JavaScript

Proxy – 김근형 강사

Proxy

- Proxy Object
  - 프록시는 메서드의 기본적인 오퍼레이션(Operation)과 행위(Behavior)를 중간에서 가로채서 이를 대신하는 일련의 처리를 의미한다.
  - 중간에서 가로채기를 하기 위해서는 사전에 약속된 방법을 따라야 한다.
  - 프록시 객체는 오퍼레이션 행위를 바꾸는 것보다 추가하는 측면이 더 강하다.

```
let target = {food: "밥"};
let left = target.food;
console.log(left);
```

밥

```
let target = {food: "밥"};
let middle = new Proxy(target, {});
let left = middle.food;
```

밥

Proxy

- Proxy Object 프로세스 정리
  - let target = {food:"밥"};
    - {food:"밥"} 이란 오브젝트를 생성하여 target 변수에 할당한다. Object 오브젝트에는 빌트인으로 getter([[Get]])가 포함되어 있으며 proxy를 통하지 않고 직접 getter가 호출되는 것은 target 오브젝트에 getter가 있기 때문이다.
  - let middle = new Proxy(target, {});
    - new Proxy() 의 첫번째 파라미터에 target 오브젝트를 지정하여 Proxy 인스턴스를 생성하고 middle에 할당한다. middl은 프록시 인스턴스가 된다.
  - let = middle.food;
    - Proxy 인스턴스(middle)의 getter()를 호출하면서 프로퍼티 이름(food)을 파라미터 값으로 넘겨준다. 현재 Proxy 인스턴스에 getter를 작성하지 않았으므로 디폴트 [[Get]]이 호출된다.

Proxy

- Proxy Object 프로세스 정리
  - Proxy 인스턴스의 getter가 target 오브젝트의 [[Get]]() 을 호출하면서 프로퍼티 이름(food)을 파라미터의 값으로 넘겨준다. Proxy 인스턴스를 생성할 때 파라미터에 target 오브젝트를 지정했으므로 getter로 호출할 대상 오브젝트를 알 수 있다.
  - 대상 오브젝트(target)에서 [[Get]]을 호출하면 프로퍼티 이름(food)에 해당하는 프로퍼티 값("밥")을 반환한다.
  - Proxy 인스턴스가 target 오브젝트에서 반환된 값을 받는다. 다시 자신을 호출한 코드(left = middle.food;)로 값을 반환한다. Proxy 인스턴스의 내부 getter 처리이다.
  - left = middle.food;
    - 반환 받은 값("밥")을 left에 할당한다.

Proxy

- Target
  - target은 Proxy에서 사용하는 용어이다.
  - new Proxy(target)의 첫 번째 파라미터에 지정한 대상 오브젝트를 타깃이라고 한다.

Proxy

- Trap
  - 트랩은 OS에서 사용하는 용어로 실행 중인 프로그램에 이상이 발생했을 때 프로그램을 중단하고 사전에 정의된 제어로 이동하는 동작을 의미한다.
  - 위의 (let left = middle.food;) 형태로 getter가 호출되면 targe의 getter가 호출되기 전에 Proxy의 getter가 중간에서 가로챈다.
  - 이때 Proxy에 getter에 대응하기 위해 작성한 get() 트랩이 호출된다.
  - 핸들러에 트랩을 작성하지 않으면 내부 메서드[[Get]]이 호출된다.
  - 핸들러에 get() 트랩을 작성하여[[Get]]이 호출되는 것을 가로채기 한 것.

Proxy

- Trap

```
let target = {food: "밥"};
let handler = {
  get(target, key){
    return target[key] + ",수저";
  },
  set(target, key){}
};

let middle = new Proxy(target, handler);
let left = middle.food;
console.log(left);
```

밥,수저

Proxy

- Trap
  - ES6는 getter/setter와 같이 기본 오퍼레이션을 위한 13개의 내부 메서드를 제공한다.
  - 기본 오퍼레이션이란 프로퍼티 추출, 할당, 열거, 호출 등을 의미한다.
  - 모든 빌트인 오브젝트에 내부 메서드가 설정되므로 모든 오브젝트에서 기본 오퍼레이션을 수행할 수 있다.
  - 오브젝트에 따라 함수를 호출하는 [[call]]과 인스턴스를 생성하는 [[Constructor]]가 포함되지 않을 수 있다.

