

臺北市立松山高中 學生科學作品展覽研究報告書

應用類（電腦與資訊學科）

區塊鏈研究暨

開發現行勞基法適用之智能合約

作者：二年十班 牟展佑

曾子屏

顏紹宇

指導老師：于思寧

摘要

隨著科技的發展，網路逐漸引領世界，而如何在網路上取得信任，便成為了眾人注目的難題。

區塊鏈的概念，使我們能夠在沒有任何第三方機構的認可下，達到節點間的信任。藉由從底層研究區塊鏈，我們得以了解其作為數字貨幣，真正的基礎架構與優缺點；甚至能應用此觀念與技術，透過以太坊及智能合約，打造一平台，來解決勞資糾紛的問題，讓雙方在此合約裡，完全的服從勞基法的規定，技術上達到真正的和諧共處！

壹、研究動機

近年來，人們口口聲聲喊著比特幣，看著節節高漲的價格，各方的聲音與意見在網路上來回穿梭，究竟，比特幣是什麼？區塊鏈又是什麼？

又，區塊鏈的特性除了作為貨幣，難道近期僅能成為一炒作的產品嗎？是否在「去中心化」、「透明公開」、「不可否認」的特色裡，我們能夠找到一條不一樣的路，將這門技術發揚光大呢？於此同時，眼看勞資新聞越演越烈，權力、資訊的不對等；政府、勞方、資方三者間的信任關係劣化。區塊鏈與智能合約，是否可以成為其另一出路？

貳、研究目的

依照現行勞基法的規範，將可能出現：資方強迫勞工加班，卻不會給予加班費，而是給予無限延遲之假期；資方給予的假期期限內公司倒閉，而勞方得不到其既得利益；資方給予之加班費不符合約定等等問題。

為了瞭解區塊鏈與改善上述問題，我們從根本研究區塊鏈，並希望透過以太坊與智能合約，期望能達到以下目的：

- 一、根本了解區塊鏈架構：利用 python 實作一區塊鏈，根本上理解與認識之。
- 二、提供勞資一平等平台：提供勞資雙方一公平、公正、公開之平台與介面。
- 三、提供去中心合約架構：利用以太坊與智能合約，撰一份不可否認的合約。
- 四、推廣區塊鏈與其價值：藉由真正應用區塊鏈，讓大眾認識與學習應用之。

參、實驗器材

一、硬體設備：

紙、筆、電腦、伺服器

二、軟體與其配置：

1. Anaconda3：python3、jupyter notebook、json 套件、textwrap 套件、time、uuid 套件、flask 套件、urllib.parse 套件、requests 套件

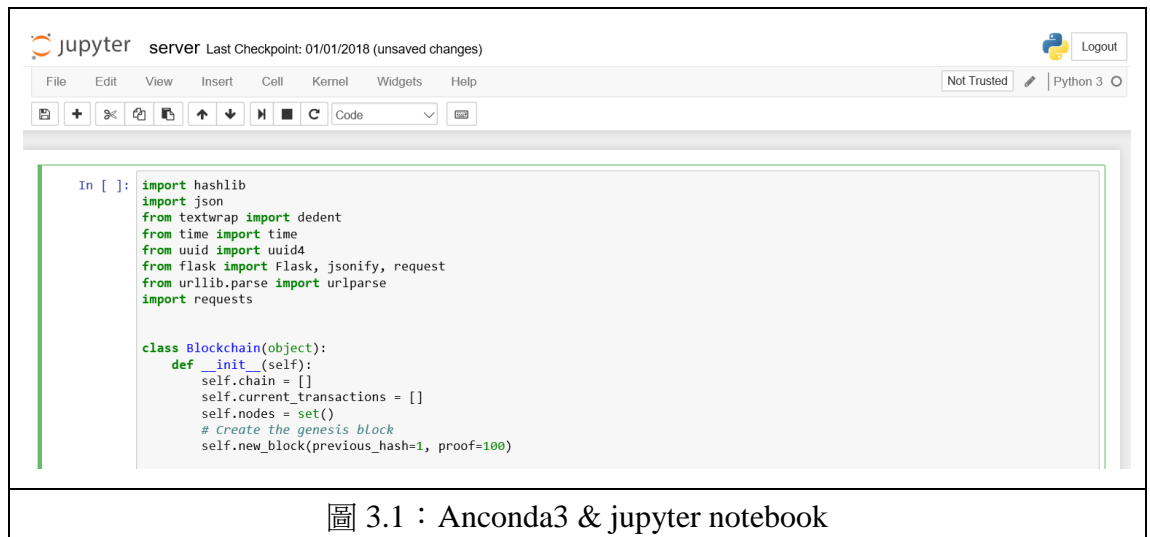
2. visual studio code：python 0.9.1、solidity 0.0.31

3. Mist：3 Accounts

4. Ethereum Remix：https://remix.ethereum.org/

5. Geth：https://geth.ethereum.org/

三、環境截圖：



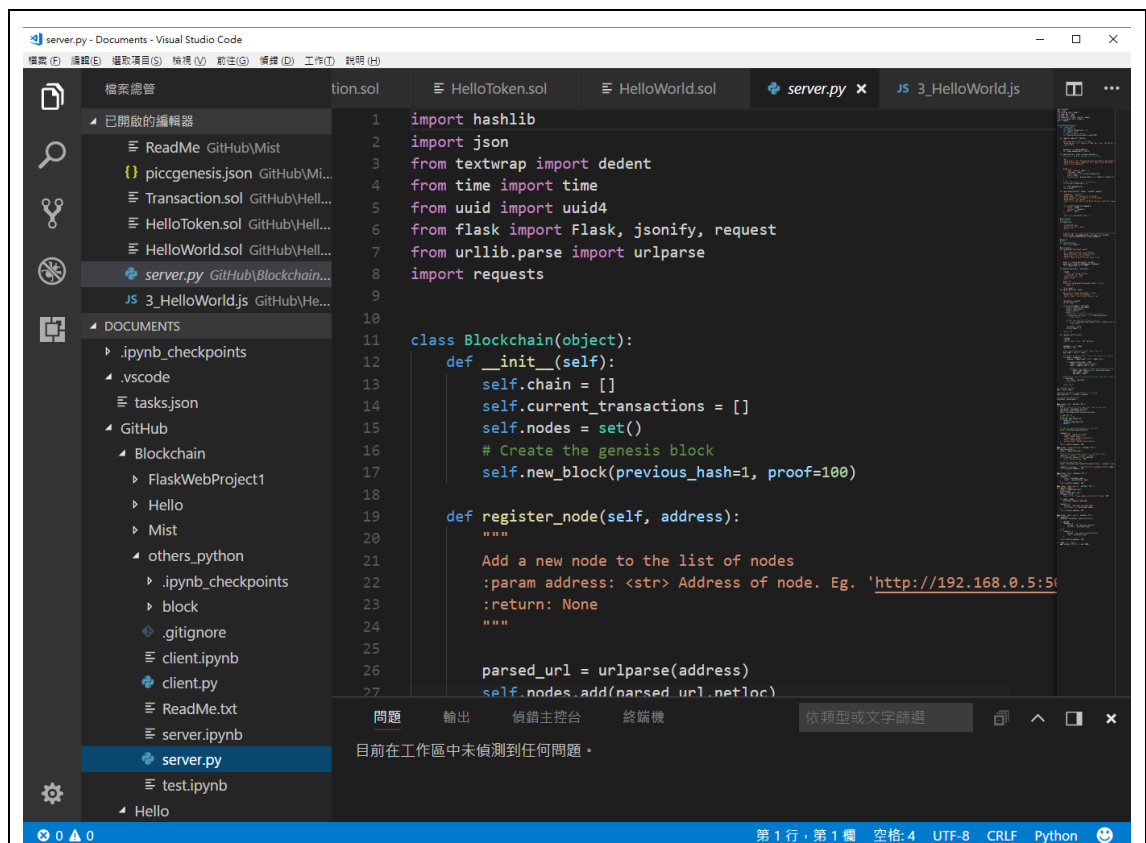


圖 3.2 : visual studio code with python 0.9.1



圖 3.3 : Mist

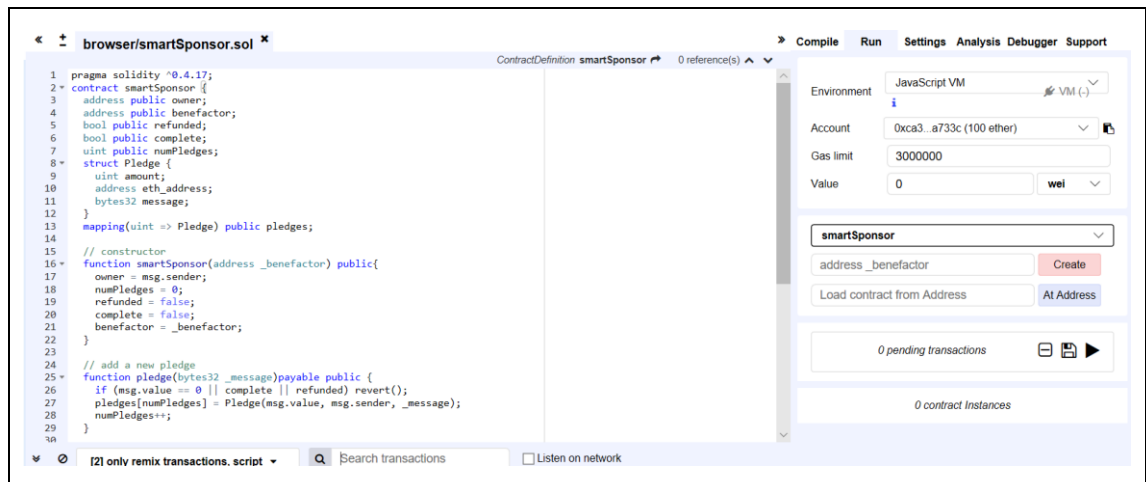


圖 3.4 : Ethereum Remix

[Go Ethereum](#)
[Install](#)
[Downloads](#)

Go Ethereum


Official Go implementation of the Ethereum protocol

[View on GitHub](#)
[Chat on Gitter](#)

What is Ethereum?

