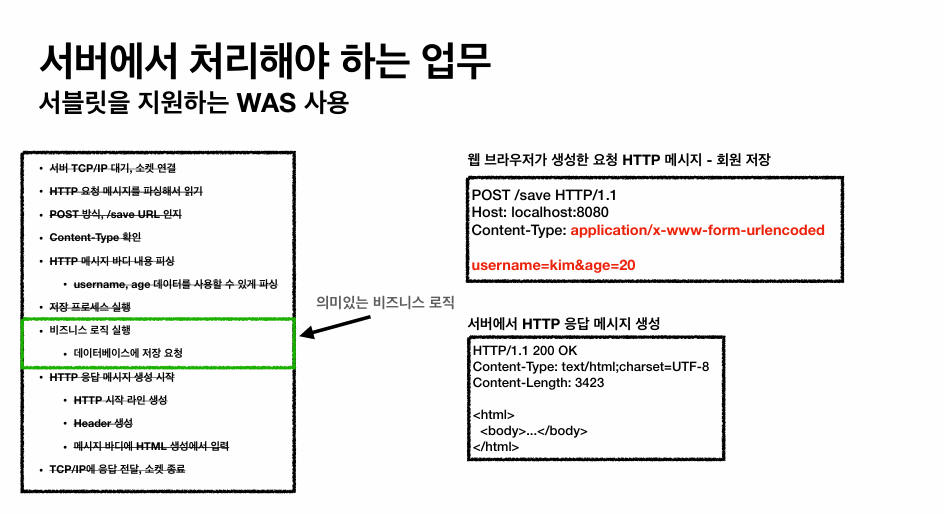
form을 작성한 뒤 전송을 누르면 브라우저가 HTTP메시지를 만듦.

Application/x-www-form-unlencoded : form방식으로 보낼 때의 content-type

name이 key로 입력값이 value로 되어 http 바디가 만들어짐.

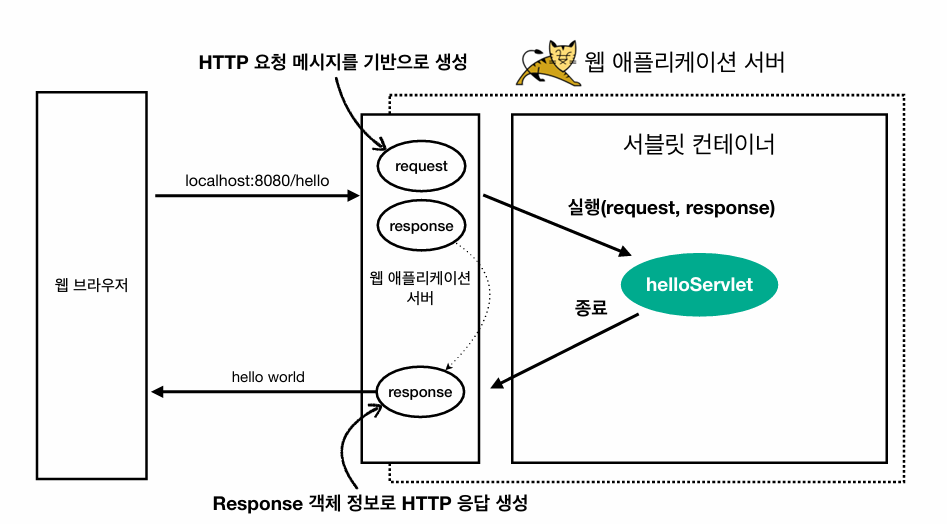


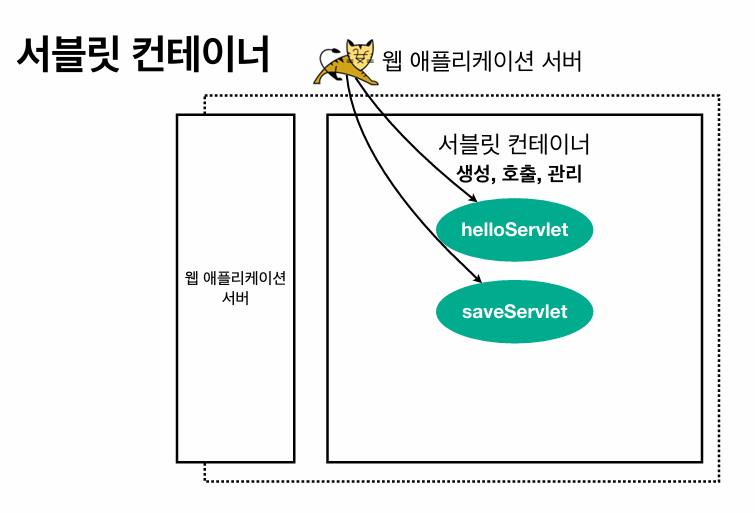
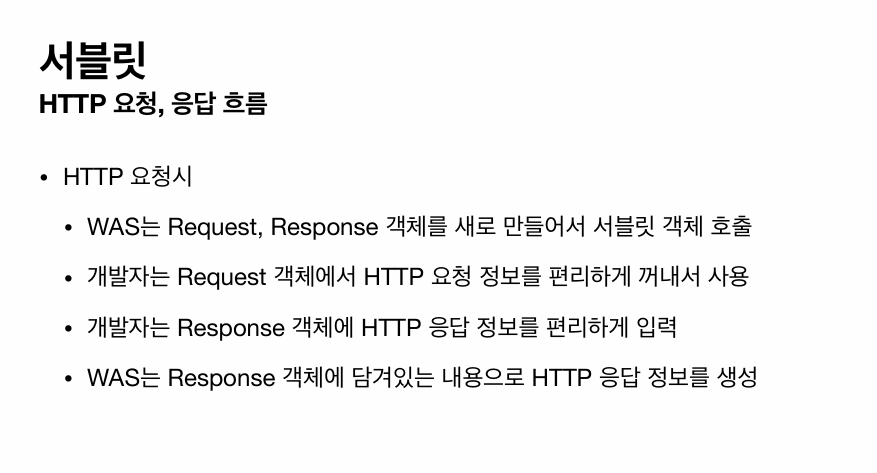
웹 어플리케이션을 직접 구현해야한다 하면 http 통신을 위한 tcp/ip 소켓 연결부터 http 요청 메시지 파싱, 응답 메시지 생성 소켓 종료 등 핵심 비즈니스 로직과 관련 없는 내용들도 직접 구현해야한다.



