Can we perform DML using wire service ?

April 18, 2020 auraenabled Lightning 2

> *If you are reading this post, it means you already know what is wire service. If you are not familiar yet, i would recommened you to visit **this** post for quick recap.*

In short, NO, we can only receive the data and we cannot mutate/change the data using wire service.

There is a tip in documentation which says..

> *The wire service delegates control flow to the Lightning Web Components engine. Delegating control is great for read operations, but it isn't great for create, update, and delete operations. As a developer, you want complete control over operations that change data. That's why you perform create, update, and delete operations with a **JavaScript API** instead of with the wire service.*

It means that wire services actually deliegates the control to lwc engine, which internally perform read operation to access the data. We do not have the permission to perform DML operation using wire service.

To understand this behaviour in detail, lets take a simple example of getting account information using apex method.

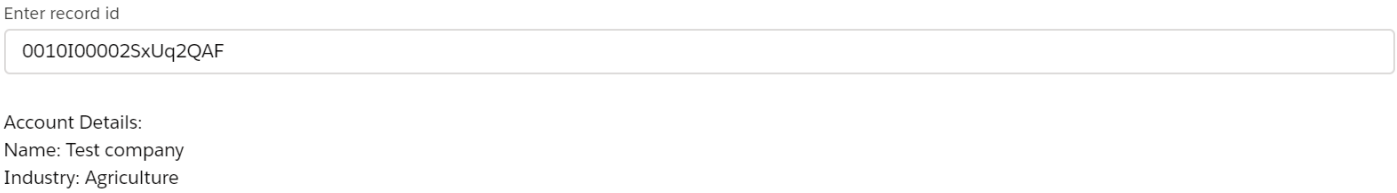
In this example, we are trying to get the account information with given record id – using an apex method. As we know, to expose an apex method to a lightning component, we must decorate the method with @AuraEnabled (which is ofcourse the name of this website 😉).
To access this method **using wire service**, we must set **cacheable=true**.

```
public with sharing class AccountRecordController {

    @AuraEnabled(cacheable = true)
    public static Account getSingleAccount(Id accId){
        return [SELECT Id, Name, Industry
                FROM Account
                WHERE Id =: accId];
    }
}
```

```
import { LightningElement, wire, api } from 'lwc';
import getSingleAccount from
'@salesforce/apex/AccountRecordController.getSingleAccount';

export default class AccountRecordDetails extends LightningElement {

    recId;

    account;
    errormessage;

    @wire(getSingleAccount, {accId : '$recId'})
    wiredSingleAccount({data, error}){
        console.log(JSON.stringify(data));
        console.log(data);
        if(data){
            this.account = data;
            this.errormessage = undefined;
        }else if(error){
            this.account = undefined;
            this.errormessage = error.body.message;
        }
    }

    handleChange(event){
        this.recId = event.target.value;
    }

}
```

```
<template>

    <lightning-layout horizontal-align={horizontalAlign}>
        <lightning-layout-item flexibility="auto" padding="around-small">
            <lightning-input label="Enter record id"
                             type="text"
                             name="recid"
                             value ={recId}
                             onchange={handleChange}>
            </lightning-input>
        </lightning-layout-item>
    </lightning-layout>

    <div class="slds-p-around_small">
        <template if:true={account}>
            Account Details: <br>
            Name: {account.Name} <br>
            Industry: {account.Industry}
        </template>

        <template if:true={errormessage}>
            {errormessage}
        </template>
    </div>

</template>
```

Output:

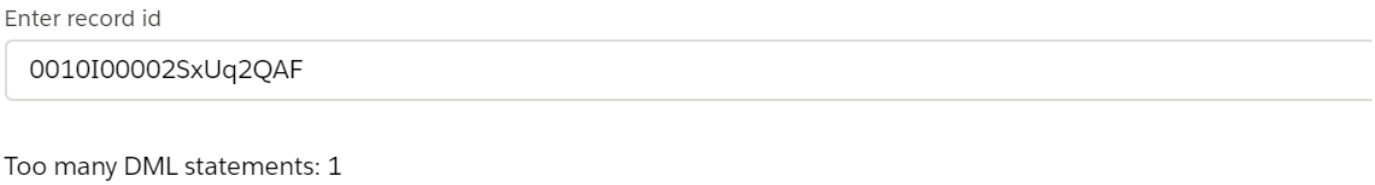


All good, it is working perfectly.

What happens when we add dml in this method ?

```
public with sharing class AccountRecordController {

    @AuraEnabled(cacheable = true)
    public static Account getSingleAccount(Id accId){
        //Ignore the functionality of this code. intention is to perform DML
        Log__c newlog = new Log__c();
        newlog.Error_Message__c = 'no error, just searching -->' + accId;
        Insert newlog;

        return [SELECT Id, Name, Industry
                FROM Account
                WHERE Id =: accId];
    }
}
```

Output:



We get the following error once populate the record Id.

Too many DML statements: 1

It is one of the famous governor limit error in Apex. But wait, it is not ‘Too many DML statements: 151’ it says 1.

why ?

That is because the request that is made using wire services does not have limits to perform DML operation. If we look at the limit usage from apex log, this is how it will be:

20:19:27.2 (2761552)|LIMIT_USAGE_FOR_NS|(default)|
Number of SOQL queries: 0 out of 100
Number of query rows: 0 out of 50000
Number of SOSL queries: 0 out of 20
Number of DML statements: 1 out of 0 * CLOSE TO LIMIT
Number of DML rows: 0 out of 10000
Maximum CPU time: 0 out of 10000
Maximum heap size: 0 out of 6000000
Number of callouts: 0 out of 100
Number of Email Invocations: 0 out of 10

Number of future calls: 0 out of 50
Number of queueable jobs added to the queue: 0 out of 50
Number of Mobile Apex push calls: 0 out of 10

It means we cannot make a request to apex method which has DML statement although it is decorated using @AuraEnabled(cacheable=true).

Can we invoke this apex method imperatively then ?

- Actually NO. we cannot invoke this method imperatatively too. It is a limitation that an apex method which is annotated with @AuraEnabled(Cacheable=true) cannot have DML method when it is invoking from lightning component.
- If we want to do this, then remove **cacheable=true** and use it imperatively.

Can i write a DML statement in another method (which is not AuraEnabled) and call this method from @AuraEnabled method ?

No the behaviour is same. It will not work. It is not where it is written, it is about how it is getting invoked.

What happens if we invoke this method using anonymous window ?

It works perfectly without any issue. Because it has nothing to do with annotations as the annotations are only to provide additional behaviour when working with other features of the platform.

Why do we need to set cacheable=true to a method ?

To improve runtime performance, set @AuraEnabled(cacheable=true) to cache the method results on the client. To set cacheable=true, a method must only get data. It can’t mutate data.

Marking a method as storable (cacheable) improves your component’s performance by quickly showing cached data from client-side storage without waiting for a server trip. If the cached data is stale, the framework retrieves the latest data from the server. Caching is especially beneficial for users on high latency, slow, or unreliable connections such as 3G networks.