PREFACE

序言

林代序

回顧人類社會文明發展，幾次工業革命對人類社會造成鉅大的改變，從而加速人類文明的發展。工業 4.0 產業革命始於本世紀，同樣對人類社會文明造成鉅大的衝擊。此革命以 AI 人工智慧為基礎，數位網路為載具，大幅改變人類生活、工作、溝通、服務等日常生活模式，AI 人工智慧的發展成為新一波產業革命重心，區塊鏈技術的開發與運用，更是 AI 人工智慧發展的核心環節。

區塊鏈的發展約略可分為幾個階段，一是區塊鏈 1.0，以數字貨幣，去中心化為發展重點，比特幣（Bitcoin）可作為該階段代表。二是區塊鏈 2.0，以太坊（Ethereum）之智能合約為代表。三是區塊鏈 3.0，以超級帳本（Hyperledger Project）或針對物聯網而設計的區塊鏈，此階段區塊鏈技術以結合產業為發展目標。區塊鏈發展至今，已呈現多元研究面向，本書將著重在探討智能合約（Smart Contract），該技術堪稱實踐理性世界的工具，可與人工智慧結合，激盪出更多人工智慧應用的可能性。

資誠聯合會計事務所於 2018 年 8 月，在全球同步發布『2018 全球區塊鏈調查報告』（PWC Global Blockchain Survey 2018）。該報告揭示，84%的全球企業正在運用區塊鏈的技術，未來三年到五年具有發展區塊鏈潛力的產業，主要是金融服務業、能源和公用事業、工業製造業、醫療保健業、政府部門等，區塊鏈在產業的應用日趨廣泛，意味著產業型態將因區塊鏈的發展而發生巨大的改變。此外，該報告亦指出區塊鏈的成功，源自四大策略：1) 創造成功案例；2) 建立生態系統；3) 建立使用者的信任；4) 保持靈活以符合法規。區塊鏈藉由前述四大策略進行研究發展十餘年，世界各國正如火如荼地多方面應用嘗試及發展區塊鏈技術，臺灣

在此方面的發展尚待有識之士整合策進。雖然世人更期待區塊鏈在產業界的實務應用，對學術理論的深究關注不深，但從學術研究立場觀之，任何事物變遷成功並非旦夕之功，「掌握原則，保持靈活」才是區塊鏈研究與產業實務運用的最佳準則，可確保區塊鏈技術在趨勢洪流中屹立不搖。

本書主要闡述區塊鏈重要的應用技術-智能合約，內容涵蓋廣泛，析論精闢，為方便讀者感受區塊鏈改變產業發展的魅力，本書透過實務案例，如供應鏈金融、共享經濟與點數經濟、醫療理賠等，帶領讀者深入了解並感受區塊鏈為未來社會所帶來的生活衝擊，並為區塊鏈未來的發展勾勒出美麗的藍圖。作者李昇暾特聘教授本身具備深厚的學理基礎與實務經驗，以其負責踏實的行事風格與專業的能力，相信本書能為讀者提供豐盛的知識饗宴，打開深入了解與認識區塊鏈新興科技的大門，帶給讀者滿滿的收穫。

最後，藉此感謝李昇暾教授對本院 FinTech 商創研究中心的竭力付出與卓越貢獻，也祝福作者與讀者福慧並進，平安喜樂！

林正章

國立成功大學管理學院院長暨
FinTech 商創研究中心計畫主持人

史代序

隨著金融科技的爆紅，奠定比特幣基礎的區塊鏈技術，成為火紅的技術焦點。區塊鏈是一套去中心化、分散式、高可用性的共享交易帳本運作系統，整體網路交易內容無需倚賴統一的資料處理中心，由參與各方多節點共同組成，每個節點保存的資料帳本內容均相同，系統中所有具有維護功能的節點可共同維護數據塊(Block)，數據塊完整儲存在全球各個節點上，因此區塊鏈技術又稱為分散式帳本技術(Distributed Ledger Technology, 簡稱 DLT)。

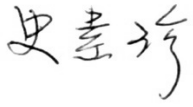
區塊鏈在交易面提供完整性的服務，包括稽核、監管追蹤和所有權轉移等功能。然而最重要的是區塊鏈的出現，使得任何交易都能以最簡單快速的零成本方式發生，無需任何中介協助處理，進而打破了人類社會千百年來交易的潛規則，陌生單位之間無法信任買賣，或者需要有財力支持或權威的中間商來促成才能進行交易的前提。比特幣不需要中央銀行就能誕生發行，此技術可應用到所有原先需要中介作保認證的市場，任何產業領域都可以打造自己的區塊鏈，交易夥伴一起上鏈共治，除了大幅提升交易速度外，亦可避免中間商的層層成本附加，而在價格上帶來更透明、低廉的好處。

