Bitcore

2022-03-17

# Bitcore

Bitcore est une suite d'outils fournis par BitPay. Son objectif est de fournir des outils faciles à utiliser pour les développeurs Bitcoin. Presque tout le code de Bitcore est écrit en JavaScript. Il existe des modules écrits spécifiquement pour NodeJS. Enfin, le module "node" de Bitcore inclut le code C++ de Bitcoin Core. Veuillez consulter <https://bitcore.io> pour plus d'informations.

# Liste des fonctionnalités de Bitcore

* Nœud complet Bitcoin (nœud bitcore)
* Explorateur de blocs (aperçu)
* Utilitaires de blocage, de transaction et de portefeuille (bitcore-lib)
* Communiquer directement avec le réseau P2P de Bitcoin (bitcore-p2p)
* Génération mnémonique d'entropie de graine (bitcore-mnémonique)
* Protocole de paiement (bitcore-payment-protocol)
* Vérification et signature des messages (bitcore-message)
* Schéma de chiffrement intégré à courbe elliptique (bitcore-ecies)
* Service de portefeuille (bitcore-wallet-service)
* Client de portefeuille (bitcore-wallet-client)
* Intégration des services directement avec Bitcoin Core (bitcore-node)

# Exemples de bibliothèque Bitcore

## Prérequis

* NodeJS >= 4.x

Si vous utilisez NodeJS et le nœud REPL :

$npm install -g bitcore-lib bitcore-p2p

## Exemples de portefeuille utilisant bitcore-lib

Création d'une nouvelle adresse Bitcoin avec clé privée associée :

&gt; bitcore = require(&#39;bitcore-lib&#39;)  
&gt; privateKey = nouveau bitcore.PrivateKey()  
&gt; adresse = cléprivée.toAddress().toString()

Création d'une clé privée déterministe hiérarchique et d'une adresse :

&gt; hdPrivateKey = bitcore.HDPrivateKey()  
&gt; hdPublicKey = bitcore.HDPublicKey(hdPrivateKey)  
&gt; hdAddress = nouveau bitcore.Address(hdPublicKey.publicKey).toString()

Création et signature d'une transaction depuis un UTXO :

&gt; utxo = {  
txId : txId, // identifiant de transaction contenant une sortie non dépensée  
outputIndex : outputIndex, // index de sortie (par exemple 0)  
adresse : addressOfUtxo,  
script : bitcore.Script.buildPublicKeyHashOut(addressOfUtxo).toString(),  
satoshis : satoshis // montant envoyé à l&#39;adresse  
}  
&gt; frais = 3000 //défini de manière appropriée pour les conditions sur le réseau  
&gt; tx = nouveau bitcore.Transaction()  
.de(utxo)  
.à(adresse, 35000)  
.frais(frais)  
.enableRBF()  
.sign(CléPrivéeDeUtxo)

Remplacer la dernière transaction dans le mempool (replace-by-fee) :

&gt; rbfTx = nouvelle Transaction()  
.de(utxo)  
.à(adresse, 35000)  
.frais(frais\*2)  
.enableRBF()  
.sign(cléprivéed&#39;Utxo);  
&gt; tx.sérialiser();  
&gt; rbfTx.sérialiser();

Diffuser une transaction sur le réseau Bitcoin (remarque : ne diffusez que des transactions valides ; reportez-vous à <https://bitnodes.21.co/nodes> pour les hôtes homologues) :

1. Copiez le code ci-dessous dans un fichier appelé *broadcast.js*.
2. Les variables tx et rbfTx sont la sortie de tx.serialize() et rbfTx.serialize(), respectivement.
3. Afin de remplacer par des frais, l'homologue doit prendre en charge l'option bitcoind mempoolreplace et la définir sur 1.
4. Exécutez le nœud de fichier *broadcast.js* :

var p2p = require(&#39;bitcore-p2p&#39;);  
var bitcore = require(&#39;bitcore-lib&#39;);  
var tx = new bitcore.Transaction(&#39;sortie de la fonction de sérialisation&#39;);  
var rbfTx = new bitcore.Transaction(&#39;sortie de la fonction de sérialisation&#39;);  
var host = &#39;adresse IP&#39;; // utilise une écoute homologue valide sur tcp 8333  
var pair = new p2p.Peer({hôte : hôte});  
var messages = new p2p.Messages();  
peer.on(&#39;prêt&#39;, fonction() {  
var txs = [messages.Transaction(tx), messages.Transaction(rbfTx)] ;  
indice var = 0 ;  
var intervalle = setInterval(fonction() {  
pair.sendMessage(txs[index++]);  
console.log(&#39;tx : &#39; + index + &#39; envoyé&#39;);  
si (index === txs.longueur) {  
clearInterval(intervalle);  
console.log(&#39;déconnexion du pair : &#39; + host);  
pair.disconnect();  
}  
}, 2000);  
});  
pair.connect();