**웹 앱 개발**

**\* 생성자 패턴**

객체를 동적으로 생성하려면?? 이런 경우에 생성자를 사용할 수 있다. 아래 코드를 보면 Java와 비슷한 방식으로 객체를 생성할 수 있다는 것을 알 수 있다.

|  |
| --- |
| function Health(name, lastTime) {  this.name= name;  this.lastTime= lastTime;  showHealth: function() {  console.log(this.name + “님, 오늘은 “ + lastTime + “에 운동하심.”;  }  }  var obj= new Health(“crong”, “AM 10:10”); // new 연산자를 통해 객체를 만든다. |

단 위의 Health 오브젝트는 객체가 생성될 때마다 showHealth라는 function이 계속 생성되므로 메모리가 낭비되게 된다. 이를 위해 prototype으로 메서드를 선언하면 메서드를 공유해서 사용할 수 있다.

|  |
| --- |
| function Health(name, lastTime) {  this.name= name;  this.lastTime= lastTime;  }  **Health.prototype.showHealth= function() {**  **console.log(this.name + “님, 오늘은 “ + lastTime + “에 운동하심.”;**  **}**  var crong= new Health(“crong”, “AM 10:10”);  var bruce= new Health(“bruce”, “AM 11:10”);  console.log(crong === bruce) // false  console.log(crong.showHealth === bruce.showHeatl) // true |

**\* form 데이터 보내기**

회원가입과 같은 기능 구현을 위해선 서버에 데이터를 전송할 필요가 있는데, HTML에선 form 태그를 통해 이런 데이터 전송을 지원한다. 뿐만 아니라 페이지를 다시 그리지 않기 위해 form 데이터를 Ajax로 처리할 수도 있다.

그렇다면 form 태그를 통해 데이터를 전송하기 전에 forntend에서 사용자가 입력한 값 검증을 해야 하는 상황도 있을 수 있다. 아래와 같은 방식으로 할 수 있다.

|  |
| --- |
| <form ..... id=”myForm”>    <input class=”sendBtn” type=”submit”>  </form>  <script>  document.querySelector(“.sendBtn”).addEventListener(“click”, function(evt) {  **evt.preventDefault();** form 버튼의 기본 기능을 안하도록 설정  // 검증로직  // var email= document.querySelector(“[name=’email’]”).value;  **document.querySelector(“#myForm”).submit;**  }  </script> |

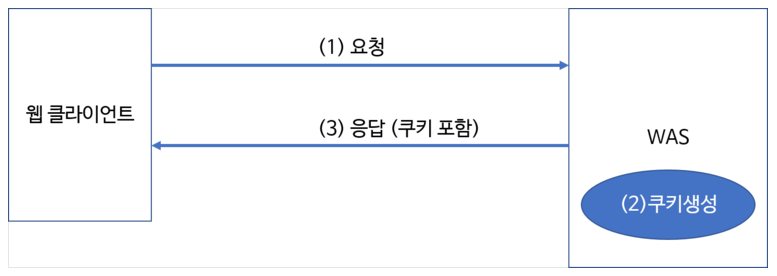
**\* 상태 정보란**

http은 서버 클라이언트 간의 연결을 유지하지 않으므로 상태 유지가 되지 않는 프로토콜이다. 때문에 이전에 무엇을 했는지, 지금은 무엇을 했는지 알 수 없다. 때문에 상태 유지를 하기 위해서 쿠키와 세션 기술이 등장하게 된다.

|  |  |
| --- | --- |
| **쿠키** | 데이터를 사용자 컴퓨터에 저장.  저장된 정보를 다른 사람이나 시스템이 볼 수 있음.  유효시간이 지나면 사라짐. |
| **세션** | 데이터를 서버에 저장.  서버가 종료되거나 유효시간이 지나면 사라짐. |

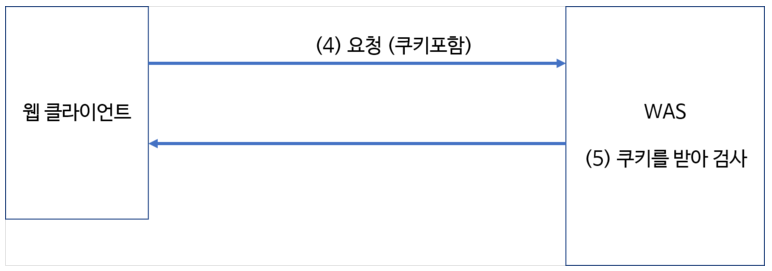
**\* 쿠키의 동작 과정**

1. 클라이언트의 요청을 받으면 서버는 쿠키 이름과 값 데이터와 유지시간과 같은 추가적인 정보를 가진 쿠키 데이터(문자열 데이터)를 생성하여 응답 데이터에 포함하여 클라이언트에 전송한다.