Ethereum is a decentralized platform that runs smart contracts, applications that run exactly as programmed without possibility of downtime, censorship, fraud or third party interference.

See [our website](#) or [read the docs](#) for more infos!



What is Go Ethereum?

Go Ethereum is one of the three original implementations (along with C++ and Python) of the Ethereum protocol. It is written in Go, fully open source and licensed under the GNU LGPL v3.

See [our repository](#) and [downloads section](#) for the code!




圖 3.5 : Geth

肆、研究過程與方法

一、基本名詞介紹：

（一）區塊鏈（BlockChain）：

區塊鏈（BlockChain）原本只是比特幣網路的一種記帳技術，近幾年來卻在金融、知識產權、數據交易、電子證照、慈善機構、新能源等領域引起了廣泛的關注。其不外乎有兩大特色：

- 1．區塊鏈具有去中心化的特徵，不以參與交易任何一方為中心，這帶來了效率的提升與成本的降低，直接增加了企業的利潤。
- 2．區塊鏈具有去信任的特徵，也就是假定交易中的任何一方都是不可信任的。透過記錄不可抵賴的交易的信息，來使交易各方遵守誠信。

區塊鏈（BlockChain）是透過「去中心化」和「去信任化」的方式來共同維護資料庫的技術，其將資料庫分別放在不同的節點裡保存，並互相監控數據，資料更動須其餘節點的共識同意。它讓交易過程中每個節點的每一筆交易，都能透明、安全地被紀錄下來。而各交易被廣播到全網後一段時間，會被收進一個區塊（Block），和全部的節點複製共享；隨著時間區塊一個一個生成形成一條鏈（Chain）。而所有節點都會擁有所有交易紀錄，共同驗證鏈上資料的正確性，故極難利用少數節點竄改歷史數據。而後續我們將對此進行研究與討論。

（二）Python：

Python，是一種廣泛使用的高階程式語言，屬於通用型程式語言，由 Guido van Rossum 創造，第一版釋出於 1991 年。作為一種直譯式語言，Python 的設計哲學強調代碼的可讀性和簡潔的語法。相比 C++ 或 Java，Python 讓開發者能夠用更少的代碼表達想法。

（三）Anaconda3：

Anaconda 是一種 Python 語言的免費增值開源發行版，用於進行大規模數據處理，預測分析，和科學計算，致力於簡化包的管理和部署。

二、區塊鏈代碼實作：

（一）區塊架構：

1 摘要：

完全以端對端（peer-to-peer）技術實現的電子現金系統允許線上支付可以不必透過金融機構而直接從某一方傳送。雖然數位簽章（digital signature）解決了部分的問題，但若需要信任的第三方才能避免雙重支付（double-spending）的話，這個系統就沒有價值了。我們提出了一個端對端的網路來解決雙重支付的問題，此網路透過以下方式對每一筆交易做時間戳記（timestamp），也就是將每一筆交易雜湊（hash）到一個「基於雜湊的工作量證明」（hash-based-proof-work）組成的一條不斷延伸的鏈（chain）上，除非重新完成所有工作量證明，這些紀錄將不能被更改。最長的鏈不僅扮演了一連串目擊事件的證明，也證明了它來自於一個 CPU 運算能力很大的池。只要多數的 CPU 運算能力都由不會聯合攻擊網路的節點（node）所控制的話，誠實的節點就會產生一條最長且超越其他攻擊者的鏈。這個網路本身需要的基礎建設很少，訊息會依據最大努力原則（best effort basis）廣播出去，而節點可自由選擇離開或重新加入網路且接受最長的工作量證明所組成的鏈作為節點離開時所發生的交易事件之證明。

2．介紹

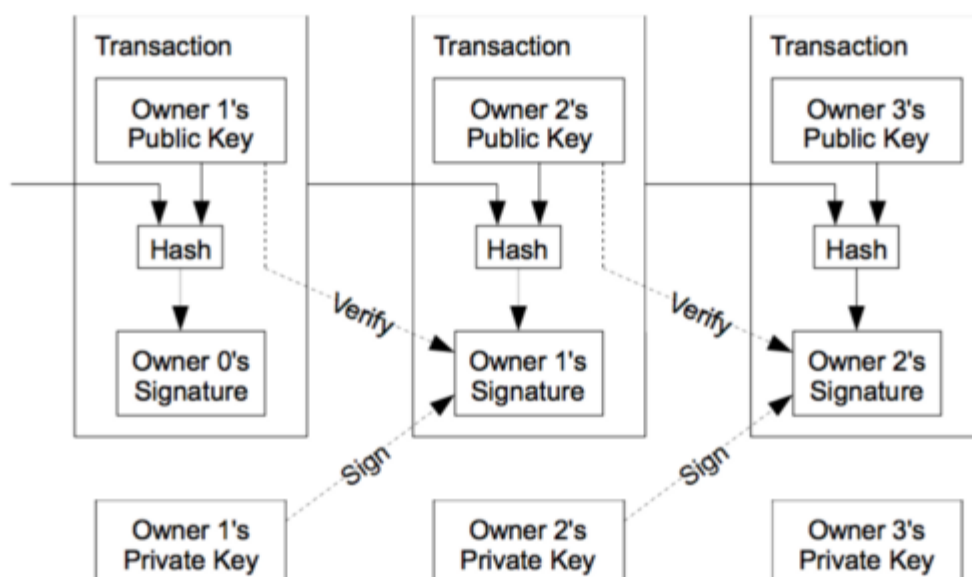
網際網路的商業應用幾乎都仰賴金融機構作為處理電子支付之信任的第三方。雖然系統在大多數情況下都能運作良好，但這類系統仍有以信任為基礎的模式這項固有的缺點。由於金融機構無法避免出面調解爭議，因此完全不可逆的交易並不真的可行。因為調解成本而提高的交易成本，將會限制最小的實際交易規模、限制了日常小額支付、失去為不可逆之服務進行的不可逆支付的能力進而造成更廣泛的成本。可逆性的服務讓信任的需求增加，商人必須更加警惕自己的客戶，因此會向他們索取完全不必要的個人資訊。一定比例的詐騙可以被接受為無法避免的。以上這些成本及支付的不確定性在使用實體貨幣時皆可避免，但透過一個缺乏第三方信任的溝通渠道進行支付的機制並不存在。

我們所需要的是一個以加密證明取代信任的電子支付系統，且允許任何達成共識的雙方能夠直接交易，而不需信任的第三方參與。計算不可逆（computationally impractical to reverse）的交易可以避免賣方受騙，而常規性托管機制（routine escrow mechanisms）則可以輕易地

讓買方被保護。在這篇論文中，我們透過一個端對端的分散式時間戳記伺服器去產生按時間排序的交易之計算證明進而解決了雙重支付的問題。只要誠實的節點能夠共同合作去控制比攻擊者族群更多的 CPU 運算能力，這個系統就會是安全的。

3 · 交易 (Transactions)

我們定義一顆電子貨幣為一串數位簽章。每位擁有者為前一個交易及下一位擁有者的 公開金鑰 (public key) 簽署一個雜湊的數位簽章，並將之加入一顆貨幣的尾端，透過上述的方式將貨幣發送給下一位擁有者。收款者可以透過檢查簽章 (signature) 來驗證該鏈的擁有者。



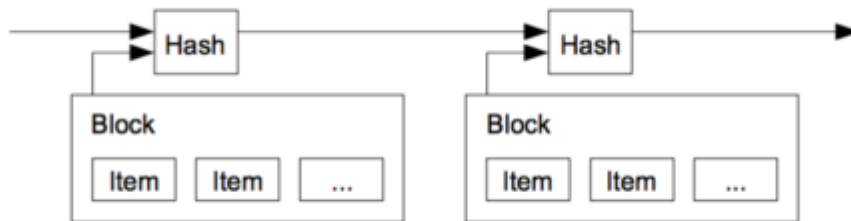
▲ 交易 (Transactions)

這個過程的問題在於收款者無法驗證擁有者是否對這顆貨幣進行雙重支付。一般解法為導入一個信任的中央權威機構或是造幣廠來驗證每一筆交易，以防止雙重支付。在每一筆交易之後，貨幣必須送回造幣廠以發行新貨幣，且只有直接從造幣廠發行的貨幣才會被信任為沒有雙重支付。然而，這個解法的問題在於整個金錢系統的命運皆仰賴在像銀行一樣曾經手每筆交易的造幣廠公司。我們需要一個方法讓收款者知道前一位擁有者並沒有簽署更早發生的交易，由於我們計較的是在此之前發生的交易，因此不用在意在此之後是否有雙重支付。證實交易存在的唯一方法是去意識到所有交易的發生，而在以造幣廠為基礎的模式中，造幣廠必須意識到所有交易並決定交易完成的先後

順序。為了在沒有信任的第三方的之情況下達到目的，交易必須被公告，且我們需要一個讓所有參與者在他們接收的順序上達成共識的系統。收款者需要確保在交易期間絕大多數的節點都認同該交易是首次出現。

4 · 時間戳記伺服器 (Timestamp Server)

我們提出的解法先由時間戳記伺服器說起。時間戳記伺服器會將由很多交易組成的區塊 (block) 所對應之雜湊加上時間戳記，並如同在報紙或 Usenet 中發送的方法將雜湊進行廣播。時間戳記證明了資料在進入雜湊時必然存在。每個時間戳記應將前一個時間戳記加入其雜湊中，之後的每一個時間戳記都和前一個時間戳記進行增強 (reinforce)，而形成了一條鏈。

