6. Session Tracking

Session Tracking

- 등장배경
  - HTTP 프로토콜의 상태비유지 특징으로 인해서 동일한 사용자의 여러 요청에 대한 하나의 상태 유지가 불가능하다.
- Session Tracking
  - 일정 시간동안 동일한 사용자의 여러 요청들을 하나의 상태로 보고 유지 관리하기 위한 기술
- Cookie
  - 상태정보를 client 측에 저장하여 유지하는 기술
- Session
  - 상태정보를 server측에 저장하여 유지하는 기술

Cookie

- 서버와 브라우저 간에 교환되는 작은 데이터(이름/값의 String 쌍)
- 서버는 브라우저로 쿠키를 보내고, 이후 클라이언트는 매번 요청에 이 값을 전송한다.
- 브라우저의 설정에서 Cookie를 허용하지 않으면 사용할 수 없다.
- 세션쿠키
  - 브라우저의 메모리에 저장되는 쿠키
- 일반쿠키
  - 시스템에 파일로 저장되는 쿠키

Cookie

Cookie 객체를 생성합니다.

```
Cookie cookie = new Cookie("username", name);
```

Cookie 클래스 생성자는 "이름/값" 쌍을 인자로 받습니다.

쿠키가 클라이언트에 얼마나 오랫동안 살아 있을지 설정합니다.

```
cookie.setMaxAge(30*60);
```

setMaxAge()는 초 단위로 설정합니다. 이 코드가 의미하는 바는 "30*60초(60분) 동안 클라이언트에 살아 있어라"입니다. -1을 설정하면 쿠키는 파일 같은 곳으로 영구적으로 저장되지 않으며, 브라우저가 빠져나가는 대로 지우라는 의미입니다. "JSESSIONID" 쿠키의 getMaxAge()는 무엇을 리턴할지 알겠나요?

쿠키를 클라이언트로 보냅니다.

```
response.addCookie(cookie);
```

클라이언트 Request에서 쿠키(들)를 읽어옵니다.

```
Cookie[] cookies = request.getCookies();
for (int i = 0; i < cookies.length; i++) {
    Cookie cookie = cookies[i];
    if (cookie.getName().equals("username")) {
        String userName = cookie.getValue();
        out.println("Hello " + userName);
        break;
    }
}
```

getCookie(String) 메소드는 존재하지 않습니다. Cookie에서 getCookies()만 호출 가능합니다. 원하는 쿠키를 찾으려면 배열을 루프로 돌려야 합니다.

Session

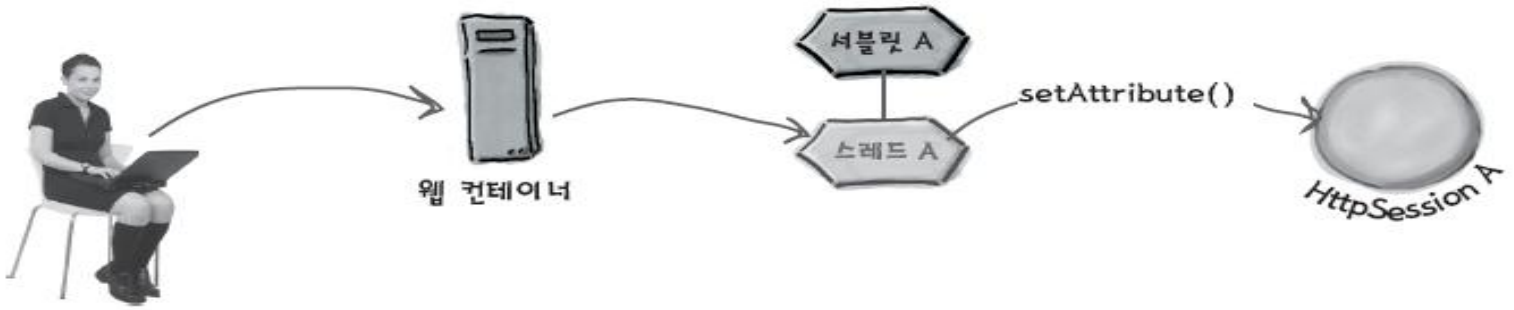
- 서버측에서 사용자의 상태정보를 유지하기 위해 정보를 저장하는 객체.
- 내부적으로 Cookie를 사용한다.
- 일정 시간동안만 유지된다.