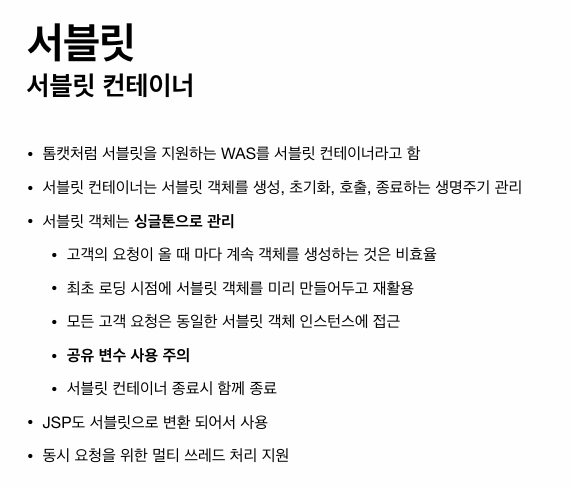
그렇기 때문에 서블릿이 등장했고, 서블릿이 TCP/IP 연결부터 요청 HTTP 파싱 및 HTTP 응답 메시지 만들기 등의 HTTP통신에 필요한 일련의 서브 작업들을 지원해줌.







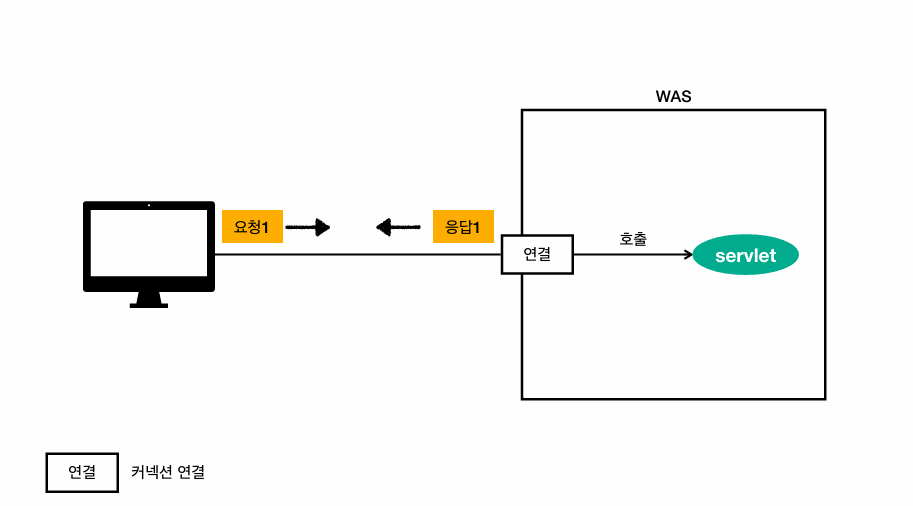
서블릿 객체는 개발자가 직접 만드는 것이 아니다. HttpServlet 클래스를 상속한 클래스의 service 메소드만 작성하면 WAS가 서블릿 객체를 만들어서 서블릿 컨테이너를 통해 관리한다. WAS가 서블릿 컨테이너를 통해 서블릿 객체를 만들고, 호출하고 관리(생명주기 관리)해준다.



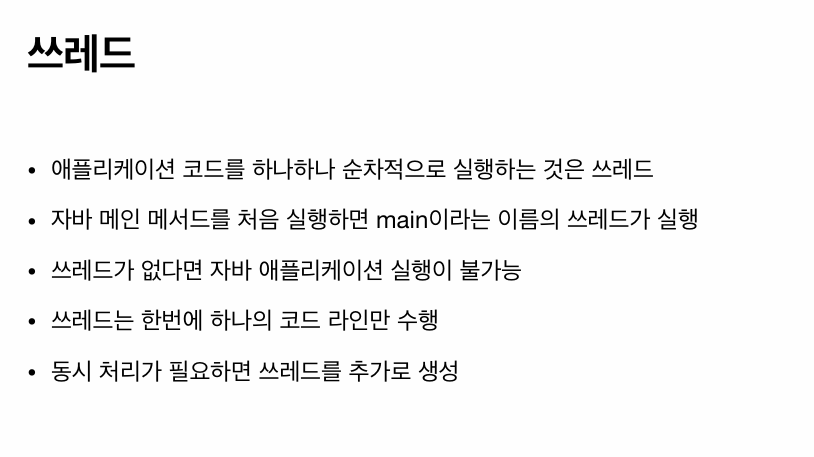
Request 나 response 객체는 요청 데이터가 다르기 때문에 요청이 들어올 때마다 새로 만들어지는 것이 맞음. 하지만 서블릿 객체는 요청이 올 때마다 다를 필요가 없기 때문에 싱글톤으로 관리하면 된다.

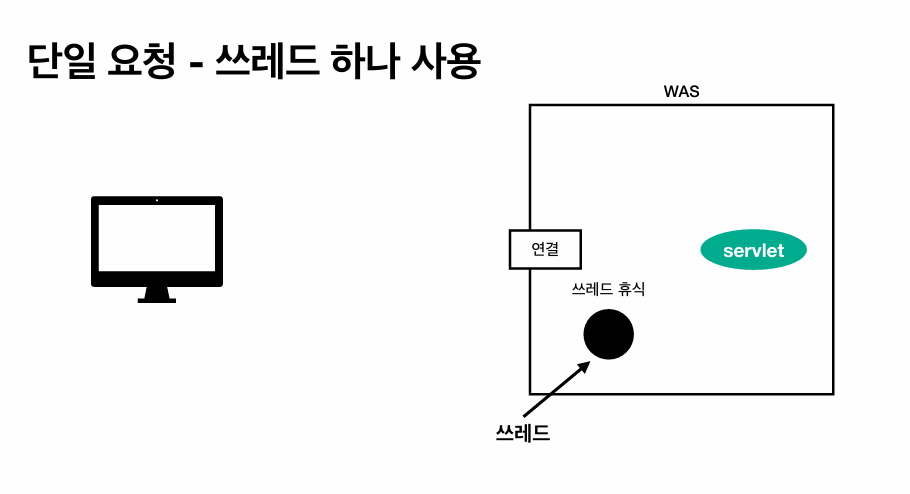
WAS의 큰 특징 : 동시 요청을 위한 멀티 쓰레드 처리 지원을 해줌. 개발자가 멀티쓰레드 처리를 신경쓰지 않아도 WAS 가 해결을 해줌.