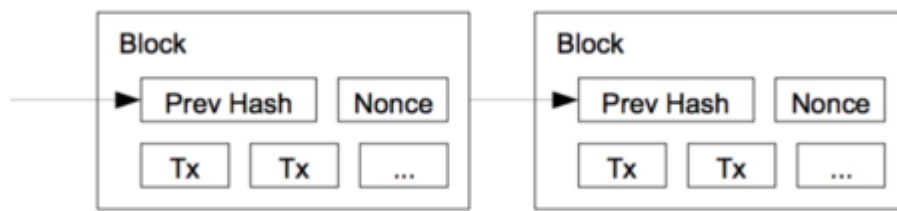


▲時間戳記伺服器 (Timestamp Server)

5 · 工作量證明 (Proof-of-Work)

為了在端對端的基礎上去實現一個分散式時間戳記伺服器，我們需使用一個類似 Adam Back 提出的 Hashcash 技術的工作量證明系統，而非像報紙或 Usenet 的發送方法。工作量證明包括掃描一個數值，以 SHA-256 安全雜湊演算法 (Secure Hash Algorithm) 為例，雜湊值以一個或多個 0 開始。平均而言，隨著 0 的數目上升，所需之工作量將呈指數遞增，且可藉由執行一次雜湊運算即完成驗證。

在我們的时间戳記網路中，我們藉由增加區塊中 nonce 的數量直到提供區塊所對應之雜湊足夠數量的 0 之 nonce 值被發現來完成工作量證明。一旦 CPU 的貢獻擴大到足以滿足工作量證明，除非重新完成一定的工作量否則該區塊無法被更改。當後來的區塊在此之後被鏈結上，若想更改區塊，則需要完成之後所有區塊的工作量。



▲工作量證明（Proof of Work）

工作量證明也解決了多數決代表的問題。如果多數是建立在一個 IP 位址對應一票的基礎上，則會被任何可以擁有許多 IP 者所推翻。工作量證明本質上是一個 CPU 對應一票。多數決代表為最長的鏈，因為最長的鏈包含了最大的工作量。如果大多數的 CPU 運算能力是由誠實的節點所控制，則此誠實的鏈將會成長為最快速且超越任何一條競爭者的鏈。若要修改先前的區塊，攻擊者必須重新完成該區塊及在它之後的所有區塊的工作量證明，並趕上且超越誠實節點之工作量。我們稍後會說明較慢的攻擊者趕上的機率會隨著之後的區塊增加而呈指數消遞減。

為了解決因硬體速度的增加及節點參與網路的程度起伏之問題，工作量證明的難度是由每小時平均產生之區塊數量而決定。如果區塊的產生越快速，則難度也會隨之增加。

6 · 網路（Network）

以下是網路運作的步驟：

- 1.新的交易會向所有節點廣播
- 2.每個節點會蒐集加入區塊的新交易
- 3.每個節點負責為其區塊找到一條夠困難的工作量證明
- 4.當節點找到工作量證明後，會向所有節點廣播
- 5.節點只有在節點中的所有交易為有效且尚未存在過時，才會接受區塊為有效的

6.節點會使用已接納之區塊的雜湊作為之前的雜湊來創造鏈上的新區塊，來表示接受該區塊

節點始終會將最長的鏈視為正確的且持續延長。若有兩個節點同時廣播兩個不同版本的新區塊，部分節點可能會先接收到其中一種。在這種情況下，節點會先處理先接收到的區塊，但也會保存另一條分支鏈以免它變成最長的鏈。當下一個工作量證明被發現且其中一條鏈

被證實為較長的一條時，這樣的僵局會被打破，此時，原本在另一條分支鏈上工作的節點會轉而處理較長的這條鏈。

新交易的廣播並不一定要觸及所有節點，只要這些交易廣播到足夠多的節點，不久後這些交易就會被整合進一個區塊。區塊的廣播同樣可以容許漏掉訊息，若某個節點沒有接收到特定區塊，當這個節點接收到下一個區塊時，就會認知到它遺漏了一個區塊並可提出自己下載該區塊的請求。

7．獎勵（Incentive）

按照慣例，區塊中的第一個交易較特別，因為它產生了一顆由該區塊建立者所擁有的新貨幣。這激勵了節點去支持整個網路，且在沒有中央權威機構發行貨幣的情況下，提供了初期貨幣流通的方法。新貨幣數量的穩定增加和礦工為了增加黃金流通而耗費資源去挖礦類似。而在我們的案例中，耗費的資源是 CPU 運算時間和消耗的電力。

獎勵也可以來自於交易手續費。交易的輸入值減掉輸出值即為交易的手續費，提供交易所在之區塊的獎勵。一旦既定數量的貨幣進入流通，獎勵機制就可以完全依靠交易手續費且能免於通貨膨脹。

獎勵機制也有助於鼓勵節點保持誠實。若有貪婪的攻擊者能夠裝備比所有誠實節點還要多的 CPU 運算能力，那麼他將面臨以下選擇：進行雙重支付的攻擊或是用於誠實工作來產生新貨幣，他將會發現遵循規則將是更有利可圖的，因為比起破壞系統及自身財產的合法性，遵守規則能夠使他擁有更多的電子貨幣。

以上引用自比特幣白皮書與其原文翻譯：

原文：<https://bitcoin.org/bitcoin.pdf>

譯文：<https://medium.com/taipei-ethereum-meetup/%E6%AF%94%E7%89%B9%E5%B9%A3-%E7%AB%AF%E5%B0%8D%E7%AB%AF%E9%9B%BB%E5%AD%90%E7%8F%BE%E9%87%91%E7%B3%BB%E7%B5%B1-bitcoin-a-peer-to-peer-electronic-cash-system-i-8a52de003c9>

(二) 代碼實做：

在此階段，我們將實現除了第 6 · 以外的所有特色，而第 6 · 則將交由下個部分的以太坊做實作與解釋。

1 · 引入需要使用的套件

```
import hashlib
import json
from textwrap import dedent
from time import time
from uuid import uuid4
from flask import Flask, jsonify, request
from urllib.parse import urlparse
import requests
```

2 · 初始化類別

```
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.nodes = set()
        # 創建創世塊
        self.new_block(previous_hash=1, proof=100)
```

3 · 註冊節點

```
def register_node(self, address):
    """
    在清單中加入新的節點
    :param address: <str> Address of node. Eg. 'http://192.168.0.5:5000'
    :return: None
    """

    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

4 · 新增一個新區塊

```
def new_block(self, proof, previous_hash=None):
    # 創建一個新的塊並將其添加到鏈中
    """
    生成新塊
    :param proof: <int> The proof given by the Proof of Work algorithm
    :param previous_hash: (Optional) <str> Hash of previous Block
    :return: <dict> New Block
    """

    block = {
        'index': len(self.chain) + 1,
        'timestamp': time(),
        'transactions': self.current_transactions,
        'proof': proof,
        'previous_hash': previous_hash or self.hash(self.chain[-1]),
    }

    # 重置當前的交易列表
    self.current_transactions = []

    self.chain.append(block)
    return block
```

5 · 新增一筆新交易

```
def new_transaction(self, sender, recipient, amount):
    """
    生成新交易信息，信息將加入到下一個待挖的區塊中
    :param sender: <str> Address of the Sender
    :param recipient: <str> Address of the Recipient
    :param amount: <int> Amount
    :return: <int> The index of the Block that will hold this transaction
    """

    self.current_transactions.append({
        'sender': sender,
        'recipient': recipient,
        'amount': amount,
    })

    return self.last_block['index'] + 1
```

6 · 得到一個一個區塊鏈的 Hash 值

```
@staticmethod
# 從外部非實體化調用函數
def hash(block):
    """
    生成塊的 SHA-256 hash值
    :param block: <dict> Block
    :return: <str>
    """

    # 我們必須確保字典是有序的，否則我們將有不一致的哈希值
    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest()
```

7 · 回傳最後一個區塊

驗證 proof 的結果是否正確

```
@property
# 檢查參數後再傳入
def last_block(self):
    return self.chain[-1]

@staticmethod
def valid_proof(last_proof, proof):
    """
    驗證證明：是否hash(last_proof, proof)以4個0開頭？
    :param last_proof: <int> Previous Proof
    :param proof: <int> Current Proof
    :return: <bool> True if correct, False if not.
    """

    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"
```

8 · 驗證整條區塊鏈是否合法

```
def valid_chain(self, chain):  
    """  
    Determine if a given blockchain is valid  
    :param chain: <list> A blockchain  
    :return: <bool> True if valid, False if not  
    """  
  
    last_block = chain[0]  
    current_index = 1  
  
    while current_index < len(chain):  
        block = chain[current_index]  
        print(f'{last_block}')  
        print(f'{block}')  
        print("\n-----\n")  
        # 檢查塊的hash是否正確  
        if block['previous_hash'] != self.hash(last_block):  
            return False  
  
        # 检查工作證明是否正確  
        if not self.valid_proof(last_block['proof'], block['proof']):  
            return False  
  
        last_block = block  
        current_index += 1  
  
    return True
```

9 · 尋找更長的鏈，並解決共識衝突

```
def resolve_conflicts(self):  
    """  
    共識算法解決衝突  
    使用網路中最長的鏈。  
    :return: <bool> True 如果鏈被取代，否則為False  
    """  
  
    neighbours = self.nodes  
    new_chain = None  
  
    # 只尋找比我們更長的鏈條  
    max_length = len(self.chain)  
  
    # 抓取並驗證我們網路中所有節點的鏈  
    for node in neighbours:  
        response = requests.get(f'http://{node}/chain')  
  
        if response.status_code == 200:  
            length = response.json()['length']  
            chain = response.json()['chain']  
  
