객체(1)

- 자바스크립트 객체는 단순한 키-값의 쌍이다.
  - ES6에서는 클래스가 존재하지만 내부적으로는 생성자 함수 방식이다.
  - 키-값 쌍 하나를 속성(Property)이라 부름
  - 속성 중에서 키에 대한 값이 함수일 때 메서드를 부름
- 객체의 생성
  - 생성자 함수 방식
  - 객체 리터럴 방식

[예제04-01]

```
01: var p1 = new Object(); //생성자 함수를 이용한 객체 생성
02: p1.name = "홍길동";
03: p1.phone = "010-2222-3331";
04: p1.age = 20;
05: console.log(p1);
06:
07: var p2 = { name : "홍길동", phone : "010-2222-3331", age : 20 }; //객체 리터럴을 이용한 객체 생성
08: console.log(p2);
```

객체(2)

객체의 속성

- 속성의 추가/삭제는 자유롭다

[예제 04-02]

```
01: var p1 = { name:"이몽룡", age:20 };
02: p1.email = "mrlee@opensg.net";
03: delete p1.age
04: console.log(p1);
```

- 객체의 속성에 접근하는 방법
  - 해시 표기법과 마침표(.) 표기법
- 속성을 삭제할 때는 delete!
  - 삭제할 수 없으면 false리턴
  - 속성이 없어도 오류발생(X)

[예제 04-03]

```
01: var p1 = { name:"비문수", age:30 };
02: p1["email"] = "mspark@hotmail.com";
03: p1.address = "서울시 강남구 역삼동";
04:
05: console.log(p1.email);
06: console.log(p1["address"]);
07:
08: delete p1.email;
09:
10: p1.getInfo = function() {
11:     return this.name + "님의 나이 : " + this.age;
12: };
13:
14: console.dir(p1);
```

```
> var A1 = "hello";
< undefined
> A2 = "world";
< "world"
> window.A1
< "hello"
> window.A2
< "world"
> delete window.A1;
< false
> delete window.A2
< true
```

객체(3)

객체의 메서드

- 자바스크립트 객체의 메서드는 속성 값으로 함수가 할당된 것
- ES2015에서는 클래스를 지원하지만 내부적으로는 함수임.

```
//ECMAScript 2015 클래스 방식
class Contact {
    constructor (id, name, phone) {
        this.id = id;
        this.setContact(x, y);
    }
    setContact (name, phone) {
        this.name = name;
        this.phone = phone;
    }
}
```

```
//앞 코드를 ECMAScript 5 생성자 함수 형태로 표현
var Contact = function(id, name, phone) {
    this.id = id;
    this.setContact(name, phone);
}
Contact.prototype.setContact = function(name, phone) {
    this.name = name;
    this.phone = phone;
}
```

```
> class Contact {
      constructor (id, name, phone) {
          this.id = id;
          this.setContact(x, y);
      }
      setContact (name, phone) {
          this.name = name;
          this.phone = phone;
      }
  }
< function class Contact {
      constructor (id, name, phone) {
          this.id = id;
          this.setContact(x, y);
      }
      setContact (name, phone) {
          this.name = name;
          this.phone = phone;

> typeof(Contact)
< "function"
> Contact.prototype.setContact
< function setContact(name, phone) {
          this.name = name;
          this.phone = phone;
      }
```

객체(4)

객체의 속성 열거

- for ~ in ~ 문 사용
- 메서드만 열거하고 싶다면 typeof() 연산자 사용

[예제 04-04]

```
01: var p1 = { name:"홍길동", age:20, email:"gdhong@gmail.com" };
02: p1.getInfo = function() {
03:     return this.name + "님의 나이 : " + this.age;
04: }
05:
06: for (var k in p1) {
07:     console.log(k + " ---> " + p1[k]);
08: }
```

[실행 결과]

```
name ---> 홍길동
age ---> 20
email ---> gdhong@gmail.com
getInfo ---> function () {
    return this.name + "님의 나이 : " + this.age;
}
```

```
for (var k in p1) {
    if (typeof(p1[k]) != "function") {
        console.log(k + " ---> " + p1[k]);
    }
}
```

배열(1)

