INTERPARK

2017 NOV.

{ .JS }

JavaScript Lab

함수와 라이브러리 설계시 유의점

JavaScript Lab 6

01. Undefined 활용

> 부가적인 인자값 체크 시
undefined 사용

> 이를 인자로 0,
빈 문자열 수용 시
트루시니스 테스트 방법 사용 금지하고
undefined로 체크

```
1
2   //1. undefined 활용
3   var s1 = new Server(80, "interpark.com");
4   var s2 = new Server(80); //localhost 처리
5
6   function Server(port, host){
7     if(arguments.length < 2){
8       host = "localhost";
9     }
10    this.hostName = host;
11  }
12
13  var s1 = new Server(80, "interpark.com");
14  var s2 = new Server(80); //localhost 처리
15  var s3 = new Server(80, config.hostName); //??
16
17  //undefined 활용
18  function Server(port, host){
19    if(host === undefined){
20      host = "localhost";
21    }
22    this.hostName = host;
23    /* 또는 투루시니스 사용 => undefined는 false처리
24    this.hostName = host || "localhost";
25    */
26  }
27
28  /*
29  트루시니스 사용 시 주의점
30  빈문자열 또는 숫자 0을 의미있는 값으로 사용할 경우
31  */
32
```

02. Prameter 정의 시 옵션객체 활용

> 인수 정의 보다 옵션객체 활용 시 코드 가독성 및 확장성 증대

> 순서에 대한 의존성 제거 => 의도치 않은 오류 제거빈 문자열 수용 시

> 필요한 Prameter 세팅 용이

> 필수 인자는 분리처리 필요

```
1
2   //기능확장을 통한 인자증식의 결과
3   var alert = new Alert(
4   100, 75, 300, 200, "Error", message
5   ,"blue", "white", "black", "error", true
6   );
7
8   //옵션객체 사용으로 가독성 및 확장성 증대
9   var alert = new Alert(
10  {x: 100, y: 75, width: 300, height: 200, title:
11  "Error", message: message
12  ,titleColor: "blue", bgColor: "white",
13  textColor: "black", icon: "error", modal: true}
14  );
15
16  //필수인자 필요시 옵션객체와 분리
17  var alert = new Alert(
18  message, Error, {x: 100, y: 75, width: 300, height:
19  200, title: "Error",
20  titleColor: "blue", bgColor: "white",
21  textColor: "black", icon: "error", modal: true}
22  );
23
24
25  //Alert 내부구현
26  function Alert(message, title, opts){
27    opts = opts || {};
28    this.width = opts.width === undefined ? 320 :
29  opts.width;
30    this.height = opts.height === undefined ? 240 :
31  opts.height;
32    this.titleColor = opts.titleColor || "gray";
```

02. Prameter 정의 시 옵션객체 활용

> 인수 정의 보다 옵션객체 활용 시 코드 가독성 및 확장성 증대

> 순서에 대한 의존성 제거 => 의도치 않은 오류 제거빈 문자열 수용 시

> 필요한 Prameter 세팅 용이

> 필수 인자는 분리처리 필요

```
1
2   //Alert 내부구현(extend 사용)
3   function Alert(message, title, opts){
4     this.message = message;
5     this.title = title;
6     opts = Extend({
7     width: 320, height: 240, titleColor: "gray"
8     }, opts);
9   }
10
11  //extend 구현
12  function Extend(target, source){
13    if(source){
14      for(var key in source){
15        var val = source[key];
16        if(typeof val !== "undefined"){
17          target[key] = val;
18        }
19      }
20    }
21    return target;
22  }
23
24
25
26
27
28
29
30
31
32
```