Session

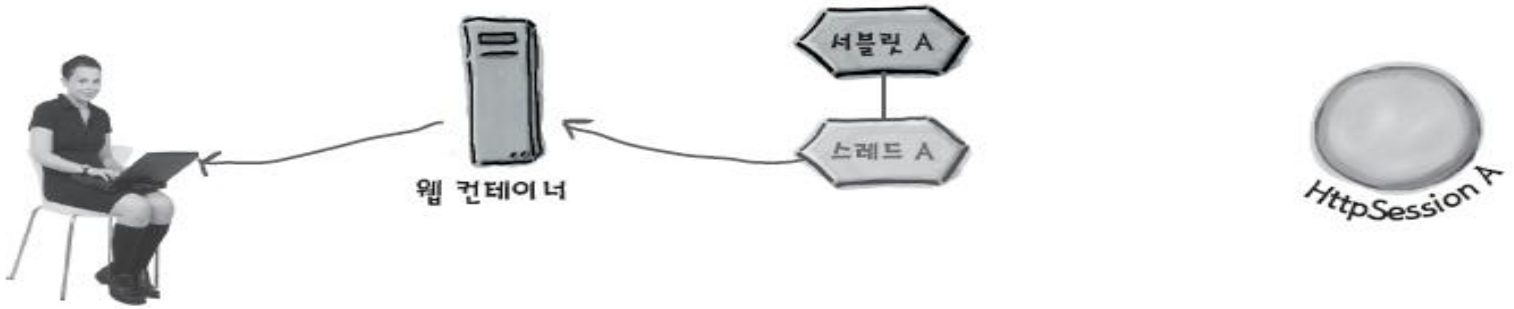
① 다이아나는 Dark를 선택한 다음, Submit 버튼을 클릭합니다.

컨테이너는 BeerApp 서블릿의 새로운 스레드로 요청을 보냅니다.

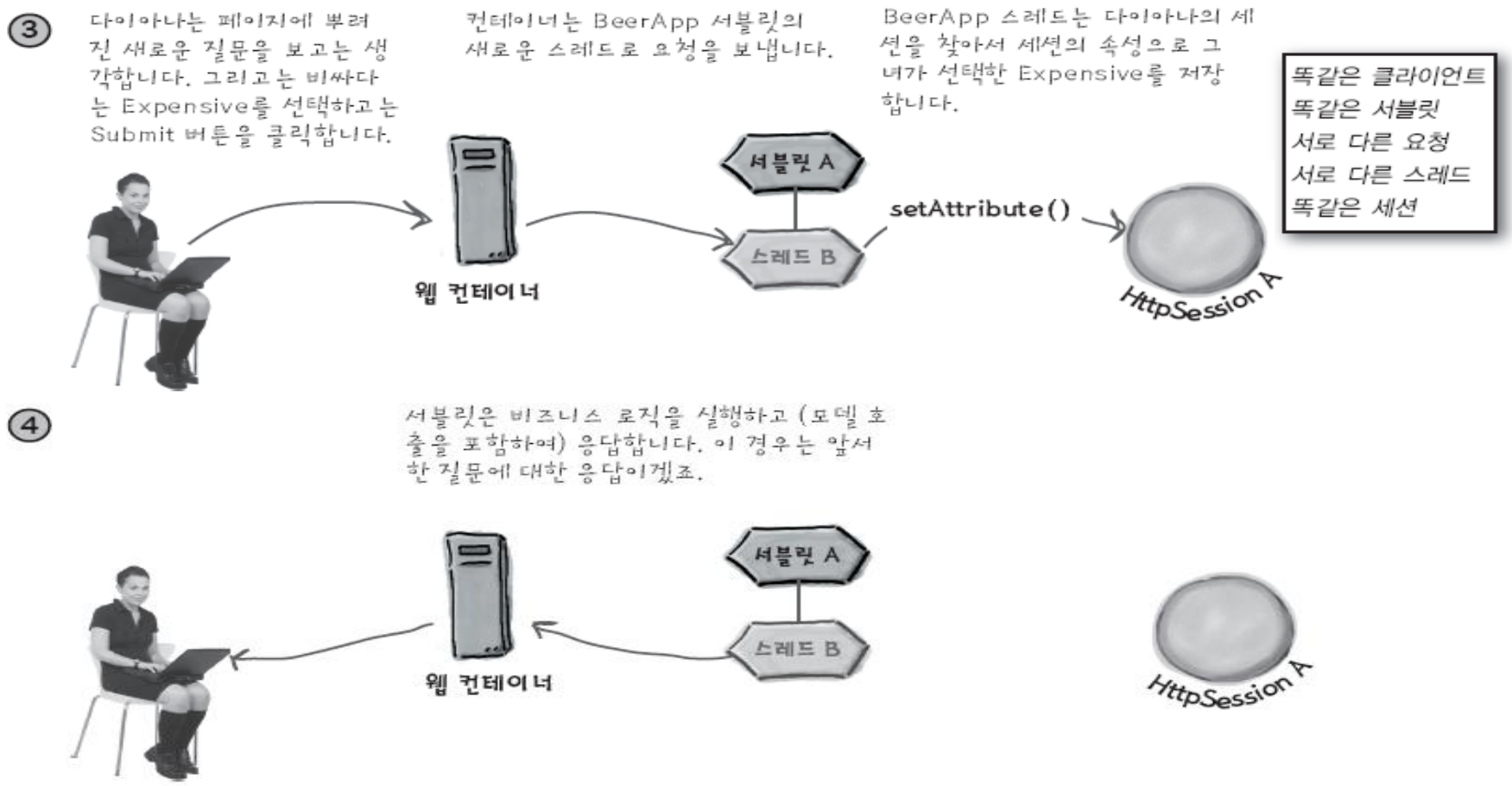
BeerApp 스레드는 다이아나의 세션을 찾아서 세션의 속성에 그녀가 선택한 Dark를 저장합니다.