關貿網路由一開始經營的加值網路業者(Value-Added services Network, VAN)到應用系統網路服務提供者(Application Service Provider, ASP)，都是基於使用者的信任所建構之中心端服務，區塊鏈的出現，是否意謂關貿網路也可能在去中心化的浪潮下失去其位置，其實不然，相反的，關貿網路一直也扮演國內業者對外的溝通橋樑，區塊鏈為跨國貿易結帳所需貿易文件的真實性提供保證，如區塊鏈技術應用於電子化國際結算，數據與貨幣的流動可以同時流動，將使整個貿易結算流程更為縮短、降低成本、並具備高度安全性。因此，區塊鏈技術不但不會威脅關貿的存在，反可能是關貿網路在跨國貿易資訊服務的無限商機。

區塊鏈技術係屬於一種高度跨領域整合的先進技術，許多行業包括金融從業人員第一次接觸區塊鏈技術時，都表示不易理解透徹，甚至容易產生誤解的認知。正巧關貿的長年好友兼 Java 的啟蒙師—李昇暾教授出版了這本書，以深入淺出的

方式，說古道今，帶領讀者迅速進入區塊鏈的世界，由區塊鏈技術衍伸的新商業模式，再進入區塊鏈的核心—智能合約，並以實例，讓讀者實際體會發布 ICO(Initial Coin Offering)投資的過程。最後以當前廣受重視的議題，實際帶領讀者開發相關的 DApp (decentralized application，去中心化應用程式)，讓讀者能以實例領略區塊鏈之奧妙，值得一讀。

區塊鏈技術從 2015 年起受到所有人的關注，2016 年起金融業更大量投入區塊鏈的應用研究與試驗計劃，雖然區塊鏈已有多項應用經實證可行，仍面臨多種挑戰，但投入技術研發改進者越來越多，可預期的，未來的技術勢必愈來愈成熟，只是大家在專注於此新技術時，更應思考用這項新技術到底要解決什麼問題的目的是什麼，而非為了追求區塊鏈技術，尋找應用場景。其實，無論是中心化或去中心化模型，都可能有相當的互補空間，或許隨著時間的推移及實際應用的發展，區塊鏈技術可能存在一個兼具中心化及去中心化模式的生態系，而非一味去中心化。



關貿網路公司技術長

CHAPTER

03

初探智能合約

智能合約（Smart Contract）是以太坊區塊鏈最具特色的地方，也具有引領塑造新時代的潛力。若要用一句話來介紹智能合約，最簡單的可說是：「一種執行在區塊鏈虛擬機器（VM）上的電腦程式」。

透過智能合約可去中心化，使得離烏托邦世界更進一步。舉例來說，有兩個人想打賭天氣狀況，在過去中心化世界為求公正性，兩人會選擇一位第三方人士擔任中心化的仲裁者，分別將賭金交給公正的第三方，一旦確定賭局結果，再由公正的第三方將獲勝獎金交給勝利的一方。

這聽起來頗為合理，畢竟這種交易方式已運作數千年之久。然而公正的第三方會不會想要抽成呢？而在最極端的情況下，公正的第三方若變得不再公正，反而捲款潛逃，那又該如何是好呢？

請暫時忽略執行智能合約需花費 gas 燃料費的事實。假設同樣的打賭遊戲搬到智能合約場景中，智能合約會依程式邏輯所設定的規則運作，當某個狀態達到設定的條件，例如天氣結果為晴天時自動將賭金移轉給勝利的一方。這樣實在是太好了，從此以後世界運作將不再需要中心化的仲裁者，只要擁抱智能合約即可！

本章架構如下：

- 淺談智能合約
- Hello World 智能合約
- JSON-RPC 遠端存取智能合約

3-1 淺談智能合約

以撰寫 Java 程式為例，程式設計師會依高內聚、低耦合（high cohesion, low coupling）的觀念設計類別（class），當運作需要時便會根據類別所勾勒出來的藍圖將類別實例化，形成可執行的物件實體。智能合約的概念與物件導向程式設計如出一轍，可把它當成是類別設計。

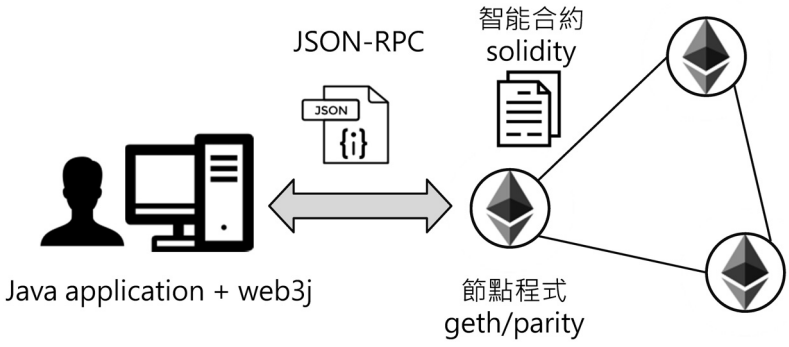
以太坊的智能合約撰寫完後同樣須經過編譯的動作，而順利編譯後即可得到 binary code 及 JSON 格式的 ABI。ABI（application binary interface）是用以描述智能合約所提供的介面（interface）應如何解讀的資訊，即是告知合約使用者該如何取用合約所提供之函數的說明書。