- 명칭은 배열이지만 자바의 ArrayList와 같은 컬렉션의 성격
  - 자바스크립트의 배열은 컬렉션과 같이 요소를 추가하고 삭제하는 것이 자유롭다.
- **배열의 생성**
  - Array 생성자 함수 방식, 배열 리터럴 방식

[예제04-05]

```
01: var a1 = new Array(); //생성자 함수를 이용해 배열 생성
02: a1[0] = "hello";
03: a1[1] = "world";
04: a1.push(1000);
05: console.log(a1);
06:
07: var a2 = [ "hello", "world", 1000 ]; //리터럴 방식을 이용해 배열 생성
08: console.log(a2);
```

배열(2)

배열 요소의 변경

[예제 04-06]

```
01: var a = [100,200,300];
02: a[0] =  "hello";    //기존 요소값 변경
03: a[4] = "world";     // ["hello", 200, 300, undefined, "world" ]
04: console.log(a.length);      // 5
05: console.log(a);
06:
07: a.push(400);    //push는 마지막 위치에 추가
08: console.log(a);     // ["hello", 200, 300, undefined, "world", 400 ]
09: console.log(a.length);      //6
10:
11: a.splice(3, 2);     //3번 위치(4번째)에서 2개를 삭제한다.
12: console.log(a);     // ["hello", 200, 300, 400 ]
13:
14: a.splice(3, 0, "jquery");   //3번 위치에서 'jquery' 문자열 삽입
15: console.log(a);     // ["hello", 200, 300, "jquery", 400 ]
16:
17: a.length = 10;
18: console.log(a);     // ["hello", 200, 300, "jquery", 400, undefined x 5 ]
19: console.log(a.length);  //10
20:
21: a.length = 2;
22: console.log(a);     // ["hello", 200 ]
23: console.log(a.length);  //2
```

배열(3)

배열 요소의 변경(이어서)

- push() 메서드 : 요소 추가
- splice() 메서드 : 요소의 삽입, 삭제, 치환
- length 속성 변경 가능
  - 기존 배열의 길이보다 작은 값으로 설정하면 배열의 길이도 줄어들지만 설정된 length 범위 밖의 요소값은 삭제된다는 점에 주의!!
  - length 속성을 증가시키면 비어있는 요소의 값으로 undefined

```
> var a = [10,20,30];
< undefined
> a.length
< 3
> a.length = 5;
< 5
> a
< [10, 20, 30, undefined × 2]
```

splice(startIndex, deleteCount, insertItems)

startIndex : 시작 위치

deleteCount : 삭제할 요소 개수

insertItems : 삽입할 값들

```
> var a = [10,20,30,40,50];
< undefined
> a[-1]
< undefined
> a[9] = "hello";
< "hello"
> a.length
< 10
> a
< [10, 20, 30, 40, 50, undefined × 4, "hello"]
```

배열(4)

- 기타 배열 메서드
  - 자세한 내용은 5장에서

```
▼__proto__: Array[0]
  ▶concat: function concat()
  ▶constructor: function Array()
  ▶copyWithin: function copyWithin()
  ▶entries: function entries()
  ▶every: function every()
  ▶fill: function fill()
  ▶filter: function filter()
  ▶find: function find()
  ▶findIndex: function findIndex()
  ▶forEach: function forEach()
  ▶indexOf: function indexOf()
  ▶join: function join()
  ▶keys: function keys()
  ▶lastIndexOf: function lastIndexOf()
   length: 0
  ▶map: function map()
  ▶pop: function pop()
  ▶push: function push()
  ▶reduce: function reduce()
  ▶reduceRight: function reduceRight()
  ▶reverse: function reverse()
  ▶shift: function shift()
  ▶slice: function slice()
  ▶some: function some()
  ▶sort: function sort()
  ▶splice: function splice()
  ▶toLocaleString: function toLocaleString()
  ▶toString: function toString()
  ▶unshift: function unshift()
  ▶Symbol(Symbol.iterator): function ArrayValues()
  ▶Symbol(Symbol.unscopables): Object
  ▶__proto__: Object
```

배열과 객체

배열도 객체 중의 하나

- 따라서 배열에 임의의 속성을 추가하는 것이 가능함.
- for ~ in ~ 문은 배열, 임의의 추가된 속성까지 모두 접근
- for 문은 배열 요소값만 접근

