Javascript 키보드 이벤트

키워드 : 키보드 이벤트 (https://www.w3schools.com/jsref/obj_keyboardevent.asp)

이번 장에서는 지난 장에 이어서 웹 사이트에 방문한 클라이언트가 가장 많이 할 수 있는 액션 중 하나인, 키보드로 할 수 있는 행동에 대한 이벤트 처리에 대해서 다루도록 하겠습니다.

키보드 이벤트

먼저, 이번 절에서는 사용자의 키보드 액션에 대한 다양한 처리를 할 수 있는 키보드 이벤트에 대해서 알아보겠습니다.

메서드	의미
onblur	<input> 태그에서 초점(focus)이 벗어날 때 발생하는 이벤트
onchange	<input> 태그의 내용이 변경될 때 발생하는 이벤트
onfocus	<input>태그의 초점(focus)이 생길 때 발생하는 이벤트
onselect	<input>태그의 텍스트를 선택할 때 발생하는 이벤트
onsubmit	submit 버튼을 눌렀을 때 발생하는 이벤트
onreset	reset 버튼을 눌렀을 때 발생하는 이벤트
onkeydown	키보드를 눌렀을 때(혹은, 누르고 있을 경우) 발생하는 이벤트
onkeypress	키보드를 눌렀을 때(혹은, 누르고 있을 경우) 발생하는 이벤트
onkeyup	키보드를 눌렀다가 땔 때 발생하는 이벤트

키보드 이벤트도 마우스 이벤트와 마찬가지로, 다양한 이벤트가 존재합니다. 위의 내용을 어렵지 않게 이해하실 수 있을 거라 생각합니다. 다만, 한 가지 주의깊게 보셔야할 부분이 있습니다. 바로 이벤트가 발생하는 순서입니다.

아래 코드는 키보드를 누를 때 발생하는 이벤트 발생 순서가 어떻게 동작하는지 확인할 수 있는 예시를 나타냅니다.

```
//1번코드
function log(event){
    console.log(event.type);
}
//2번코드
window.addEventListener("keyup", log, false);
window.addEventListener("keypress", log, false);
window.addEventListener("keydown", log, false);
```

우선 1번 코드는 발생한 이벤트의 객체 정보를 반환할 수 있는 event란 인자를 사용하고 있습니다. 여기서 type이라는 속성을 이용하면 어떤 이벤트가 발생했는지 확인할 수 있습니다.

2번 코드는 최상위 객체인 window 객체에 addEventListener 메서드로 keyup, keypress, keydown 에 대한 이벤트를 전역(Global)으로 등록하고 있습니다.

위 테스트 코드를 실행하고, 키보드를 클릭하면 아래와 같은 결과를 개발자 도구 콘솔로 확인할 수 있습니다.

```
keydown -> 1번
keypress -> 2번
keyup -> 3번
```

위 결과에서 알 수 있듯이, 키보드 이벤트의 순서는keydown 이벤트가 먼저 발생하고, keypress 이벤트가 나중에 발생합니다. 마지막으로, 키보드 자판을 눌렀다가 땔 경우에 keyup 이벤트가 발생합니다. 마찬가지로, 특정 키를 꾹 누르고(holding) 있어도 아래와 같은 형태로 콘솔에 나타나게 됩니다.

```
keydown
keypress
keydown
keypress
...
```

위 내용과 별개로 keydown 이벤트와 keypress 이벤트는 한 가지 큰 차이점이 존재합니다. 바로 기능키에서 차이가 있습니다. 여기서 기능키란 일반적으로, `ctrl`, `shift`, `alt` 등이 해당합니다. (Mac OS의 경우, `cmd`, `shift`, `opt`)

결론부터 말하면, 기능키를 누르는 이벤트의 경우엔 keydown과 keyup 이벤트만 발생하게 됩니다.

```
keydown
keyup
keydown
keyup
...
```

그렇기 때문에, 기능키에 대한 처리가 필요한 경우엔 keypress 이벤트로는 처리가 불가능하기 때문에, keydown 이벤트로 처리해주어야 합니다.

키보드 속성 & 메서드

키보드 이벤트는 이벤트 객체를 통해 다양한 키보드 관련과 속성을 반환받고, 활용할 수 있습니다.

속성/메서드	의미
altKey	`Alt`키가 눌렸는지 반환
charCode	눌려진 문자의 유니코드 값을 반환(keydown)
code	눌려진 키의 코드를 문자열로 반환
ctrlKey	`Ctrl`키가 눌렸는지 반환
getModifierState("key")	특정 "key"가 활성화 되어있는지 반환
key	눌려진 키의 대표 이름을 반환
keyCode	눌려진 키의 유니코드 값을 반환 (keydown, keyup)
location	키가 눌려진 위치 반환(e.g., 왼쪽 `ctrl` 키와 오른쪽 `ctrl` 키는 서로 다른 값을 반환)
metaKey	`meta`키가 눌렸는지 반환(e.g., `Mac`: 커멘드 키, `Windows`: 윈도우 키)
repeat	반복적으로 키가 눌려졌는지 반환
shiftKey	`Shift`키가 눌렸는지 반환
which	눌려진 키의 유니코드 값을 반환 (keydown, keyup)

위의 표에 보이듯, 키보드 이벤트와 관련된 다양한 속성과 메서드가 있습니다. 단, 모든 브라우저에서 정상적으로 동작하진 않는다는 점도 함께 기억해주세요. 여기선 유용하게 사용될 수 있는 몇 가지만 정리해보도록 하겠습니다.