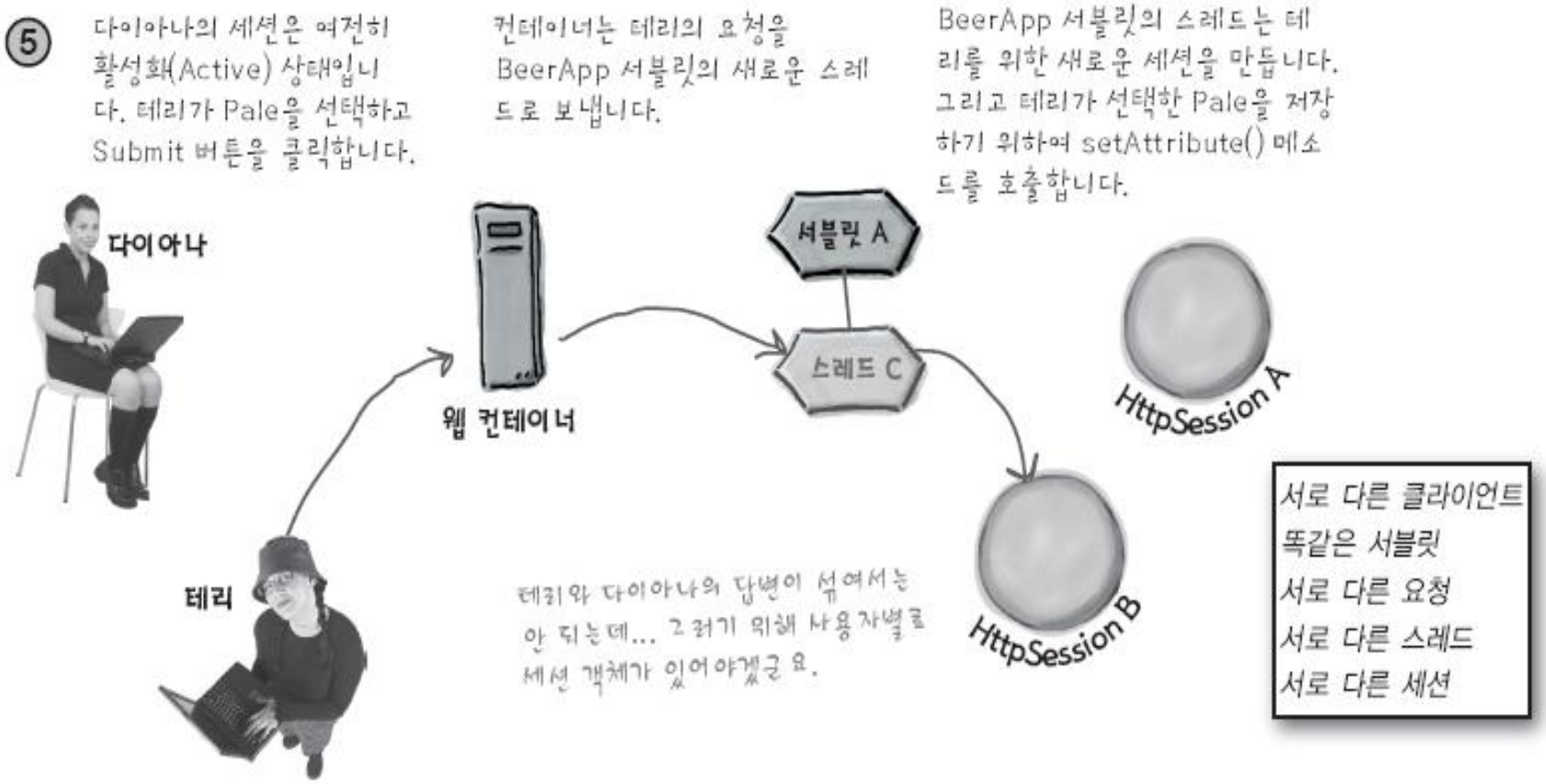
② 서블릿은 비즈니스 로직을 실행하고 (모델을 호출하여), 응답합니다. 여기서 질문을 하나 더 하죠. "생각하는 가격대는 어떻게 됩니까?"



