폼과 폼 조작

폼 프로퍼티와 메서드

- 폼과 관련된 요소(form, input, select 등등)에는 해당 요소만을 위한 특별한 속성과 이벤트가 존재함
- 문서에 삽입된 폼은 특수한 콜렉션인 **document.forms 콜렉션에 포함**됨
  - document.forms는 **이름과 순서가 있는 기명 컬렉션(named collection)**으로 개발자는 다음과 같이 이름이나 순서를 사용해 문서 내의 폼에 접근할 수 있음
  - 물론 기존에 사용하던 요소 조회 메서드(querySelector 등)를 이용해서도 접근 가능

```
document.forms.my
=> 이름 속성값이 "my"인 폼 => <form name="my"></form>

document.forms[0]
=> 문서 내의 첫 번째로 등장하는 폼
```

form_manipulation1.html

```
<form name="my">
    <h1>h1</h1>
    <input name="one" value="1" /><br />
    <p>p</p>
    <input name="two" value="2" /><br />
    <div>
        <input type="radio" name="age" value="10" checked />
        <input type="radio" name="age" value="20" />
    </div>
</form>
<script>
let form = document.forms.my;
// 기존 조회 메서드 활용 가능
from = document.querySelector('form[name="my"]');
// 폼 내용 전송과 관련된 요소만 elements 콜렉션에 포함됨을 유의!
console.log(form.elements);
// children은 모든 요소 포함
console.log(form.children);
// <input name="one"> 요소
let elem = form.elements.one;
console.log(elem.value); // 1
// 두 개의 라디오버튼이 포함된 콜렉션
console.log(form.elements.age);
for(let age of form.elements.age) {
    console.log(age.checked, age.value);
}
// 짧은 표기법으로 접근 가능
console.log(form.one, form.two, form.age[0]);
// 역으로 폼 참조 가능
console.log(form.one.form, form.two.form, form.age[0].form);
```

```
</script>
```

- 이름이나 순서를 사용해 원하는 폼을 가져온 다음에는 기명 컬렉션 elements를 통해서 **폼에 포함된 폼 관련 요소**를 얻어올 수 있음 (폼안에 포함된 모든 요소가 아님을 유의)
- 개발을 하다 보면 이름이 같은 요소 여러 개(ex: 라디오버튼)를 다뤄야 하는 경우가 생기기도 하는데 이 경우 elements[name]은 컬렉션이 된다는 사실을 이용하면 됨
- 폼 요소 탐색에 쓰이는 프로퍼티는 **태그 구조에 의존하지 않으며** 폼을 조작하는 데 쓰이는 요소들은 모두 태그 깊이에 상관없이 elements 콜렉션을 통해서 접근 가능
    - 코드에서 라디오 버튼은 모두 div에 포함되어있지만 바깥에서 elements를 통해 참조할 때에는 부모 요소 div에 대해서 상관하지 않아도 됨
- 짧은 표기법인 form[index 혹은 name]으로도 요소에 접근할 수 있음
    - 즉, form.elements.login 대신 form.login처럼 접근이 가능
- 폼이 내부에 있는 요소 모두를 참조할 수 있듯이, 각 요소 또한 **역으로 폼을 참조**할 수 있음

폼 요소

input과 textarea

- input과 textarea 요소의 값은 **value 속성또는 checked 속성**을 통해서 얻을 수 있으며 변경시 해당 속성에 값을 대입

```
// textContent나 innerHTML이 아님을 주의
input.value = "New value";
textarea.value = "New text";
// checked 속성 통해서 체크 여부 확인 혹은 상태 변경 가능
input.checked = true;
```

select와 option

- select 요소에는 세 가지 중요 프로퍼티가 존재함

```
1. select.options       - <option> 하위 요소를 담고 있는 컬렉션
2. select.value         - 현재 선택된 <option> 값
3. select.selectedIndex - 현재 선택된 <option>의 번호(인덱스)
```