다음 절에서 알아볼 속성은 모두 읽기 전용(Read-only)입니다. 즉, 아래와 같이 값을 변경하는 코드는 불가합니다.

```
event.which = "key"; //불가
event.keyCode = "key"; //불가
```

keyCode(Firefox 에선 정상 동작 불가)

이번 절에선, 위 여러 가지 속성 중에 keyCode 속성에 대해서 알아보도록 하겠습니다. keyCode 속성은 keypress 이벤트가 발생했을 때 키에 해당하는 숫자 코드를 Unicode character code로 반환해줍니다. 반대로, keydown, keyup에 대응하는 이벤트는 Unicode key code라는 것을 반환합니다. 말이 조금 어려울 수 있는데요, 하나하나 알아보도록 하겠습니다.

- Unicode character code : 눌려진 키에 대응하는 아스키(ASCII) 값 반환
- Unicode key code : 눌려진 키에 실제 대응하는 값 반환

위에 나와있는 아스키코드는 영문 알파벳을 사용하는 대표적인 문자 인코딩입니다. 쉽게 생각해서, 영어 알파벳 대소문자 각각에 대응하는 숫자를 부여한 것이라고 할 수 있습니다.

영화 마션에서 주인공 멧 데이먼이 화성에 홀로 갇혔을 때, 지구와 조금 더 빠르고, 원활하게 통신하기 위해 사용했던 부호이기도 합니다. 조금 뜬금없죠..?^^

아무튼, 우리가 알고 있는 알파벳 문자 집합 중에 기억하고 있으면 좋은 아스키코드 값이 있습니다.

- 알파벳 소문자 a: 97
- 알파벳 대문자 A: 65
- 숫자 0: 48

위의 아스키코드 값은 문자열 처리에 있어서 기준이 될 수 있는 중요한 값입니다. 예를 들어, 알파벳 소문자 a는 97부터 시작하여 소문자 b는 98, 소문자 c는 99... 식으로 값을 표현할 수 있습니다. 마찬가지로, 숫자 0은 48부터 시작하여 숫자 1은 49, 숫자 2는 50... 식으로 값을 표현할 수 있습니다.

더 자세한 유니코드와 관련된 내용은 [여기 (https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C)](https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C)를, 아스키코드와 관련된 내용은 [여기 (https://ko.wikipedia.org/wiki/ASCII)](https://ko.wikipedia.org/wiki/ASCII) 참고해주세요.

다시 돌아와서, keyCode 속성은 keypress 이벤트가 발생하면 아스키코드로 값을 반환한다고 말씀드렸습니다. 여기엔 한 가지 문제점이 있습니다. 바로 영문 대소문자를 구분하지 못하는 점입니다. 예를 들어, 아스키코드 값으로 소문자 a는 97이고, 대문자 A는 65입니다. 하지만, keyCode 값은 두 값 모두 소문자 a의 값인 97이 나오게 됩니다. (모든 문자 키가 동일합니다.)

또한, 사용자가 기능키를 사용하는 경우에 발생시켜야 하는 이벤트는 위에서 알아보았듯이, keydown과 keyup 이벤트에 대응하도록 해야 합니다. e.g., Home, F1, etc.

참고로, 최신 버전의 DOM 이벤트를 제공하는 브라우저에서는 가능하다면 위 코드 대신 key 속성을 사용하는 것을 권장하기도 합니다.

위의 내용은 이벤트에 따른 특정 키의 동작을 위해 어떤 개념들이 나오는지 설명하기 위해 작성된 글이라고 생각해주시고, key 속성을 직접 사용해보세요. :)

```
function log(e){
    console.log("keyCode: ",e.keyCode);
    console.log("key: ", e.key);
}

window.addEventListener('keypress', log, false);
```

getModifier(Safari 10 & IE 8.0 이하 버전에선 정상 동작 불가)

키보드의 키 중에는 Modifier 키라는 것이 존재합니다. 이 키는 '활성화'(activated)와 '비활성화'(deactivated) 두 가지 상태를 가질 수 있는 키를 말합니다. 활성화 상태를 가질 수 있는 키는 크게 세 가지 종류가 있습니다.

- CapsLock
- NumLock
- ScrollLock

키보드 이벤트가 발생할 때, getModifier("key") 메서드를 이용하면 위 키가 활성화(true) 되었는지 아닌지(false) 확인할 수 있습니다. 위 키 중에서 영문 대소문자 구분을 위해 많이 사용되는 CapsLock 키를 예로 들어보겠습니다.

아래 코드는 test라는 아이디 속성 값을 갖는 <input> 태그가 html 코드에 짜여있다고 가정합니다.

```
document.getElementById("test").addEventListener("keydown", function(e){
    var isCapsLock = e.getModifier("CapsLock");
    if(isCapsLock){
        //CapsLock 키가 활성화 상태
    }else{
        //CapsLock 키가 비활성화 상태
    }
}, false);
```

위 코드의 내용은 어렵지 않게 이해하고 활용할 수 있을 거라 생각합니다. 더 많은 키보드 이벤트 속성과 메서드에 대한 내용은 여기 (https://www.w3schools.com/jsref/obj_keyboardevent.asp)를 참고해주세요. :)

다른 파트도 마찬가지지만, 이벤트와 관련된 내용은 굉장히 방대합니다. 이벤트와 관련된 성능 이슈(효율적인 메모리 관리)에 대한 내용도 중요하며, 어떤 식으로 사용자 액션을 처리할 것인지를 결정하는 것도 시간이 오래 걸리는 작업입니다.

프론트엔드 개발에 있어서 UI&UX에 대한 이야기를 많이 하게 되는데요. 디자인부터 시작해서, 사용자의 접근성을 높이기 위한 방법은 상당히 많이 존재합니다. 이벤트를 잘 활용해서 웹 페이지에 방문한 사용자의 접근성을 높이기 위한 고민을 해보세요! :)