            # 如果我們發現一個比我們的更長的、新的有效鏈條，就取代我們的鏈條  
            if length > max_length and self.valid_chain(chain):  
                max_length = length  
                new_chain = chain  
  
    # 如果我們發現一個比我們的更長的新的有效鏈條，就取代我們的鏈條  
    if new_chain:  
        self.chain = new_chain  
        return True  
  
    return False
```

1 0 · 藉由 Flask 部署各種方法於 http 協議上

```
# 實體化我們的節點
app = Flask(__name__)

# 建立一個node節點
node_identifier = str(uuid4()).replace('-', '')

# 實體化區塊
blockchain = Blockchain()
```

1 1 · 透過驗證 proof 建立區塊，並將交易包含於其中

```
@app.route('/mine', methods=['GET'])
def mine():
    # 使用 proof of work 演算法來得到下一個 proof...
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    # 給工作量證明的節點提供獎勵。
    # 發送者為 "0" 表明是新挖出的幣
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )

    # 通過將其添加到鏈中來創造新的塊
    block = blockchain.new_block(proof)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```


1 2 · 傳送一筆新的交易，並置於交易清單中等待包覆。

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # 確定是一個可實行的交易 (資料充足與完備)
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # 創建一新的交易
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200
```

1 3 · 傳送己身節點位置，使節點彼此串連

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()
    print(values)
    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

1 4 · 透過遍歷尋找最長的鏈並取代之。

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

1 5 · 執行於區網之中。

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

三、區塊鏈運行結果：

```
127.0.0.1 - - [22/Jan/2018 22:14:29] "POST /transactions/new HTTP/1.1" 500 -
127.0.0.1 - - [22/Jan/2018 22:14:40] "POST /transactions/new HTTP/1.1" 201 -
127.0.0.1 - - [22/Jan/2018 22:18:15] "POST /nodes/register HTTP/1.1" 201 -

{'nodes': ['http://0.0.0.0:5001']}
{'0.0.0.0:5001'}

127.0.0.1 - - [22/Jan/2018 22:22:37] "GET /nodes/resolve HTTP/1.1" 200 -

{'index': 1, 'previous_hash': 1, 'proof': 100, 'timestamp': 1516629852.086288, 'transactions': []}
{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'timestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender': '0'}]}

-----

{'index': 2, 'previous_hash': 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', 'proof': 35293, 'timestamp': 1516630877.7717729, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender': '0'}]}
{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'timestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender': '0'}]}

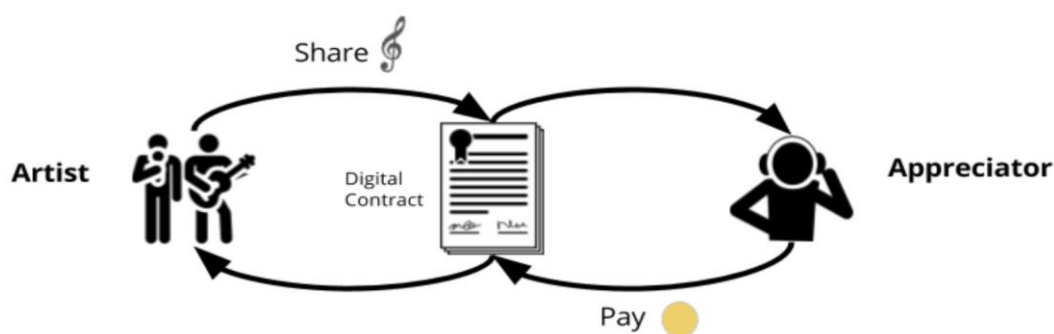
-----

{'index': 3, 'previous_hash': 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', 'proof': 35089, 'timestamp': 1516630879.711094, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender': '0'}]}
{'index': 4, 'previous_hash': '229728353739f40ac78711de0ffec649539ec5cbe3587c66eb434a2e67992fea', 'proof': 119678, 'timestamp': 1516630881.634832, 'transactions': [{'amount': 1, 'recipient': '052cefe98e61491696f0166d8570a782', 'sender': '0'}]}
```

四、以太坊名詞介紹：

（一）以太坊：

以太坊的目的是基於腳本、競爭幣和鏈上元協議（**on-chain meta-protocol**）概念進行整合和提高，使得開發者能夠創建任意的基於共識的、可擴展的、標準化的、特性完備的、易於開發的和協同的應用。以太坊通過建立終極的抽象的基礎層-內置有圖靈完備編程語言的區塊鏈-使得任何人都能夠創建合約和去中心化應用並在其中設立他們自由定義的所有權規則、交易方式和狀態轉換函數。域名幣的主體框架只需要兩行代碼就可以實現，諸如貨幣和信譽系統等其它協議只需要不到二十行代碼就可以實現。智能合約-包含價值而且只有滿足某些條件才能打開的加密箱子-也能在我們的平台上創建，並且因為圖靈完備性、價值知曉（**value-awareness**）、區塊鏈知（**blockchain-awareness**）和多狀態所增加的力量而比特幣腳本所能提供的智能合約強大得多。



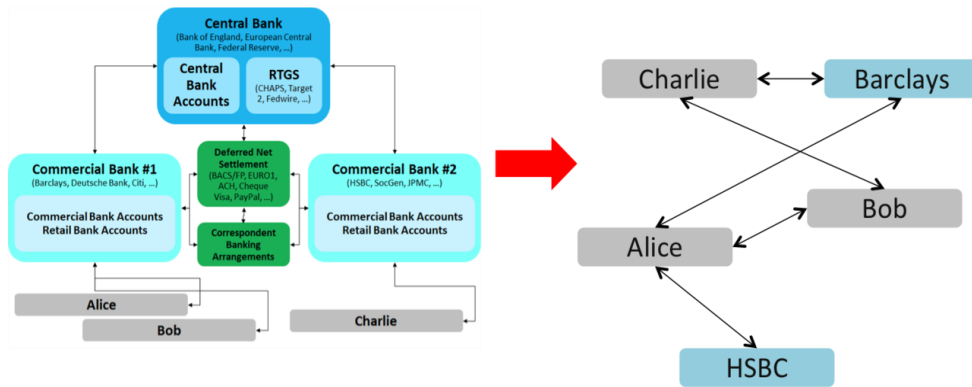
（二）智能合約：

智能合約程序不只是一個可以自動執行的計算機程序：它自己就是一個系統參與者。它對接收到的信息進行回應，它可以接收和儲存價值，也可以向外發送信息和價值。

這個程序就像一個可以被信任的人，可以臨時保管資產，總是按照事先的規則執行操作。

下面這個示意圖就是一個智能合約模型：一段代碼（智能合約），被部署在分享的、複製的賬本上，它可以維持自己的狀態，控制自己的資產和對接收到的外界信息或者資產進行回應。





(三) Solidity 語言：

Solidity 是一種語法類似 JavaScript 的高級語言。它被設計成以編譯的方式生成以太坊虛擬機代碼。在後續內容中你將會發現，使用它很容易創建用於投票、眾籌、封閉拍賣、多重簽名錢包等等的合約。

(四) Geth 系統：

Geth 又名 Go Ethereum. 是以太坊協議的三種實現之一，由 Go 語言開發，完全開源的項目。Geth 可以被安裝在很多作業系統上，包括 Windows、Linux、Mac 的 OSX、Android 或者 IOS 系統。

Geth 工具是 Go Ethereum, 是以太坊的官方客戶端（Go 語言實現）通過 Geth 的一些基本命令，可以很方便的創建出一個以太坊的私有鏈條。

Geth 的命令行中包含了大多數的以太坊的命令，包括帳戶新建，帳戶之間的以太幣轉移，挖礦，獲取餘額，部署以太坊合約等。

以上資料引用自：

每日頭條：<https://kknews.cc/zh-tw/tech/oygbxlm.html>

極客學院：<http://wiki.jikexueyuan.com/project/solidity-zh/introduction.html>

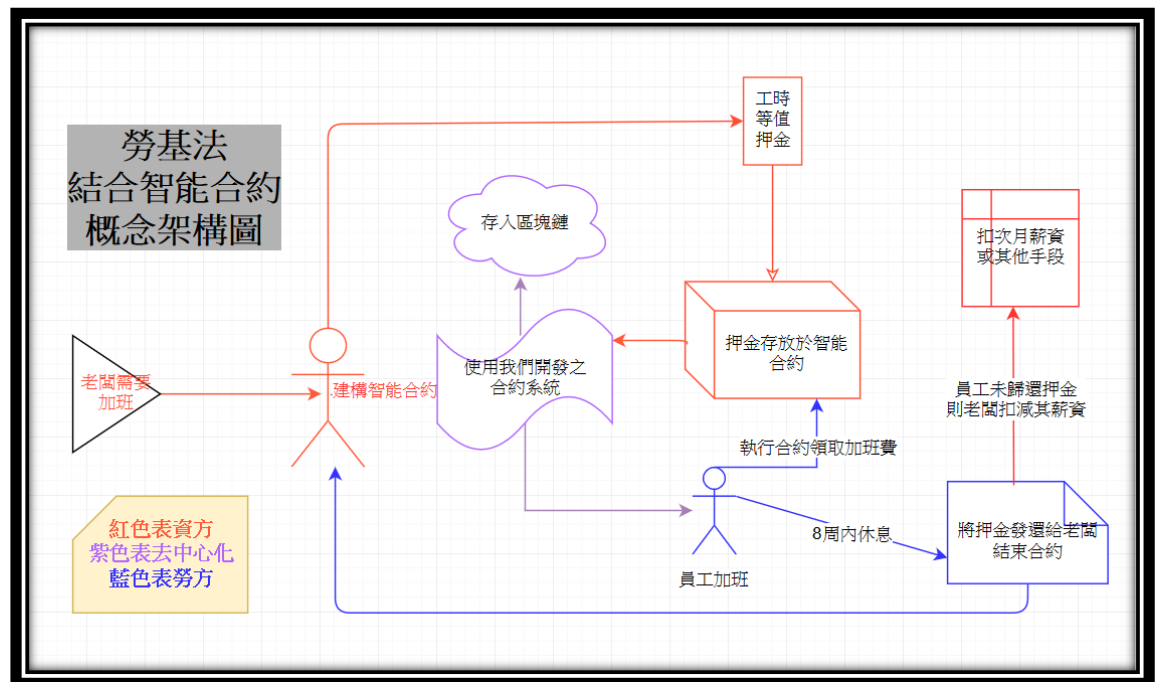
Github<https://github.com/EthFans/wiki/wiki/%E6%99%BA%E8%83%BD%E5%90%88%E7%BA%A6>

以太坊白皮書：

<https://github.com/ethereum/wiki/wiki/%5B%E4%B8%AD%E6%96%87%5D-%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%99%BD%E7%9A%AE%E4%B9%A6>

五、代碼實做：

(一) 合約架構：



(二) 代碼實做：

1. 定義各項參數以便後續使用

