链滴

Hyperledger Fabric Node.js 如何使用基于通道的事件服务

作者：jimilai

原文链接：https://ld246.com/article/1555937653010

来源网站：链滴

本文说明了基于通道的事件的使用。这些事件与现有事件类似，但是特定于单个通道。在设置侦听器，客户端处理基于通道的事件有一些新选项。从v1.1开始，基于通道的事件是Hyperledger Fabric Node.js客户端的新功能。

以下假设了解Fabric网络(orderers和peer)以及Node应用程序开发，包括使用Javascript Promise。

概述

客户端应用程序可以使用Fabric Node.js客户端注册“侦听器”以在将块添加到通道分类帐时接收块我们将这些称为“基于通道的事件”，它们允许客户端开始接收来自特定块编号的块，从而允许事件理在可能已丢失的块上正常运行。Fabric Node.js客户端还可以通过处理传入的块并查找特定的交易链代码事件来协助客户端应用程序。这允许客户端应用程序被通知交易完成或任意链代码事件，而不在接收时执行多个查询或搜索块。

该服务允许任何用户接收“过滤的”块事件(换句话说，不包含敏感信息)。接收“未过滤”的块事件要对通道进行读访问。默认行为是连接以接收过滤的块事件。要连接以接收未过滤的块事件，请调用connect(true)(参见下文)。

请注意，如果你注册块事件然后提交交易，则不应对包含交易的块进行任何假设。特别是，你不应该设你的交易处于与注册到对等方基于通道的事件服务之后收到的第一个块事件关联的块中。相反，你以只注册一个交易事件。

通道上的API

- newChannelEventHub(peer)：获取ChannelEventHub的新实例的Channel实例方法。
- getChannelEventHubsForOrg：获取基于组织的ChannelEventHubs列表。如果省略组织名称，使用当前用户的当前组织。

在v1.1中新增的ChannelEventHub和API：

- registerBlockEvent(eventCallBack，errorCallBack，options)：注册块事件。
- unregisterBlockEvent(reg_num)：删除块注册。
- registerTxEvent(tx_id，eventCallBack，errorCallBack，options)：注册特定的交易事件。
- unregisterTxEvent(tx_id)：删除特定的交易注册。
- registerChaincodeEvent(ccid，eventCallBack，errorCallBack，options)：注册链代码事件。
- unregisterChaincodeEvent(cc_handle)：删除链代码事件注册。
- connect(full_block)：使客户端通道事件中心与基于结构通道的事件服务连接。必须在您的ChanneEventHub实例接收事件之前进行此调用。当基于通道的事件中心与服务连接时，它将请求接收块或滤的块。如果省略full_block参数，则默认为false，并且将请求过滤的块。调用connect()后，无法更接收块或已过滤的块。
- disconnect()：使客户端通道事件路由关闭与基于结构网络通道的事件服务的连接，并使用已注册的rrorCallBacks通知所有当前通道事件注册的关闭。

节点参数

获取ChannelEventHub的新实例时必须包含此参数。使用连接配置文件时，该值可以是Peer实例或点的名称，请参阅如何使用公共网络配置文件。

eventCallback参数

必须包含此参数。这是当此通道接收新块时，在侦听特定交易或链代码事件时要通知的回调函数。

errorCallback参数

这是一个可选参数。这是在此通道事件路由关闭时要通知的回调函数。关闭可能是由结构网络错误，络连接问题或通过调用disconnect()方法引起的。

options参数

这是一个可选参数。此参数将包含以下可选属性：

- {integer} startBlock(可选)：此选项设置为事件检查的起始块号。包含时，将要求对等方基于通道事件服务开始从此块编号发送块。这也是如何恢复监听或重放添加到分类帐的遗漏块。默认值是分类上最后一个块的编号。replay事件可能会混淆其他事件监听器；因此，当使用startBlock和endBlock，ChannelEventHub上只允许一个侦听器。当排除此参数时(因为它将正常)，将要求事件服务开始从类帐上的最后一个块发送块。
- {integer} endBlock(可选)：此选项设置为事件检查的结束块编号。当包含时，将要求对等方的基通道的事件服务在交付此块后停止发送块。这是如何重播添加到分类帐的遗漏块。如果未包含startBlock，则endBlock必须等于或大于当前通道块高度。replay事件可能会混淆其他事件监听器;因此，当使用tartBlock和endBlock时，ChannelEventHub上只允许一个侦听器。
- {boolean} unregister(可选)：此选项设置指示在看到事件时应删除(取消注册)注册。当应用程序使超时仅等待指定的时间来查看交易时，超时处理应包括交易事件侦听器的手动“取□□消注册”，以避意外调用事件回调。对于不同类型的事件侦听器，此设置的默认值是不同的。对于块侦听器，将end_block设置为选项时，默认值为true。对于交易侦听器，默认值为true。对于链码侦听器，默认值为fals，因为匹配过滤器可能适用于许多交易。
- {boolean} disconnect(可选)：此选项设置指示ChannelEventHub实例在看到事件后自动断开自身对等方基于通道的事件服务的连接。除非设置了endBlock，否则默认值为false，那么它将为true。

