3. 자바스크립트를 활용한 함수형 프로그래밍 (2)

Prof. Seunghyun Park (sp@hansung.ac.kr)

Division of Computer Engineering

한성대학교 HANSUNG UNIVERSITY

학습 목표: 3장. 자바스크립트를 활용한 함수형 프로그래밍

- 함수형 프로그래밍
  - 함수
    - 1급 객체와 고차 함수
  - 함수형 프로그래밍 특징
    - 명령형 프로그래밍과 선언적 프로그래밍 비교
    - 불변성
    - 순수 함수
    - 데이터 변환
    - 고차 함수
    - 재귀
    - 합성

순수 함수 (pure functions, 계속) side effect : 객체의 변화를 바로 알 수 없다

```
/* ch03/05/01-pure-functions.html */

var frederick = {
  name: "Frederick Douglass",
  canRead: false,
  canWrite: false
}

function selfEducate() {
  frederick.canRead = true
  frederick.canWrite = true
}

selfEducate()

console.log( frederick )
```

순수하지 않은 함수
1) 매개변수 없음
2) 반환 값 없음
3) 함수 밖 객체의 값 (속성)을 변경

side effect 발생

```
{name: 'Frederick Douglass', canRead: true, canWrite: true}
```

- 매개변수에 의해서만 반환 값이 결정되는 함수
  > **데이터 원본은 수정되지 않음**

```
/* ch03/05/02-pure-functions.html */

var frederick = { (생략) }

const selfEducate = person => {
  person.canRead = true
  person.canWrite = true
  return person
}

console.log( selfEducate(frederick) )
console.log( frederick )
```

순수하지 않은 함수
- 매개변수와 반환 값은 존재하나,
1) 매개변수로 전달된 함수 밖 객체의 (속성) 값을 변경

side effect 발생

```
{name: 'Frederick Douglass', canRead: true, canWrite: true}
{name: 'Frederick Douglass', canRead: true, canWrite: true}
```

순수 함수 (pure functions, 계속)

```
/* ch03/05/03-pure-functions.html */

var frederick = {
    name: "Frederick Douglass",
    canRead: false,
    canWrite: false
}

const selfEducate = person =>
    ({
        ...person,
        canRead: true,
        canWrite: true
    })

console.log( selfEducate(frederick) )
console.log( frederick )
```

side effect 를 막으려면

1) 새로운 객체 생성
2) 스프레드 연산자 ... 활용
 - 매개변수로 전달받은 객체의 모든 요소 나열
3) 같은 이름의 element 값 수정
4) 복사본에 작업하고 결과 반환 (원본 유지)

순수 함수

1) 매개변수 전달
2) 결과 값 반환
3) 함수 밖 객체의 값을 변경하지 않음

→ side effect 발생하지 않음

```
{name: "Frederick Douglass", canRead: true, canWrite: true}
{name: "Frederick Douglass", canRead: false, canWrite: false}
```

순수 함수 (pure functions)

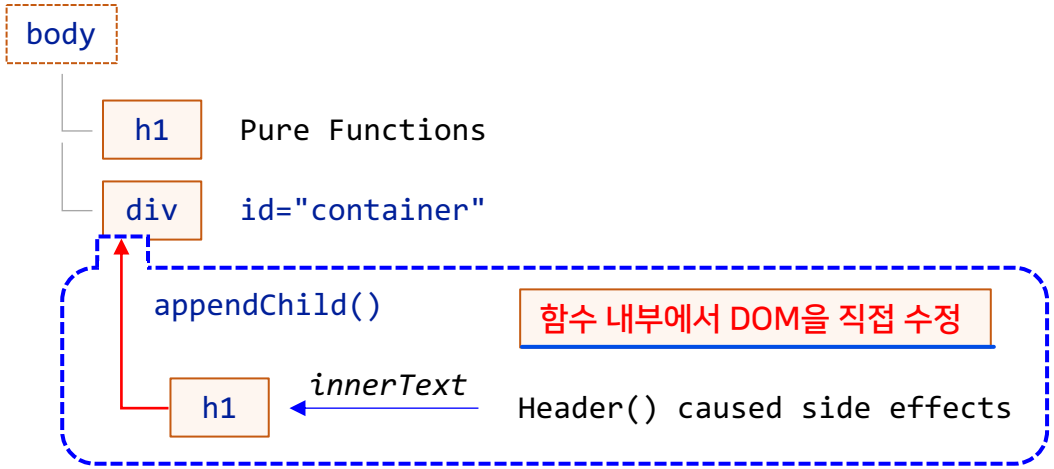
```
/* ch03/05/04-2-pure-functions.html */
function Header(text) {
    const h1 = document.createElement('h1')
    h1.innerText = text
    const div = document.getElementById("container")
    div.appendChild(h1)
}

Header("Header() caused side effects")
```

