1. 区块链V1保存字符串

本小节实现简单功能, 使用传统技术实现一个账本功能

1. 实现添加和查询功能

// 生活中的账本 = 区块链

public class NoteBook {

// 用于保存数据的集合

private ArrayList<String> list = new ArrayList<>();

// 添加封面 = 创世区块

// 添加封面的时候,必须保证账本是新的

private void addGenesis(String genesis) {

if (list.size() > 0) {

throw new RuntimeException("添加封面的时候,必须保证账本是新的");

}

list.add(genesis);

}

// 添加交易记录 = 普通区块

// 添加交易记录的时候,必须保证账本已经有封面了

private void addNote(String note) {

if (list.size() < 1) {

throw new RuntimeException("添加交易记录的时候,必须保证账本已经有封面了");

}

list.add(note);

}

// 展示数据

private void showlist() {

for (String s : list) {

System.out.println(s);

}

}

// 保存到本地硬盘

private void save2Disk() {

}

public static void main(String[] args) {

NoteBook noteBook = new NoteBook();

noteBook.addGenesis("封面");

noteBook.addNote("222给2转账了一百块钱");

noteBook.showlist();

}

}

1. 实现保存功能

增加加载本地数据的方法

private void loadFile() {

try {

ObjectMapper objectMapper = new ObjectMapper();

File file = new File("a.json");

// 判断文件是否存在

if (file.exists() && file.length() > 0) {

// 如果文件存在,读取之前的数据

JavaType javatype = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, String.class);

list = objectMapper.readValue(file, javatype);

}

} catch (IOException e) {

e.printStackTrace();

}

}

在构造函数中加载本地数据

public NoteBook() {

loadFile();

}

增加保存数据到本地的方法

// 保存数据到本地硬盘

private void save2Disk() {

try {

ObjectMapper objectMapper = new ObjectMapper();

File file = new File("a.json");

objectMapper.writeValue(file, list);

} catch (IOException e) {

e.printStackTrace();

}

}

在添加数据时, 调用保存方法

private void addGenesis(String genesis) {

if (list.size() > 0) {

throw new RuntimeException("添加封面的时候,必须保证账本是新的");

}

list.add(genesis);

save2Disk();

}

private void addNote(String note) {

if (list.size() < 1) {

throw new RuntimeException("添加交易记录的时候,必须保证账本已经有封面了");

}

list.add(note);

save2Disk();

}

1. 区块链V2增加Http访问接口

本小节为账本功能提供网络接口, 使用户可以通过页面对账本进行增删改查的操作

1. 增加网络访问接口

@RestController

public class BlockController {

private NoteBook book = new NoteBook();

@RequestMapping(value = "/addGenesis", method = RequestMethod.POST)

public String addGenesis(String genesis) {

try {

book.addGenesis(genesis);

return "success";

} catch (Exception e) {

return "fail:" + e.getMessage();

}

}

@RequestMapping(value = "/addNote", method = RequestMethod.POST)

public String addNote(String note) {

try {

book.addNote(note);

return "success";

} catch (Exception e) {

return "fail:" + e.getMessage();

}

}

@RequestMapping(value = "/showlist", method = RequestMethod.GET)

public ArrayList<String> showlist() {

return book.showlist();

}

}

1. 增加前台页面

引入资料中的静态资源

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8"/>

<title></title>

<link rel="stylesheet" type="text/css" href="css/bootstrap.css"/>

<link rel="stylesheet" type="text/css" href="css/pb.css"/>

<style type="text/css">

body {

margin: 30px;

}

#result {

padding: 15px;

}

</style>

</head>

<body>

<!--输入框-->

<input type="text" class="form-control" id="inputCtr" placeholder="请输入内容" style="width: 600px"> <br>

<!--按钮组-->

<div class="btn-group btn-group-lg" role="group">

<button type="button" class="btn btn-default" onclick="addGenesis()">添加封面</button>

<button type="button" class="btn btn-default" onclick="addNote()">添加记录</button>

<button type="button" class="btn btn-default" onclick="showlist()">展示数据</button>

</div>

<br>

<!--用于展示结果-->

<p class="bg-info" id="result">

</p>

<script src="js/jquery-1.12.4.min.js" type="text/javascript" charset="utf-8"></script>

<script src="js/bootstrap.js" type="text/javascript" charset="utf-8"></script>

<script src="js/pb.js" type="text/javascript" charset="utf-8"></script>

<script src="js/jsrsasign-all-min.js" type="text/javascript" charset="utf-8"></script>