获取基于通道的事件路由

Fabric Node.js客户端Channel对象中添加了新方法，以简化ChannelEventHub对象的设置。使用以命令获取将设置为与对等方基于通道的事件服务一起使用的ChannelEventHub实例。ChannelEventub实例将使用对等实例正在使用的所有相同端点配置设置，例如tls证书以及主机和端口地址。

使用连接配置文件（请参阅参考资料）时，可以使用节点的名称来获取新的通道事件路由。

```
var channel_event_hub = channel.newChannelEventHub('peer0.org1.example.com');
```

以下是在使用连接配置文件时如何获取通道事件路由列表的示例。以下内容将根据连接配置文件的当活动客户端client部分中定义的当前组织获取列表。组织中定义的将eventSource设置为true的对象添加到列表中。

```
var channel_event_hubs = channel.getChannelEventHubsForOrg();
```

创建节点实例时，可以使用节点实例获取ChannelEventHub实例。

```
let data = fs.readFileSync(path.join(__dirname, 'somepath/tlscacerts/org1.example.com-cert.pm'));
let peer = client.newPeer(
    'grpcs://localhost:7051',
    {
```

```
        pem: Buffer.from(data).toString(),
        'ssl-target-name-override': 'peer0.org1.example.com'
    }
);
let channel_event_hub = channel.newChannelEventHub(peer);
```

区块侦听器

当需要监视要添加到分类帐的新块时，请使用块事件侦听器。当新块被提交给节点上的分类帐时，将知Fabric客户端Node.js.然后，客户端Node.js将调用应用程序的已注册回调。回调将传递新添加的块JSON表示。请注意，当未使用true值调用connect()时，回调将接收过滤块。注册接收完整块的用户访问权限将由节点的基于信道的事件服务检查。当需要查看先前添加的块时，回调的注册可以包括起块号。回调将开始从此号码接收块，并在添加到分类帐时继续接收新块。这是应用程序resume和reply在应用程序脱机时可能已丢失的事件的一种方法。应用程序应记住它已处理的最后一个块，以避免relay整个分类帐。

以下示例将注册块侦听器以开始接收块。

```
// keep the block_reg to unregister with later if needed
block_reg = channel_event_hub.registerBlockEvent((block) => {
    console.log('Successfully received the block event');
    <do something with the block>
}, (error)=> {
    console.log('Failed to receive the block event ::'+error);
    <do something with the error>
});
```

以下示例将使用起始块编号进行注册，因为此应用程序需要在特定块中恢复并replay丢失的块。应用序回调将像当前事件一样处理同一区域中的replay块。块侦听器将继续接收块，因为它们已提交到节的分类帐。

```
// keep the block_reg to unregister with later if needed
block_reg = channel_event_hub.registerBlockEvent((block) => {
    console.log('Successfully received the block event');
    <do something with the block>
}, (error)=> {
    console.log('Failed to receive the block event ::'+error);
    <do something with the error>
},
    {startBlock:23}
);
```

以下示例将使用起始块编号和结束块进行注册。应用程序需要replay丢失的块。应用程序回调将处理当前事件相同的区域中的replay块。当侦听器看到结束块事件时，块侦听器将自动取消注册，并且ChannelEventHub将关闭。申请将不必处理此句柄。

```
block_reg = channel_event_hub.registerBlockEvent((block) => {
    console.log('Successfully received the block event');
    <do something with the block>
}, (error)=> {
    console.log('Failed to receive the block event ::'+error);
    <do something with the error>
},
    // for block listeners, the defaults for unregister and disconnect are true,
```

```
    // so the they are not required to be set in the following example
    {startBlock:23, endBlock:30, unregister: true, disconnect: true}
);
```

交易监听器

当需要监视组织对等方的交易完成时，请使用交易侦听器。当新块被提交给节点上的分类帐时，将通客户端Node.js.然后，客户端将检查块是否已注册的交易标识符。如果找到交易，则将通过交易ID，易状态和块编号通知回调。过滤的块包含交易状态，因此无需连接到对等方的基于通道的事件服务即接收完整的块。由于大多数非管理员用户将无法看到完整的块，因此当这些用户只需要监听其提交的易时，连接到接收过滤的块将避免访问问题。

