Private Methods

When Using Callbacks

Usually, when a contract has to have a callback for a remote cross-contract call, this callback method should only be called by the contract itself to avoid someone else calling it and changing the state. A common pattern is to have an assertion that validates that the direct caller (predecessor account ID) matches to the contract's account (current account ID). The({ privateFunction: true }) decorator simplifies this by making it a single line decorator while improving readability.

Use this annotation within the designated contract class with the NearBindgen({}) decorator as follows:

@ call ({

privateFunction :

true

}) my_method ({ })

{ // ... } Which is equivalent to:

@ call ({ }) my_method ({ })

{ if near . currentAccountId ( )

!= near . predecessorAccountId ( )

{ throw

new

Error ("Method method is private") ; } // ... } Now with this annotation, only the account of the contract itself can call this method, either directly or through a promise.

Writing Internal Methods

Not all functions need to be exposed publicly. It may be beneficial to write private methods for helper or utility functions, for instance. There are three approaches to write internal methods:

1. Declare the method without using thecall
2. orview
3. decorators.

helperMethod (a , b)

{ // ... } 1. Using an internal helper function in the module scope.

// Function that can be called in another JS file function

getFirstName (account)

{ // ... } 1. Importing a helper function or class from another module.

Another way of not exporting methods is by having a separate class, that is not marked withNearBindgen({}) .

import

{ getFirstName }

from

"./helpers.js" ;

@ NearBindgen ({ }) export

class

Contract

{ // ... }

class

Helpers

{ // ... }Edit this page Last updatedonJan 20, 2023 byDennis Was this page helpful? Yes No

Previous Public Methods Next Payable Methods