<script>

// 添加封面

function addGenesis() {

// 显示进度条

loading.baosight.showPageLoadingMsg(false)

// 用户输入的内容

var content = $("#inputCtr").val();

// 发起请求

$.post("/addGenesis", "genesis=" + content, function (data) {

//清空输入框

$("#inputCtr").val();

// 展示结果

$("#result").html(data)

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

// 查询最新的数据

showlist();

})

}

// 添加记录

function addNote() {

// 显示进度条

loading.baosight.showPageLoadingMsg(false)

// 用户输入的内容

var content = $("#inputCtr").val();

// 发起请求

$.post("/addNote", "note=" + content, function (data) {

//清空输入框

$("#inputCtr").val();

// 展示结果

$("#result").html(data)

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

// 查询最新的数据

showlist();

})

}

// 展示数据

function showlist() {

// 显示进度条

loading.baosight.showPageLoadingMsg(false)

// 发起请求

$.get("/showlist", function (data) {

// 展示数据

$("#result").html(data)

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

})

}

// 页面加载成功后,查询已有数据

$(function () {

showlist();

})

</script>

</body>

</html>

1. 进度条控件的使用
2. 拷贝资源pb.css和pb.js
3. 将CSS和Js文件引入页面
4. 示例代码

loading.baosight.showPageLoadingMsg(false) // 显示进度条

loading.baosight.hidePageLoadingMsg() // 隐藏进度条

1. 区块链V3增加Hash校验

上一小节实现的功能,很容易导致黑客攻击,篡改数据,这一小节,为数据增加Hash校验功能,增加黑客攻击的难度.

1. 创建Block实体类

public class Block {

public int id;

public String content;

public String hash;

public Block() {

}

public Block(int id, String content, String hash) {

this.id = id;

this.content = content;

this.hash = hash;

}

}

1. Notebook中增加校验方法

// 校验数据

public String check() {

StringBuilder sb = new StringBuilder();

for (Block block : list) {

// 获取内容

String content = block.content;

// 生成Hash

String hashNew = HashUtils.sha256(content);

// 比对hash,如果不一样说明数据内篡改

if (!hashNew.equals(block.hash)) {

sb.append("编号为" + block.id + "的数据有可能被篡改了,请注意防范黑客<br>");

}

}

return sb.toString();

}

1. Controller中增加方法

@RequestMapping(value = "/check", method = RequestMethod.GET)

public String check() {

String check = book.check();

if (StringUtils.isEmpty(check)) {

return "数据是安全的";

}

return check;

}

1. 页面中增加表格用于展示数据

<table class="table">

<thead>

<tr>

<th>编号</th>

<th>内容</th>

<th>哈希值</th>

</tr>

</thead>

<tbody id="idTbody">

</tbody>

</table>

1. 修改页面显示数据的逻辑

function showlist() {

// 获取用户在输入框中输入的内容

var content = $("#idInput").val();

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.get("showlist", function (data) {

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

// 清空数据

$("#idTbody").html("")

// 遍历数据,添加数据

for (var i = 0; i < data.length; i++) {

var id = data[i].id

var content = data[i].content

var hash = data[i].hash

$("#idTbody").append("<tr><th>" + id + "</th><td>" + content + "</td><td>" + hash + "</td></tr>")

}

})

}

1. 页面中增加校验数据的逻辑

// 校验数据

function check() {

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.get("check", function (data) {

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

})

}

1. 区块链V4增加工作量证明

上一小节增加Hash校验后,如果黑客在篡改数据的同时,又修改了Hash,就无法达到保护数据的目的.因此本节,增加工作量证明,即生成的Hash必须符合特定的规则,如必须以0000开头.这个获取特定Hash的过程就是比特币中的挖矿.