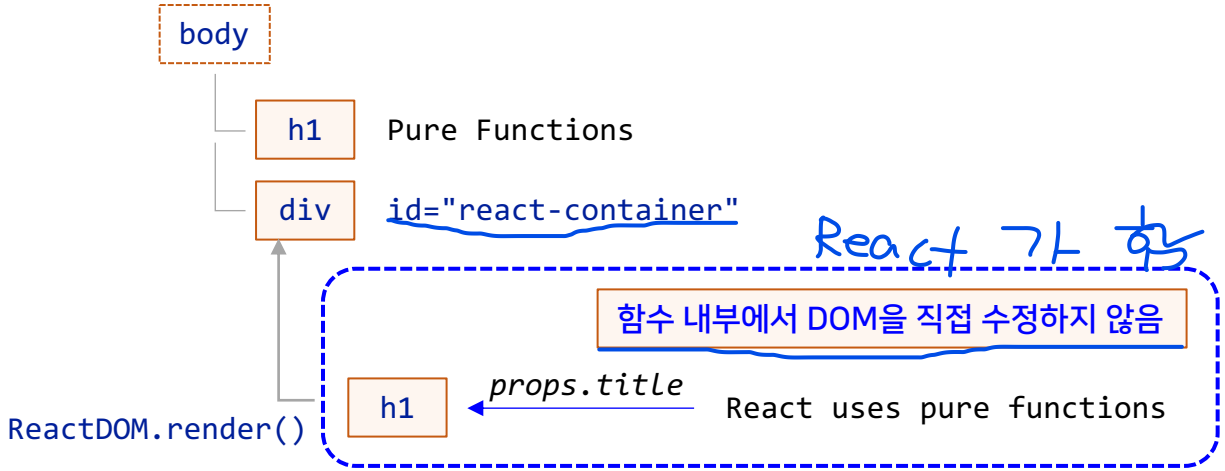
함수 내부에서 h1 element를 생성하였으나, DOM을 변화시킴 → side effect 발생

```
/* ch03/05/05-pure-functions.html */
const Header = (props) => <h1>{props.title}</h1>

ReactDOM.render(
    <Header title="React uses pure functions" />,
    document.getElementById('react-container')
)
```

Header():
- props를 매개변수로 받아서 새로운 <h1> 객체를 반환
- DOM에 직접적인 변화시키지 않음

Header 컴포넌트를 element에 렌더링





데이터 변환 (계속)

- 순수 함수를 사용한 데이터 변경
  - 원본의 복제본을 생성하여 처리하고 결과를 반환

```
/* ch03/06/01-data.html */

const schools = [ "Yorktown", "Washington & Lee", "Wakefield" ]
console.log( schools )

console.log( schools.join(", ") )
console.log( schools )
```

배열의 모든 원소를 연결하여 하나의 문자열로 반환
원본은 수정하지 않음

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
Yorktown, Washington & Lee, Wakefield
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

- `Array.prototype.join()`
  - 배열의 모든 요소를 연결해 하나의 문자열로 만들어 반환

데이터 변환 (계속)

```
const schools = [ "Yorktown", "Washington & Lee", "Wakefield" ]
```

```
/* ch03/06/02-data.html */
console.log( schools )

const wSchools = schools.filter(sch => sch[0] === "W")
console.log( wSchools )

console.log( schools )
```

true 배열

return true/false

schools 배열의 원소 중
첫 글자가 W인 원소만 필터링하여 wSchool에 할당
원본은 수정하지 않음

- Array.prototype.filter(*callback*)
  - 주어진 함수의 테스트를 통과하는 모든 요소를 모아 새로운 배열로 반환

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
(2) ['Washington & Lee', 'Wakefield']
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

wSchool

```
/* ch03/06/03-data.html */
console.log(schools)

const cutSchool = (cut, list) => list.filter(sch => sch !== cut)

console.log(cutSchool("Washington & Lee", schools).join(", "))
console.log(schools)
```

