Salesforce LWC learning (twenty-nine) getRecordNotifyChange (LDS expansion and enhancement)

zero.zhang posted on 2020-12-18

Reference for this article:

https://developer.salesforce.com/docs/component-library/documentation/en/lwc/data_ui_api

https://developer.salesforce.com/docs/component-library/documentation/en/lwc/data_guidelines

https://developer.salesforce.com/docs/component-library/documentation/en/lwc/lwc.reference_get_record_notify

LDS (Lightning Data Service) has been introduced in aura and lwc articles before. In short, LDS realizes that the records are shared across components, and the versions of the current records are the same in the cross components, so that different components can display the same content of the current record. In lwc, there are two parts that automatically implement LDS.

- lightning-record-form/lightning-record-view-form/lightning-record-edit-form
- `lightning/ui*Api`模組中得所有得`wire adapter`得方法。

It's silly to say this, but it's more intuitive and understandable to give an example.

RecordNotifyChangeController.cls

```apex
public with sharing class RecordNotifyChangeController {
    @AuraEnabled
    public static String saveAccount(String recordId,String industry,String phone) {
        Account accountItem = new Account();
        accountItem.Id = recordId;
        accountItem.industry = industry;
        accountItem.phone = phone;
        accountItem.Name = industry + phone;
        try {
            update accountItem;
            return 'success';
        } catch(Exception e) {
            return 'error';
        }
    }

    @AuraEnabled(cacheable=true)
    public static Account getAccount(String recordId) {
        Account accountItem = [SELECT Name,Industry,Phone from Account where Id = :recordId limit 1];
        return accountItem;
    }
}
```

recordNotifyChangeSample.js

```
import { LightningElement, wire,api,track } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import { refreshApex } from '@salesforce/apex';
import saveAccount from '@salesforce/apex/RecordNotifyChangeController.saveAccount';
import getAccount from '@salesforce/apex/RecordNotifyChangeController.getAccount';
import PHONE_FIELD from '@salesforce/schema/Account.Phone';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';
import NAME_FIELD from '@salesforce/schema/Account.Name';
export default class RecordNotifyChangeSample extends LightningElement {
    @api recordId;

    @track phone;

    @track industry;

    @track accountName;

    fields=[PHONE_FIELD,INDUSTRY_FIELD];

   accountRecord;

    // Wire a record.
    @wire(getRecord, { recordId: '$recordId', fields: [PHONE_FIELD, INDUSTRY_FIELD,NAME_FIELD]})
    wiredAccount(value) {
        this.accountRecord = value;
        const { data, error } = value;
        if(data && data.fields) {
            this.industry = data.fields.Industry.value;
            this.phone = data.fields.Phone.value;
            this.accountName = data.fields.Name.value;
        } else if(error) {
            //TODO
        }
    }


    handleChange(event) {
        if(event.target.name === 'phone') {
            this.phone = event.detail.value;
        } else if(event.target.name === 'industry') {
            this.industry = event.detail.value;
        }
    }

    handleSave() {
        saveAccount({ recordId: this.recordId, industry : this.industry, phone : this.phone})
        .then(result => {
            if(result === 'success') {
                refreshApex(this.accountRecord);
            } else {
                //TODO
            }
        })
        .catch(error => {
            //TODO
        });
    }
```