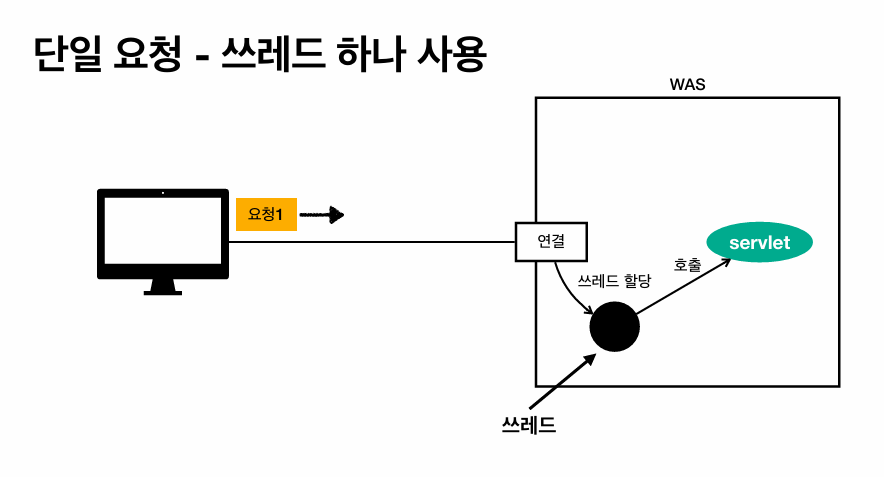
**동시 요청 - 멀티 쓰레드**

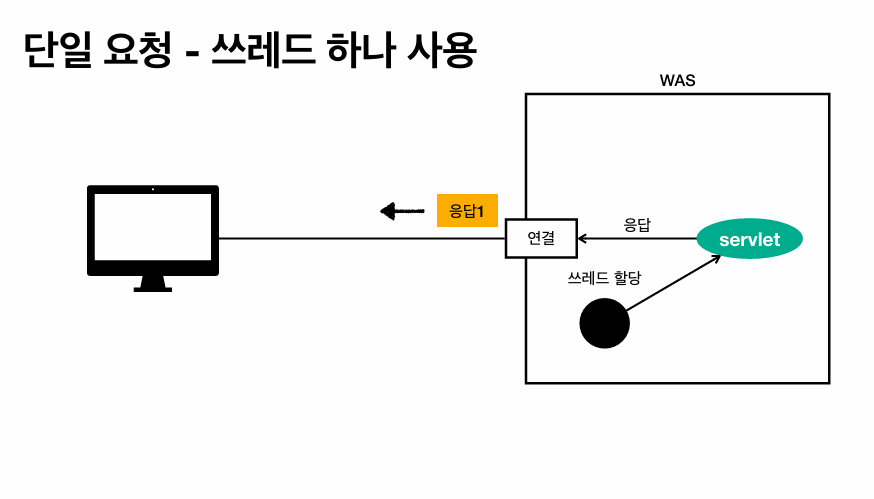


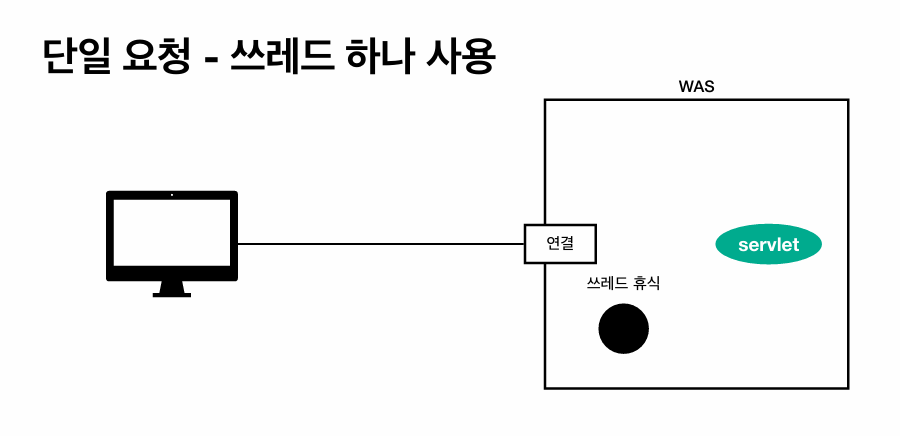
서블릿 호출은 누가 해주는 것이지? : 쓰레드가 해줌.



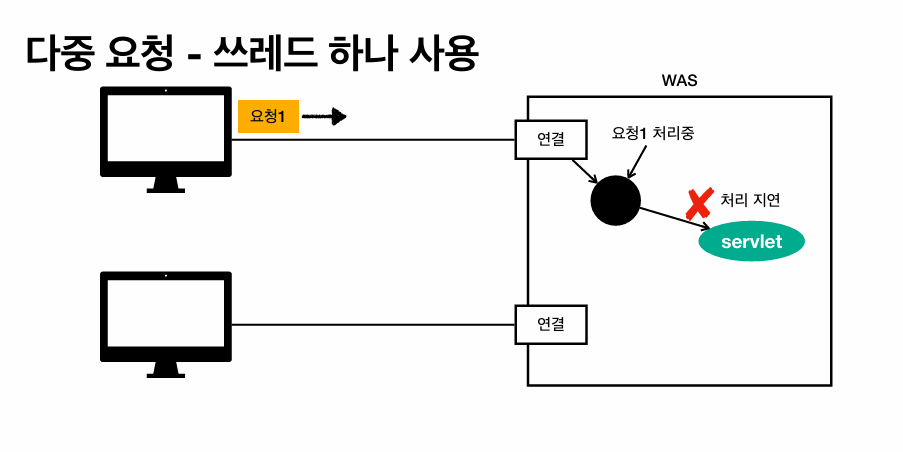


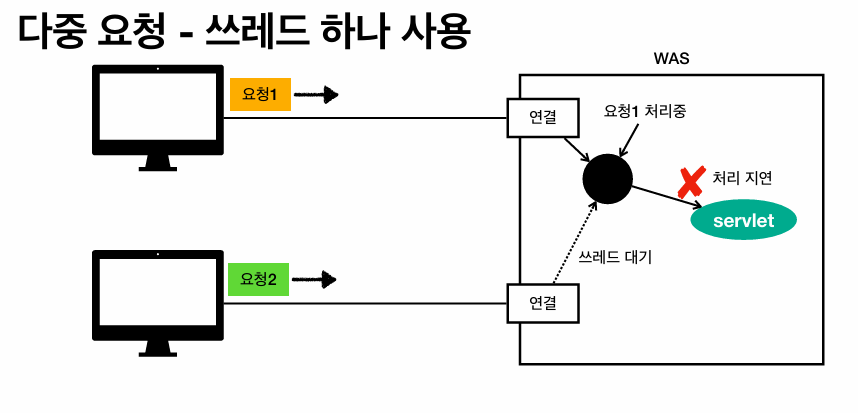


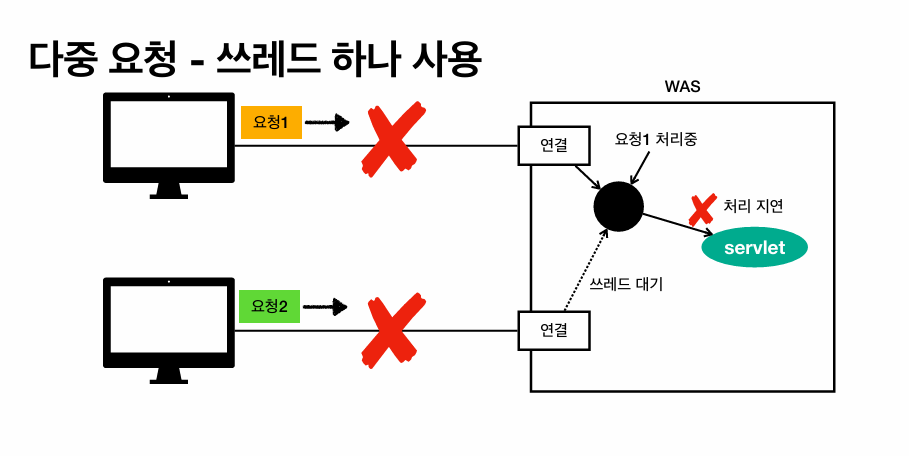




쓰레드를 만들고, 요청이 들어오면 쓰레드를 통해 요청을 분석 후 응답까지 처리한 후 다시 대기 상태에 돌입.