以下示例将显示在javascript承诺中注册交易ID并构建另一个将交易发送到订购者的承诺。这两个承将一起执行，以便一起收到两个行动的结果。使用交易侦听器，取消注册的默认可选设置为true。因，在以下示例中，在侦听器看到交易之后，将注册的侦听器将自动取消注册。

```
let tx_object = client.newTransactionID();

// get the transaction ID string for later use
let tx_id = tx_object.getTransactionID();

let request = {
    targets : targets,
    chaincodeId: 'my_chaincode',
    fcn: 'invoke',
    args: ['doSomething', 'with this data'],
    txId: tx_object
};

return channel.sendTransactionProposal(request);
}).then((results) => {
// a real application would check the proposal results
console.log('Successfully endorsed proposal to invoke chaincode');

// start block may be null if there is no need to resume or replay
let start_block = getBlockFromSomewhere();

let event_monitor = new Promise((resolve, reject) => {
    let handle = setTimeout(() => {
        // do the housekeeping when there is a problem
        channel_event_hub.unregisterTxEvent(tx_id);
        console.log('Timeout - Failed to receive the transaction event');
        reject(new Error('Timed out waiting for block event'));
    }, 20000);

    channel_event_hub.registerTxEvent((event_tx_id, status, block_num) => {
        clearTimeout(handle);
        //channel_event_hub.unregisterTxEvent(event_tx_id); let the default do this
        console.log('Successfully received the transaction event');
        storeBlockNumForLater(block_num);
        resolve(status);
    }, (error)=> {
        clearTimeout(handle);
        console.log('Failed to receive the transaction event ::'+error);
```

```
        reject(error);
    },
        // when this `startBlock` is null (the normal case) transaction
        // checking will start with the latest block
        {startBlock:start_block}
        // notice that `unregister` is not specified, so it will default to true
        // `disconnect` is also not specified and will default to false
    );
});
let send_trans = channel.sendTransaction({proposalResponses: results[0], proposal: results[1]})


return Promise.all([event_monitor, send_trans]);
}).then((results) => {
```

Chaincode事件监听器

当需要监控将在您的链代码中发布的事件时，请使用链代码事件监听器。当新块提交到分类帐时，将知客户端Node.js.然后，客户端将在链代码事件的名称字段中检查已注册的链代码模式。监听器的注包括用于检查链代码事件名称的正则表达式。如果发现链代码事件名称与侦听器的正则表达式匹配，将通过链代码事件，块编号，交易ID和交易状态通知侦听器的回调。过滤的块将不具有链码事件有效荷信息；它只有chaincode事件名称。如果需要有效载荷信息，则用户必须能够访问完整块，并且通事件中心必须连接（true）以从对等方的基于通道的事件服务接收完整块事件。

以下示例演示如何在javascript承诺中注册链代码事件侦听器，并构建另一个将交易发送到订购者的诺。这两个承诺将一起执行，以便一起收到两个行动的结果。如果长期监视需要chaincode事件侦听，请遵循上面的块侦听器示例。

```
let tx_object = client.newTransactionID();
let request = {
    targets : targets,
    chaincodeId: 'my_chaincode',
    fcn: 'invoke',
    args: ['doSomething', 'with this data'],
    txId: tx_object
};

return channel.sendTransactionProposal(request);
}).then((results) => {
// a real application would check the proposal results
console.log('Successfully endorsed proposal to invoke chaincode');

// Build the promise to register a event listener with the NodeSDK.
// The NodeSDK will then send a request to the peer's channel-based event
// service to start sending blocks. The blocks will be inspected to see if
// there is a match with a chaincode event listener.
let event_monitor = new Promise((resolve, reject) => {
    let regid = null;
    let handle = setTimeout(() => {
        if (regid) {
            // might need to do the clean up this listener
            channel_event_hub.unregisterChaincodeEvent(regid);
            console.log('Timeout - Failed to receive the chaincode event');
        }
```

```
        reject(new Error('Timed out waiting for chaincode event'));
    }, 20000);

    regid = channel_event_hub.registerChaincodeEvent(chaincode_id.toString(), '^evtsender*',
        (event, block_num, txnid, status) => {
        // This callback will be called when there is a chaincode event name
        // within a block that will match on the second parameter in the registration
        // from the chaincode with the ID of the first parameter.
        console.log('Successfully got a chaincode event with transid:'+ txnid + ' with status:'+sta
us);

        // might be good to store the block number to be able to resume if offline
        storeBlockNumForLater(block_num);

        // to see the event payload, the channel_event_hub must be connected(true)
        let event_payload = event.payload.toString('utf8');
        if(event_payload.indexOf('CHAINCODE') > -1) {
            clearTimeout(handle);
            // Chaincode event listeners are meant to run continuously
            // Therefore the default to automatically unregister is false
            // So in this case we want to shutdown the event listener once
            // we see the event with the correct payload
            channel_event_hub.unregisterChaincodeEvent(regid);
            console.log('Successfully received the chaincode event on block number '+ block_num
;
            resolve('RECEIVED');
        } else {
            console.log('Successfully got chaincode event ... just not the one we are looking for on
block number '+ block_num);
        }
    }, (error)=> {
        clearTimeout(handle);
        console.log('Failed to receive the chaincode event ::'+error);
        reject(error);
    }
        // no options specified
        // startBlock will default to latest
        // endBlock will default to MAX
        // unregister will default to false
        // disconnect will default to false
    );
});

// build the promise to send the proposals to the orderer
let send_trans = channel.sendTransaction({proposalResponses: results[0], proposal: results[1]})


// now that we have two promises all set to go... execute them
return Promise.all([event_monitor, send_trans]);
}).then((results) => {
```

身份证书与MSP服务、权限策略、信道配置与启动、链码通信接口等核心概念，也包含Fabric网络设
、java链码与应用开发的操作实践，是java工程师学习Fabric区块链开发的最佳选择。

原文链接：Hyperledger Fabric Node.js 如何使用基于通道的事件服务