[예제 04-07]

```
01: var a = [100,200,300];
02: a.name = "홍길동";
03: a.age = 20;
04: a.email = "gdhong@gmail.com";
05:
06: console.dir(a);
```

[예제 04-08 : 배열 요소의 열거]

```
//예제 04-07에 이어서 작성함.
for (var k in a) {
    console.log(k + "--->" + a[k]);
}
```

[실행 결과]

```
0--->100
1--->200
2--->300
name--->홍길동
age--->20
email--->gdhong@gmail.com
```

[예제 04-09]

```
// 예제 04-08에 이어서 작성
for (var i=0; i < a.length; i++) {
    console.log(a[i]);
}
```

[실행 결과]

```
100
200
300
```

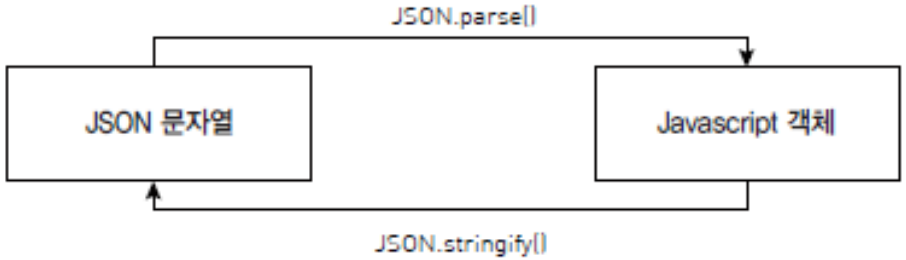
JSON(1)

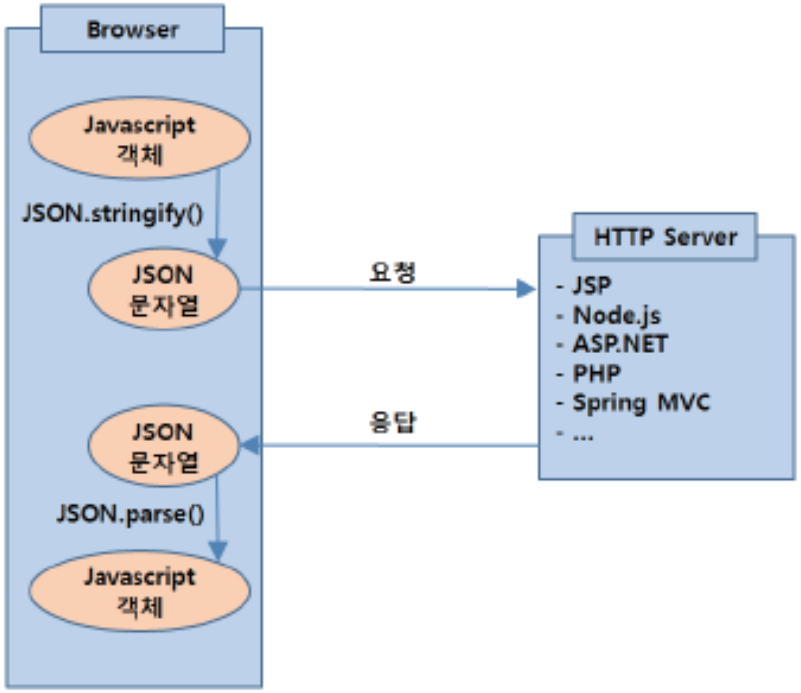
- **원래는** 자바스크립트가 객체를 표기하는 표기법
  - JSON은 프로그래밍언어가 아니라 데이터 표기법
  - 현재는 N/W에서 데이터를 교환하는 경량 데이터 전송 표준의 의미
  - 자바스크립트의 객체 리터럴과 배열 리터럴을 결합하여 표기하는 표기법
    - 속성의 키가 큰따옴표로 묶여야 하고 속성의 값과 배열의 값이 함수여서는 안 됨.

"JSON(JavaScript Object Notation)은 속성-값 쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다. 비동기 브라우저/서버 통신을 위해, 넓게는 XML(AJAX가 사용)을 대체하는 주요 데이터 포맷이다. 특히, 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 알려져 있다. 자료의 종류에 큰 제한은 없으며, 특히 컴퓨터 프로그램의 변수값을 표현하는 데 적합하다."