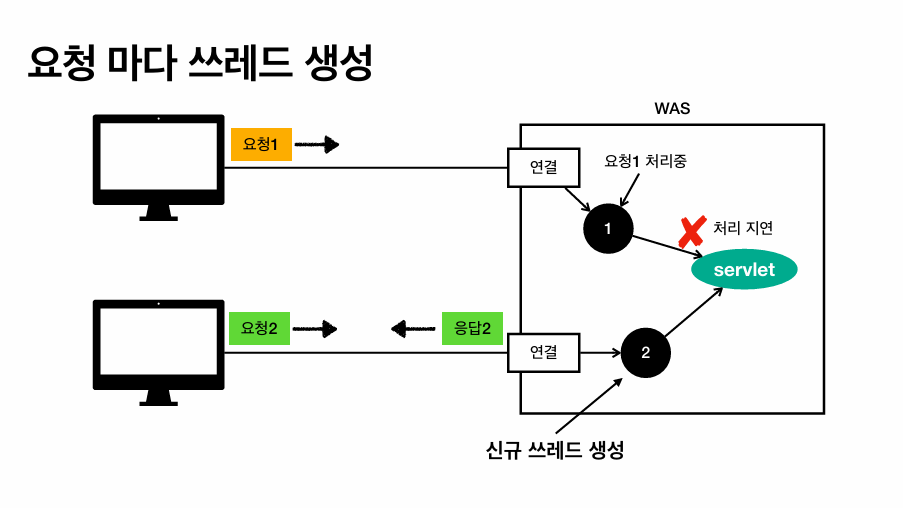




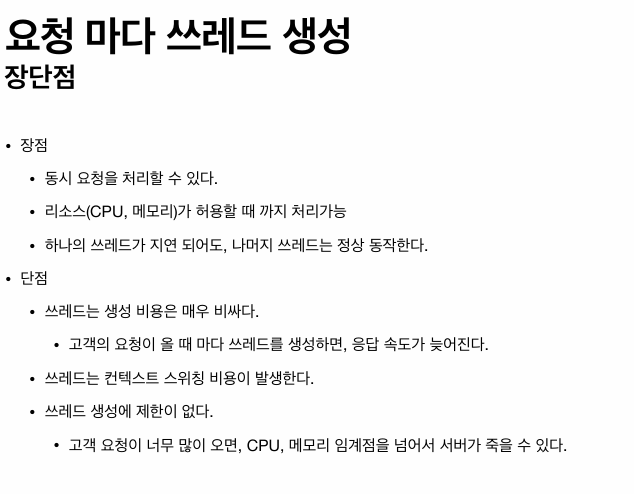
단일 쓰레드 사용할 때 다중요청이 들어오는 경우

요청 1이 들어왔는데, 쓰레드 내에서 처리가 지연되고 있는 상황에서 요청 2가 들어오면 쓰레드를 대기하고 있어야한다.

이런 경우 요청1과 요청2가 모두 동작을 하지 않게 되어 결과적으로 타임아웃 에러가 발생하게 된다.

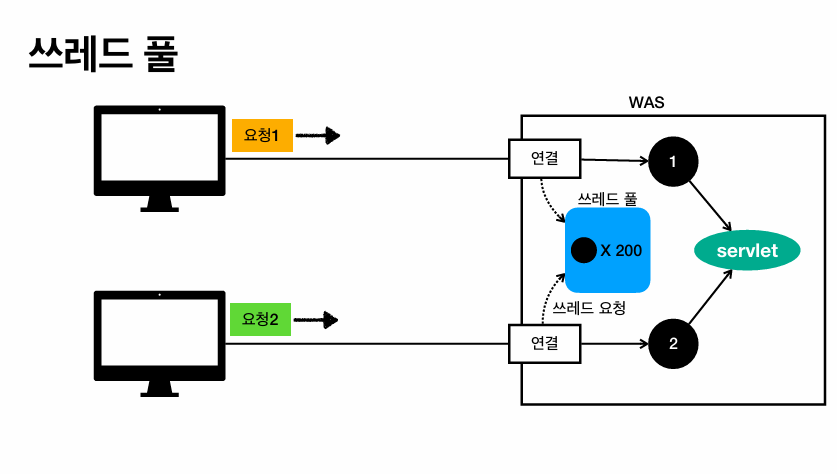


요청이 올 때마다 쓰레드를 새로 만들게 되면, 요청 1의 쓰레드에서 처리가 지연되더라도 요청2의 쓰레드를 독립적으로 만들어서 요청1과 관련 없이 요청2의 요청 응답 처리가 가능하게 된다.

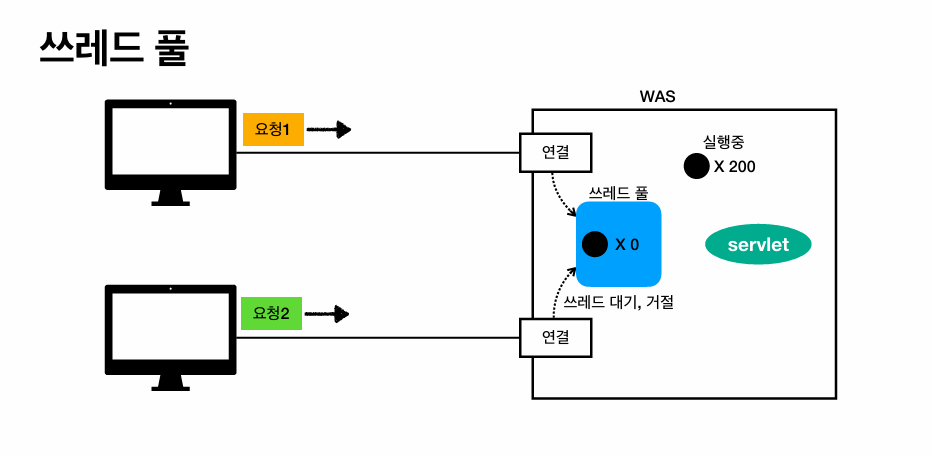


쓰레드를 생성할 때 시간이 걸리기 때문에 응답 속도가 늦어질 수 있다.