Proxy

- Trap

Internal Method	Handler Method	Internal Method	Handler Method
[[Call]]()	apply()	[[HasProperty]]()	has()
[[Construct]]()	constructor()	[[IsExtensible]]()	isExtensible()
[[DefineOwnProperty]]()	defineProperty()	[[OwnPropertyKeys]]()	ownKeys()
[[Delete]]()	deleteProperty()	[[PreventExtensions]]()	preventExtensions()
[[Get]]()	get()	[[Set]]()	set()
[[GetOwnProperty]]()	getOwnPropertyDescriptor()	[[SetPrototypeOf]]()	setPrototypeOf()
[[GetPrototypeOf]]()	getPropertyOf()		

Proxy

- Trap

```
let target = {food: "밥"};
let middle = new Proxy(target, {
  get(target, key){
    return target[key] + ",수저";
  }
});

let left = middle.food;
console.log(left);
```

밥,수저

Proxy Trap

- new Proxy()
  - Proxy 인스턴스를 생성하여 반환한다.

구분	타입	데이터(값)
형태		new Proxy()
파라미터	object	Target, 대상 오브젝트(Object, Array, Function, Proxy…)
	object	Handler, 트랩
반환	Proxy	생성한 Proxy 인스턴스

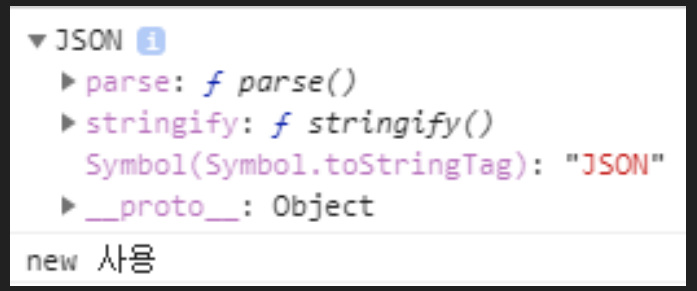
```
let obj = Proxy;
console.log(obj);
```

```
ƒ Proxy() { [native code] }
```

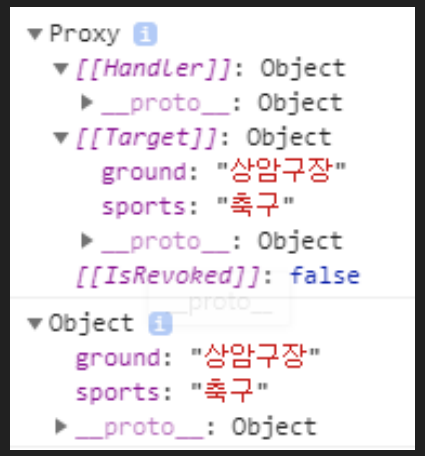
Proxy Trap

- new Proxy()
  - JSON, Math, 글로벌 오브젝트는 new 연산자로 인스턴스를 생성할 수 없다.

```
let json = JSON;
console.log(json);

try {
  let proxyObj = Proxy();
} catch (e) {
  console.log("new 사용");
};
```




```
let target = {ground: "상암구장"};
let newProxy = new Proxy(target, {});

newProxy.sports = "축구";
console.log(newProxy);

console.log(target);
```



Proxy Trap

- set()
  - target 오브젝트에 프로퍼티 키와 값을 설정한다.

구분	타입	데이터(값)
형태		handler.set()
파라미터	object	Target, 대상 오브젝트
	String	Key, 프로퍼티 키
	any	Value, 프로퍼티 값
	object	Recover, Proxy 또는 Proxy를 상속받은 object
반환	boolean	처리성공 true, 아니면 false

Proxy Trap

- set() 준수사항(Invariant)
  - 준수 사항은 set() 트랩에서 지켜야 할 내용이며 이를 지키지 않으면 에러가 발생하거나 정상으로 처리되지 않는다.
    - 트랩의 처리 성공을 나타내려면 true를 반환하고 실패를 나타내려면 false를 반환한다. strict mode에서 false를 반환하면 TypeError가 발생한다.
    - target 오브젝트의 프로퍼티가 데이터 디스크립터일 때, [[Writable]]:false, [[Configurable]]:false 이면 프로퍼티 값을 설정할 수 없다.
    - target 오브젝트의 프로퍼티가 액세서 디스크립터일 때, [[Configurable]]:false 이면 프로퍼티 값을 설정할 수 없다.