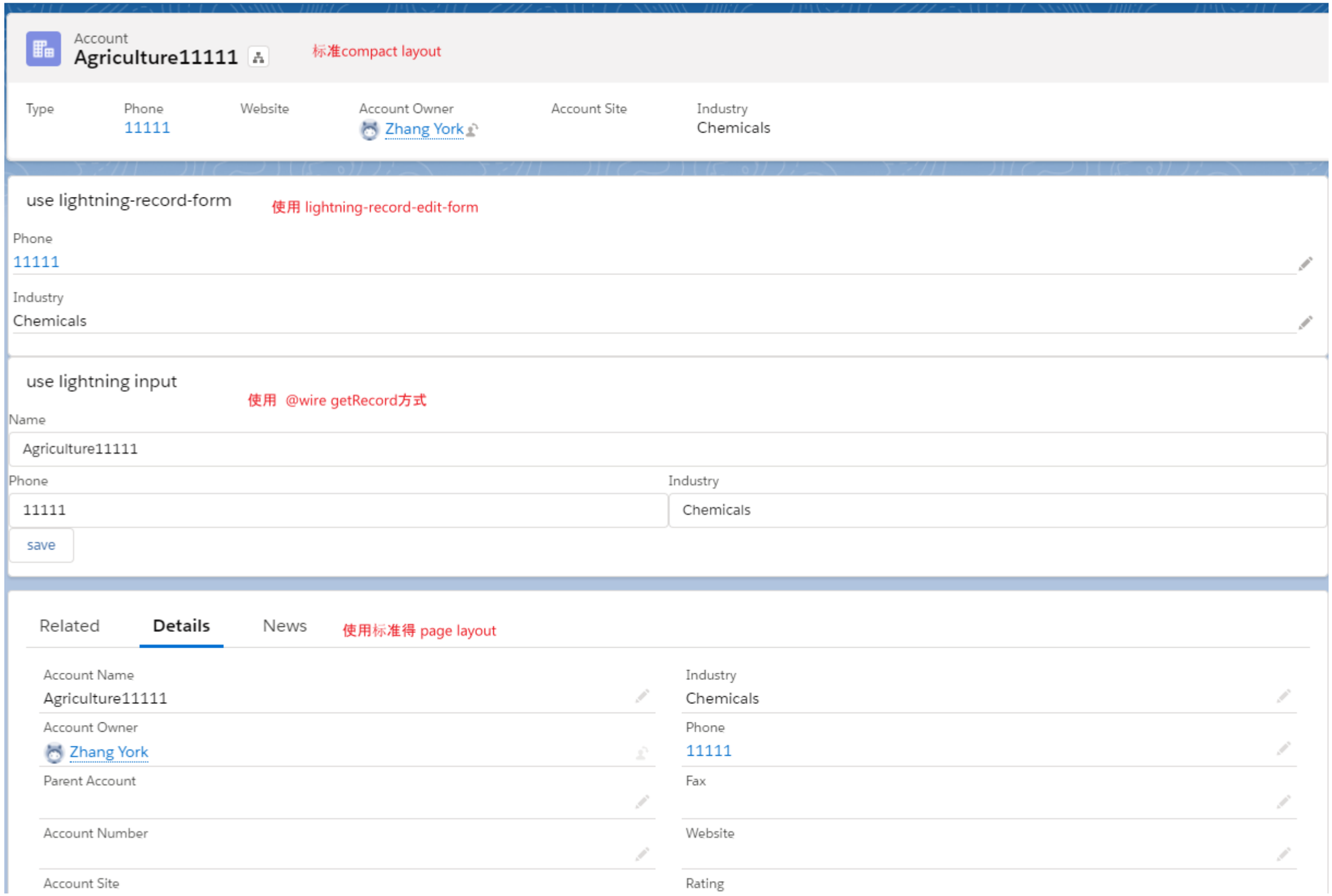
```
58
59 }
```

recordNotifyChangeSample.html

```
<template>
    <lightning-card title="use lightning-record-form">
        <lightning-record-form object-api-name="Account" fields={fields} record-id={recordId} mode="view"></lightning-record-form>
    </lightning-card>

    <lightning-card title="use lightning input">
        <lightning-layout multiple-rows="true">
            <lightning-layout-item size="12">
                <lightning-input value={accountName} label="Name"></lightning-input>
            </lightning-layout-item>
            <lightning-layout-item size="6">
                <lightning-input value={phone} label="Phone" name="phone" onchange={handleChange}></lightning-input>
            </lightning-layout-item>
            <lightning-layout-item size="6">
                <lightning-input value={industry} name="industry" label="Industry"></lightning-input>
            </lightning-layout-item>
            <lightning-layout-item size="12">
                <lightning-button onclick={handleSave} label="save"></lightning-button>
            </lightning-layout-item>
</20         lightning-layout>
    </lightning-card>
</template>
```

Show results:

1. The page below consists of several parts, because in lightning, a page may contain multiple components, and multiple components may share data. The advantage of using LDS is that all caches are of the same version, that is, a modification changes the version. In the future, all those using the current LDS will refresh the version to the latest and display the latest content.



2. We use inline edit to change the value of the industry. After the change, there is no need to rearrange the current page. The quoted content of the above two parts will be changed automatically.