조건: 배열에서 매개변수로 전달된 요소를 제외하도록 필터링

조건에 따라 필터링 한 결과를 반환하고,
각 요소를 ,으로 연결하여 출력
원본은 수정하지 않음

schools

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

Yorktown, Wakefield

cutSchool() 결과

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

schools

데이터 변환 (계속)

```
const schools = [ "Yorktown", "Washington & Lee", "Wakefield" ]
```

- Array.prototype.map(*callback*)
  - 배열 내 모든 요소에 대해
    주어진 함수를 호출한 결과를 모아 새로운 배열로 반환

```
/* ch03/06/04-data.html */
console.log(schools)
const highSchools = schools.map(sch => `${sch} High School`)

console.log(highSchools.join("\n"))
console.log(schools)
```

배열의 모든 요소에 지정된 문자열을 추가하고
연산 결과를 highSchools로 반환
원본은 수정하지 않음

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
Yorktown High School
Washington & Lee High School
Wakefield High School
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

highSchools

```
/* ch03/06/05-data.html */
console.log(schools)

const highSchools = schools.map(sch => ({ name: sch }))
console.log(highSchools)

console.log(schools)
```

배열의 모든 요소에 대하여,
키가 name이고 값이 위 배열의 요소인 객체를 각각 생성하여
배열 highSchools의 요소로 할당
원본은 수정하지 않음

```
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
(3) [{…}, {…}, {…}]
  0: {name: 'Yorktown'}
  1: {name: 'Washington & Lee'}
  2: {name: 'Wakefield'}
  length: 3
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
```

schools
highSchools
schools

데이터 변환 (계속)

```
/* ch03/06/06-data.html */
let schools = [
  { name: "Yorktown"},
  { name: "Stratford" },
  { name: "Washington & Lee"},
  { name: "Wakefield"}
]
const editName = (oldName, name, arr) =>
  arr.map(item => {
    if (item.name === oldName) {
      return {
        ...item,
        name
      }
    } else {
      return item
    }
  })

console.log( schools )

let updatedSchools = editName("Stratford", "HB Woodlawn", schools)
console.log( updatedSchools )

console.log( schools )
```

매개변수로 전달받은 배열의 모든 요소 (각 객체)에 대하여
객체의 name 속성의 값과 전달받은 매개변수1를 비교하여
- 같으면, **객체의 키가 name이고 값이 매개변수2인 객체**를
- 다르면, 현재의 객체를
배열 highSchools의 요소로 할당하여 반환
원본은 수정하지 않음

```
(4) [{…}, {…}, {…}, {…}]
  0: {name: 'Yorktown'}
  1: {name: 'Stratford'}
  2: {name: 'Washington & Lee'}
  3: {name: 'Wakefield'}
  length: 4
  [[Prototype]]: Array(0)
```
schools

```
(4) [{…}, {…}, {…}, {…}]
  0: {name: 'Yorktown'}
  1: {name: 'HB Woodlawn'}
  2: {name: 'Washington & Lee'}
  3: {name: 'Wakefield'}
  length: 4
  [[Prototype]]: Array(0)
```
updatedSchools

```
(4) [{…}, {…}, {…}, {…}]
  0: {name: 'Yorktown'}
  1: {name: 'Stratford'}
  2: {name: 'Washington & Lee'}
  3: {name: 'Wakefield'}
  length: 4
  [[Prototype]]: Array(0)
```
schools

데이터 변환 (계속)

```
/* ch03/06/06-data.html */

const editName = (oldName, name, arr) =>
  arr.map(item => {
    if (item.name === oldName) {
      return {
        ...item,
        name
      }
    } else {
      return item
    }
  })
```

item은 { name: "..." } 형태의 객체로 schools의 요소를 탐색

{ name: name } 은 { name }으로 축약 가능

```
/* ch03/06/07-data.html */

const editName = (oldName, name, arr) =>
  arr.map(item => (item.name === oldName) ? {name} : item)

let updatedSchools = editName("Stratford", "HB Woodlawn", schools)
```

3항 연산자를 활용하여 기존 코드 축약 가능
> arr 의 모든 요소를 item으로 탐색

```
// callback에서 2번째 매개변수는 array 요소의 인덱스를 의미
const editNth = (n, name, arr) =>
    arr.map((item, i) => (i === n) ? { name } : item )

let updatedSchools2 = editNth(2, "Mansfield", schools)
```

현재 item의 인덱스 (생략 가능)