Proxy Trap

- set()
  - 핸들러와 트랩을 작성하지 않은 형태 / 핸들러와 트랩을 작성한 형태

```
let target = {event: "축구"};
let sportsProxy = new Proxy(target, {});

sportsProxy.sports = "스포츠";
```

```
let target = {};
let musicProxy = new Proxy(target, {
  set(target, key, value, receiver){
    console.log(target);
    console.log(key, value);
    return true;
  }
});
musicProxy.music = "음악";
console.log(musicProxy.music);
```

```
▼Object
  ▶__proto__: Object
music 음악
undefined
```

Proxy Trap

- set()
  - 핸들러와 트랩을 작성한 형태
    - 아래의 예제에서 key는 music이 value는 음악이 세팅된다

```
let target = {};
let musicProxy = new Proxy(target, {
  set(target, key, value, receiver){
    target[key] = value;
    return true;
  }
});
musicProxy.music = "음악";
console.log(musicProxy.music);
```

```
음악
```

Proxy Trap

- set()
  - 아래 예제에서 targe은 target 오브젝트이며 receiver는 Proxy 인스턴스 이므로 false가 나온다
  - receiver. event는 getter이며 여기서는 getter가 없으므로 [[Get]]이 호출된다

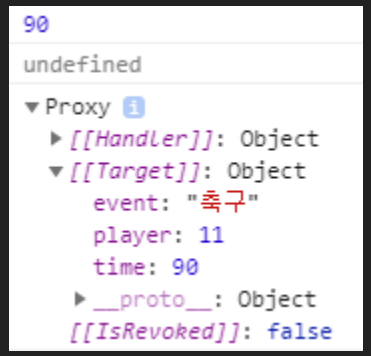
```
let target = {event: "축구"};
let handler = {
  set(target, key, value, receiver){
    console.log(target == receiver);
    console.log(receiver.event);
    return true;
  }
}
let sportsProxy = new Proxy(target, handler);
sportsProxy.sports = "스포츠";
```

```
false
축구
```

Proxy Trap

- set()

```
let target = {event: "축구"};
let newProxy = new Proxy(target, {
  set(target, key, value, receiver){
    target[key] = value;
    console.log(receiver.time);
    console.log(target.time);
    target["time"] = receiver.time;
    return true;
  }
});

let createObj = Object.create(newProxy, {
  time: {value: 90}
});
createObj.player = 11;
console.log(newProxy);
```



Proxy Trap

- this
  - 트랩 안에서 this는 new Proxy()의 두 번째 파라미터인 handler 오브젝트를 참조한다.

```
let target = {event: "축구"};
let handler = {
  ground: "상암구장",
  set(target, key, value, receiver){
    console.log(this.ground);
    this.home = "서울";
    return true;
  }
};

let sportsProxy = new Proxy(target, handler);
sportsProxy.sports = "스포츠";

console.log(handler.home);
```

```
상암구장
서울
```

Proxy Trap

- get()
  - target 오브젝트에서 프로퍼티 값을 반환한다.

구분	타입	데이터(값)
형태		handler.get()
파라미터	object	Target, 대상 오브젝트(Object, Array, Function, Proxy…)
	String	Key, 프로퍼티 key
	object	receiver, Proxy 또는 Proxy를 상속받은 object
반환	any	반환 값

Proxy Trap

- get() 준수사항
  - 프로퍼티가 데이터 디스크립터일 때.
    - [[Writable]]:false,[[Configurable]]:false 이면 target 오브젝트의 프로퍼티 값을 변경하여 반환할 수 없으며 값을 그대로 반환해야 한다.
  - 프로퍼티가 엑세서 디스크립터일 때
    - [[Configurable]]: false 이면 target 오브젝ㄱ트의 프로퍼티 값을 변경하여 반환할 수 없다.

```
let sports = {soccer: "축구"};
let handler = {
  get(target, key, receiver){
    return target[key] + ",11명";
  }
}
let sportsProxy = new Proxy(sports, handler);
console.log(sportsProxy.soccer);

let desc = Object.getOwnPropertyDescriptor(sports, "soccer");
console.log(desc);
```

```
축구,11명
▼Object
    configurable: true
    enumerable: true
    value: "축구"
    writable: true
  ▶__proto__: Object
```