```
1  pragma solidity ^0.4.17;
2
3  contract Transaction {
4      //定義勞方，資方
5      address employee;
6      address employer;
7      //定義合約創建時間
8      uint create_time;
9      //定義基本工資
10     uint base_wage;
11     //定義應付工資
12     uint payables;
13     //定義補修期限
14     uint expire = 8;
15     //定義工作時數
16     uint working_hours;
17     //定義每wei對台幣之轉換
18     uint ETH_to_TWD_100 = 300000000000000;
19     //定義補修期限是否結束
20     bool ended = false;
21     //定義合約是否合格
22     bool used = false;
```

2 · 建置初始化方法，用來實現基礎功能

```
24 | //合約啟動時之函數（勞方地址,工作時數,基礎薪資）[設定為可傳入以太幣][設定為公開函數]
25 | function Transaction(address new_employee, uint new_working_hours ,uint new_base_wage)payable public{
26 |     //導入變數
27 |     employee = new_employee;
28 |     employer = msg.sender;
29 |     working_hours = new_working_hours;
30 |     base_wage=new_base_wage;
31 |
32 |     //依勞基法判斷加班薪資
33 |     if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+2*233*base_wage;
34 |     else payables = working_hours*233*base_wage;
35 |
36 |     //判斷薪資是否合格
37 |     if (msg.value>=payables/ETH_to_TWD_100) usaged=true;
38 |     else return;
39 | }
```

3 · 建立補充方法，補充初始化時誤輸的資料

```
44 | //合約修改之函數（新增工作時數）[設定為可傳入以太幣][設定為公開函數]
45 | function add_more(uint new_working_hours )payable public{
46 |     //如果合約已結束，則回傳以太幣
47 |     if (ended == true) revert();
48 |
49 |     //否則導入變數
50 |     working_hours+=new_working_hours;
51 |
52 |     //依勞基法判斷加班薪資
53 |     if (working_hours >= 2) payables = (working_hours-2)*base_wage*266+working_hours*233*base_wage;
54 |     else payables = working_hours*233*base_wage;
55 |
56 |     //判斷薪資是否合格
57 |     if (msg.value>=payables/ETH_to_TWD_100) usaged=true;
58 |     else return;
59 |
60 |     //合格及建立合約
61 |     create_time=now;
62 | }
```

4 · 建立勞工領錢方法，與超越八周末休假時使用

```
64 | // 勞工依法取回加班薪資 [設定為公開函數]
65 | function collectMoney() public {
66 |     //如果時間已超過8周
67 |     if ((now - create_time) > (expire * 1 weeks)) {
68 |         //設定合約以結束
69 |         ended = true;
70 |     }
71 |     //否則設定合約繼續
72 |     else {
73 |         ended = false;
74 |     }
75 |
76 |     //若申請者非勞方，或合約未結束，或合約不可使用，則不執行
77 |     if (msg.sender!=employee || ended == false || usaged == false) revert();
78 |     //否則執行函數transfer傳送押金(薪資)給勞方
79 |     employee.transfer(this.balance);
80 | }
```

5 · 建立返回方法，使勞方有權力能控制薪資。

```
82 // 若已休假，勞方回傳押金給資方【設定為公開函數】
83 function returnMoney () public {
84     //若申請者非勞方，或合約未結束，則不執行
85     if (msg.sender!=employee || ended == true ) revert();
86     //否則執行函數transfer傳送押金給資方
87     employer.transfer(this.balance);
88 }
89 }
```

六、智能合約結果展示：

The screenshot displays the Ethereum Wallet interface for deploying a contract. The top bar shows the wallet balance as 17,095.00 ETH. The main area is divided into two tabs: 'SOLIDITY 契約原始碼' (Solidity Contract Source Code) and '契約 BYTECODE' (Contract Bytecode). The Solidity code is visible, showing a contract with functions like `add_more` and `is_expire`. On the right side, there are deployment parameters: 'New_employee - address' with the value `0x51E63BA95FBCF403b2803DFa922`, 'New_working_hours - 256 bits unsigned integer' with the value `10`, and 'New_base_wage - 256 bits unsigned integer' with the value `140`. The bottom of the interface shows the transaction details, including the gas limit and the amount of ETH to be sent (14 ETH).

伍、研究結果

一、成功用 Python 建造一個區塊鏈架構，並測試之。詳情如圖 5.1.1~5.1.4



圖 5.1.1 藉由/mine 挖礦：http://localhost:5000/mine

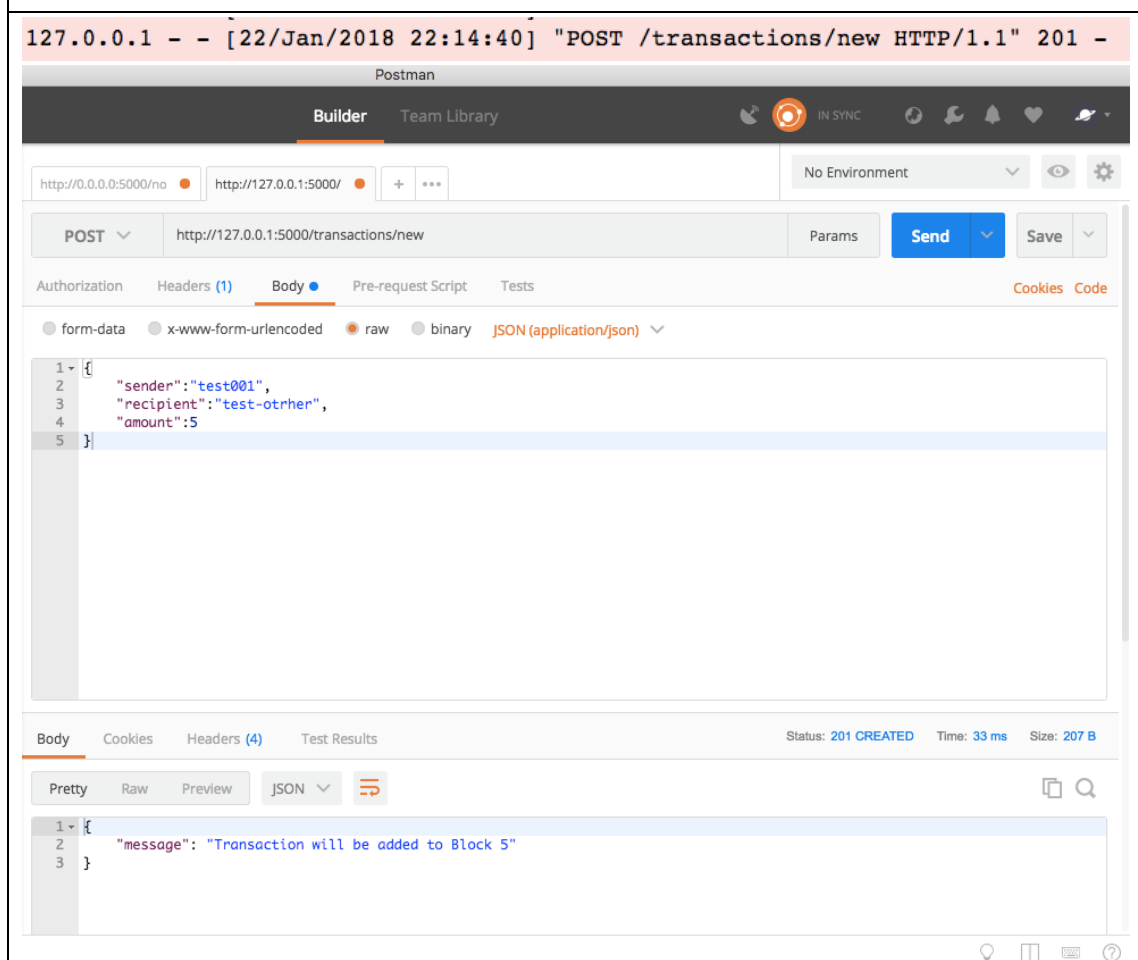
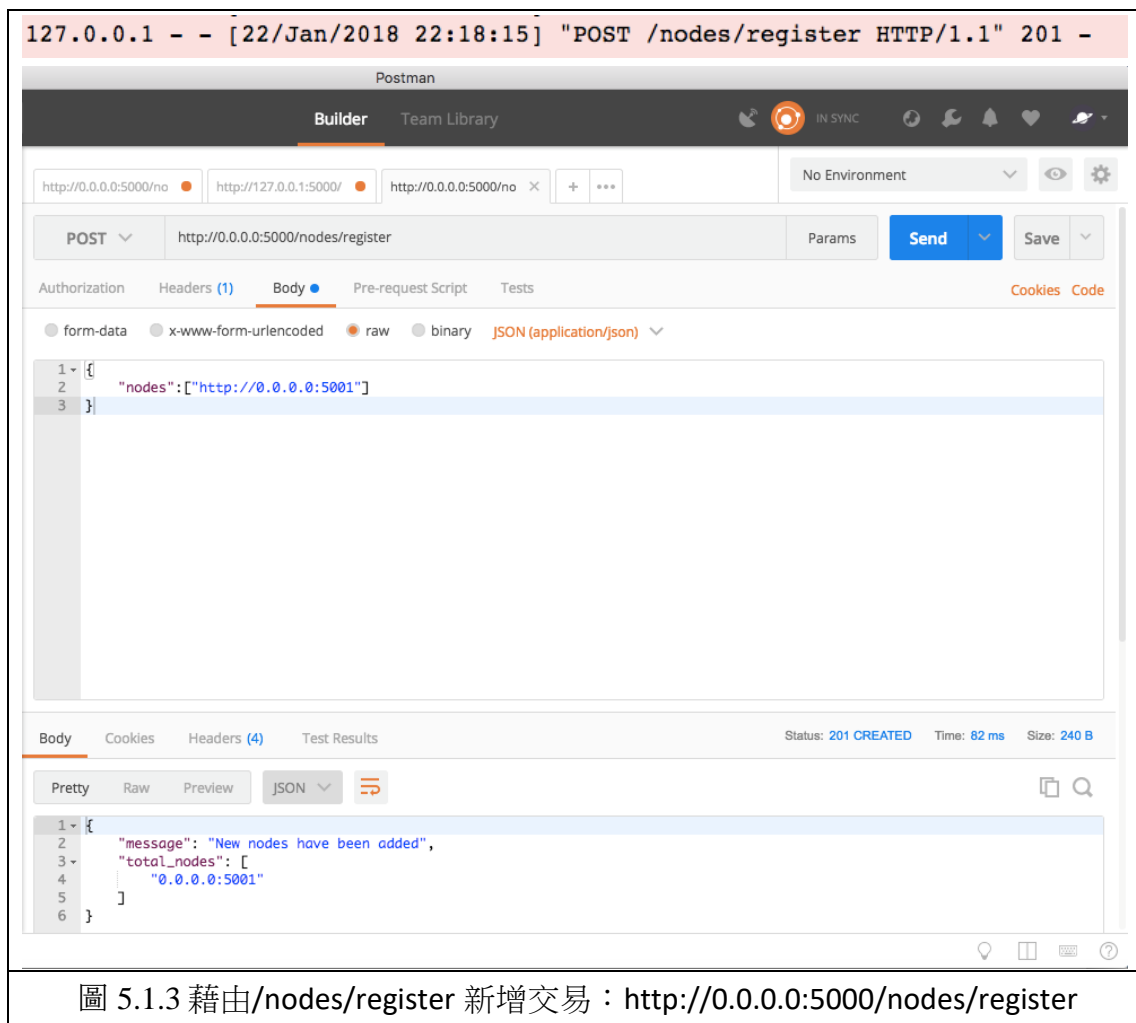


圖 5.1.2 藉由/transactions/new 新增交易：http://127.0.0.1:5000/transactions/new