데이터 변환 (계속)

```
/* ch03/06/08-data.html */

const schools = {
    "Yorktown": 10,
    "Washington & Lee": 2,
    "Wakefield": 5
}

console.log(schools)

const schoolArray = Object.keys(schools).map(key =>
    ({
        name: key,
        wins: schools[key]
    })
)
console.log(Object.keys(schools))

console.log(schoolArray)
```

- Object.keys(*object*)
  - 주어진 객체의 속성 이름을 열거하는 배열을 반환

원본은 변하지 않는다

```
{Yorktown: 10, Washington & Lee: 2, Wakefield: 5}
  Wakefield: 5
  Washington & Lee: 2
  Yorktown: 10
  [[Prototype]]: Object
(3) ['Yorktown', 'Washington & Lee', 'Wakefield']
  0: "Yorktown"
  1: "Washington & Lee"
  2: "Wakefield"
  length: 3
[[Prototype]]: Array(0)
(3) [{…}, {…}, {…}]
  0: {name: 'Yorktown', wins: 10}
  1: {name: 'Washington & Lee', wins: 2}
  2: {name: 'Wakefield', wins: 5}
  length: 3
  [[Prototype]]: Array(0)
```

schools

Object.keys(schools)

schoolArray

데이터 변환 (계속)

```
/* ch03/06/09-data.html */
const ages = [21,18,42,40,64,63,34]

const maxAge = ages.reduce((max, age, idx) => {
  console.log(`${age} > ${max} = ${age>max}, ${idx}`)
  if (age > max) {
    return age
  } else {
    return max
  }
}, 0)

console.log('maxAge', maxAge)
```

배열 탐색

반환 값을 max에 누적

callback:
- max: *accumulator*
- age: *currentValue*

initValue: 0

*initValue*를 지정하지 않은 경우, 인덱스는 1부터 시작

```
21 > 0 = true, 0
18 > 21 = false, 1
42 > 21 = true, 2
40 > 42 = false, 3
64 > 42 = true, 4
63 > 64 = false, 5
34 > 64 = false, 6
maxAge 64
```

```
18 > 21 = false, 1
42 > 21 = true, 2
40 > 42 = false, 3
64 > 42 = true, 4
63 > 64 = false, 5
34 > 64 = false, 6
maxAge 64
```

- Array.prototype.reduce(*callback*, *initValue*)
  - 배열 내 모든 요소에 대해 *callback*을 실행하고 하나의 결과값 반환

* *callback*: (*accumulatgor*, *currentValue*, *currentIndex*, *array*) => { ... }

- *accumulator*: 콜백의 반환 값 누적 (mandatory)
- *currentValue*: 현재 처리할 요소 (mandatory)
- *currentIndex*: 현재 처리할 요소의 인덱스 (optional)
- *array*: reduce()를 호출한 배열 (optional)

```
/* ch03/06/10-data.html */

const ages = [21,18,42,40,64,63,34];

const max = ages.reduce(
    (max, value) => (value > max) ? value : max,
    0
)

console.log('max', max)
```

callback:
- max: *accumulator*
- value: *currentValue*

initValue: 0

데이터 변환 (계속)

- Array.prototype.indexOf()
  - 배열 내 지정된 요소를 찾을 수 있는 첫 번째 인덱스 반환
  - 존재하지 않으면 -1 반환

```
/* ch03/06/11-data.html */

const colors = [
  {
    id: '-xekare',
    title: "과격한 빨강",
    rating: 3
  },
  {
    id: '-jbwsof',
    title: "큰 파랑",
    rating: 2
  },
]

const hashColors = colors.reduce(
    (hash, {id, title, rating}) => {
        hash[id] = {title, rating}
        return hash
    },
    {}
)

console.log(hashColors)
```

객체가 4개

currentValue

callback

accumulator

initValue

```
Object
  -jbwsof: {title: '큰 파랑', rating: 2}
  -prigbj: {title: '큰곰 회색', rating: 5}
  -ryhbhsl: {title: '바나나', rating: 1}
  -xekare: {title: '과격한 빨강', rating: 3}
  [[Prototype]]: Object
```

```
/* ch03/06/12-data.html */

const colors = ["red", "red", "green", "blue", "green"];

const distinctColors = colors.reduce(
    (distinct, color) =>
        (distinct.indexOf(color) !== -1) ?
            distinct : [...distinct, color],
    []
)

console.log(distinctColors)
```

currentValue

callback

accumulator

initValue

반환 값:
- 배열에 color 요소가 존재하면 인덱스 반환, (-1이 아니므로) 유지
- 존재하지 않으면 -1을 반환, 배열에 해당 요소 추가

```
(3) ['red', 'green', 'blue']
```