Proxy Trap

- get()
  - createObj 인스턴스 프로퍼티에서 sports 프로퍼티를 검색한다
  - 프로퍼티가 존재하면 createObj 인스턴스의 [[Get]]()을 호출하여 프로퍼티 값을 반환한다
  - 프로퍼티가 존재하지 않으면 상속받은 newProxy가 있는 __proto__ 에서 sports 프로퍼티를 검색한다.
  - __proto__에 sports 프로퍼티가 존재하므로 newProxy의 get() 트랩을 호출하여 프로퍼티 값을 반환한다.

```
let hobby = {sports: "스포츠", music: "음악"};
let newProxy = new Proxy(hobby, {
  get(target, key, receiver){
    return target[key] + ",get()";
  }
});

let createObj = Object.create(newProxy, {
  music: {
    value: "클래식"
  }
});

console.log(createObj.music);
console.log(createObj.sports);
```

```
클래식
스포츠,get()
```

Proxy Trap

- get()

```
let sportsObj = Object.defineProperty({}, "sports", {
  set(){
    this.value = 123;
  },
  configurable: false
});

let newProxy = new Proxy(sportsObj, {
  get(target, key){
    return target[key] || 123;
  }
});

try {
  newProxy.sports;
} catch (e) {
  console.log("에러");
};
```

에러

```
let newProxy = new Proxy([10, 20], {
  get(target, key, receiver){
    return target[0] + target[1];
  }
});

console.log(newProxy.listArray);  30
```

Proxy Trap

- has()
  - target 오브젝트에서 프로퍼티 key 존재 여부를 반환한다.

구분	타입	데이터(값)
형태		handler.has()
파라미터	object	Target, 대상 오브젝트(Object, Array, Function, Proxy…)
	String	Key, 프로퍼티 key
반환	any	프로퍼티가 있으면 true, 아니면 false

  - 오브젝트에 프로퍼티가 존재하면서 프로퍼티 추가 금지 상태이거나 [[Configurable]]:false 일 때, 의도적으로 return false를 할 수 없다.

Proxy Trap

- has()

```
let newProxy = new Proxy({sports: "스포츠"}, {
  has(target, key){
    return key in target ? true : false;
  }
});

console.log("sports" in newProxy);
console.log("music" in newProxy);
```

```
true
false
```

```
let sportsObj = {soccer: "축구"};
Object.preventExtensions(sportsObj);

let newProxy = new Proxy(sportsObj, {
  has(target, key){
    return target[key];
    // return false;
  }
});

console.log("baseball" in newProxy);
console.log("soccer" in newProxy);
```

```
false
true
```

Proxy Trap

- defineProperty()
  - target 오브젝트에 프로퍼티를 추가하거나 프로퍼티 값을 변경한다.

구분	타입	데이터(값)
형태		Handler.defineProperty()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
	String	Key,추가/변경할 프로퍼티 키
	Array	추가 변경할 attributes(스크립터)
반환	Boolean	처리 성공 : true , 아니면 false

  - strict 모드일 때 false를 반환하면 TypeError가 발생한다.

Proxy Trap

- defineProperty()

```
let target = {};
let newProxy = new Proxy(target, {
  defineProperty(target, key, descriptor) {
    descriptor.value = descriptor.value + ":축구";
    Object.defineProperty(target, key, descriptor);
    return true;
  }
});

Object.defineProperty(newProxy, "sports", {
  value: "스포츠", writable: true, configurable: true
});

console.log(target);
```

```
▼Object
    sports: "스포츠:축구"
  ▶__proto__: Object
```

Proxy Trap

- deleteProperty()
  - target 오브젝트에서 프로퍼티를 삭제한다.

구분	타입	데이터(값)
형태		Handler.deleteProperty()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
	String	Key,삭제할 프로퍼티 키
반환	Boolean	처리 성공 : true , 아니면 false

  - 프로퍼티가 [[Configrable]] : false 이면 삭제할 수 없으며 TypeError가 발생한다.

Proxy Trap

- deleteProperty()

```
let target = {sports: "스포츠", music: "음악"};
let newProxy = new Proxy(target, {
  deleteProperty(target, key) {
    return delete target[key];
  }
});

console.log(delete newProxy.sports);
console.log(delete newProxy.dummy);

Object.seal(target);
let desc = Object.getOwnPropertyDescriptor(target, "music");
if (desc.configurable){
  console.log(delete newProxy.music);
} else {
  console.log("삭제 불가");
};
```

```
true
true
삭제 불가
```

Proxy Trap

- preventExtensions()
  - target 오브젝트에서 프로퍼티 추가 금지를 설정한다.