1. 修改实体类,增加工作量证明字段

public class Block {

public int id;

public String content;

public String hash;

public int nonce;

public Block() {

}

public Block(int id, String content, String hash, int nonce) {

this.id = id;

this.content = content;

this.hash = hash;

this.nonce = nonce;

}

}

1. Notebook中增加挖矿方法

// 挖矿

private int mine(String content) {

// 求取一个符合特定规则的hash值,并将运算次数返回

for (int i = 0; i < Integer.MAX\_VALUE; i++) {

String s = HashUtils.sha256(i + content);

if (s.startsWith("0000")) {

System.out.println("挖矿成功:" + i);

return i;

} else {

System.out.println("挖矿失败:" + i);

}

}

throw new RuntimeException("挖矿失败");

}

1. Notebook中修改添加的逻辑

int nonce = mine(genesis);

Block block = new Block(list.size() + 1, genesis, HashUtils.sha256(nonce + genesis), nonce);

1. Notebook中修改校验的逻辑

// 校验数据

public String check() {

StringBuilder sb = new StringBuilder();

for (Block block : list) {

// 获取内容

String content = block.content;

// 工作量证明

int nonce = block.nonce;

// 生成Hash

String hashNew = HashUtils.sha256(nonce + content);

// 比对hash,如果不一样说明数据内篡改

if (!hashNew.equals(block.hash)) {

sb.append("编号为" + block.id + "的数据有可能被篡改了,请注意防范黑客<br>");

}

}

return sb.toString();

}

1. 修改页面,表格增加列

<table class="table">

<thead>

<tr>

<th>编号</th>

<th>内容</th>

<th>哈希值</th>

<th>工作量</th>

</tr>

</thead>

<tbody id="idTbody">

</tbody>

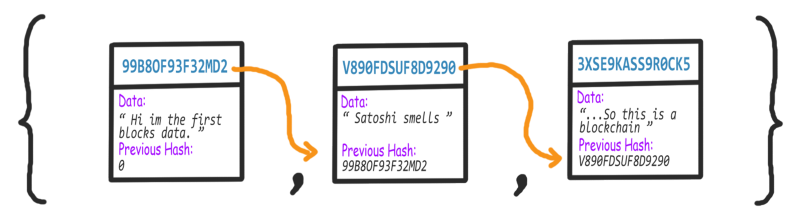
</table>

1. 修改页面,展示数据的逻辑

$("#idTbody").append("<tr><th>" + id + "</th><td>" + content + "</td><td>" + hash + "</td><td>" + nonce + "</td></tr>")

1. 区块链V4形成区块链

本节继续增加黑客攻击的难度, 每个区块都持有上个区块的Hash值,互相咬合,形成一个链条.这样当链条足够长的时候,会大大增加黑客攻击的难度.



1. 修改实体类,增加preHash字段

public class Block {

public int id;

public String content;

public String hash;

public int nonce;

public String preHash;

public Block() {

}

public Block(int id, String content, String hash, int nonce, String preHash) {

this.id = id;

this.content = content;

this.hash = hash;

this.nonce = nonce;

this.preHash = preHash;

}

}

1. Notebook中修改添加的逻辑

public void addGenesis(String genesis) {

if (list.size() > 0) {

throw new RuntimeException("添加封面的时候,必须保证账本是新的");

}

String preHash = "0000000000000000000000000000000000000000000000000000000000000000";

int nonce = mine(preHash + genesis);

Block block = new Block(list.size() + 1, genesis, HashUtils.sha256(nonce + preHash + genesis), nonce, preHash);

list.add(block);

save2Disk();

}

public void addNote(String note) {

if (list.size() < 1) {

throw new RuntimeException("添加交易记录的时候,必须保证账本已经有封面了");

}

Block preBlock = list.get(list.size() - 1);

String preHash = preBlock.hash;

int nonce = mine(preHash + note);

Block block = new Block(list.size() + 1, note, HashUtils.sha256(nonce + preHash + note), nonce, preHash);

list.add(block);

save2Disk();

}

1. Notebook中修改校验的逻辑

public String check() {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < list.size(); i++) {

Block currentBlock = list.get(i);

String savedHash = currentBlock.hash;

String content = currentBlock.content;

int nonce = currentBlock.nonce;

String preHash = currentBlock.preHash;

int id = currentBlock.id;

if (i == 0) {

// 创世区块,校验hash

preHash = "0000000000000000000000000000000000000000000000000000000000000000";

String caculatedHash = HashUtils.sha256(nonce + preHash + content);

if (!savedHash.equals(caculatedHash)) {

sb.append("编号为" + id + "的数据有可能被篡改了,请注意防范黑客<br>");

}

} else {

// 其他区块,校验hash,preHash

String caculatedHash = HashUtils.sha256(nonce + preHash + content);

if (!savedHash.equals(caculatedHash)) {

sb.append("编号为" + id + "的hash有问题,请注意防范黑客<br>");

}

Block preBlock = list.get(i - 1);

String preBlockHash = preBlock.hash;

if (!preBlockHash.equals(preHash)) {

sb.append("编号为" + id + "的preHash有问题,请注意防范黑客<br>");