```
127.0.0.1 - - [22/Jan/2018 22:22:37] "GET /nodes/resolve HTTP/1.1" 200 -
{"index": 1, "previous_hash": 1, "proof": 100, "timestamp": 1516629852.086288, "transactions": []}
{"index": 2, "previous_hash": 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', "proof": 35293, "timestamp": 1516630877.7717729, "transactions": [{"amount": 1, "recipient": '052cefe98e61491696f0166d8570a782', "sender": '0'}]}

-----

{"index": 2, "previous_hash": 'a62effa8f9a07bd39b3d7abfc4fd770b0d14cf10321bad0a17412ee7dcfaf637', "proof": 35293, "timestamp": 1516630877.7717729, "transactions": [{"amount": 1, "recipient": '052cefe98e61491696f0166d8570a782', "sender": '0'}]}
{"index": 3, "previous_hash": 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', "proof": 35089, "timestamp": 1516630879.711094, "transactions": [{"amount": 1, "recipient": '052cefe98e61491696f0166d8570a782', "sender": '0'}]}

-----

{"index": 3, "previous_hash": 'c484a441e458caefd67daa78727c8166a289d684a4d5126d091a0436148904d9', "proof": 35089, "timestamp": 1516630879.711094, "transactions": [{"amount": 1, "recipient": '052cefe98e61491696f0166d8570a782', "sender": '0'}]}
{"index": 4, "previous_hash": '229728353739f40ac78711de0ffec649539ec5cbe3587c66eb434a2e67992fea', "proof": 119678, "timestamp": 1516630881.634832, "transactions": [{"amount": 1, "recipient": '052cefe98e61491696f0166d8570a782', "sender": '0'}]}
```

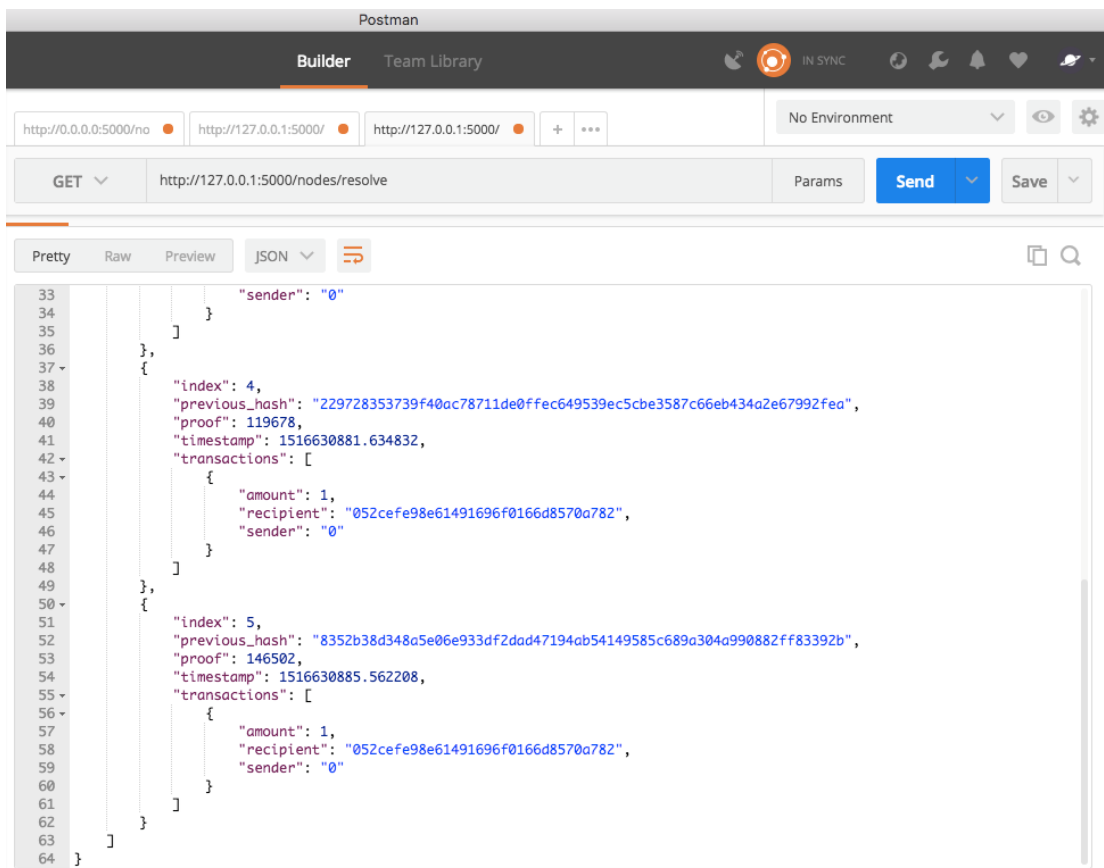
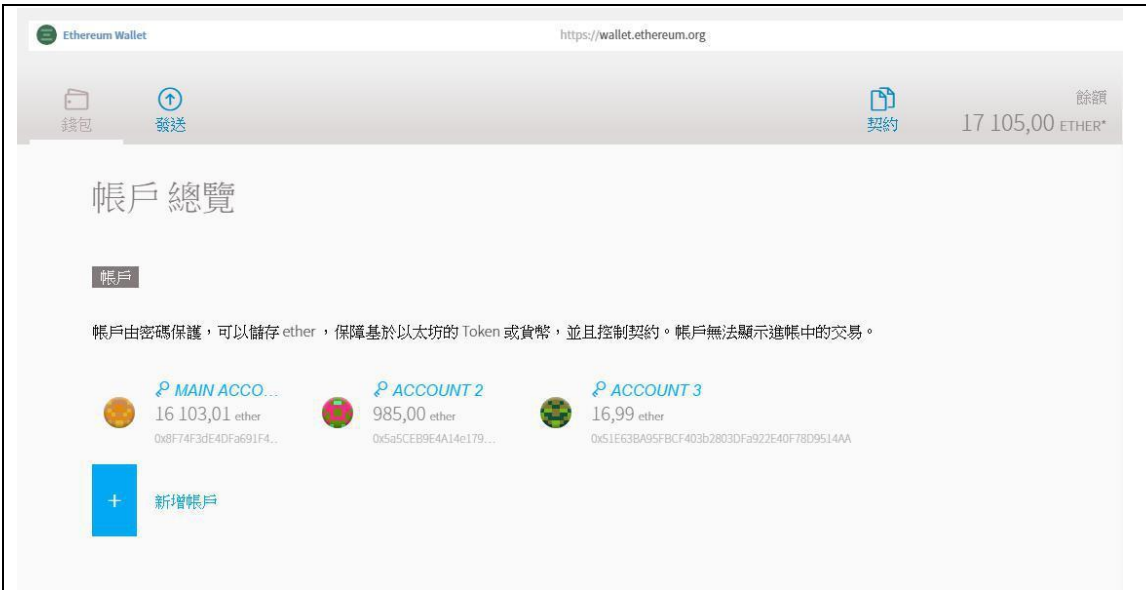
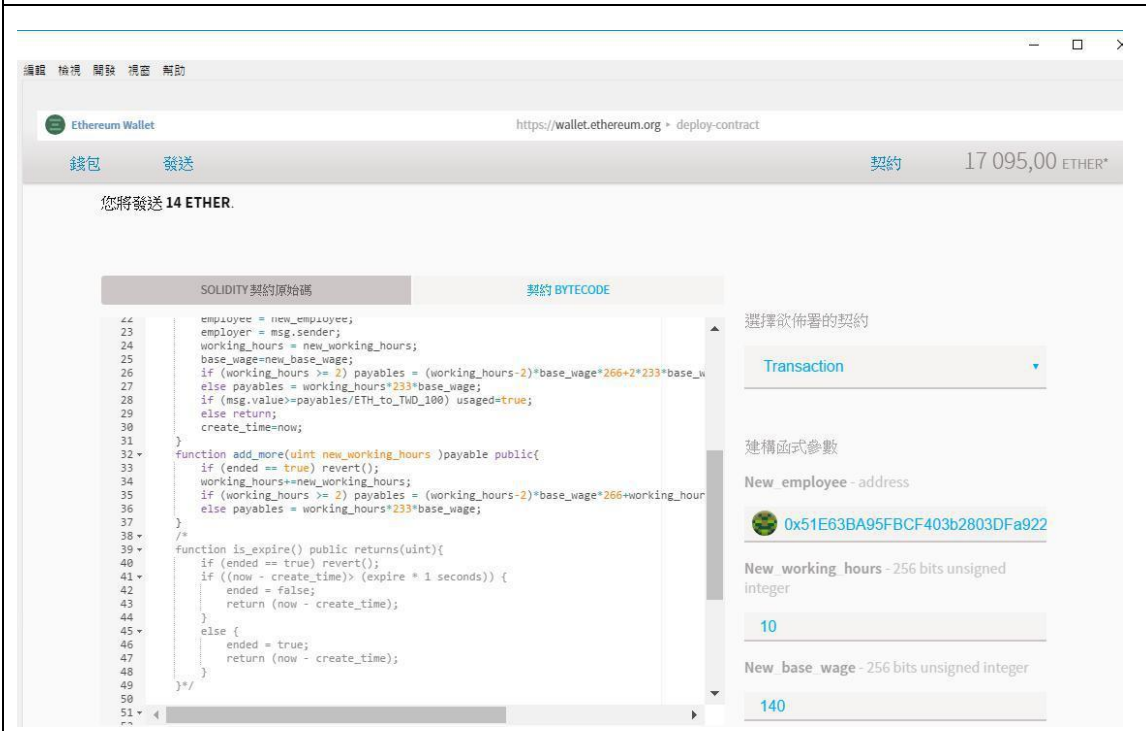


圖 5.1.4 藉由/nodes/resolve 新增交易：http://127.0.0.1:5000/nodes/resolve

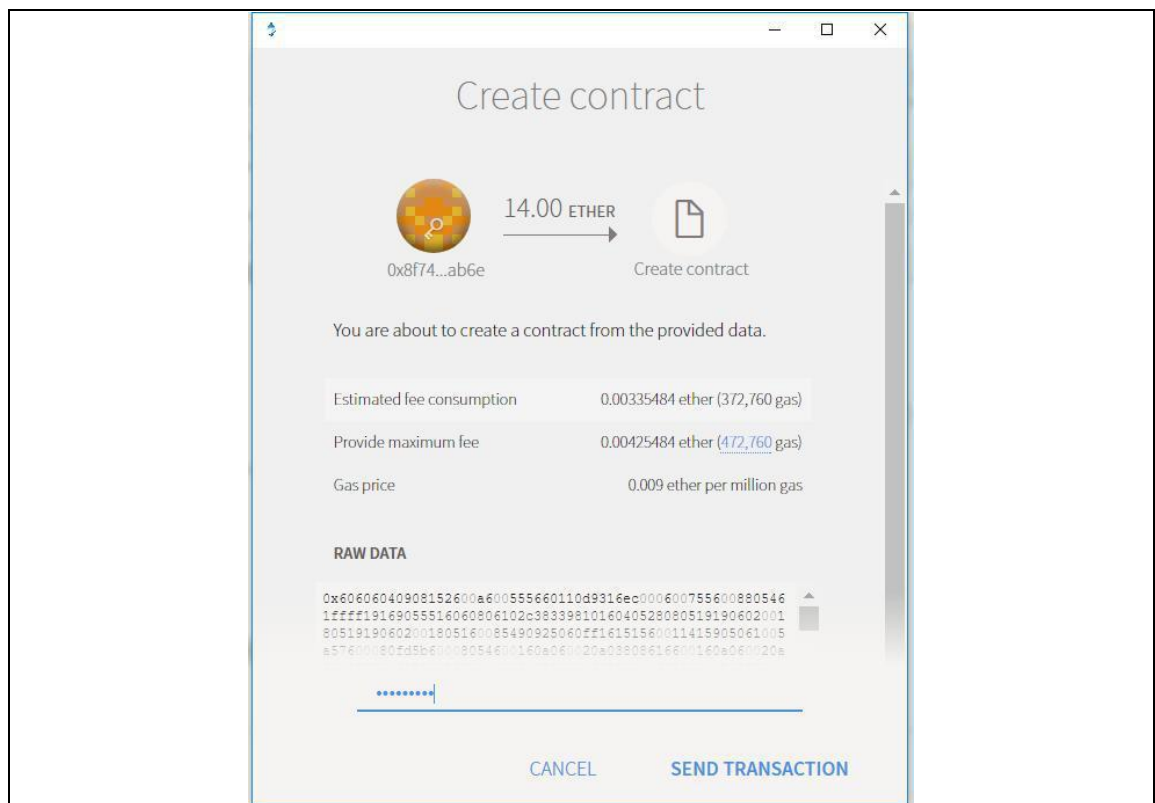
二、成功建構一智能合約，得已應用於現行法律之中。詳情如圖 5.2.1~5.2.10



5.2.1 起初帳戶總覽



5.2.2 建設智能合約



5.2.3 建設合約認證



5.2.4 合約寫入區塊鏈

| | |
|---|---|
| <div> <div>寫至契約</div> <div>Select function</div> <div>Add_more</div> <div>New_working_hours - 256 bits unsigned integer</div> <div>10</div> <div>Execute from</div> <div>Main Account (Etherbase) - 16 256</div> <div>Send ether*</div> <div>10</div> <div>執行</div> </div> | <div> <div>Execute contract</div> <div> <div>0x8f74...ab6e</div> <div>0.00 ETHER</div> <div>0X69A191CB</div> <div>0x0558...7297</div> </div> <div>It seems this transaction will fail. If you submit it, it may consume all the gas you provide.</div> <div> <div>Estimated fee consumption</div> <div>We couldn't estimate the gas.</div> </div> <div> <div>Provide maximum fee</div> <div>ether (0 gas)</div> </div> <div> <div>Gas price</div> <div>0.009 ether per million gas</div> </div> <div> <div>RAW DATA</div> <div>TRY TO DECODE DATA</div> </div> <div>0x69a191cb</div> <div>Enter password to confirm the transaction</div> <div> <div>CANCEL</div> <div>SEND TRANSACTION</div> </div> </div> |
| 5.2.5 Add 方法新增押金 | 5.2.6 合約驗證 |
| <div> <div> <div>Jan 22</div> <div> <div>契約執行</div> <div> <div>Main account (Etherbase)</div> <div>Transaction 0558</div> </div> </div> <div> <div>6 的 12 確認區塊</div> <div>-10,00 ETHER</div> <div></div> </div> </div> <div> <div>Jan 22</div> <div> <div>已創造契約</div> <div> <div>Main account (Etherbase)</div> <div>契約創造在 Transaction 0558</div> </div> </div> <div> <div>3 minutes ago</div> <div>-14,00 ETHER</div> <div></div> </div> </div> </div> | |
