Gentle Explanation of the Keyword `this` in Javascript

https://web.archive.org/web/20180209163541/https://dmitripavlutin.com/gentle-explanation-of-this-in-javascript/

The Mystery of `this`

- `this` is the current execution context of a function

- JS has four invocation types

  - function invocation `alert('Hello world')`
  - method invocation `console.log('Hello world')`
  - constructor invocation `new RegExp('\d')
  - indirect invocation `alert.call(undefined, 'Hello world')`

- Invocation: executing the code that makes the body of a function (calling the function)

- Context: the value of `this` within the function body
- Scope: set of variables, objects, and functions accessible within a function body

Function Invocation

- Function invocation cannot be a property accessor (that's reserved for methods)
- IIFE (immediately-invoked function expression) is an expression that evaluates to a function object, followed by an argument

`this` in Function Invocation: The Global Object

- `this` is the **global object** in a function invocation

`this` in Function Invocation: Strict Mode

- `this` is undefined in a function invocation in strict mode
- To use strict mode, place directive `'use strict';` at the top of a function body
- Strict mode is active in inner scopes of all functions declared inside

Pitfall: `this` in an Inner Function

- `this` is not the same in an inner function as it is in the outer function
- The context of the innner function depends on invocation, not on the outer function's context
- To have the expected `this`, modify the inner function's context with indrect invocation (`call`, `apply`) or create a bound function using `bind`
  - Since all function invocations have `this` as the global object, it doesn't matter where it's being called from, it will always be the global object by default

Method Invocation

- A method is a function stored in a property of an object
- Method invocation is performed when an expression in a form of property accessor
- Method invocation is different from function invocation primarily because method invocation requires a property accessor form to call the function

`this` in Method Invocation

- `this` is the object that owns the method

Pitfall: separating method from its object

- A method can be extracted into a separate variable, and then when called alone, it loses it's context and `this` no longer referes to the object on which the method was defined
- Since the method is being callled without an object, its really a function invocation, for which `this` is always the global object
- Using `bind` when you assign the method to a variable will fix that context
- `var alone = someObject.someMethod.bind(someObject)`

Indirect Invocation

- Indirect invocation is performed when a function is called using `call` or `apply` methods
- Functions are first-class, which means a function is an object, and its type is `Function`
- `call` and `apply` are `Function` methods that are used to invoke the function with a configurable context
- `call` accepts the first argument `thisArg` as the context, followed by a list of comma separated arguments passed as arguments to the called function
- `apply` accepts the first argument `thisArg` as the context, followed by an array-like object that are passed as arguments to the called function
- `apply` and `call` both invoke the function right away (unlike `bind`)

`this` in Indirect Invocation

- `this` is the first argument of `call` or `apply` in an indirect invocation
- Indirect Invocation is useful when a function should be executed with a specific context

Bound Function

- A bound function is a function connected with an object, usually created from the orignal function using `bind`
- The original and bound functions share teh same code and scope, but different contexts on execution
- The method `bind` accepts the first argument `thisArg` and an optional list of arguments that are then passed as arguments to the called function

- The role of `bind` is to create a new function, **it does NOT invoke the function- it returns a new function bound with `thisArg`**

`this` in Bound Function

- `this` is the first argument of `bind` when invoking a bound function

Tight Context Binding

- `bind` makes a permanent context link and will always keep it
- A bound function cannot change its linked context when using `call` or `apply` with a different context

Conclusion

- Don't ask "Where is `this` taken from?"
- Ask "How is the function invoked?"