}

}

}

return sb.toString();

}

1. 修改页面,表格增加列

<tr>

<th>编号</th>

<th>内容</th>

<th>哈希值</th>

<th>工作量</th>

<th>上一个Hash</th>

</tr>

1. 修改页面,展示数据的逻辑

$("#idTbody").append("<tr><th>" + id + "</th><td>" + content + "</td><td>" + hash + "</td><td>" + nonce + "</td><td>" + preHash + "</td></tr>")

1. 钱包功能

前面的章节,存储数据时,直接可见转账双方的信息,本节通过非对称加密技术和签名技术隐藏交易双方的真实信息.

可以简单理解为 非对称加密中的公钥就是钱包地址,非对称加密中的私钥就是钱包的密码.

1. 钱包实体类

public class Wallet {

// 公钥

public PublicKey publicKey;

// 私钥

public PrivateKey privateKey;

public Wallet(String name) {

// 保存公私钥的文件

File pubFile = new File(name + ".pub");

File priFile = new File(name + ".pri");

// 如果文件不存在,说明没有公私钥,就创建公私钥文件

if (!pubFile.exists() || pubFile.length() == 0 || !priFile.exists() || priFile.length() == 0) {

RSAUtils.generateKeys("RSA", name + ".pri", name + ".pub");

}

// 从文件中读取公私钥

publicKey = RSAUtils.getPublicKeyFromFile("RSA", name + ".pub");

privateKey = RSAUtils.getPrivateKey("RSA", name + ".pri");

}

// 转账

public Transaction sendMoney(String receiverPublickKey, String content) {

// 将公钥转为字符串

String publicKeyEncode = Base64.encode(publicKey.getEncoded());

// 生成签名

String signature = RSAUtils.getSignature("SHA256withRSA", privateKey, content);

// 生成交易对象

Transaction transaction = new Transaction(publicKeyEncode, receiverPublickKey, signature, content);

return transaction;

}

}

1. 交易实体类

public class Transaction {

// 付款方的公钥

public String senderPublickKey;

// 收款方的公钥

public String receiverPublickKey;

// 签名

public String signature;

// 转账信息

public String content;

public Transaction() {

}

public Transaction(String senderPublickKey, String receiverPublickKey, String signature, String content) {

this.senderPublickKey = senderPublickKey;

this.receiverPublickKey = receiverPublickKey;

this.signature = signature;

this.content = content;

}

// 校验交易是否正确

public boolean verify() {

PublicKey sender = RSAUtils.getPublicKeyFromString("RSA", senderPublickKey);

return RSAUtils.verify("SHA256withRSA", sender, content, signature);

}

public String getSenderPublickKey() {

return senderPublickKey;

}

public void setSenderPublickKey(String senderPublickKey) {

this.senderPublickKey = senderPublickKey;

}

public String getReceiverPublickKey() {

return receiverPublickKey;

}

public void setReceiverPublickKey(String receiverPublickKey) {

this.receiverPublickKey = receiverPublickKey;

}

public String getSignature() {

return signature;

}

public void setSignature(String signature) {

this.signature = signature;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

@Override

public String toString() {

return "Transaction{" +

"senderPublickKey='" + senderPublickKey + '\'' +

", receiverPublickKey='" + receiverPublickKey + '\'' +

", signature='" + signature + '\'' +

", content='" + content + '\'' +

'}';

}

}

1. 区块链V5增加转账功能

本小节实现网页转账的功能

jsrsasign插件 : <https://github.com/kjur/jsrsasign>

1. 在页面引入插件

<script src="js/jsrsasign-all-min.js" type="text/javascript" charset="utf-8"></script>

1. 修改页面,增加输入key的文本域

<label>

发送方的私钥

</label>

<textarea class="form-control" id="senderPrivateKey">

</textarea>

<label>

发送方的公钥

</label>

<textarea class="form-control" id="senderPublicKey">

</textarea><label>

接收方的公钥

</label>

<textarea class="form-control" id="receiverPublicKey">

</textarea>

<label>

转账信息

</label>

1. 修改页面添加记录的逻辑

function addNote() {

// 获取输入框的内容

var senderPrivateKey = $("#senderPrivateKey").val();

var senderPublicKey = $("#senderPublicKey").val();

var receiverPublicKey = $("#receiverPublicKey").val();

var content = $("#idInput").val()