Account
Agriculture11111

Type	Phone	Website	Account Owner	Account Site	Industry
	11111		Zhang York		Chemicals

use lightning-record-form

Phone
11111

Industry
Chemicals

use lightning input

Name
Agriculture11111

Phone
11111

Industry
Chemicals

save

Related | **Details** | News

*Account Name
Agriculture11111

Industry
Agriculture

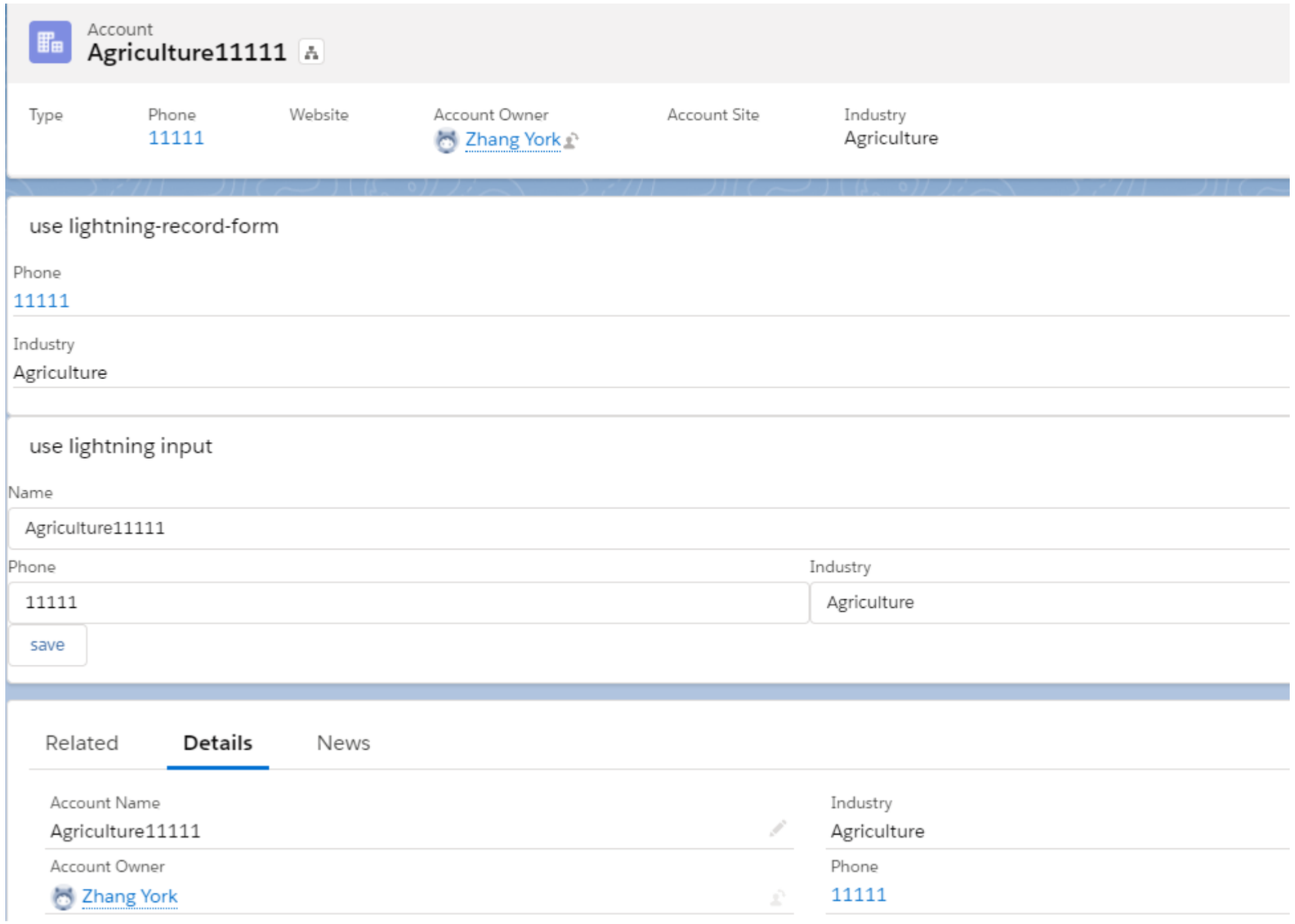
Account Owner
Zhang York

Phone
11111

Parent Account
Search Accounts...

Fax

Cancel | Save



Although LDS is cool to use, it has limitations after all, because the LDS cache can only be shared when the above conditions are met. If you use @wire to call the background apex code, you cannot share LDS, which leads to the display of each component on a page problem. Speaking of this, mention the usual order of use of work with data in lwc.