원본은 변하지 않는다

고차함수 (계속)

- 고차 함수 (high order function)
  - 함수를 매개변수로 받거나, 함수를 결과로 반환하는 함수

```
/* ch03/07/01-higher-order-fns.html */

const invokeIf = (condition, fnTrue, fnFalse) => (condition) ? fnTrue() : fnFalse()

const showWelcome = () => console.log("Welcome!!!")

const showUnauthorized = () => console.log("Unauthorized!!!")

invokeIf(true, showWelcome, showUnauthorized)
invokeIf(false, showWelcome, showUnauthorized)
```

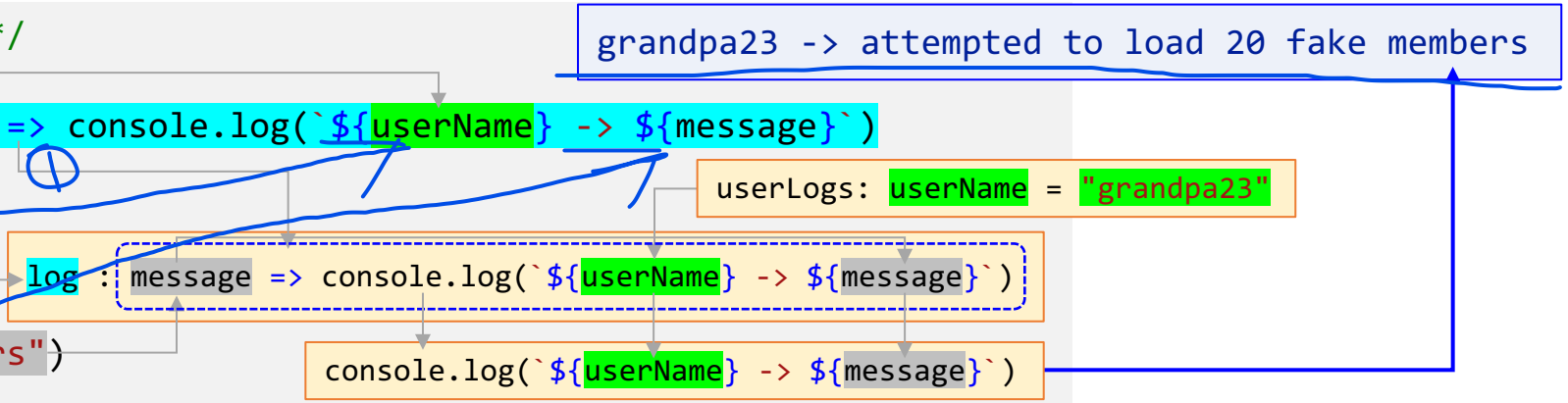
화살표 함수 showWelcome():
매개변수 없이 "Welcome!!!" 문구를 화면에 출력

화살표 함수 showUnauthorized():
매개변수 없이 "Unauthorized!!!" 문구를 화면에 출력

```
Welcome!!!
Unauthorized!!!
```

고차함수 (계속)

```
/* ch03/07/02-higher-order-fns.html */

const userLogs = userName => message => console.log(`${userName} -> ${message}`)

const log = userLogs("grandpa23")

log("attempted to load 20 fake members")

const getFakeMembers = count => new Promise((resolves, rejects) => {
  const api = `https://api.randomuser.me/?nat=US&results=${count}`
  const request = new XMLHttpRequest()
  request.open('GET', api)
  request.onload = () => (request.status === 200) ?
      resolves(JSON.parse(request.response).results) : reject(Error(request.statusText))
  request.onerror = (err) => rejects(err)
  request.send()
})

getFakeMembers(20).then(
    members => log(`successfully loaded ${members.length} members`),
    error => log("encountered an error loading members")
)
```



학습 정리: 3장. 자바스크립트를 활용한 함수형 프로그래밍

- 함수형 프로그래밍
  - 함수
    - 1급 객체와 고차 함수
  - 함수형 프로그래밍 특징
    - 명령형 프로그래밍과 선언적 프로그래밍 비교
    - 불변성
    - 순수 함수
    - 데이터 변환
    - 고차 함수
    - 재귀
    - 합성