// 生成私钥

var prvKey = KEYUTIL.getKey(senderPrivateKey);

// 指定生成签名使用的算法

var sig = new KJUR.crypto.Signature({"alg": "SHA256withRSA"});

// 初始化私钥

sig.init(prvKey);

// 传入原文

sig.updateString(content)

// 生成签名数据

var sigValueHex = sig.sign()

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.post("addNote", {

senderPublickKey: senderPublicKey,

receiverPublickKey: receiverPublicKey,

signature: sigValueHex,

content: content,

}, function (data) {

// 清空输入框

$("#idInput").val("");

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

// 展示最新的数据

showlist()

})

}

1. 修改RsaUtils,增加js生成签名,验证签名的方法

/\*\*

\* 生成公私钥,并保存在文件中,JS版本

\*

\* @param algorithm : 算法

\* @param privateKeyPath : 保存私钥的文件路径

\* @param publicKeyPath : 保存公钥的文件路径

\*/

public static void generateKeysJS(String algorithm, String privateKeyPath, String publicKeyPath) {

try {

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(algorithm);

KeyPair keyPair = keyPairGenerator.generateKeyPair();

PrivateKey privateKey = keyPair.getPrivate();

PublicKey publicKey = keyPair.getPublic();

byte[] privateKeyEncoded = privateKey.getEncoded();

byte[] publicKeyEncoded = publicKey.getEncoded();

String encodePrivateKey = Base64.encode(privateKeyEncoded);

String encodePublicKey = Base64.encode(publicKeyEncoded);

FileUtils.writeStringToFile(new File(privateKeyPath), "-----BEGIN PRIVATE KEY-----\n" + encodePrivateKey + "\n-----END PRIVATE KEY-----", "UTF-8");

FileUtils.writeStringToFile(new File(publicKeyPath), encodePublicKey, "UTF-8");

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 校验签名,JS版本

\*

\* @param algorithm : 加密算法(SHA256withRSA)

\* @param publicKey : 公钥

\* @param originalData : 原文

\* @param signaturedData : 签名

\* @return : 签名是否正确

\*/

public static boolean verifyDataJS(String algorithm, PublicKey publicKey, String originalData, String signaturedData) {

try {

// 获取签名对象

Signature signature = Signature.getInstance(algorithm);

// 传入公钥

signature.initVerify(publicKey);

// 传入原文

signature.update(originalData.getBytes());

// 校验数据

return signature.verify(toBytes(signaturedData));

} catch (Exception e) {

e.printStackTrace();

}

return false;

}

//转换方法

public static byte[] toBytes(String str) {

if (str == null || trim(str) == "") {

return new byte[0];

}

byte[] bytes = new byte[str.length() / 2];

for (int i = 0; i < str.length() / 2; i++) {

String subStr = str.substring(i \* 2, i \* 2 + 2);

bytes[i] = (byte) Integer.parseInt(subStr, 16);

}

return bytes;

}

public static String trim(String str) {

int startIndex = 0;

int endIndex = str.length() - 1;

boolean startFound = false;

while (startIndex <= endIndex) {

int index;

if (!startFound) {

index = startIndex;

} else {

index = endIndex;

}

char it = str.charAt(index);

boolean match = it <= ' ';

if (!startFound) {

if (!match)

startFound = true;

else

startIndex += 1;

} else {

if (!match)

break;

else

endIndex -= 1;

}

}

return str.substring(startIndex, endIndex + 1);

}

修改Controller,添加记录的逻辑

@RequestMapping(value = "/addNote", method = RequestMethod.POST)

public String addNote(Transaction transaction) {

try {

// 校验交易数据

if (transaction.verify()) {

// 将交易数据转为字符串

ObjectMapper objectMapper = new ObjectMapper();

String note = objectMapper.writeValueAsString(transaction);

// 添加数据

book.addNote(note);

return "success";

} else {

throw new RuntimeException("交易数据校验失败");