Session

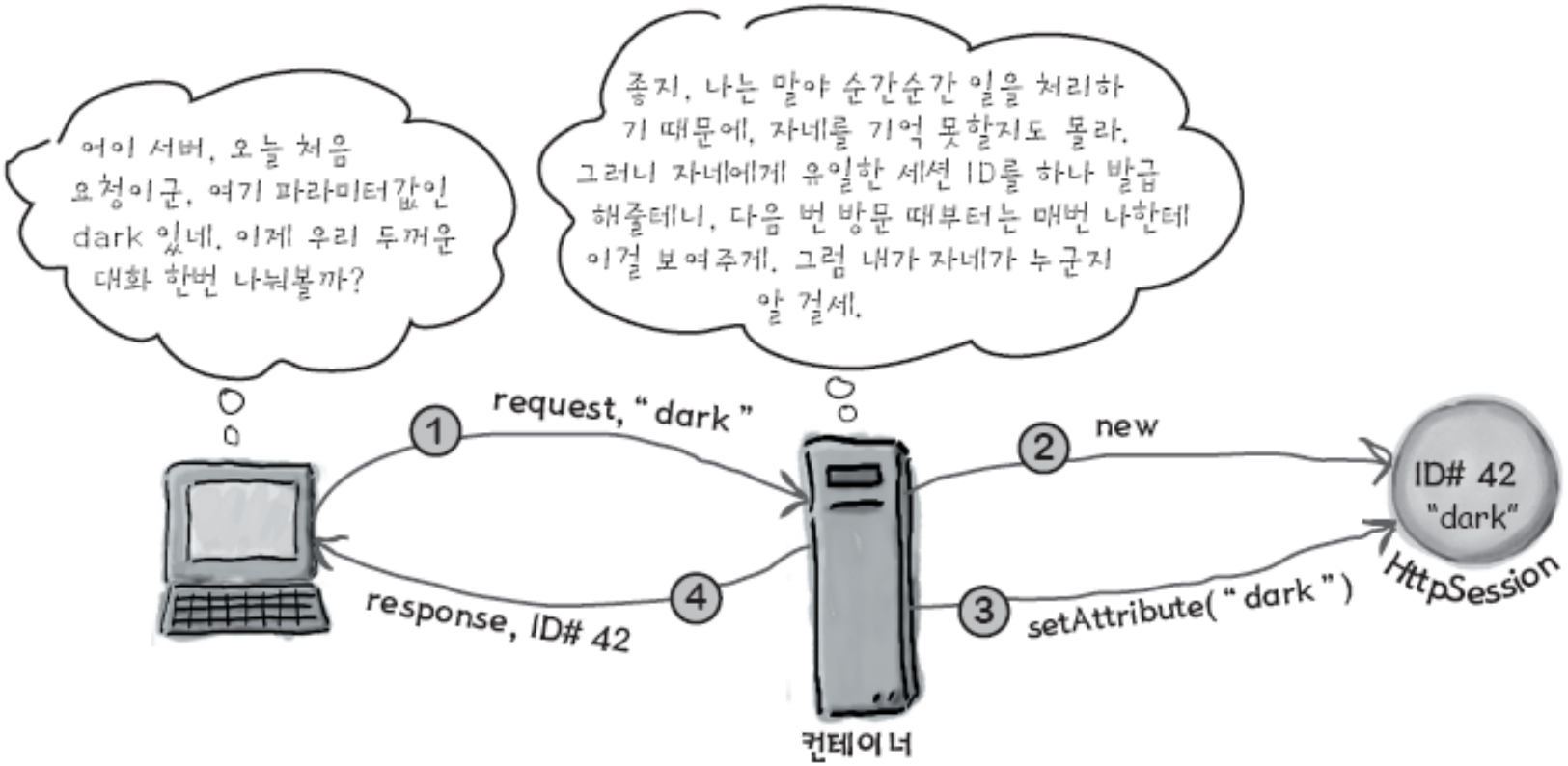

3
다이아나는 페이지에 뿌려진 새로운 질문을 보고는 생각합니다. 그리고는 비싸다는 Expensive를 선택하고는 Submit 버튼을 클릭합니다.
컨테이너는 BeerApp 서블릿의 새로운 스레드로 요청을 보냅니다.
BeerApp 스레드는 다이아나의 세션을 찾아서 세션의 속성으로 그녀가 선택한 Expensive를 저장합니다.
서블릿 A
스레드 B
웹 컨테이너
setAttribute()
HttpSession A
똑같은 클라이언트
똑같은 서블릿
서로 다른 요청
서로 다른 스레드
똑같은 세션
4
서블릿은 비즈니스 로직을 실행하고 (모델 호출을 포함하여) 응답합니다. 이 경우는 앞서 한 질문에 대한 응답이겠죠.
서블릿 A
스레드 B
웹 컨테이너
HttpSession A

Session


⑤
다이아나의 세션은 여전히 활성화(Active) 상태입니다. 테리가 Pale을 선택하고 Submit 버튼을 클릭합니다.
컨테이너는 테리의 요청을 BeerApp 서블릿의 새로운 스레드로 보냅니다.
BeerApp 서블릿의 스레드는 테리를 위한 새로운 세션을 만듭니다. 그리고 테리가 선택한 Pale을 저장하기 위하여 setAttribute() 메소드를 호출합니다.
다이아나
웹 컨테이너
서블릿 A
스레드 C
HttpSession A
HttpSession B
테리
테리와 다이아나의 답변이 섞여서는 안 되는데... 그러기 위해 사용자별로 세션 객체가 있어야겠군요.
서로 다른 클라이언트
똑같은 서블릿
서로 다른 요청
서로 다른 스레드
서로 다른 세션