| 5.2.7 合約寫入區塊鏈 | |

寫至契約

Select function

Collect Money

Execute from




Account 3 - 16,99 ETH

執行

5.2.8 勞方使用 Collect 方法，拿到工資

| | | | | |
|--------|--|---------------|------------|--|
| Jan 22 | 契約執行 Account 3 → Transaction 0558 | 5 的 12 確認區塊 | -0,00 ETH | |
| Jan 22 | 契約執行 Main account (Etherbase) → Transaction 0558 | 4 minutes ago | -10,00 ETH | |
| Jan 22 | 已創建契約 Main account (Etherbase) → 契約創建在 Transaction 0558 | 6 minutes ago | -14,00 ETH | |

5.2.9 合約寫入區塊鏈

| | | |
|---|--|--|
|  <div> <div>MAIN ACCO...</div> <div>16 394,01 ether</div> <div>0x8F74F3dE4DFa691F4...</div> </div> |  <div> <div>ACCOUNT 2</div> <div>985,00 ether</div> <div>0x5a5CEB9E4A14e179...</div> </div> |  <div> <div>ACCOUNT 3</div> <div>40,99 ether</div> <div>0x51E63BA95FBCF403b2803DFa922E40F78D9514AA</div> </div> |
| <div> <div>+</div> <div>新增帳戶</div> </div> | | |

5.2.10 勞方獲得工資（ACCOUNT3）

陸、討論

| |
|---|
| Q：為何不使用 python 區塊鏈製作合約而採用以太坊？ |
| A：因為 python 區塊鏈製作合約時，我們無一統一而有價值的代幣；若要於 python 區塊鏈實現，則必須交由政府發行，那便失去去中心化的意義，且若政府非可信的一方，那勞工的資產將有危險。 |
| Q：Solidity 語言中的 now 不一定準確，以太坊的挖礦也可能造成時間的延遲，請問如何解決時間差的問題？ |
| A：因為區塊鏈必須由礦工共同確認，故在安全的基礎上時間延遲是不可避免的，但因合約時間單位較久（數週），不因此影響此操作。 |
| Q：為何不採用紙本合約？紙本合約具有法律效力及法律的保障。 |
| A：有鑒於現今社會勞方常遇到打壓，勞方多無法得到一個有力的回覆。透過法律途徑卻往往勞民傷財，最後又回歸集體罷工或上街抗議，造成兩敗俱傷的結局，故採用智能合約，以不信任任何一個節點為出發點，那麼將不存在毀約的問題，也能促進社會和諧。 |
| Q：為何最後是由勞方返回押金，而不是由資方主動贖回？ |
| A：原因同上，社會勞方常處於不利的立場，故若由資方贖回，可能會造成紛爭，例如勞方未確實休假等等。故選擇由勞方贖回，給予權力的平等。 |
| Q：若勞方為返回押金，資方權利是否受損？ |
| A：資方因掌有較多的權力，可藉由扣減次月薪水，扣減福利或其他手段避免此類事情發生。 |
| Q：若程式設定錯誤，是否押金將永存於合約中不得取出？ |

A：是的，因為是去中心化的系統，將沒有任何客服可以協助，技術上更是不允許，否則會造成區塊鏈的崩毀。但為防範此狀況，設立 add 方法，避免資方在輸入過少金額時，造成的合約不成立（`used == false`），藉此給資方一個更改合約的機會。

Q：勞方如何得知資方已確實創立合約，而非只是隨口胡言？

A：資方可傳送合約地址及其 json 檔案，供勞方於 Mist 錢包中查看合約概況。

Q：資方如何得知勞方已確實返還押金，而非只是隨口胡言？

A：資方可查看合約地址或己身錢包紀錄，於 Mist 錢包中確認金流。

Q：紙本合約與智能合約的比較。

A：

| | 現今紙本合約 | 此研究之智能合約 |
|--------------------|------------|-----------|
| 公正第三方存在 | 需要 | 不需要 |
| 合約建立是否平等 | 不平等，多利資方 | 平等 |
| 合約用途與效益 | 往往說一套做一套 | 強制遵守，不得抗命 |
| 合約不成立時的影響 | 法律訴訟、上街抗議 | 呼叫方法，強制履行 |
| 社會普及與應用程度 | 成熟，但有紛爭與疑慮 | 發展中 |
| 若實際引入職場，能否符合勞基法規範？ | 不可 | 可以 |

柒、結論

本研究總共利用了 BlockChain、Python、Ethereum、Solidity、Smart Contract、Geth、Mist 等科技，開發出對勞資雙方皆公平可靠的合約與平台。

BlockChain 使一切都變得公開透明，使一切資料與記錄都不可否認，更確保了資料的安全性，而利用自行開發之 Smart Contract，使未來勞資雙方的合約不再是由任何一方控制，金流更不僅只掌握於資方之手，重新賦予了勞資雙方平等的權力！

透過我們的 Smart Contract，給予了勞基法一個適當的歸所。除了提供了政府一條新的道路，可以用不同的角度去正視勞資問題，尋找新的解決方案；更能從技術與科技上，讓整個勞資生態圈與區塊鏈技術共同成長，欣欣向榮。

捌、未來展望

- 一、因使用以太幣作為交易貨幣，可能因貨幣價值不穩定，導致價格差異。期望能引進即時價格，將差異減至最小，提供更穩定的合約。
- 二、合約內容可能含有漏洞，需要公布或請駭客審查之，才能夠正式使用。
- 三、盼望與政府合作，推廣此合約與技術，確實應用於勞資市場，實際達成目的。
- 四、於部分公司嘗試使用合約，蒐集資料與數據，更加完善系統。
- 五、考慮更多勞資方需求與實務細節，打造更完善的系統。

玖、參考資料

一、科展作品

[57 屆科展：使用區塊鏈開發可監督捐款之公開募捐平台](<http://activity.ntsec.gov.tw/activity/race-1/57/pdf/052505.pdf>)

二、觀念建構

[區塊鏈技術概觀（一）：讓我們從歷史文本說起](<https://finance.technews.tw/2017/08/01/block-chain-technology-overview/>)

[區塊鏈技術概觀（二）：資料分割與密碼學](<https://technews.tw/2017/08/09/blockchain-principle-part-two/>)

[區塊鏈技術概觀（三）：多數決的運作結構](<https://technews.tw/2017/08/16/blockchain-principle-part-three/>)

[區塊鏈技術指南](https://yeasy.gitbooks.io/blockchain_guide/content/)

三、Python 語法