}

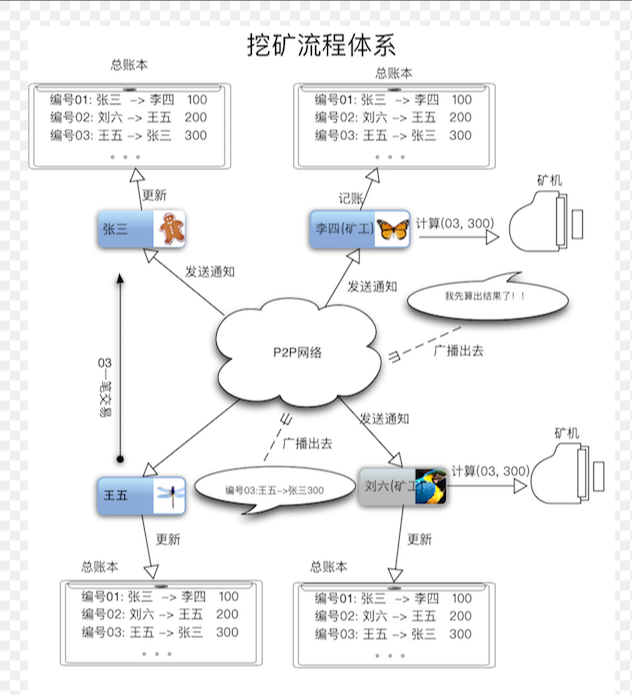
} catch (Exception e) {

return "fail:" + e.getMessage();

}

}

1. WebSocket简介



前面的章节章节中,我们的转账信息和生成的区块只是保存在本地,事实上我们需要把这些信息广播出去,告知其他节点.其他节点上的转账信息和生成的区块也要告知我们,这样我们就需要通过WebSocket和其他节点之间进行通信.

WebSocket是一种通信协议,可以实现双向通信.

具体介绍 : <http://www.ruanyifeng.com/blog/2017/05/websocket.html>

本项目使用Java WebSockets框架. <https://github.com/TooTallNate/Java-WebSocket>

* 快速入门案例

1. 引入坐标

compile "org.java-websocket:Java-WebSocket:1.3.8"

1. 创建Server

public class MyServer extends WebSocketServer {

// 服务器端口

private int port;

public MyServer(int port) {

super(new InetSocketAddress(port));

this.port = port;

}

@Override

public void onOpen(WebSocket conn, ClientHandshake handshake) {

System.out.println("WebSocket服务器\_\_" + port + "\_\_打开了一个连接,对方是:" + conn.getRemoteSocketAddress().toString());

}

@Override

public void onClose(WebSocket conn, int code, String reason, boolean remote) {

System.out.println("WebSocket服务器\_\_" + port + "\_\_关闭了一个连接,对方是:" + conn.getRemoteSocketAddress().toString());

}

@Override

public void onMessage(WebSocket conn, String message) {

System.out.println("WebSocket服务器\_\_" + port + "\_\_收到了消息,对方是:" + conn.getRemoteSocketAddress().toString() + "\_\_消息的内容是:" + message);

}

@Override

public void onError(WebSocket conn, Exception ex) {

System.out.println("WebSocket服务器\_\_" + port + "\_\_发生了错误\_\_原因:" + ex.getMessage());

}

@Override

public void onStart() {

System.out.println("WebSocket服务器\_\_" + port + "\_\_启动成功");

}

// 开启服务器

public void startServer() {

new Thread(this).start();

}

}

1. 创建Client

public class MyClient extends WebSocketClient {

private String name;

/\*\*

\* @param serverUri : 要连接的服务器的地址

\* @param name : 本客户端的名字

\*/

public MyClient(URI serverUri, String name) {

super(serverUri);

this.name = name;

}

@Override

public void onOpen(ServerHandshake handshakedata) {

System.out.println("WebSocket客户端\_\_" + name + "\_连接成功");

}

@Override

public void onMessage(String message) {

System.out.println("WebSocket客户端\_\_" + name + "\_收到消息,内容是:" + message);

}

@Override

public void onClose(int code, String reason, boolean remote) {

System.out.println("WebSocket客户端\_\_" + name + "\_连接关闭");

}

@Override

public void onError(Exception ex) {

System.out.println("WebSocket客户端\_\_" + name + "\_发生错误");

}

}

1. 创建测试类

public class MyTest {

public static void main(String[] args) {

try {

// 创建并开启服务器

MyServer server = new MyServer(8000);

server.startServer();

// 指定服务器地址

URI uri = new URI("ws://localhost:8000");

// 创建客户端

MyClient client1 = new MyClient(uri, "1111");

MyClient client2 = new MyClient(uri, "2222");

// 客户端连接服务器

client1.connect();

client2.connect();

// 避免连接尚未成功,就发送消息,导致的发送失败

Thread.sleep(1000);

// 服务器发送广播

// server.broadcast("这是来自服务器的广播");