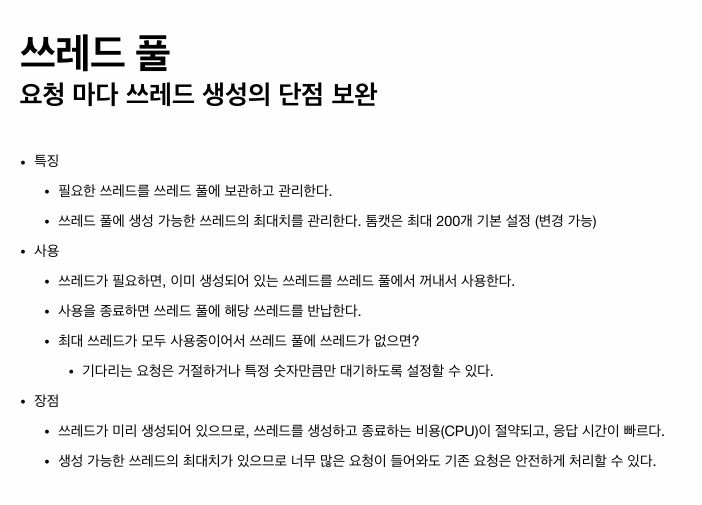
쓰레드 동시 처리는 실제로 동시 처리는 아니다. CPU 코어가 빠른 시간 내에 쓰레드를 전환하면서 처리를 해주기 때문에 동시처리로 보이는 것이고, 쓰레드를 전환하여 처리할 때 컨텍스트 스위치 비용이 발생한다.



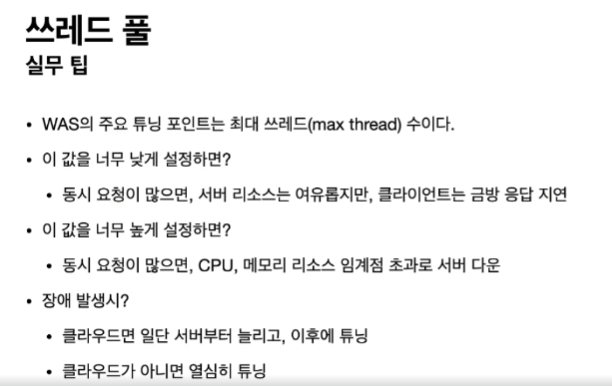
쓰레드 풀에 쓰레드를 미리 만들어 두고, 요청이 들어오면 쓰레드를 갖다가 씀. 요청에 대한 응답이 끝나면 쓰레드 풀에 다시 쓰레드를 반납을 한다. 쓰레드를 새로 생성하고 죽이는 과정을 하는 것이 아니라, 쓰레드를 미리 만들어 놓고 요청에 대해 쓰레드를 돌려 쓰는 것이 가능하다.

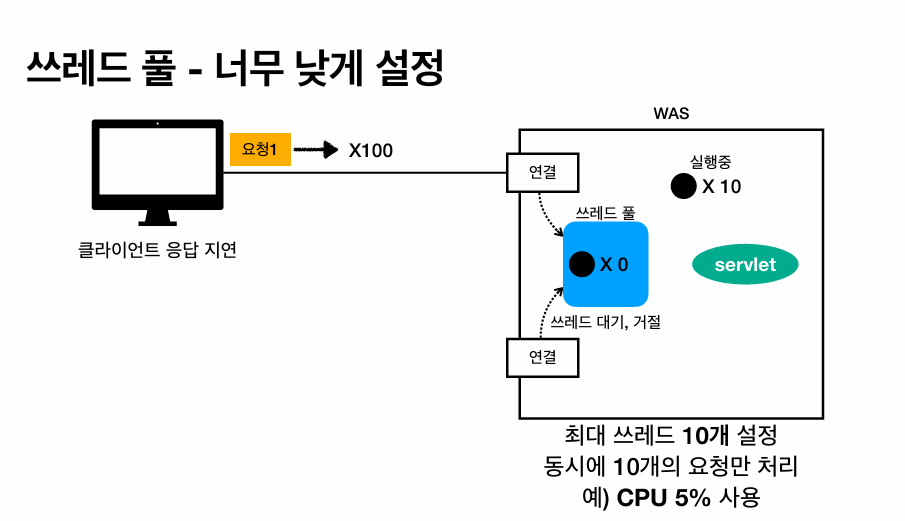


쓰레드 풀에 생성된 쓰레드 수가 모두 사용중인 상태에서 추가 요청이 들어온다면, 쓰레드 풀에 사용가능한 쓰레드가 존재하지 않게 된다. 따라서 요청에 필요한 쓰레드를 대기 시키거나 요청을 거절할 수 있다.



쓰레드 풀 최대치 설정 변경은 톰캣 MaxConnection 설정 방법을 찾아보면 된다.



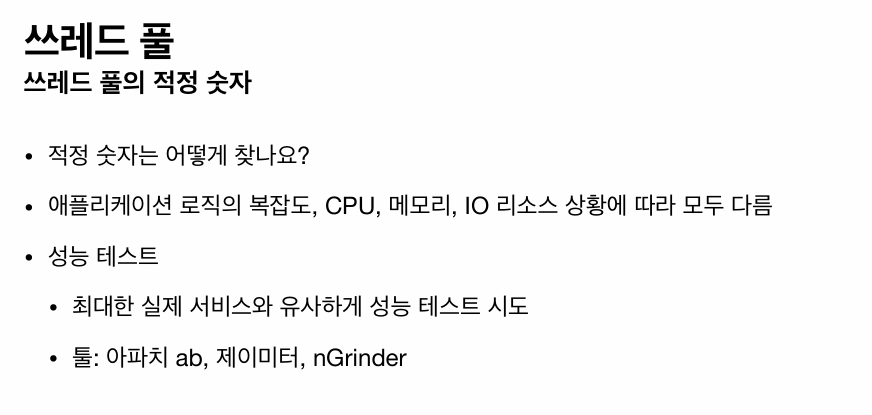


사용 가능 CPU리소스보다 더 적은 리소스만 사용하면서, 고객의 요청 응답은 지연될 수 있는 가능성이 존재한다.

-> 개발자가 설정을 잘못한 것. 서버를 늘리기 전에 MaxConnection수를 확인해보자.

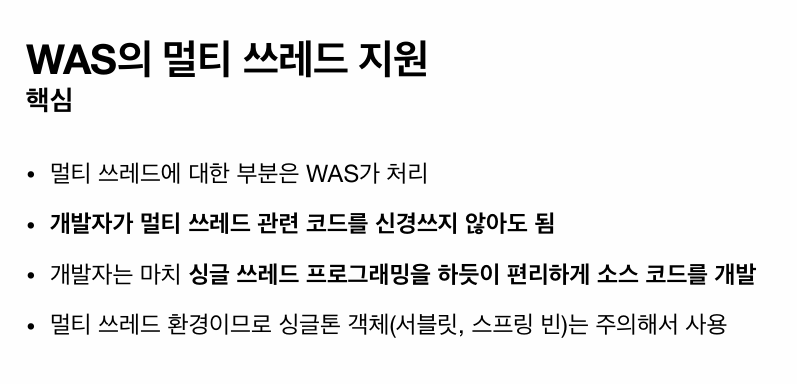
설정을 잘 해주면, 한 서버로 10배는 더 많이 처리할 수 있다. (CPU 50%까지는 사용해야함)

쓰레드 풀 설정 값이 너무 높으면, 동시 요청이 많을 때, CPU 메모리 리소스 임계점 초과로 서버가 죽게 된다. 서버가 죽으면 복구가 쉽지 않다.