**주의** 클라이언트 브라우저에 따라 쿠키의 크기 및 저장할 수 있는 쿠기 갯수도 제한되어 있으므로 서버는 이 점에 주의해야 한다. 또한 하나의 클라이언트가 같은 이름의 쿠기를 가질 수 없으므로 나중에 서버로부터 받은 쿠키의 데이터가 덮어써지게 된다.

이 후 클라이언트는 다음 요청부터 응답 받은 쿠키 데이터를 서버에 같이 보내게 되고 서버는 쿠키 데이터를 받으면 이전에 만들었던 쿠키인지 검사하여 이전 사용자인지 아닌지 파악한다.

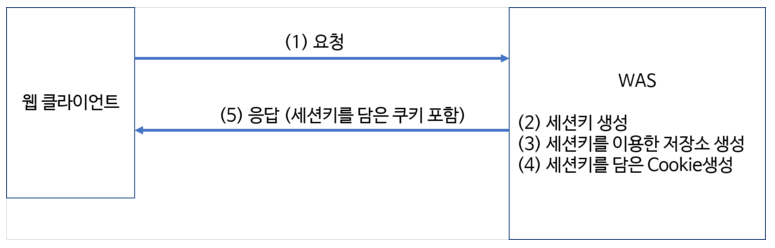


쿠키를 사용한 코드를 간략히 살펴보자.

|  |
| --- |
| Cookie cookie= new Cookie(“name”, “value”);  cookie.setPath(“/”) // 명시된 경로 이하에 모두 쿠기 적용.  response.addCookie(cookie);  Cookie[] cookies= request.getCookies();  cookies[0].setMaxAge(0); // 쿠키는 클라이언트가 관리하므로 클라이언트에게 쿠키 삭제 요청 |

**\* 세션의 동작 과정**

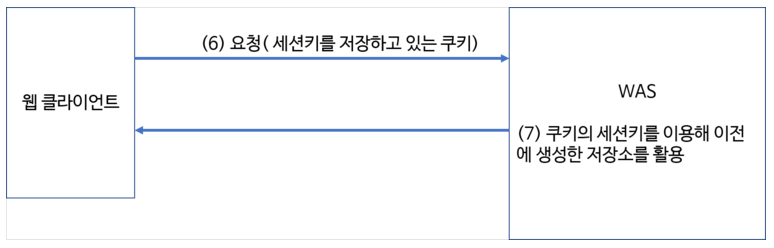
세션은 클라이언트에게 요청 받으면 세션키 및 키, 값 형태로 데이터를 저장할 수 있는 저장소를 생성하고 클라이언트의 응답에 세션키 정보를 쿠키에 담아 전송한다.



세션 사용 코드를 간략히 보자.

|  |
| --- |
| HttpSession session= request.getSession(); // 세션이 없다면 새로 만들어 준다.  HttpSession session= request.getSession(false); // 세션을 새로 안만들고 null 리턴.  session.setAttribute(“키”, “값”); // 값은 Object  session.getAttribute(“키”); |

이후 클라이언트의 요청이 오면 쿠키에 있는 세션키를 통해 상태 정보를 알아낼 수 있다. 참고로 세션의 기본 유지 시간은 30분이고, 세션이 한 번 사용되면 해당 시간은 다시 초기화되게 된다.



**\* 인터셉터란?**

인터셉터는 Dispatcher Servlet에서 Handler(Controller)로 요청을 보낼 때와 Handler로부터 응답을 받을 때 동작하는 모듈을 의미한다. 사용 예제는 아래와 같다.

|  |
| --- |
| class LogInterceptor extends HandlerInterceptorAdapter {  @Override  public void postHandle(HttpServletRequest...) { } // Controller 종료 후 동작  @Override  public void preHandle(...) { } // Controller 호출 전 동작  } |
| @Configuration  class Configuration {  @Override  public void addInterceptors(InterceptorRegistry registry) {  registry.addInterceptor(new LogInterceptor());  }  } |

**\* Argument Resolver란?**

컨트롤러 메소드의 인자로 사용자가 임의의 값을 전달하는 방법을 제공할 때 사용하는 기능이다. 예를 들어 세션에 저장되어 있는 값 중 특정 이름의 값을 메소드 인자로 전달하는 경우가 있다. 사용 예제는 아래와 같다.

|  |
| --- |
| class MyObjResolver implements HandlerMethodArgumentResolver {  @Override  public boolean supportsParameter(MethodParameter parameter) {  return parameter.getParameterType() == MyObj.class;  }  @Override  public Object resolveArgument(MethodParameter, ModelAndView....) {  return new MyObj(...);  }  } |
| @Configuration  class Configuration {  @Override  public void addArgumentResolver(List<Handler...> argumentResolvers) {  argumentResolvers.add(new MyObjResolver());  }  } |