JSON(2)

JSON 문자열과 자바스크립트 객체간의 변환






```
> var a = { name:"홍길동", age:20 };
< undefined
> console.dir(a)
  ▼Object i
     age: 20
     name: "홍길동"
    ▶__proto__: Object
< undefined
> JSON.stringify(a)
< "{"name":"홍길동","age":20}"
```

JSON(3)

(이어서)

- receive나 replacer를 사용해 변환시에 데이터를 가공한다.

JSON.parse(jsontext [,receiver])

jsontext : 필수 입력값. JSON 문자열

receiver : 옵션값. 각 필드에 대해 이 함수가 호출된다. 함수로 전달하는 파라미터는 key, value 값이며, 리턴하는 값이 변환된 자바스크립트 객체의 속성값이 된다.

JSON.stringify(object [,replacer])

object : 필수 입력값. JSON 문자열로 변환하려는 자바스크립트 객체

replacer : 옵션값. 객체의 속성 이름/값이 파라미터가 되며, 리턴하는 값이 JSON 문자열 필드의 값이 된다.

JSON(4)

- receiver, replacer를 사용하지 않았을 때

```
> var p1 = { name:"홍길동", email:"gdhong@opensg.net", hiredate: new Date() };
< undefined
> var json = JSON.stringify(p1);
< undefined
> console.log(json);
  {"name":"홍길동","email":"gdhong@opensg.net","hiredate":"2015-11-24T23:50:38.229Z"}
< undefined
> var p2 = JSON.parse(json);
< undefined
> console.dir(p2);
  ▼Object
    email: "gdhong@opensg.net"
    hiredate: "2015-11-24T23:50:38.229Z"
    name: "홍길동"
    ▶__proto__: Object
```

JSON(5)

- replacer 사용

[예제 04-10 : JSON.stringify( )의 replacer 기능]

```
01: var data = {
02:     "_id" : "mspark11",
03:     "name" : "박명수",
04:     "phones" : [ "010-2452-8864", "02-2214-3521" ],
05:     "title" : "수석 컨설턴트",
06:     "hiredate" : new Date("2010-09-01")
07: };
08: var str = JSON.stringify(data, function(key, value) {
09:     if (key == "hiredate") {
10:         return new Date(value).getTime();
11:     } else {
12:         return value;
13:     }
14: });
15:
16: console.log(str);
```

```
{"_id":"mspark11","name":"박명수","phones":["010-2452-8864","02-2214-3521"],"title":"수석 컨설턴
트","hiredate":1283299200000}
```

[그림 04-14 : 예제 04-10 실행 결과]

JSON(6)

- receiver 사용

[예제 04-11 : JSON.parse( )의 receiver 기능]

```
01: var json = '{"_id":"mspark11","name":"박명수","phones":["010-2452-8864","02-
2214-3521"],"title":"수석 컨설턴트","hiredate":1283299200000}';
02:
03: var p2 = JSON.parse(json, function(key, value) {
04:     if (key == "hiredate") {
05:         return new Date(value);
06:     } else {
07:         return value;
08:     }
09: });
10:
11: console.dir(p2);
```

```
▼Object
    id: "mspark11"
  ▶ hiredate: Wed Sep 01 2010 09:00:00 GMT+0900 (대한민국 표준시)
    name: "박명수"
  ▶ phones: Array[2]
    title: "수석 컨설턴트"
  ▶ __proto__: Object
```

[그림 04-15 : 예제 04-11 실행 결과]

속성, 메서드, this(1)

속성은 연관배열

- p1.name
- p1[“name”]
- 속성의 존재 여부는 hasOwnProperty() 메서드를 이용함

[예제04-12]

```
01: var p1 = { name:"홍길동", age: 20, email:"gdhong@opensg.net" };
02: console.log(p1.name);
03: console.log(p1["name"]);
04:
05: for (var key in p1) {
06:     console.log(key + "-->" + p1[key]);
07: }
08:
09: console.log(p1.hasOwnProperty("age"));
```

속성, 메서드, this(2)

this

- this가 결정되는 시점은 함수 호출될 때마다...
  - 클래스 기반의 객체를 다루는 언어는 객체가 생성될 때 결정됨.
- 함수 호출시 호이스팅 단계 막바지에 this 값이 바인딩됨.