// 客户端发送消息给服务器

client1.send("这是一号客户端发送的消息");

} catch (Exception e) {

e.printStackTrace();

}

}

}

1. SpringBoot集成WebSocket
2. 修改Application类,可以手动指定启动端口号

public class WsdemoApplication {

public static String port;

public static void main(String[] args) {

// 获取用户输入的内容,作为端口号

Scanner scanner = new Scanner(System.in);

port = scanner.nextLine();

// 启动应用

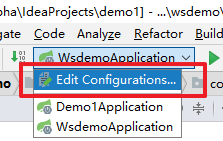
new SpringApplicationBuilder(WsdemoApplication.class).properties("server.port=" + port).run(args);

}

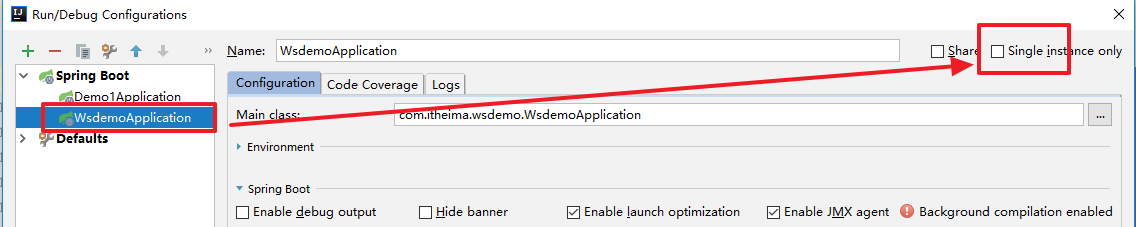
}

1. 修改Springboot配置,使其可以开启多实例

编辑配置



找到对应的Application类,修改配置



1. 创建Controller,实现注册节点 / 连接 / 广播功能

@RestController

public class DemoController {

private MyServer server;

// 创建服务器,开启服务器

@PostConstruct// 创建Controller后调用该方法

public void init() {

// webSocket服务器占用的端口号 = SpringBoot占用的端口号 + 1

server = new MyServer(Integer.parseInt(WsdemoApplication.port) + 1);

server.startServer();

}

private HashSet<String> set = new HashSet<>();

// 注册节点

@RequestMapping("/regist")

public String regist(String port) {

set.add(port);

return "节点:" + port + "注册成功";

}

// 连接

@RequestMapping("/conn")

public String conn() {

try {

// 遍历集合,连接其他WebSocket服务器

for (String port : set) {

URI uri = new URI("ws://localhost:" + port);

MyClient client = new MyClient(uri, "连接到\_\_"+port+"\_\_服务器的客户端");

client.connect();

}

return "连接成功";

} catch (URISyntaxException e) {

return "连接失败:" + e.getMessage();

}

}

// 广播

@RequestMapping("/broadcast")

public String broadcast(String msg) {

server.broadcast(msg);

return "发送消息成功";

}

}

1. 本项目集成WebSocket
2. 参考上例将WebSocket集成进本项目
3. 修改页面,增加对应的按钮

<input type="text" class="form-control" id="idNode"> <br>

<div class="btn-group btn-group-lg" role="group">

<button type="button" class="btn btn-default" onclick="regist()">注册节点</button>

<button type="button" class="btn btn-default" onclick="conn()">连接</button>

<button type="button" class="btn btn-default" onclick="broadcast()">发送广播</button>

</div>

1. 修改页面,增加对应的点击事件

function regist() {

// 获取用户在输入框中输入的内容

var content = $("#idNode").val();

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.post("regist", "port=" + content, function (data) {

// 清空输入框

$("#idNode").val("");

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

})

}

function conn() {

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.post("conn", function (data) {

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

})

}

function broadcast() {

// 获取用户在输入框中输入的内容

var content = $("#idNode").val();

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.post("broadcast", "msg=" + content, function (data) {

// 清空输入框

$("#idNode").val("");

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

})

}

1. 同步区块链数据

当一个新节点启动的时候,是没有任何数据的,此时可以发送同步请求,从其他节点获取区块链数据.本节实现垓功能.