03. 상태유지 속성 최소화

> 메소드 호출시 다른 내부 속성에
영향을 받으면 상태유지, 아니면 무상태

> 상태유지는 다른 동작으로 인해
기존동작이 영향 받을 가능성 농후 -> 버그발생

> 메소드 호출전 사전 속성 세팅을 위한 코드 필요
-> 복잡도 증가

> 속성 초기화 어려움:
모든 속성에 대해 인지필요

> 상태 메서드는 메소드체이닝 지원 시,
this return, 무 상태는 새로운 객체 생성 후 return

```
//무상태 메소드
"foo".toUpperCase(); //FOO

//상태유지 메소드
var toDay = new Date();
console.log(toDay.toLocaleDateString());
// 2017. 10. 30. 오전 12:11:00
toDay.setHours(2);
console.log(toDay.toLocaleDateString());
// 2017. 10. 30. 오전 2:11:00

//기존동작에 영향을 받는 상태유지의 예(수정사항 확인 필요)///
c.fillText("text 1", 0, 0); //default 색상 black
c.fillStype = "blue";
c.fillText("text 1", 0, 30); // blue
c.fillStype = "black";
c.fillText("text 1", 0, 60); // black

//무상태 메소드의 예(수정사항 확인 불필요)
c.fillText("text 1", 0, 0); //default 색상 black
c.fillText("text 1", 0, 0, {fillStype: "blue"});
// blue
c.fillText("text 1", 0, 0);
// black
```

04. 메소드 설계시 배열과 유사 배열 객체 구별 속성 최소화

> 유사배열객체를 확인하기 위한 명확한 방법 부재

> 이로인해 메소드 설계 시, 진짜 배열 여부만 판단하고,
유사배열을 배열로 처리 시,
사용자 측에서 변환하여 전달하도록 가이드

> 진짜배열 판단 방법 =>
x instanceof Array vs Array.isArray(x)

> instance는 Array.prototype 상속여부로 판단

> Array.isArray는
객체 내부의 [[class]] 프로퍼티 값으로 판단
(**ES5 표준, 사용권장**)

> 2개의 방법 모두 진짜 배열인지 확인.
진짜배열만 true, 유사배열(aguments)은 false

```
var set = new StringSet();
set.add("housing");
set.add(["housing" , "air", "tour"]);
set.add({housing: 11, air: 12, tour: 8});

//add구현
StringSet.prototype.add = function(x){
      if(typeof x === "string"){
        this.addString(x);
      }
      else if(Array.isArray(x)){
        x.forEach(function(s){
          this.addString(s);
        })
      }
      else{
        for(var key in x){
          this.addString(key);
        }
      }
}

//사용 예
function MyClass(){
  this.keys = new StringSet();
  ....
}

MyClass.prototype.update = function(){
  this.keys.add(arguments);
  // else 구문에서 딕셔너리처럼 처리
  //this.keys.add([].slice.call(arguments));
}

var my = new MyClass();
my.update("housing", "air", "tour"); // => ??
```

05. 메소드 체이닝 지원

> 유연성과 간결성과 가독성을 높이기 위해
메소드 체이닝 지원 필요

> 상태 메서드는 메소드체이닝 지원시 return this;

> 무상태 메서드는 새로운 객체 생성후 return

```
function escapeBasicHtml(str){
      var st2 = str.replace(/&/g, "&amp;");
      var st3 = str2.replace(/</g, "&lt;");
      var st4 = str3.replace(/>/g, "&gt;");
      var st5 = str4.replace(/"/g, "&quot;");
      var st6 = str5.replace(/'/g, "&apos;");
}

//무상태체이닝 예
function escapeBasicHtml2(str){
return str.replace(/&/g, "&amp;")
      .replace(/</g, "&lt;")
      .replace(/>/g, "&gt;")
      .replace(/"/g, "&quot;")
      .replace(/'/g, "&apos;");
}

var user = record.map(function(record){
            return record.username;
})
.filter(function(username){
            return !!username;
})
.map(function(username){
            return username.toLowerCase();
});

//상태메소드 체이닝 예
elements.setBackground("yellow")
      .setColor("red")
      .setFontWeight("bold");

$("#notification").html("server not responding")
      .removeClass("info")
      .addClass("error");
```

Fin