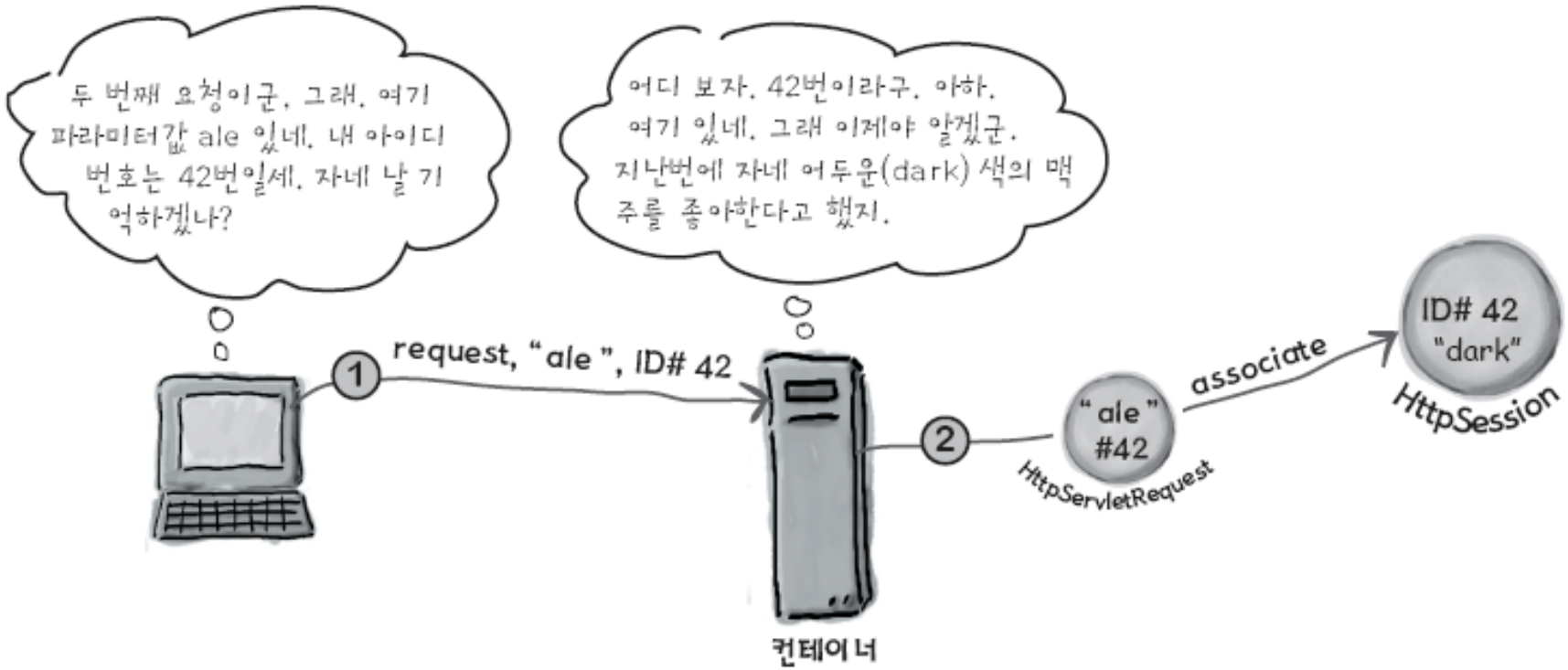
Session ID

- 클라이언트는 유일한 세션 ID를 이용하여 관리한다.