1. If you need, you can use the lightning-record-form / lightning-record-view-form / lightning-record-edit-form scenarios, and use it first. The use of this label needs to consider the issue of permissions, because the permission to use this label depends on the current user's access permissions to the current table and field. If we don't have the relevant permission for this table and field, we can't use it normally. And these three tags are not valid for

all tables. You need to check whether your table supports it when using it. For example, Event/Task does not support it. And these three tables are not suitable for particularly complex new/update scenarios.

2. If the requirements cannot be achieved using the content described in 1, you can use the related wire adapter methods provided by lwc, such as getRecord, updateRecord, etc. This is still valid for most standard objects and all custom objects, and will be handled more flexibly than 1.

3. Use wire or imperative call apex method to process logic. This applies to the following scenarios:

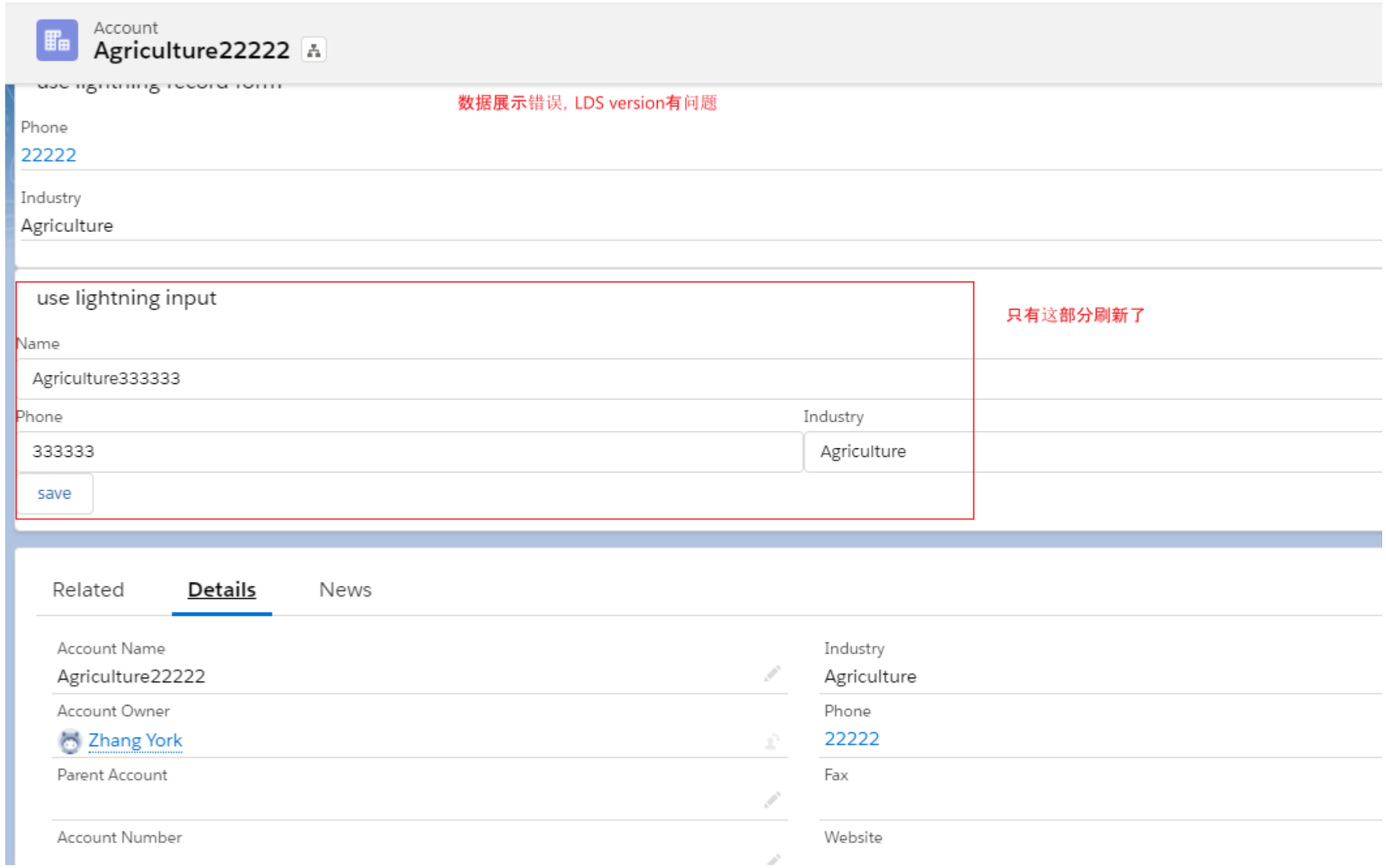
- Used for data processing of objects not supported by the wire adapter, such as Event / Task;
- Used for operations that are not supported by the user interface. For example, the wire adapter provides a method to obtain list data, but the logic of the relevant filter cannot be set. We can use apex to process complex logic in the background;
- To process a transactional logic, for example, after creating an account, you want to create a preset contact. This is not possible with the wire adapter, and you can only use apex;
- Implicit call methods, for example, we click a button or call some background methods in the life cycle function.