[예제 04-13]

```
01: function test() {
02:     console.log(this);
03: }
04:
05: console.log(this);
06: test();
07:
08: var p1 = { name:"홍길동", age:20 };
09: p1.test = test;
10: p1.test();
```

```
▶ Window
▶ Window
▼ Object i
    age: 20
    name: "홍길동"
  ▶ test: function test()
  ▶ __proto__: Object
```

속성, 메서드, this(3)

주의해야 할 부분

- 10행에서 inner()를 호출할 때의 this는 p1이 아니라 window(Global)

[예제 04-14]

```
01: var p1 = { nick:"박문수", age:20 };
02: p1.getInfo = function() {
03:   return this.nick + "님의 나이 :  " + this.age;
04: };
05:
06: p1.getInfo2 = function() {
07:     function inner() {
08:         return this.nick + " is " + this.age + " years old.";
09:     }
10:     return inner();
11: }
12:
13: console.log(p1.getInfo());
14: console.log(p1.getInfo2());
```

```
박문수님의 나이 :  20
undefined is undefined years old.
```

속성, 메서드, this(4)

this의 명시적 바인딩

- call(), apply() 메서드
- jQuery내부에서도 이벤트 처리시에 내부적으로 많이 사용됨

[예제 04-15 : apply( )와 call( ) 메서드]

```
01: function test(x,y) {
02:     var result = x+y;
03:     this.result = result;
04: }
05: test(1,2);
06: console.log(window.result);
07:
08: var p1 = { name:"홍길동" };
09: test.apply(p1, [ 3, 4 ]);
10: console.log(p1.result);
11:
12: var p2 = { name:"이몽룡" };
13: test.call(p2, 5, 6);
14: console.log(p2.result);
```

실행 결과

```
3
7
11
```

» 이전 페이지 예제 개선

[예제 04-16]

```
01: var p1 = { nick:"박문수", age:20 };
02: p1.getInfo = function() {
03:   return this.nick + "님의 나이 :  " + this.age;
04: };
05:
06: p1.getInfo2 = function() {
07:     function inner() {
08:         return this.nick + " is " + this.age + " years old.";
09:     }
10:     return inner.apply(this);
11: }
12:
13: console.log(p1.getInfo());
14: console.log(p1.getInfo2());
```

생성자 함수(1)

- **자바스크립트 생성자 함수를 이용해 객체를 생성할 수 있음**
  - ES6에는 Class가 존재하지만 실제 내용물은 함수임.
- **생성자 함수란?**
  - 바로 자바스크립트 객체를 생성하는 데 사용되는 함수이다.
  - 생성자 함수는 대부분 대문자로 시작하는 파스칼 표기법을 함수명에 적용
- **생성자 함수의 사용 목적**
  - 정형화된 객체 생성
  - 타입 확인 기능
  - prototype에 메서드를 작성

생성자 함수(2)

정형화된 객체 생성

- 생성자 함수를 사용하지 않는 Plain Object 생성
  - 모두 그냥 Object!!
  - 2행의 이름 속성명이 nama지만 오류발생하지 않음..
    - 속성을 동적으로 추가하는 것이 자유롭기 때문에...

[예제 04-17]

```
01: var p1 = { name : "홍길동", age : 20 };
02: var p2 = { nama : "아이패드", price : 299000 };
03: console.log(p1.name);
04: console.log(p2.name);   //undefined!!
05:
06: console.dir(p1);
07: console.dir(p2);
```

```
홍길동
undefined
▼Object
    age: 20
    name: "홍길동"
  ▶ __proto__: Object
▼Object
    nama: "아이패드"
    price: 299000
  ▶ __proto__: Object
```

생성자 함수(3)

(이어서)

- 생성자 함수를 이용한 객체
  - name, age 속성은 반드시 포함하는 객체를 생성함.