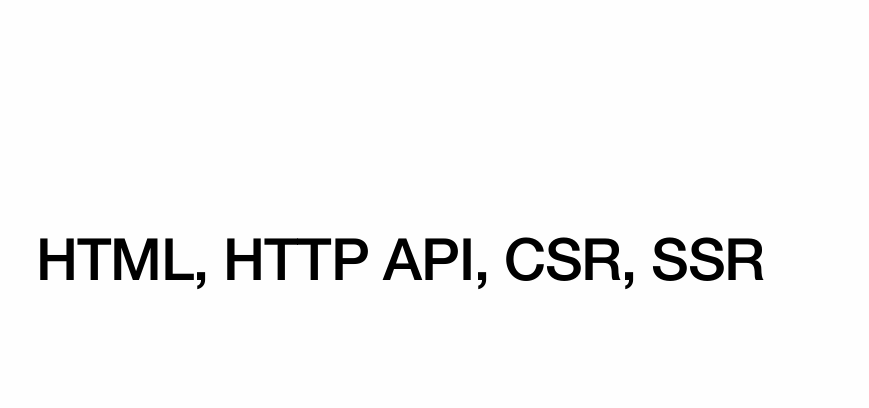
db를 몇 번 조회하는지 등의 애플리케이션 로직의 차이, IO 리소스를 얼마나 사용하는지, CPU 와 메모리 크기는 얼마나 되는지 등에 따라 다 다르게 maxConnection 수를 적용해야한다.

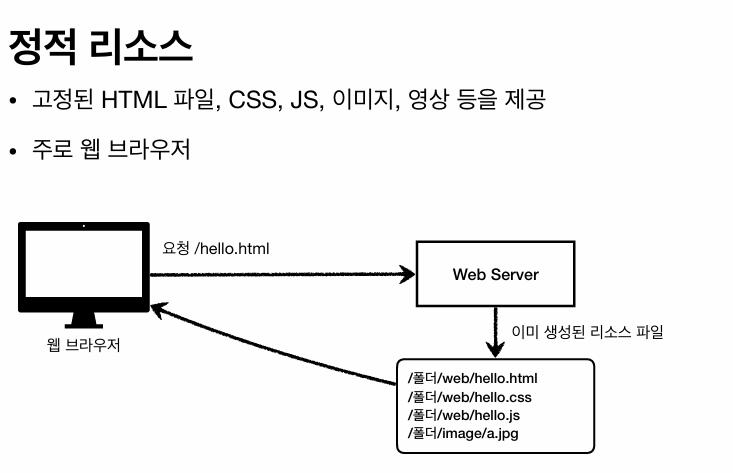
애플리케이션의 변화가 있다면 성능테스트를 해야함. 트래픽이 있는 서비스라면 성능테스트를 꼭 해봐야 함.

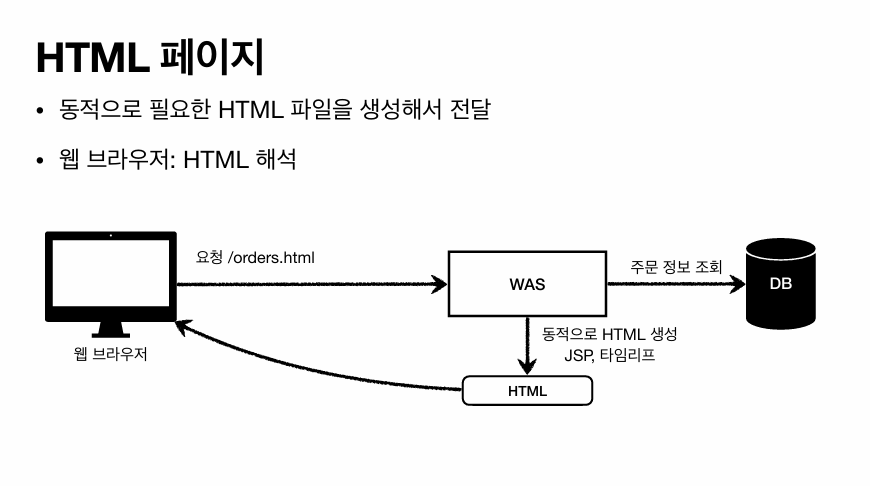


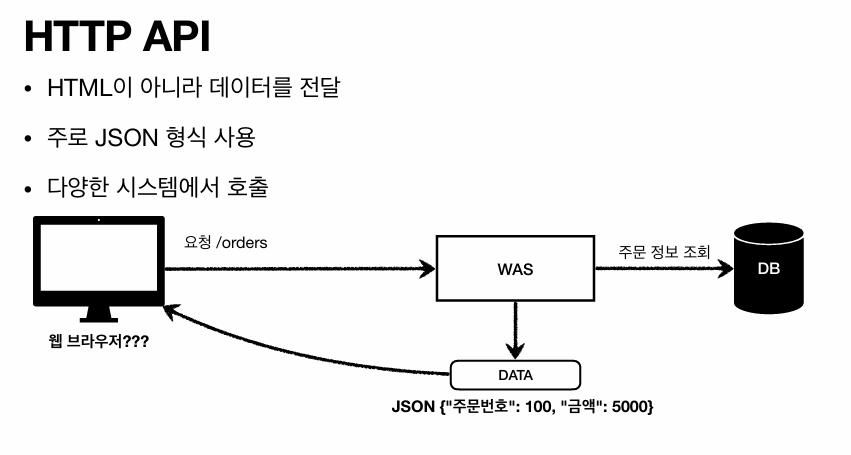
개발자가 설정 사용 방법만 이해하면 된다.

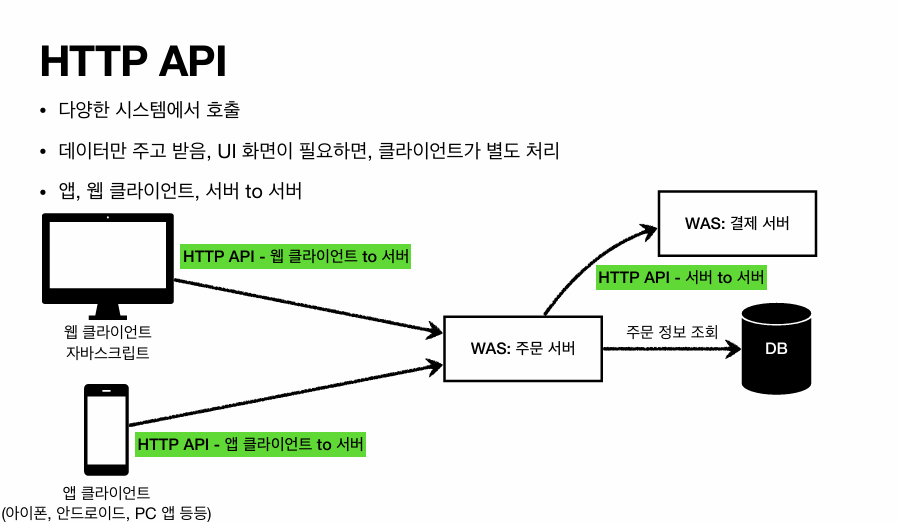
WAS의 가장 큰 핵심은 멀티쓰레드를 지원해주기 때문에, 싱글 쓰레드 프로그래밍을 하듯이 편리하게 소스 코드를 개발할 수 있다.