1. 页面上增加同步按钮,并实现点击事件

function syncData() {

// 打开进度条

loading.baosight.showPageLoadingMsg(false);

// 发起请求

$.post("syncData", function (data) {

// 隐藏进度条

loading.baosight.hidePageLoadingMsg()

$("#result").html(data)

})

}

1. 修改Notebook,使其为单例模式

private NoteBook() {

loadFile();

}

private static volatile NoteBook instance;

public static NoteBook getInstance() {

if (instance == null) {

synchronized (NoteBook.class) {

if (instance == null) {

instance = new NoteBook();

}

}

}

return instance;

}

1. 修改Controller,增加同步方法

// 向其他节点发起请求,要求获取区块链的数据

@RequestMapping("/syncData")

public String syncData() {

for (MyClient client : clients) {

client.send("请把你最新的区块链数据发给我一份");

}

return "发送消息成功";

}

1. 增加MessageBean,用于传递数据

public class MessageBean {

// 1 ,服务器发送给客户端的最新的区块链数据

public int type;

public String msg;

public MessageBean() {

}

public MessageBean(int type, String msg) {

this.type = type;

this.msg = msg;

}

}

1. 修改Server接收消息的方法

public void onMessage(WebSocket conn, String message) {

try {

System.out.println("WebSocket服务器\_\_" + port + "\_\_收到了消息,对方是:" + conn.getRemoteSocketAddress().toString() + "\_\_消息的内容是:" + message);

if ("请把你最新的区块链数据发给我一份".equals(message)) {

// 获取本地区块链数据

NoteBook noteBook = NoteBook.getInstance();

ArrayList<Block> list = noteBook.showlist();

// 封装数据

ObjectMapper objectMapper = new ObjectMapper();

String chain = objectMapper.writeValueAsString(list);

MessageBean bean = new MessageBean(1, chain);

String msg = objectMapper.writeValueAsString(bean);

// 广播数据

broadcast(msg);

}

} catch (JsonProcessingException e) {

e.printStackTrace();

}

}

1. 修改Client接收消息的方法

public void onMessage(String message) {

try {

System.out.println("WebSocket客户端\_\_" + name + "\_收到消息,内容是:" + message);

if (!StringUtils.isEmpty(message)) {

// 解析消息对象

ObjectMapper objectMapper = new ObjectMapper();

MessageBean bean = objectMapper.readValue(message, MessageBean.class);

// 判断消息类型

if (bean.type == 1) {

// 获取数据

JavaType javaType = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, Block.class);

ArrayList<Block> list = objectMapper.readValue(bean.msg, javaType);

// 对数据进行比对

NoteBook book = NoteBook.getInstance();

book.compareList(list);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

1. 修改Notebook,增加比对数据的方法

// 比对数据,如果新的数据长度更长,就替换掉本地数据

public void compareList(ArrayList<Block> newList) {

System.out.println("参数:" + newList.size());

if (newList.size() > list.size()) {

list = newList;

}

System.out.println("list:" + list.size());

}

1. 广播交易数据

我们添加完交易后,将交易数据广播给其他节点

1. 修改Controller,添加记录的方法

@RequestMapping(value = "/addNote", method = RequestMethod.POST)

public String addNote(Transaction transaction) {

try {

// 校验交易数据

if (transaction.verify()) {

// 将交易数据转为字符串

ObjectMapper objectMapper = new ObjectMapper();

String note = objectMapper.writeValueAsString(transaction);

// 封装消息Bean

MessageBean messageBean = new MessageBean(2, note);

String bean = objectMapper.writeValueAsString(messageBean);

// 广播交易数据

server.broadcast(bean);

// 添加数据

book.addNote(note);

return "success";

} else {

throw new RuntimeException("交易数据校验失败");

}

} catch (Exception e) {

return "fail:" + e.getMessage();

}

}

1. 修改Client,接收消息的方法

public void onMessage(String message) {

try {

System.out.println("WebSocket客户端\_\_" + name + "\_收到消息,内容是:" + message);

if (!StringUtils.isEmpty(message)) {

NoteBook book = NoteBook.getInstance();

// 解析消息对象

ObjectMapper objectMapper = new ObjectMapper();

MessageBean bean = objectMapper.readValue(message, MessageBean.class);

// 判断消息类型

if (bean.type == 1) {

// 获取数据

JavaType javaType = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, Block.class);

ArrayList<Block> list = objectMapper.readValue(bean.msg, javaType);

// 对数据进行比对

book.compareList(list);

} else if (bean.type == 2) {

// 获取交易数据

Transaction transaction = objectMapper.readValue(bean.msg, Transaction.class);

// 验证交易数据

if (transaction.verify()) {

// 添加到区块

book.addNote(bean.msg);

}

}

}

} catch (Exception e) {

e.printStackTrace();

}

}