구분	타입	데이터(값)
형태		Handler.preventExtensions()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
반환	Boolean	추가금지 설정 성공 : true , 아니면 false

Proxy Trap

- preventExtensions()

```
let sportsObj = {sports: "스포츠"};
let newProxy = new Proxy(sportsObj, {
  preventExtensions(target){
    Object.preventExtensions(target);
    return true;
  }
});

Object.preventExtensions(newProxy);
console.log(Object.isExtensible(sportsObj)); false
```

Proxy Trap

- isExtensions()
  - target 오브젝트에서 프로퍼티 추가 가능 여부를 설정한다.

구분	타입	데이터(값)
형태		Handler.isExtensions()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
반환	Boolean	추가가능 : true , 아니면 false

Proxy Trap

- isExtensions()

```
let sportsObj = {};
let newProxy = new Proxy(sportsObj, {
  isExtensible(target){
    return Object.isExtensible(target);
  }
});

console.log(Object.isExtensible(newProxy));

Object.preventExtensions(sportsObj);
console.log(Object.isExtensible(newProxy));
```

```
true
false
```

Proxy Trap

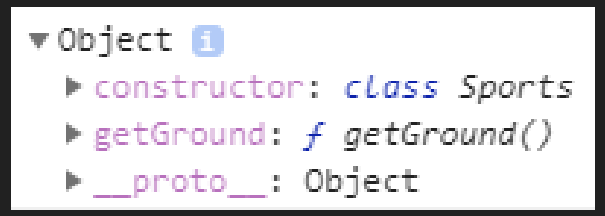
- getPrototypeOf()
  - target 오브젝트의 prototype을 반환한다

구분	타입	데이터(값)
형태		Handler.getPrototypeOf()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
반환	object	Prototype 또는 null

Proxy Trap

- getPrototypeOf()

```
class Sports{
  getGround(){
    return "상암 구장";
  }
};
let newSports = new Sports();

let newProxy = new Proxy(newSports, {
  getPrototypeOf(target){
    return Object.getPrototypeOf(target);
  }
});

console.log(Object.getPrototypeOf(newProxy));
```



Proxy Trap

- getPrototypeOf()

```
class Sports{
  getGround(){
    return "상암 구장";
  }
}
let newSports = new Sports();

let newProxy = new Proxy(newSports, {
  getPrototypeOf(target){
    return Object.getPrototypeOf(target);
  }
});

console.log(newProxy.__proto__);

console.log(Sports.prototype.isPrototypeOf(newProxy));
console.log(Object.prototype.isPrototypeOf(newProxy));
```

```
▼ Object
  ▶ constructor: class Sports
  ▶ getGround: ƒ getGround()
  ▶ __proto__: Object
true
true
```

Proxy Trap

- setPrototypeOf()
  - target 오브젝트의 __proto__에 prototype을 설정한다

구분	타입	데이터(값)
형태		Handler.setPrototypeOf()
파라미터	object	target, 대상 인스턴스 혹은 오브젝트(Object, Array, Function, Proxy…)
	Object	Prototype, 설정할 prototype 또는 null
반환	Boolean	처리성공 true, 실패 false

Proxy Trap

- setPrototypeOf()

```
class Sports{
  getSports(){
    return "스포츠";
  };
};
class Music{
  getMusic(){
    return "음악";
  };
};
let newMusic = new Music("클래식");

let newProxy = new Proxy(newMusic, {
  setPrototypeOf(target, proto){
    Object.setPrototypeOf(target, proto);
    return true;
  }
});
Object.setPrototypeOf(newProxy, Sports.prototype);

console.log(newMusic.getSports());
console.log(newMusic.getMusic);
```

```
스포츠
undefined
```

Proxy Trap

- ownKeys()
  - target 오브젝트에서 프로퍼티 키를 배열로 반환한다.