[예제 04-18 : 생성자 함수를 이용한 객체]

```
01: function Person(name, age) {
02:     this.name = name;
03:     this.age = age;
04: }
05:
06: var p1 = new Person("홍길동", 20);
07: var p2 = new Person("박문수", 30);
08:
09: console.dir(p1);
10: console.dir(p2);
```

```
▼Person
    age: 20
    name: "홍길동"
  ▼__proto__: Person
    ▶constructor: function Person(name, age)
    ▶__proto__: Object
▼Person
    age: 30
    name: "박문수"
  ▶__proto__: Person
```

생성자 함수(4)

타입 확인 기능

- 예제 04-17의 p1, p2는 각각 사람과 제품을 의미하지만 console에서 확인해보면 단지 Object!!
  - 객체리터럴을 이용해서 만든 Plain Object는 타입 확인 기능이 없음.

```
홍길동
undefined
▼Object i
    age: 20
    name: "홍길동"
  ▶ __proto__: Object
▼Object i
    nama: "아이패드"
    price: 299000
  ▶ __proto__: Object
```

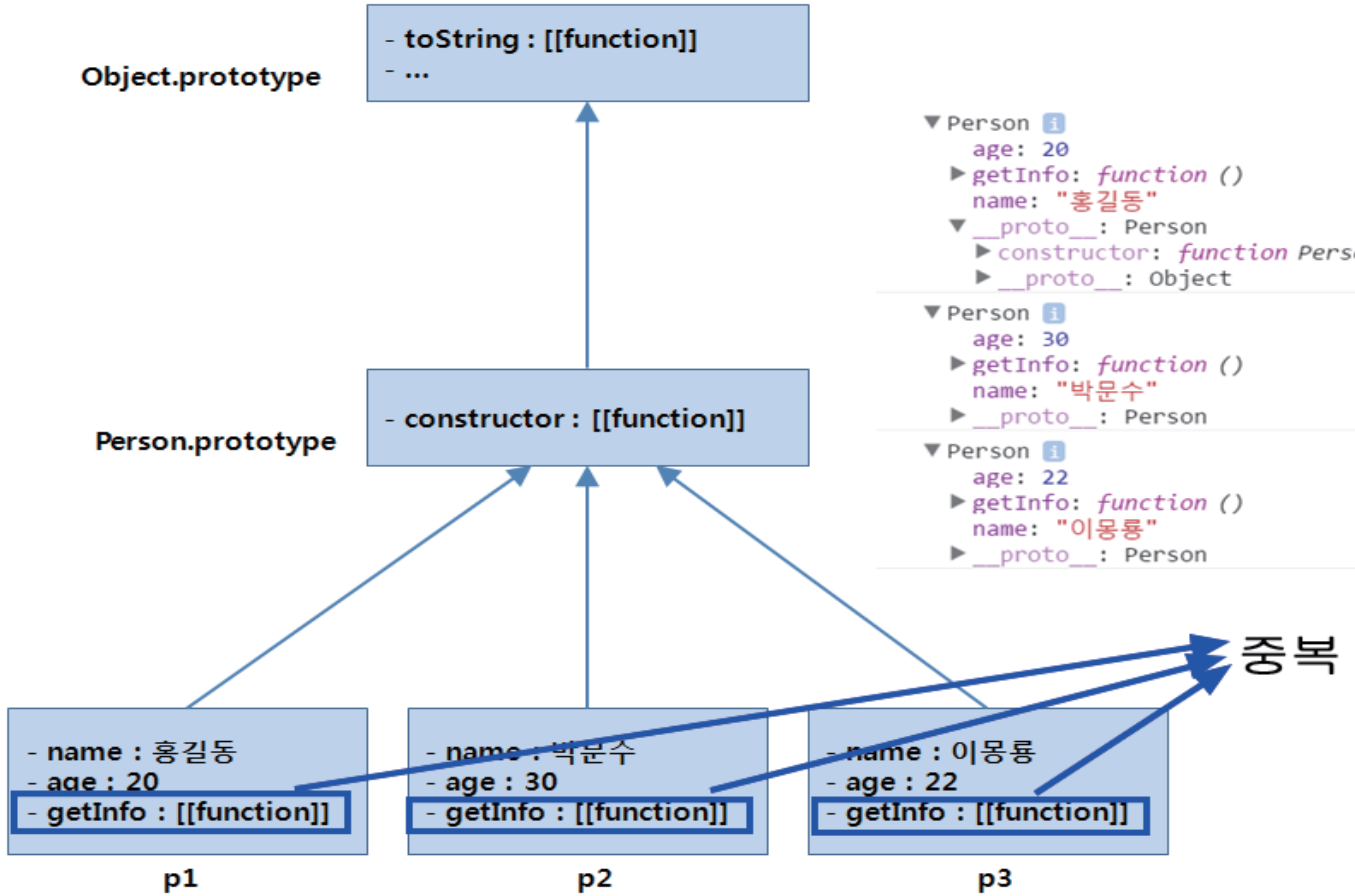
- 생성자 함수를 이용해서 만든 객체는 어느 생성자 함수로 만든 객체인지를 확인할 수 있음
  - prototype의 constructor 속성 이용

```
if (typeof(p1) ==="object" && p1.constructor == Person) { ... }
```

