Jonathan Disla

Discussion 5.1

Web 330

Prof. Krasso

When working in large applications, developers may use the same function or variable names in the global scope. This will cause what is called "namespace pollution" in the global scope. In an example used by Kudvenkat, if team A and team B have the same function but with different number of parameters, the function call may not display correctly. Much like when a variable with the same name as a previously defined variable overwrites the value of the original variable. The reason this is even more crucial with jQuery is because it is a library where syntax is written the same. I remember working with ajax and not knowing how to separate different Ajax requests. I could have really used namespaces.

Namespaces can be used in several ways however most times we have to check if the namespace already exists and if it doesn't then we can create an empty object. After that there are a multitude of patterns such as object literal notation or namespace injection. jQuery namespaces are important because it allows the namespace to be added to the global scope of the application thus giving single access to the namespace when the function is called.

We must use namespaces to avoid third party libraries colliding with functionality. There are different types of namespace patterns. As an example, I will use the object literal notation style. Object literal notation can be thought of an object containing a collection of key-value pairs[Osmani]. I will use the object literal and nested styles:

//*First we check if this namespace exists*

//*If it doesn't exists we create the global object*

**var** myApp1 = myApp1 **||** {};

//*optional: nesting*

//*checking if myApp1.nestedNamespace exists*

myApp1**.**nestedNamespace = myApp1**.**nestedNamespace **||** {};

//*Contructor function to display customer name*

//*add the customer property to the nested namespace*

myApp1**.**nestedNamespace**.**customer = ((fName, lName)**=>**{

**this.**fName = fName;

**this.**lName = lName;

**this.**getFullName = (()**=>**{

console**.log**(**this.**fName + ' ' + **this.**lName)

})

})

Reference List

Kudvenkat. “Namespace in Javascript.” YouTube. 3 February 2015.

https://www.youtube.com/watch?v=9p-uDkWqqKU

Osmani Addy. “Learning Javascript Design Patterns.” O’Riley Media Inc. 2012.