[python2 廖雪峰教程](<https://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000>)

[python3 廖雪峰教程](<https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000>)

["@"裝飾器概念](<https://foofish.net/python-decorator.html>)

[Python & Flask](<http://blog.techbridge.cc/2017/06/03/python-web-flask101-tutorial-introduction-and-environment-setup/>)

四、python 實作區塊鏈

[用 Python 从零开始创建区块链](<http://www.jianshu.com/p/cc6663cbbc41>)

[50 行 Python 代码构建一个区块链](http://blog.csdn.net/simple_the_best/article/details/75448617)

五、以太坊運作與實作：

[以太坊白皮書](<https://github.com/ethereum/wiki/wiki/%5B%E4%B8%AD%E6%96%87%5D->

%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%99%BD%E7%9A%AE%E4%B9%A6)

[智能合約](<https://blog.gasolin.idv.tw/2017/08/13/got-my-ens-domain/>)

[以太坊錢包 Parity](<https://medium.com/taipei-ethereum-meetup/%E5%9C%A8parity%E5%89%B5%E5%BB%BA%E4%BB%A5%E5%A4%AA%E5%9D%8A%E9%8C%A2%E5%8C%85-b2c2977378fc>)

[教練，我”只”想學 Solidity](<https://medium.com/taipei-ethereum-meetup/%E6%95%99%E7%B7%B4-%E6%88%91-%E5%8F%AA-%E6%83%B3%E5%AD%B8solidity-92b7ba8054f5>)

[收到我的 ENS 網域啦 gasolin.eth](<https://blog.gasolin.idv.tw/2017/08/13/got-my-ens-domain/>)

[如何撰寫智能合約(Smart Contract)?(I)](<https://blog.gasolin.idv.tw/2017/09/06/howto-write-a-smart-contract/>)

[如何撰寫智能合約(Smart Contract)?(II)建立加密代幣](<https://blog.gasolin.idv.tw/2017/09/11/howto-write-a-simple-token/>)

[如何撰寫智能合約(Smart Contract)?(III)建立標準代幣](<https://blog.gasolin.idv.tw/2017/09/16/howto-write-an-erc20-compatible-token/>)

[如何撰寫智能合約(Smart Contract)?(IV)加入單元測試](<https://blog.gasolin.idv.tw/2018/01/02/howto-write-a-contract-test/>)

[区块链开发（一）搭建基于以太坊的私有链环境](<http://blog.csdn.net/sportshark/article/details/51855007>)

[区块链开发（二）部署和运行第一个以太坊智能合约](<http://blog.csdn.net/sportshark/article/details/52249607>)

[区块链开发（三）编写调试第一个以太坊智能合约](<http://blog.csdn.net/fidelhl/article/details/52524434>)

[Solisity 文檔](<http://solidity.readthedocs.io/en/latest/>)

[Solidity 中文語法表](<http://www.tryblockchain.org/Solidity%E6%99%BA%E8%83%BD%E5%90%88%E7%BA%A6%E6%96%87%E4%BB%B6%E7%BB%93%E6%9E%84.html>)

[Solidity 語法教學](http://blog.csdn.net/diandianxiyu_geek/article/details/77877841)

[如何在私有区块链上编写、部署以及与以太坊进行交互的智能合约](http://blog.csdn.net/dev_csdn/article/details/78893014)

[台北以太坊社群專欄](<https://medium.com/taipei-ethereum-meetup>)

[Solidity 撰寫智能合約與注意事項(一)](<https://medium.com/taipei-ethereum-meetup/solidity%E6%92%B0%E5%AF%AB%E6%99%BA%E8%83%BD%E5%90%88%E7%B4%84%E8%88%87%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A0%85-%E4%B8%80-6c9eacc00168>)

[Solidity 撰寫智能合約與注意事項(二)](<https://medium.com/taipei-ethereum-meetup/solidity%E6%92%B0%E5%AF%AB%E6%99%BA%E8%83%BD%E5%90%88%E7%B4%84%E8%88%87%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A0%85-%E4%BA%8C-dd915bdeafa0>)

[合約與以太接收](<https://ethereum.stackexchange.com/questions/27052/how-to-create-a-smart-contract-to-send-an-eth-tx-on-time>)

六、其他實作

[瀏覽器挖礦分析](<https://4hou.win/wordpress/?p=9162>)

[javascript](<http://robertvmp.pixnet.net/blog/post/24544521-javascript---入門觀念基礎教學>)

[Socket](<http://beej-zhtw.netdpi.net/02-what-is-socket/2-1-two-internet-sockets>)

[Flask](<http://dormousehole.readthedocs.io/en/latest/quickstart.html>)