생성자 함수(5)

prototype 객체와 메서드

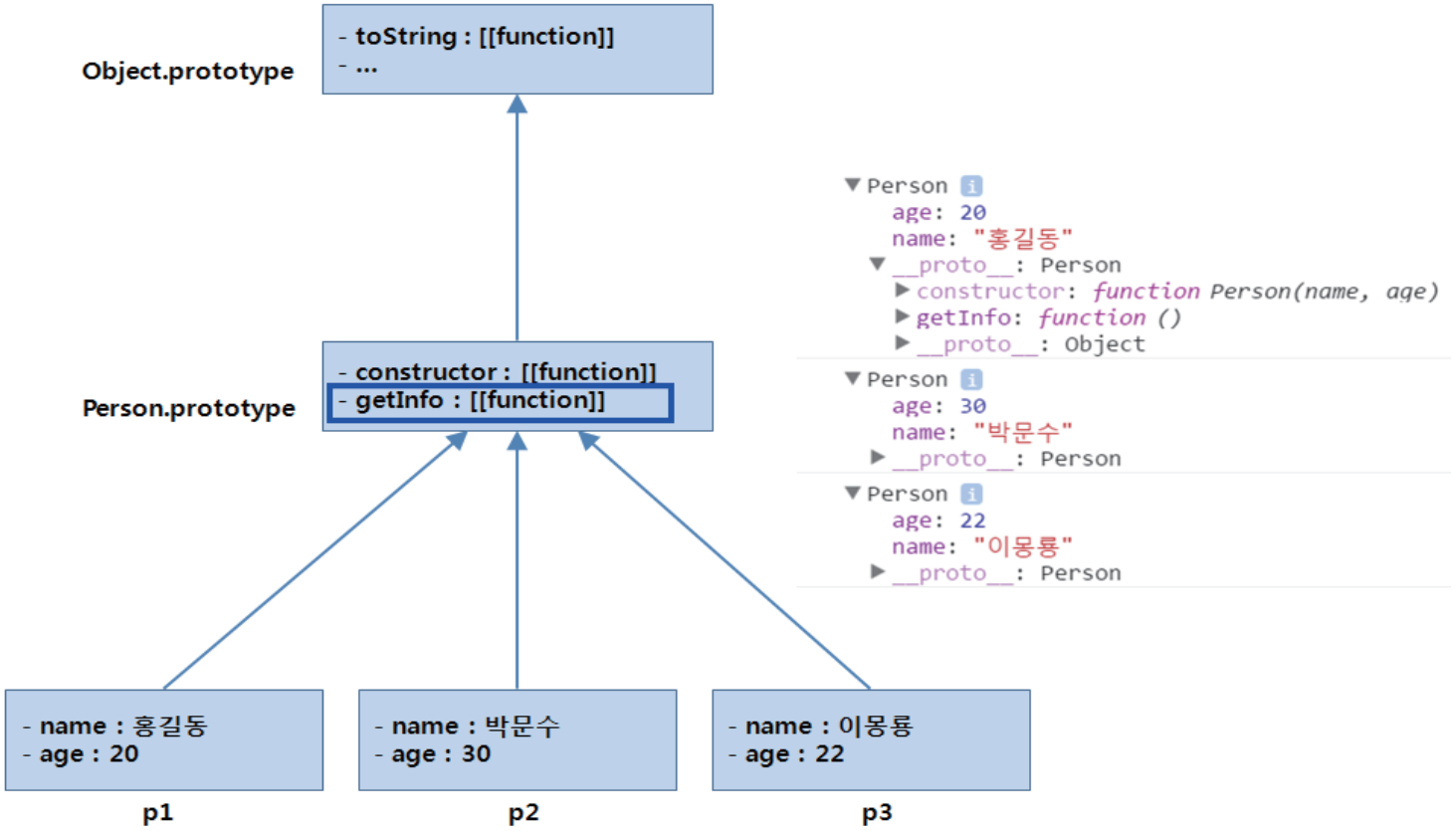
- 각 객체마다 메서드를 만들면?




```
▼ Person
    age: 20
  ▶ getInfo: function ()
    name: "홍길동"
  ▼ __proto__: Person
    ▶ constructor: function Person(name, age)
    ▶ __proto__: Object
▼ Person
    age: 30
  ▶ getInfo: function ()
    name: "박문수"
  ▶ __proto__: Person
▼ Person
    age: 22
  ▶ getInfo: function ()
    name: "이몽룡"
  ▶ __proto__: Person
```