- 세 프로퍼티를 응용하면 아래와 같은 세 가지 방법으로 select 요소의 값을 설정 가능

```
1. 조건에 맞는 <option> 하위 요소를 찾아 option.selected속성을 true로 설정
2. select.value를 원하는 값으로 설정
3. select.selectedIndex를 원하는 option 번호로 설정
```

- Option 생성자 사용하여 option 요소 생성 가능

```
option = new Option(text, value, defaultSelected, selected);
```

생성자로 전달할 매개변수의 의미

```
text            - option 내부의 텍스트
value           - option의 값
defaultSelected - true이면 HTML 속성 selected가 생성됨
selected        - true이면 해당 option이 선택됨
```

form_manipulation_2.html

```
<select id="select">
    <option value="apple">Apple</option>
    <option value="pear">Pear</option>
    <option value="banana">Banana</option>
</select>
<script>
// 세 가지 코드의 실행 결과는 모두 같습니다.
// 1. 조건에 맞는 <option> 하위 요소를 찾아 option.selected속성을 true로 설정
select.options[2].selected = true;
// 2. select.value를 원하는 값으로 설정
select.value = 'banana';
// 3. select.selectedIndex를 원하는 option 번호로 설정
select.selectedIndex = 2;

// 밑의 코드로 <option value="melon">Melon</option> 요소 생성 가능
let option = new Option("Melon", "melon");
select.append(option);
</script>
```

focus와 blur

- 사용자가 폼 요소를 클릭하거나 탭 키를 눌러 요소로 이동하게 되면 **해당 요소가 포커스(focus)**됨
    - autofocus라는 HTML 속성을 사용해도 요소를 포커스 할 수 있으며, 이 속성이 있는 요소는 **페이지가 모두 로드된 후 자동으로 포커싱** 됨
- 사용자가 다른 곳을 클릭하거나 탭 키를 눌러서 다음 입력 필드로 이동하면 포커스 상태의 요소가 **포커스를 잃게 됨(blur)**

focus, blur 이벤트

- **focus 이벤트는 요소가 포커스를 받을 때, blur 이벤트는 포커스를 잃을 때 발생**함
- 대체로 두 이벤트는 **입력 필드 값의 검증 과정에서 사용**됨

form_validation.html

```
<style>
.invalid { border-color: red; }
#error { color: red }
</style>
<body>
    <form>
        <label>이메일</label>
        <input type="email" id="input" />
        <div id="error"></div>
    </form>
```

```
</body>
<script>
input.onblur = function() {
    // @ 유무를 이용해 유효한 이메일 주소가 아닌지 검증
    if(!input.value.includes('@')) {
        input.classList.add('invalid');
        error.innerHTML = '올바른 이메일 주소를 입력하세요.'
    }
};

input.onfocus = function() {
    if(this.classList.contains('invalid')) {
        // 사용자가 새로운 값을 입력하려고 하므로 에러 메시지를 지움
        this.classList.remove('invalid');
        error.innerHTML = "";
    }
};
</script>
```

- 모던 HTML에서는 **required, pattern과 같은 속성을 사용해 입력값을 검증**할 수 있음
    - pattern 속성의 경우 **정규표현식**을 입력받아 문자열 패턴을 검증함
        - https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/pattern
- **focus와 blur 메서드**를 사용하여 요소에 강제로 포커스를 주거나 제거할 수 있음

focus_and_blur_method.html

```
<style>
input:focus {
    background: yellow;
}
</style>
<body>
    <form>
        <input type="text" id="input" />
    </form>
    <button>focus</button>
</body>
<script>
let input = document.forms[0].children[0];
let focusBtn = document.body.children[1];

input.onfocus = function() {
    // 3초 후 포커스 잃게 하기
    setTimeout(function() {
        input.blur();
    }, 3000)
}
focusBtn.onclick = function() {
    input.focus();
}
</script>
```

