Cheatography

JavaScript this binding in-depth Cheat Sheet

by [deleted] via cheatography.com/50355/cs/13903/

5 this keyword binging rules

Default binging	`this` points to a global object. Keep in mind: in the strict mode global object is `undefined`
Implicit binding	A "context object" reference gets bound to `this`
Explicit binding	`this` gets bound to an object of our choice by calling explicitly `Functi on.p ro tot ype.call()` or `Functi on.p ro tot ype.ap ply()`. Unfortunately, `this` biding is lost in = assignments.
Explicit "hard" binding	"Hard" is just an implementation of the `bind()` pattern/method that addresses problem of `this` context lost in assignments
`new f()` binding	An object, newly constructed by `new` call, is set as `this` reference in function `f`

`this` binding rules apply to how `this` keyword gets evaluated in functions. The rules above are sorted from the most generic to the most specialised. Explicit "soft" binding does not prevent us from losing `this` context

Default binding rule example

```
window.engine = "broken"; //global string
function startE ngine() {
    con sol e.l og( thi s.e ngine + " started !");
};
startE ngi ne(); //"b roken started !"
```

Implicit binding rule example

```
window.engine = "broken";
function startE ngine() {
    con sol e.l og( thi s.e ngine + " started !");
};
var raceCar = {
    engine: " 560 hp",
    sta rtE ngine: startE ngine
};
raceCa r.s tar tEn gine(); //"560hp started !"
```

Implicit binding rule broken

```
function startEngine() {
    con sol e.l og( thi s.e ngine + " started !");
};
var raceCar = {
    engine: " 560 hp",
    sta rtE ngine: startE ngine
};
var startR ace Car Engine = raceCa r.s tar tEn -
gine; //implicit binding lost !
startR ace Car Eng ine(); //"u nde fined started
!"
```

Explicit rule example

```
function startEngine() {
    con sol e.l og( thi s.e ngine + " started !");
}
var raceCar = {
    engine: " 560 hp"
};
startE ngi ne.c al l(r ace Car); // " 560hp
started !"
startE ngi ne.a pp ly( rac eCar); // " 560hp
started !"
```

Explicit binding rule broken

```
function startEngine() {
    return function() {
        con sol e.l og( thi s.e ngine + " started
!");
   }
}
var raceCar = {
    engine: " 560 hp"
};
startE ngi ne.c al l(r ace Car)(); //"u nde fined
started !"
```

Explicit "hard" rule example

```
function startEngineFactory() {
    return (funct ion() {
        con sol e.l og( thi s.e ngine + " started
!");
    }).b in d(r ace Car); //"h ard " binding !
}
var raceCar = {
    engine: " 560 hp"
};
var startE ngine = startE ngi neF act ory();
startE ngine({ engine: " 78h p" });//"560hp started
!"
startE ngi ne.c all({ engine: " 51h p" });//"560hp
started !"
```

no matter what you pass to startEngine() function, it is always bound to raceCar object instance

"New" rule example

```
function startEngine(power) {
    thi s.e ngine = power; //a new object
}
var slowCar = new startE ngi ne( " 33h p");
var fastCar = new startE ngi ne( " 260 hp");
consol e.l og( slo wCar); //{engine: " 33h p"}
consol e.l og( fas tCar); //{engine: " 260 hp"}
```

this gets bound to a newly created object (passed as *this* to the function body)

All rules in one example

```
window.engine = "broken";
function startE ngi ne( power) {
   var log = this.e ngine + " prepar ing... ";
    thi s.e ngine = power;
    con sol e.l og(log + this.e ngine + " started
!");
}
var raceCar = {
    engine: " 560 hp",
    sta rtE ngine: startE ngine
};
startE ngi ne( " 15h p"); //default binding
raceCa r.s tar tEn gin e("7 00h p"); //implicit
binding
startE ngi ne.a pply({ engine: " [to -be -se t]"
}, ["44 0hp "]); //explicit binding
var newCar = new startE ngi ne( " 33h p"); //"n -
ew" binding
//logs as follows:
//"b roken prepar ing... 15hp started !"
//"560hp prepar ing... 700hp started !"
//"[ to- be-set] prepar ing... 440hp started !"
//"u nde fined prepar ing... 33hp started !"
```