The apex method is easy to use, because it can handle any scene, but it has a shortcoming, that is, the information inquired is not LDS. If the information is updated, if the other components on this page reference LDS, the version of the information is not the latest The version shows the phenomenon of out-of-sync data. For example, we changed the wiredAccount in the above demo from the getRecord method to get data through apex in the background.

recordNotifyChangeSample.js

```
import { LightningElement, wire,api,track } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import { refreshApex } from '@salesforce/apex';
import saveAccount from '@salesforce/apex/RecordNotifyChangeController.saveAccount';
import getAccount from '@salesforce/apex/RecordNotifyChangeController.getAccount';
import PHONE_FIELD from '@salesforce/schema/Account.Phone';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';
import NAME_FIELD from '@salesforce/schema/Account.Name';
export default class RecordNotifyChangeSample extends LightningElement {
    @api recordId;

    @track phone;

    @track industry;

    @track accountName;

    fields=[PHONE_FIELD,INDUSTRY_FIELD];

   accountRecord;

   @wire(getAccount,{recordId : '$recordId'})
   wiredAccount(value) {
       this.accountRecord = value;
       const { data, error } = value;
       if(data) {
           this.industry = data.Industry;
           this.phone = data.Phone;
           this.accountName = data.Name;
       }
   }


    handleChange(event) {
        if(event.target.name === 'phone') {
            this.phone = event.detail.value;
        } else if(event.target.name === 'industry') {
            this.industry = event.detail.value;
        }
    }

    handleSave() {
        saveAccount({ recordId: this.recordId, industry : this.industry, phone : this.phone})
        .then(result => {
            if(result === 'success') {
                refreshApex(this.accountRecord);
            } else {
                //TODO
            }
        })
        .catch(error =>{
             // TODO
         });
     }

)
```

Result display



This kind of situation is full of confusion. Two versions are displayed on the same page with the same record data. Naturally, users cannot pass this level. How to solve it? Let’ s talk about today’ s protagonist: getRecordNotifyChange.

getRecordNotifyChange is used to query the specified record ID or ID list, and refresh their LDS cache and version to the latest. In this way, in the case of apex calling, even when the information is updated, the LDS of the entire page is the latest. It should be noted that **this function is only used for version 50 and above** , if it is 48/49 or other old versions, it is not supported. Specific use directly on the demo