구분	타입	데이터(값)
형태		Handler.ownKeys()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
반환	Array	프로퍼티 키

Proxy Trap

- ownKeys()

```
let sportsObj = Object.defineProperties({}, {
  baseball: {value: "축구", enumerable: true},
  swim: {value: "수영"}
});

let newProxy = new Proxy(sportsObj, {
  ownKeys(target){
    return Object.getOwnPropertyNames(target);
  }
});

console.log(Object.getOwnPropertyNames(newProxy));
console.log(Object.keys(newProxy));
```

```
▼Array(2) i
    0: "baseball"
    1: "swim"
    length: 2
  ▶ __proto__: Array(0)
▼Array(1) i
    0: "baseball"
    length: 1
  ▶ __proto__: Array(0)
```

Proxy Trap

- ownKeys()
  - 트랩의 기능에 맞는 프로퍼티를 반환하고 트랩을 호출한 곳에서 프로퍼티를 걸러내는 처리를 해야 에러가 발생하지 않는다.

```
let sportsObj = Object.defineProperties({}, {
  baseball: {value: "축구", enumerable: true},
  swim: {value: "수영"}
});

let newProxy = new Proxy(sportsObj, {
  ownKeys(target){
    return Object.keys(target);
  }
});

console.log(Object.keys(newProxy));
```

```
Uncaught TypeError: 'ownKeys' on proxy: trap result did not include 'swim'
    at Function.keys (<anonymous>)
    at ownKeys-2.js:15
```

Proxy Trap

- getOwnPropertyDescriptor()
  - target 오브젝트에서 디스크립터를 반환한다.

구분	타입	데이터(값)
형태		Handler.getOwnPropertyDescriptor()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
	String	Key, 프로퍼티 이름
반환	Object	디스크립터 또는 undefined

Proxy Trap

- getOwnPropertyDescriptor()

```
let sportsObj = Object.defineProperty({}, "sports", {
  value: "스포츠",
  configurable: true
});

let handler = {
  getOwnPropertyDescriptor(target, key){
    let desc = Object.getOwnPropertyDescriptor(target, key);
    if (desc.configurable){
      return {configurable: true, enumerable: true, value: "미술"};
    }
    return desc;
  }
};
let newProxy = new Proxy(sportsObj, handler);

console.log(Object.getOwnPropertyDescriptor(newProxy, "sports"));
```

```
▼Object
    configurable: true
    enumerable: true
    value: "미술"
    writable: false
  ▶__proto__: Object
```

Proxy Trap

- construct()
  - 인스턴스를 생성하여 반환한다

구분	타입	데이터(값)
형태		Handler.construct()
파라미터	object	target, 대상 오브젝트(Object, Array, Function, Proxy…)
	Array	ArgumentList, Array 또는 Array-like
	Object	(선택)newTarget, 인스턴스 생성에 사용할 오브젝트
반환	instance	생성한 인스턴스

Proxy Trap

- construct()

```
class Sports{
  constructor(event){
    this.event = event;
  }
};

let newProxy = new Proxy(Sports, {
  construct(target, params, proxy){
    return new target(params[0]);
  }
});

let sportsObj = new newProxy("축구");
console.log(sportsObj.event);
```

축구

Proxy Trap

- apply()
  - 함수를 호출한다

구분	타입	데이터(값)
형태		Handler.apply()
파라미터	Function	target, 호출할 함수
	Object	(선택) this, this로 참조할 오브젝트
	Array	(선택) arguments, 호출된 함수로 넘겨 줄 파라미터 값
반환	any	호출된 함수에서 반환한 값

Proxy Trap

- apply()

```
function getValue(...values){
  return values.reduce(function(previous, current){
    return previous + current;
  });
};

let newProxy = new Proxy(getValue, {
  apply(target, thisObj, args) {
    return target.apply(thisObj, args);
  }
});

console.log(newProxy(10, 20, 30));
console.log(newProxy.apply("", [10, 20, 30]));
console.log(newProxy.call({add: 100}, 10, 20, 30));
```

```
60
60
60
```

Proxy Trap

- revocable()
  - proxy 무효화를 위한 오브젝트를 생성하여 반환한다.

구분	타입	데이터(값)
형태		Proxy. revocable()
파라미터	Object	target, 대상 오브젝트
	Function	Handler, 트랩
반환	object	오브젝트

Proxy Trap

- revocable()

```
let sportsObj = {sports: "스포츠"};
let revocableObj = Proxy.revocable(sportsObj, {
  get(target, key){
    return target[key];
  }
});

console.log(revocableObj.proxy.sports);

revocableObj.revoke();

try {
  revocableObj.proxy.sports;
} catch (e) {
  console.log("사용 불가");
}
```

```
스포츠
사용 불가
```