생성자 함수(6)

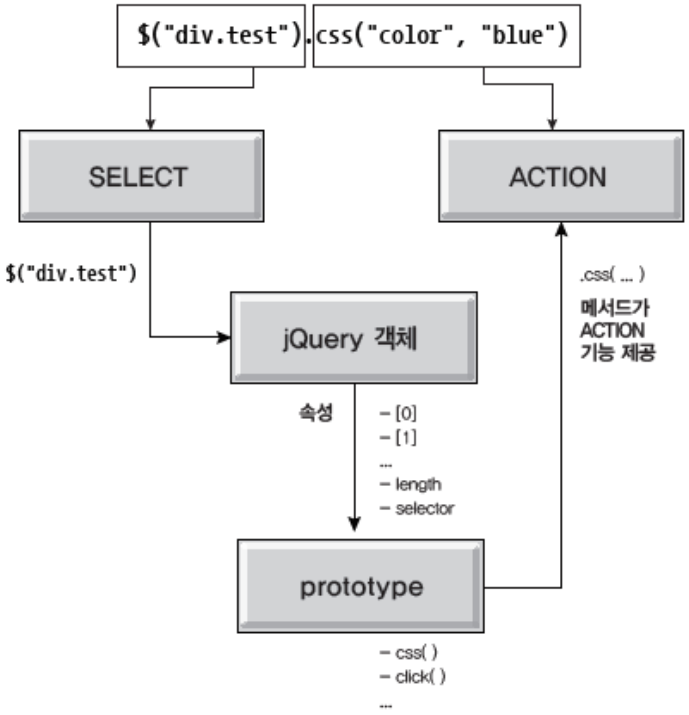
- prototype에 메서드를 만들면



- prototype : 동일한 생성자 함수로 만들어진 객체들이 서로 공유하는 공유객체

jQuery와 객체(1)

- 객체, this, prototype을 다룬 이유
  - jQuery가 내부적으로 이것들을 사용한다.
- jQuery로 하게될 기본적인 작업
  - Select
    - $() 함수, jQuery() 함수로 선택함
    - 함수가 호출되면 jQuery 객체를 리턴함
  - Action
    - jQuery객체가 제공하는 메서드를 이용해 Event, Effect, Manipulation등 다양한 Action을 수행함.
    - 이 메서드들은 jQuery 객체의 prototyp에 정의
    - 필요하다면 prototype과 jQuery 함수 객체에 메서드 추가 가능
      - 이것이 바로 jQuery Plugin!!



jQuery와 객체(2)

- jQuery 객체의 메서드 내에서의 this는 Action이 일어나는 HTML 요소를 가리킴
  - 내부적으로 Function의 apply() 메서드를 이용해 this를 직접 바인딩하였음.

```
07: <script type="text/javascript">
08: $(document).ready(function() {
09:     $("div").click(function() {
10:         alert($(this).html())
11:     });
12: });
13: </script>
```

- $() 는 함수이며 객체이므로 함수 값을 가진 속성(메서드)을 자유롭게 추가할 수 있다.

```
function jQuery(selector, context) { ... }

var $ = jQuery;

$.ajax = function(options) { ... };
```