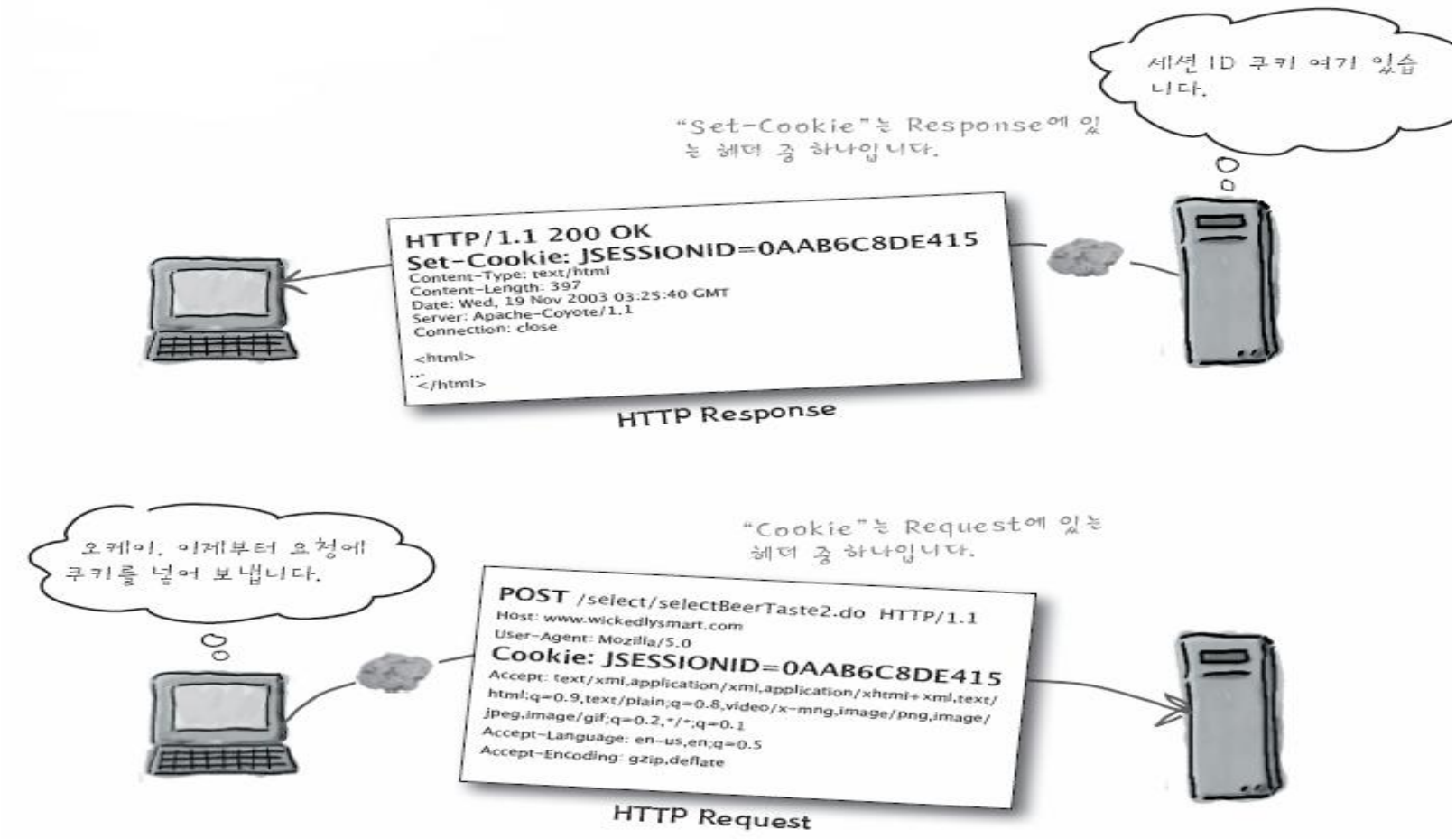
Session ID

- 두번째 요청부터는 세션 ID를 요청 시에 서버에 보내고, 서버에서는 세션 ID가 일치하는 세션을 찾아 요청과 연결한다.