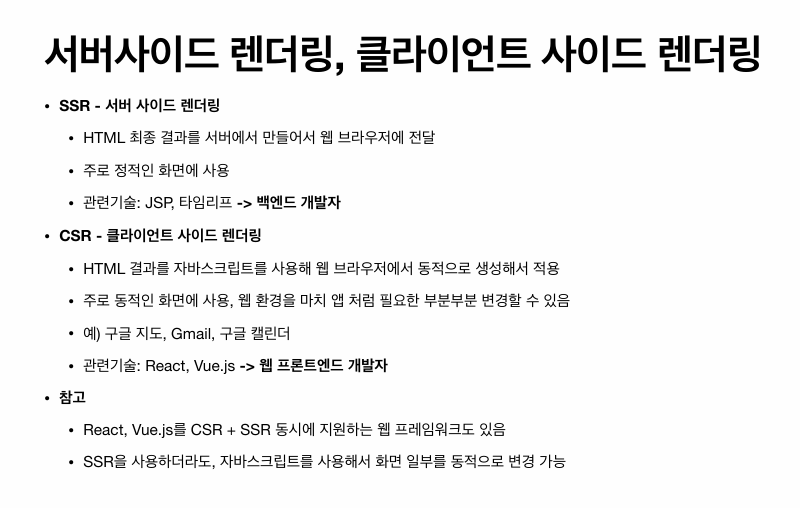
앱 클라이언트에는 화면에 대한 component는 가지고 있기 때문에 http api를 통해 데이터만 요청한다. 응답된 데이터를 화면에 넣어서 뿌려주기만 하면 됨.

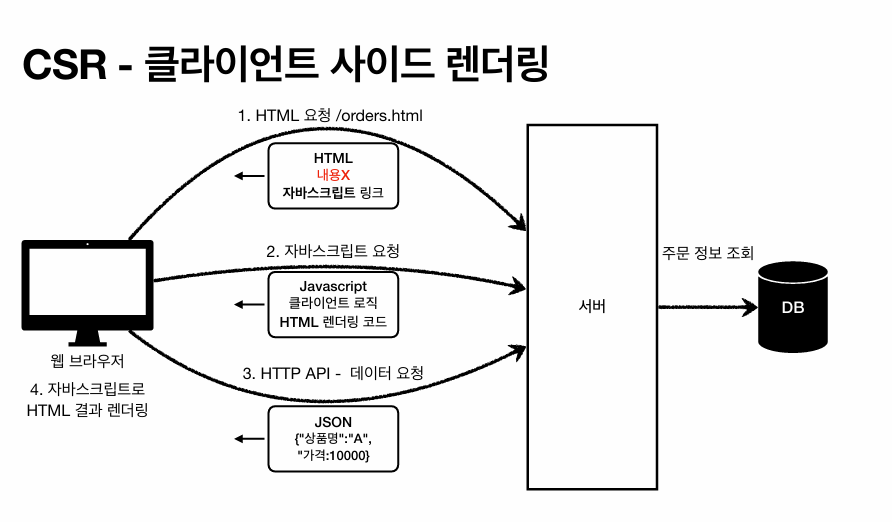
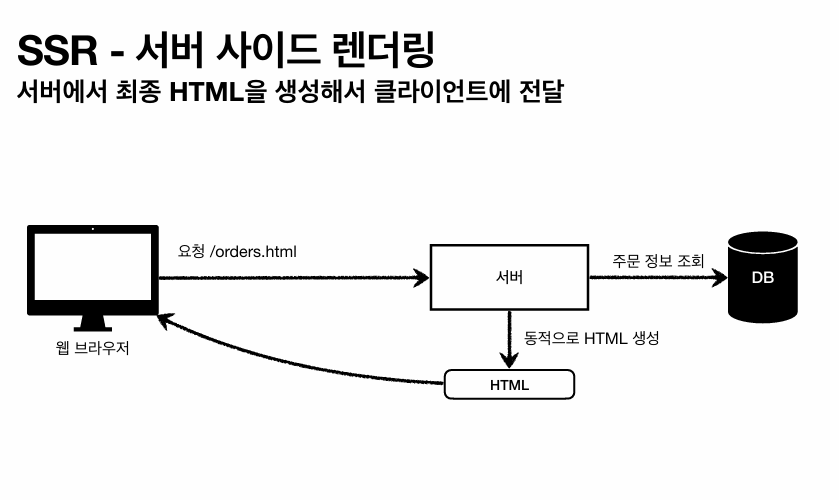
데이터를 내려주는 걸 http api라고 함.

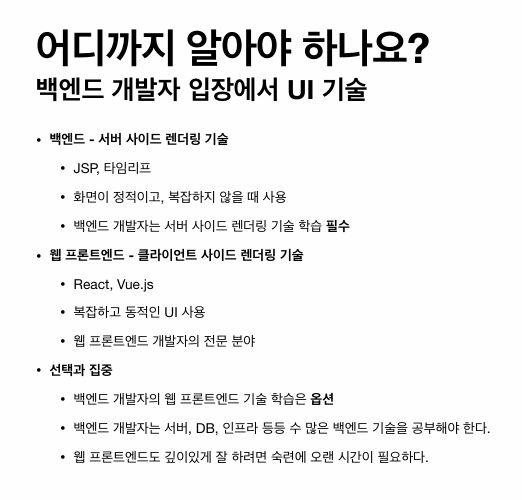
Html의 ajax와 같은 것이 http api를 호출해서 받은 데이터를 이용해서 화면을 만듦.

또는 서버 to 서버를 통해 데이터 교환이 이루어짐.