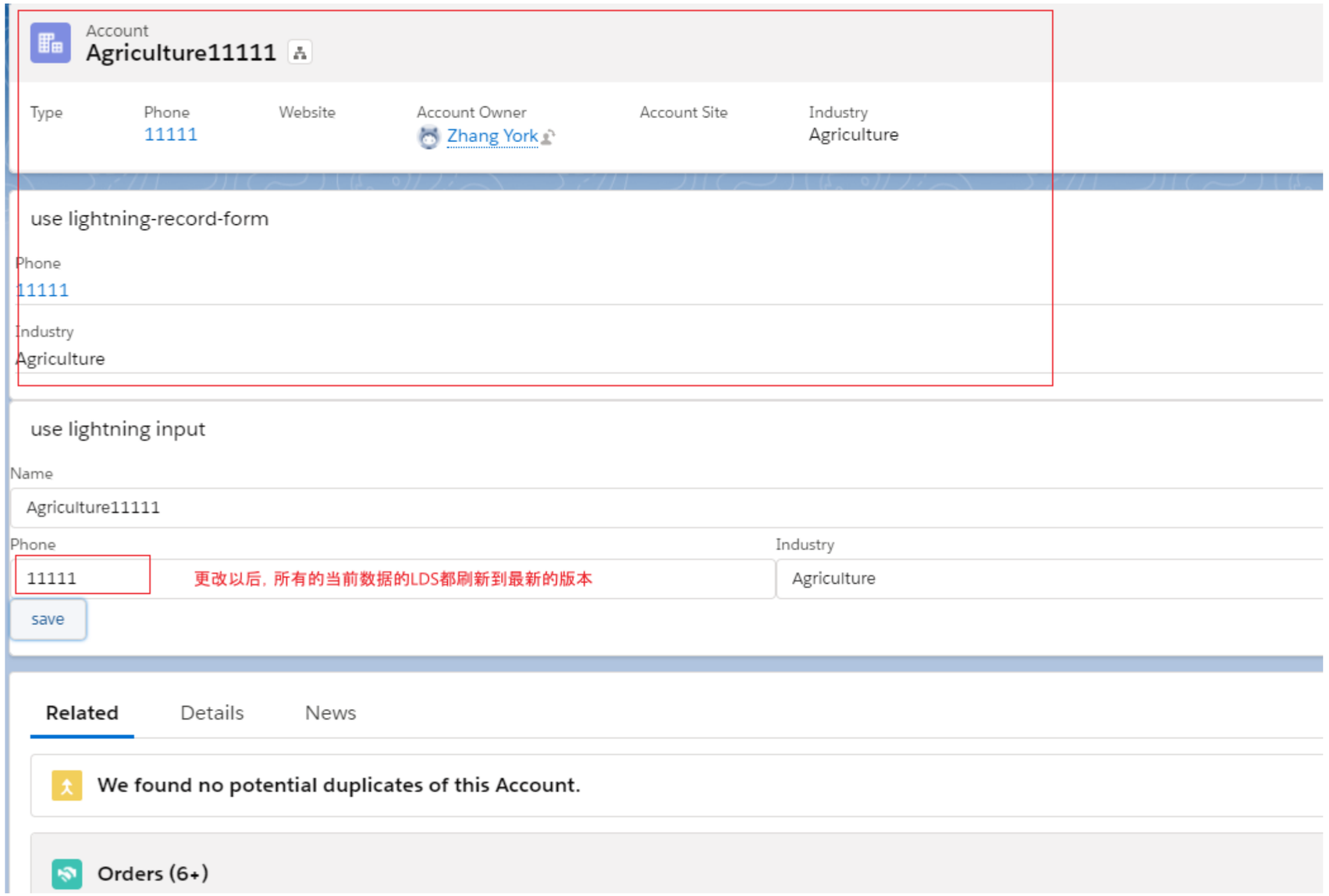
```
import { LightningElement, wire,api,track } from 'lwc';
import { getRecord,getRecordNotifyChange } from 'lightning/uiRecordApi';
import { refreshApex } from '@salesforce/apex';
import saveAccount from '@salesforce/apex/RecordNotifyChangeController.saveAccount';
import getAccount from '@salesforce/apex/RecordNotifyChangeController.getAccount';
import PHONE_FIELD from '@salesforce/schema/Account.Phone';
import INDUSTRY_FIELD from '@salesforce/schema/Account.Industry';
import NAME_FIELD from '@salesforce/schema/Account.Name';
export default class RecordNotifyChangeSample extends LightningElement {
    @api recordId;

    @track phone;

    @track industry;

    @track accountName;

    fields=[PHONE_FIELD,INDUSTRY_FIELD];

   accountRecord;

   @wire(getAccount,{recordId : '$recordId'})
   wiredAccount(value) {
       this.accountRecord = value;
       const { data, error } = value;
       if(data) {
           this.industry = data.Industry;
           this.phone = data.Phone;
           this.accountName = data.Name;
       }
   }


    handleChange(event) {
        if(event.target.name === 'phone') {
            this.phone = event.detail.value;
        } else if(event.target.name === 'industry') {
            this.industry = event.detail.value;
        }
    }

    async handleSave() {
        await saveAccount({ recordId: this.recordId, industry : this.industry, phone : this.phone})
        .then(result => {
            if(result === 'success') {
                refreshApex(this.accountRecord);
                getRecordNotifyChange([{recordId: this.recordId}]);
            } else {
                //TODO
            }
        })
        .catch(error => {
            //TODO
        });
    }

}
```

There are two main changes in the demo above:

1. Introduced getRecordNotifyChange in the header

2. HandleSave needs to use async or Promise, and asynchronous operation is used in the demo. This is a hard requirement.

Result display:



Summary: getRecordNotifyChange realizes that LDS is guaranteed to be the latest pain point in the case of using the call background apex. It may be used frequently in the project. If you don't know it, please check the official documents. If there are errors in the article, please point out that if you don’ t understand, please leave a message.