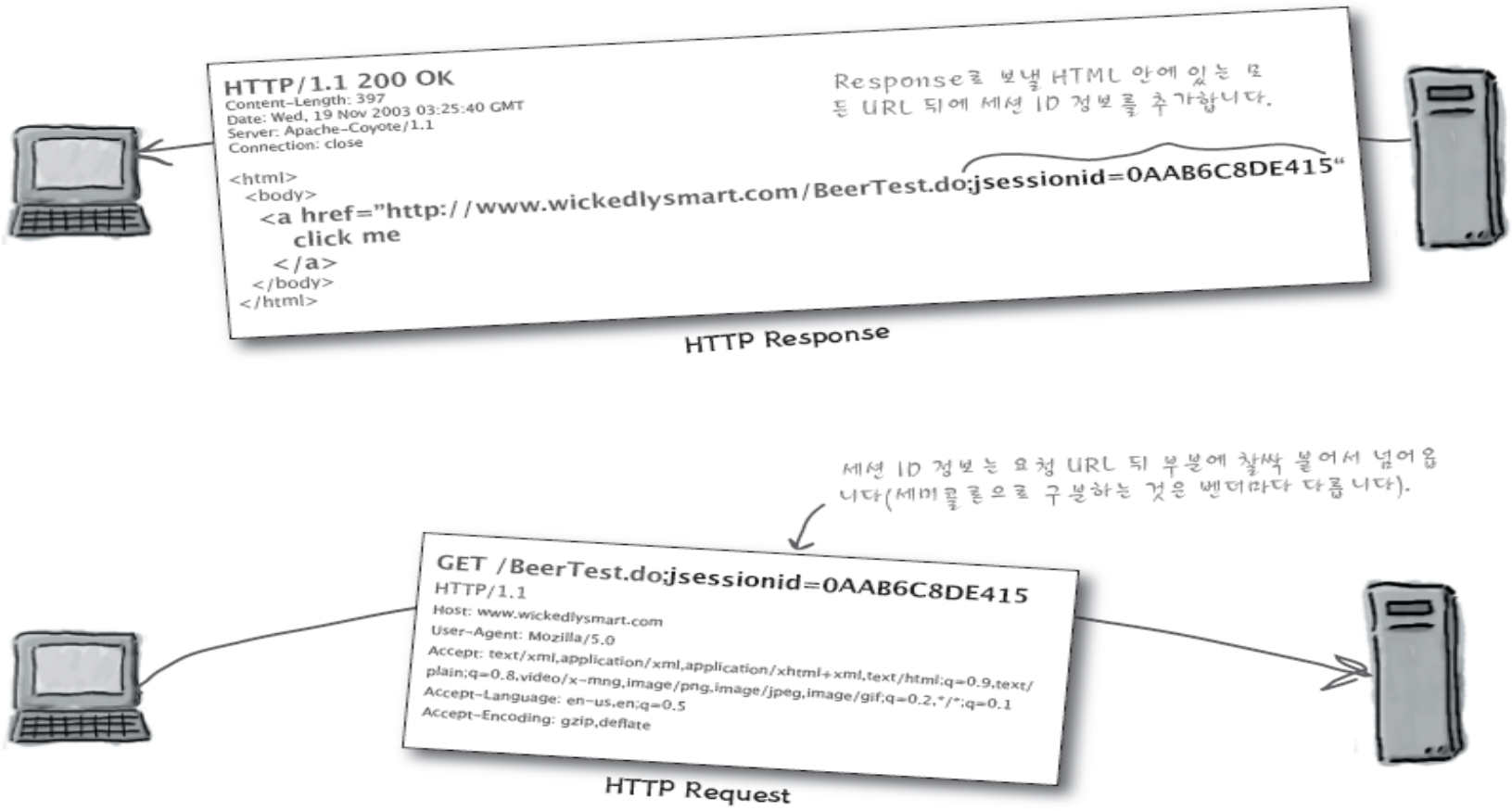
Session ID

- 세션 ID를 쿠키로 사용하는 경우



Session ID

- 세션 ID를 쿠키로 사용하지는 못하는 경우 : URL 재작성



Session 제거

- 세션이 장시간 비활성화 상태가 되면 서버에서 세션 삭제
- 서버에 세션 타임아웃 관련 설정이 있음

① DD에서 세션 타임아웃을 설정하기

DD에서 설정하는 타임아웃은 생성되는 모든 세션에 setMaxInactiveInterval() 메소드를 호출하는 것과도 같습니다.

```
<web-app ...>
  <servlet>
    ...
  </servlet>
 <session-config>
    <session-timeout>15</session-timeout>
  </session-config>
</web-app>
```

"15"는 15분을 의미합니다. "클라이언트가 15분 동안 요청이 없으면, 제거하라"는 의미입니다.*

*제거한다고 클라이언트를 제거하는 것이 아니라 세션을 제거한다는 의미입니다.

② 특정 세션만 타임아웃 설정하기

특정 세션 인스턴스만 세션 타임아웃 값을 변경할 수 있습니다(다른 세션의 타임아웃 값은 바뀌지 않습니다).

```
session.setMaxInactiveInterval(20*60);
```

이 메소드를 호출한 세션만 바뀝니다.

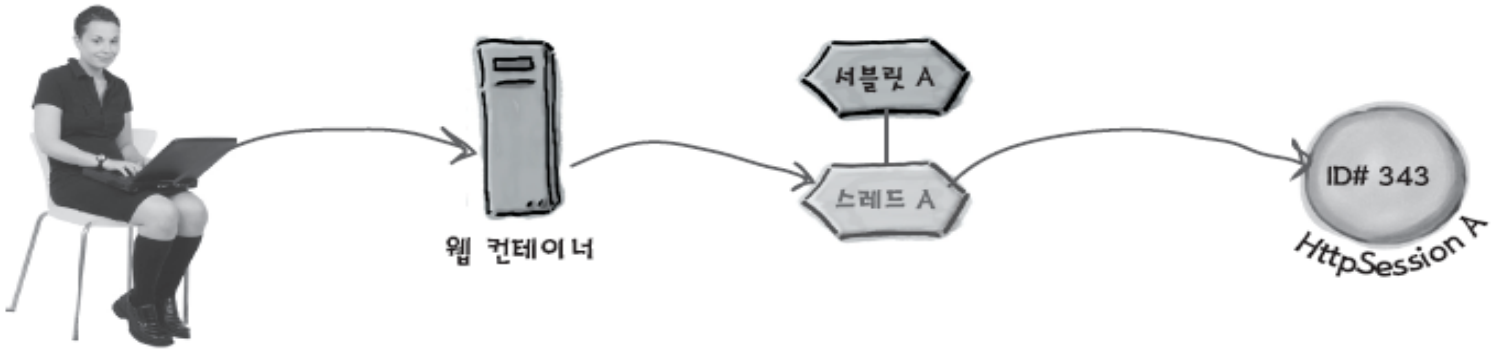
메소드의 인자는 초 단위의 시간 값입니다. 위 코드는 "클라이언트가 20분 동안 요청이 없으면, 제거하라"는 의미입니다.