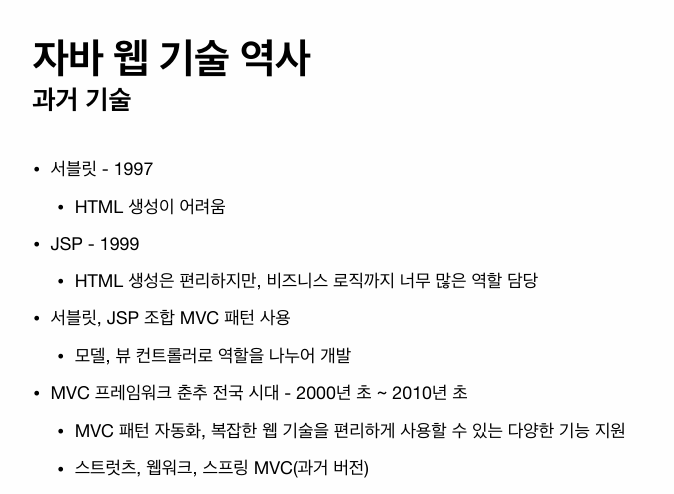
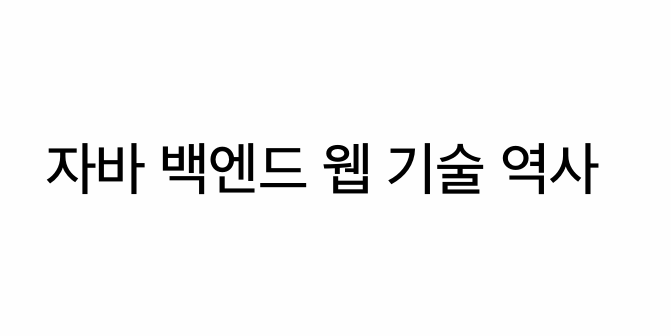








Jsp는 주로 안쓰고 타임리프를 스프링에서 밀기 때문에 타임리프를 배우는 것이 좋다.



서블릿이 나온 이유 : tcp/ip 연결이 어렵고 http 요청 정보 파싱 및 응답 처리가 어려웠음.

하지만 html생성엔 어려움이 있었다. html을 동적으로 자바코드로 만들어줘야했음.

response.write() 과 같은 코드를 사용해 html을 자바 코드로 만들어줌.

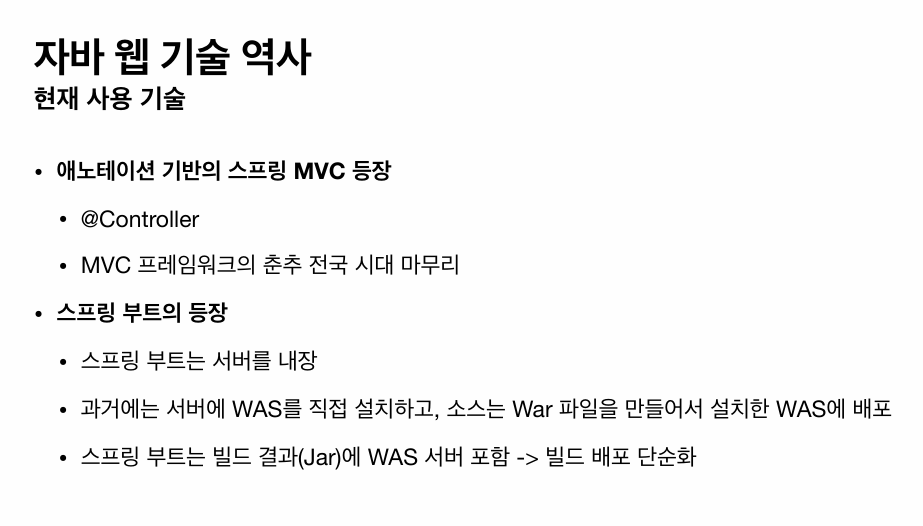
-> jsp의 등장

하지만 jsp내에서 자바 코드를 쓸 수 있기 때문에 이를 이용하여 비즈니스 로직까지 jsp 내에서 처리하는 방식이 존재했음. -> html 렌더링 뿐만 아니라 비즈니스 로직처리까지 이루어져서 너무 많은 역할을 담당하게 됨.

비즈니스로직 부분과 화면처리 관련 로직을 나눔.

MVC 패턴이 비슷하기 때문에 많은 사람들이 mvc 프레임 워크들이 많이 등장함.

과거에는 스트럿츠와 스프링 mvc를 함께 사용하는 시대가 있었음.



스프링이 제공해주는 mvc이기 때문에 스프링과 통합이 매우 잘됨.

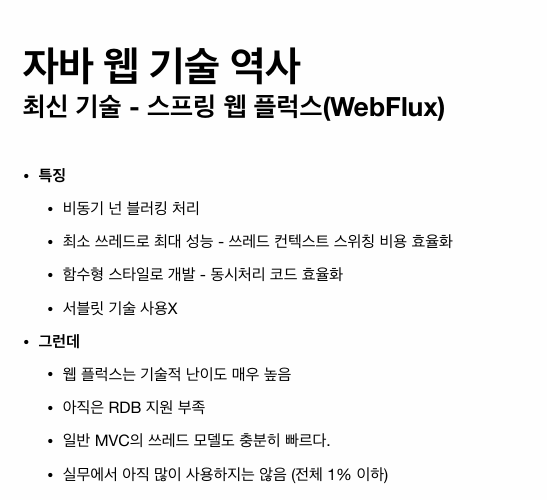
애노테이션 기반이기 때문에 유연하고 편리하게 코드 작성이 가능함.



스프링 MVC 는 기본적으로 서블릿 기반에서 동작을 한다.

서블릿 위에 스프링 mvc를 올려서 사용함.

Web Reactive 는 최신기술임. Spring WebFlux와 함께 사용.

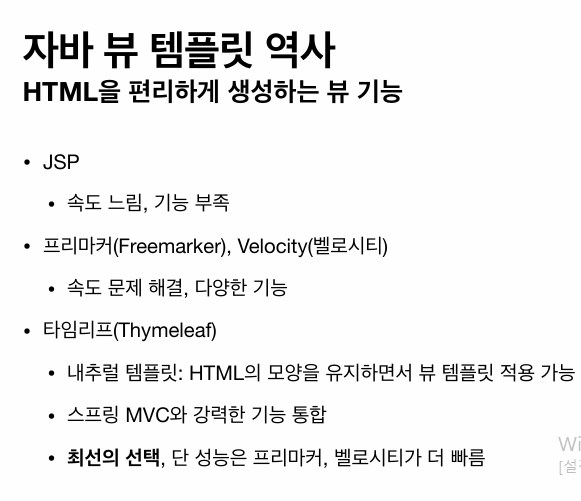


CPU 코어개수 또는 CPU 코어개수 + 1개 정도로 쓰레드 개수를 맞춤. 컨텍스트 스위칭 비용이 줄어들게 된다.

상품 서버 api조회, 주문 서버 api조회 등 여러 서버 데이터를 조회한 후 처리해야하는 경우에는 번거롭게 된다. -> 함수형 스타일로 개발하면 깔끔하게 처리 가능.

서블릿 기술 사용 x (Nettty라는 웹 프레임 워크를 사용)

자바도 nods js처럼 개발할 수 있음.



프리마커 벨로시티의 지원이 잘 이루어지지 않음.

jsp나 프리마커, 벨로시티에는 java코드가 어느정도 들어감.

타임리프는 html의 속성에 값을 넣기 때문에 굉장히 깔끔하게 적용 가능하다.

스프링 mvc와 강력한 기능 통합이 가능하다.