이벤트: change, input, cut, copy, paste

change, input 이벤트

- change 이벤트는 기존 입력값이 변경된 이후, **포커스를 잃은 시점**에 발생
- input 이벤트는 어떠한 경우이건 **입력값이 변경되었으면 무조건 실행**

change_input_event.html

```
<body>
    <form>
        <input type="text" id="onchangedemo" /><br />
        <input type="text" id="oninputdemo" /><br />
        <select id="onchangedemo2">
            <option value="">선택하세요.</option>
            <option value="1">옵션 1</option>
            <option value="2">옵션 2</option>
            <option value="3">옵션 3</option>
        </select>
    </form>
</body>
<script>
// 텍스트 입력창에 포커스가 없어질 때 이벤트 핸들러가 실행됨
onchangedemo.onchange = function(e) {
    console.log("onchangedemo.onchange", this.value);
};
// 단, select 요소의 경우에는 선택지가 선택되는 시점에 바로 실행됨
// (당연히 이전과 같은 옵션을 선택한 경우에는 실행되지 않음)
onchangedemo2.onchange = function(e) {
    console.log("onchangedemo2.onchange", this.value);
};
// 값이 조금이라도 변경되면 무조건 실행됨
oninputdemo.oninput = function(e) {
    console.log("oninputdemo.oninput", this.value);
};
</script>
```

이벤트: cut, copy, paste

- cut, copy, paste 이벤트는 각각 값을 잘라내기, 복사하기, 붙여넣기 할 때 발생함

cut_copy_paste_event.html

```
<input type="text" id="input">
<script>
// cut      - Ctrl + X
// copy     - Ctrl + C
// paste    - Ctrl + V
input.oncut = input.oncopy = input.onpaste = function(event) {
    // paste 이벤트인 경우에만 클립보드 데이터가 존재함
    alert(event.type + ' - ' + event.clipboardData.getData('text/plain'));
    return true;
};
</script>
```

submit 이벤트와 메서드

- **submit 이벤트는 폼을 제출할 때 트리거** 되는데, 주로 폼을 서버로 전송하기 전에 **내용을 검증하거나 검증 결과에 따라 폼 전송을 취소**할 때 사용함

폼을 전송하는 두 가지 방법

1. <input type="submit"> 요소나 <input type="image"> 요소를 클릭하기
2. 인풋 필드에 포커스가 잡힌 상태에서 엔터 키 누르기

form_submit_event.html

```
<body>
    <form method="GET" action="user_validate">
        <label>username</label><input type="text" name="username" /><br />
        <label>password</label><input type="password" name="password" /><br />
        <input type="submit" value="제출" />
    </form>
</body>
<script>
let form = document.forms[0];
form.onsubmit = function(e) {
    // 기본 동작(페이지 이동 및 폼 내용 제출) 막기
    e.preventDefault();

    let method = this.method;
    let action = this.action;
    let username = this.username.value;
    let password = this.password.value;
    console.log(action ,method, username, password);

    /* 폼 내용에 근거하여 ajax 요청을 보내는 코드 작성 가능 */
}
</script>
```

submit 메서드 (@)

- 폼 요소 객체에서 제공하는 **submit 메서드는 자바스크립트 코드만으로 폼 전송을 하고자 할 때 사용**됨
    - 동적으로 폼을 생성하고 submit 메서드를 호출하여 서버에 폼 내용을 전송 가능
    - 단, onsubmit에 등록한 이벤트 핸들러 함수를 통해서도 **기본 폼 전송 동작을 취소하고 ajax 요청을 통해 서버에 폼 내용 전송이 가능**하므로 잘 쓰이지 않음

form_submit_method.html

```
<body>
</body>
<script>
// 동적으로 폼 요소 생성
let form = document.createElement('form');

form.action = 'https://google.com/search';
```

```
form.method = 'GET';

form.innerHTML = '<input name="q" value="테스트">';
// 폼을 제출하려면 반드시 폼이 문서 안에 있어야하므로, 문서안에 붙이되 폼을 보이지 않게 함
form.hidden = true;
document.body.append(form);

// 1초 뒤에 코드를 통해 submit 작업 수행
setTimeout(function() {
    form.submit();
}, 1000);
</script>
```