Session 제거

① 다이아나는 Dark를 선택하고 Submit 버튼을 클릭합니다.

컨테이너는 접수한 요청을 BeerApp 서블릿의 스레드로 보냅니다.

컨테이너는 새로운 세션 ID #343을 만듭니다. "JSESSIONID" 쿠키를 Response에 넣어 다이아나에게 돌려 보냅니다(그림에는 없음).



② 에잉. 다이아나가 어디 갔지? 갑자기 사라져버렸습니다.

컨테이너는 쉬는 시간에도 자신이 할 일을 열심히 하겠죠. (다이아나 말고도 서비스해야 할 고객이 많으니까요)

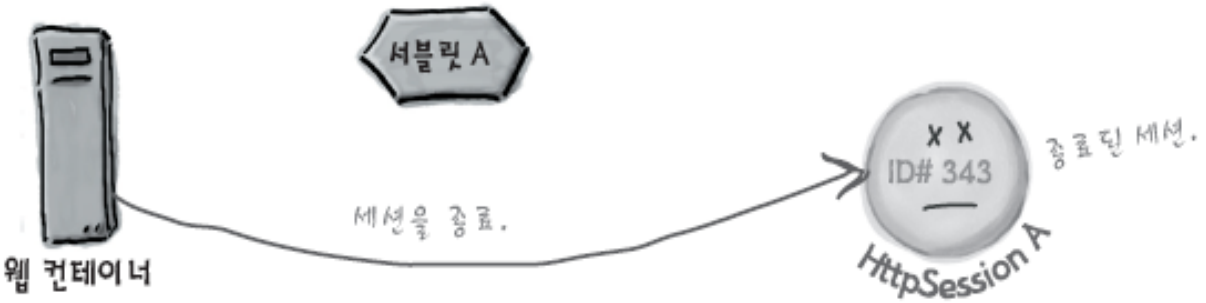
다이아나의 세션은 여전히 서버에 남아 있습니다. 기다리며... 끝없이 기다리며...

Session 제거

③ 시간이 지나도 다이아나는 돌아오지 않습니다.

컨테이너는 세션 #343의 상태를 체크해보고는 30분 동안이나 들어온 요청이 없다는 것을 확인합니다.

컨테이너는 이렇게 되뇌이죠. "그래 30분은 장시간이야. 다이아나는 안 돌아 올거야" 컨테이너는 결국 짝 잃은 불쌍한 세션을 제거합니다.



Session 주요 메소드

	하는 일이 무얼까요?	어떤 경우에 쓸까요?
getCreationTime()	세션이 생성된 시간을 리턴합니다.	세션이 얼마나 오래 되었는지 알고 싶을 때. 특정 시간만 세션을 사용하도록 제한하는 데 사용할 수 있습니다. 예를 들면 "로그인한 후 10분간만 사용하도록 하겠다"와 같은 경우.
getLastAccessedTime()	이 세션으로 들어온 마지막 요청 시간을 리턴합니다. (밀리초 단위)	클라이언트가 언제 마지막으로 세션에 접근했는지 알고 싶을 때. 클라이언트가 장시간 사용이 없을 경우, 메일을 보내 다시 사용하러 올 것인지 물어볼 때 사용할 수 있겠죠. 또는 이런 세션을 invalidate() 시킬 수도 있구요.
setMaxInactiveInterval()	해당 세션에 대한 요청과 요청 간의 최대 허용 시간(초 단위)을 지정합니다.	클라이언트의 요청이 정해진 시간이 지나도 들어 오지 않을 경우, 해당 세션을 제거하기 위하여 사용함. 서버 상에 짝 잃은 세션(stale session)을 최소화하기 위하여 사용합니다.
getMaxInactiveInterval()	해당 세션에 대한 요청과 요청 간의 최대 허용 시간(초 단위)을 리턴합니다.	세션이 얼마나 오랫동안 비활성화 상태였는지, 여전히 살아있기는 한지 알고 싶을 때. 세션이 invalidate()되기까지 시간이 얼마나 남았는지 알기 위하여 사용할 수도 있습니다.
invalidate()	세션을 종료합니다. 이 작업에는 현재 세션에 저장된 모든 세션 속성을 제거하는(unbind) 작업이 포함됩니다. (이 장 후반에 자세한 설명이 나옵니다)	클라이언트가 비활성화이거나, 세션 작업이 완료되어 강제로 세션을 종료할 때(예를 들면 장바구니 결제를 완료했을때). 세션 객체 자체는 컨테이너가 내부적으로 재사용합니다. 여기에 대해선 여러분이 신경쓸 필요가 없죠. invalidate()는 세션 ID가 더 이상 존재하지 않으니, 관련 속성을 세션 객체에서 제거하라는 의미입니다.