執行 Java 程式時可直接在記憶體中建立物件實體，然而智能合約必須經過部署（deploy，又稱上鏈）的動作。在部署過程中會先利用 ABI 創建智能合約的空殼，再將所取得的 binary code 當做充填資料。吾人可將之想像成是產生一個物件實體，接著透過以太坊的特性將實例化的智能合約廣播到整個區塊鏈網路，隨著礦工挖礦之進行，智能合約便會被寫到區塊鏈中，當部署完成後便會得到該智能合約在區塊鏈中的位址，使用者即可透過此位址及對映的 ABI 來呼叫與使用智能合約所提供的函數，進而改變合約狀態，亦似物件中變數的概念。

區塊鏈是一種分散式系統，若對某節點調用特定的智能合約並改變其狀態，其結果也將被廣播至整個區塊鏈網路。如同分散式帳本，智能合約的狀態改變具有不可否認的特性，因此在某種程度上與真實世界中的合同一樣，具有相當程度的信任基礎。也正因為如此，Ethereum 智能合約具有引領新時代的能力。

撰寫 Ethereum 智能合約有多種程式語言可供選擇，例如 Serpent、LLL、Viper 等，其中以 Solidity 程式語言最廣被使用，本書篇即採用此語言來作介紹。Solidity 的主要開發者為 Gavin Wood、Christian Reitwiessner、Alex Beregszaszi、Liana Husikyan、Yoichi Hirai 與其他幾位早期的貢獻者。Solidity 是一種合約導向式（contract-oriented），多方參考 JavaScript、C++、Python、PowerShell 等的高階程式語言。如同一些高階語言一樣，Solidity 也是一種靜態型別（statically typing）的程式語言，意指在編譯時期即必須宣告變數的型態，同時也支援繼承、函數庫化以及自訂複雜資料型別的能力。

下圖為本書各技術章節範例所採行的系統架構圖。



在此系統架構中，建構底層區塊鏈網路的節點程式可以選用 geth 或是 parity（第二章），而以 solidity 程式語言實作 Ethereum 智能合約（第三、四章），至於 DApp 的前端，則是由 Java 應用程式搭配 web3j 套件來實現（第五、六章）。

本章接下來，將以兩小節關於 solidity 的簡要介紹來讓讀者對智能合約有小試身手的機會。

3-2 Hello World 智能合約

不能免俗地，我們所實作的第一個以太坊智能合約還是先從「Hello World」開始。首先請用您習慣的文字編輯軟體編寫下列智能合約，並將程式內容儲存為.sol 結尾的純文字檔。

檔名：HelloWorld.sol

```
pragma solidity 0.4.22;

contract HelloWorld {

   address owner;
   string greetStr = "hello world";

   constructor() public {
       owner = msg.sender;
   }

    function greet() public constant returns (string) {
      if (msg.sender == owner) {
           return strConcat(greetStr, " ", "boss");
      } else {
           return strConcat(greetStr, " ", "guest");
      }
   }

   function strConcat(string _a, string _b, string _c) private pure returns
(string){
    bytes memory _ba = bytes(_a);
    bytes memory _bb = bytes(_b);
    bytes memory _bc = bytes(_c);

    string memory abcde = new string(_ba.length + _bb.length + _bc.length);
    bytes memory babcde = bytes(abcde);

    uint k = 0;
    for (uint i = 0; i < _ba.length; i++) babcde[k++] = _ba[i];
    for (i = 0; i < _bb.length; i++) babcde[k++] = _bb[i];
    for (i = 0; i < _bc.length; i++) babcde[k++] = _bc[i];
```

```
    return string(babcde);
   }
}
```

本程式之第一行 pragma solidity 0.4.22 宣告了此智能合約所適用的編譯器版本。contract HelloWorld 宣告了後面一對大括號所包裹的內容是一份智能合約程式，讀者可將之觀想成其它程式語言（如 Java）之類別宣告。

address owner 宣告一個型態為「位址型別」，且名稱為 owner 的變數。在以太坊智能合約中，address 型別可用來記錄帳號在區塊鏈中的位址，或另外一份智能合約的位址。string greetStr 則宣告一個型別為字串且名稱為 greetStr 的變數，並設定它的初始內容為「hello world」。

constructor() 望文生義可知是建構者函數，乃是智能合約在上鏈時最先執行的函數。讀者們可想像智能合約上鏈的過程，就好比是傳統物件導向程式語言在記憶體中建立物件實體，因此可將一些初始化的工作放在智能合約的建構者函數中。如範例所示，乃是將執行智能合約上鏈動作的用戶帳號（即 msg.sender）記錄在 owner 變數中。

function greet() public constant returns (string) 乃是宣告一個名稱為 greet 的函數，因函數宣告成 public，任何帳號或智能合約皆可呼叫與使用此函數；constant 則是用來保證執行此函數時，不會更動區塊鏈的任何狀態，意即不會改變任何合約的變數內容，因此呼叫此類型函數的交易毋須經過節點間的共識；最後的 returns (string) 則表示函數執行後將會回傳字串型別的結果。

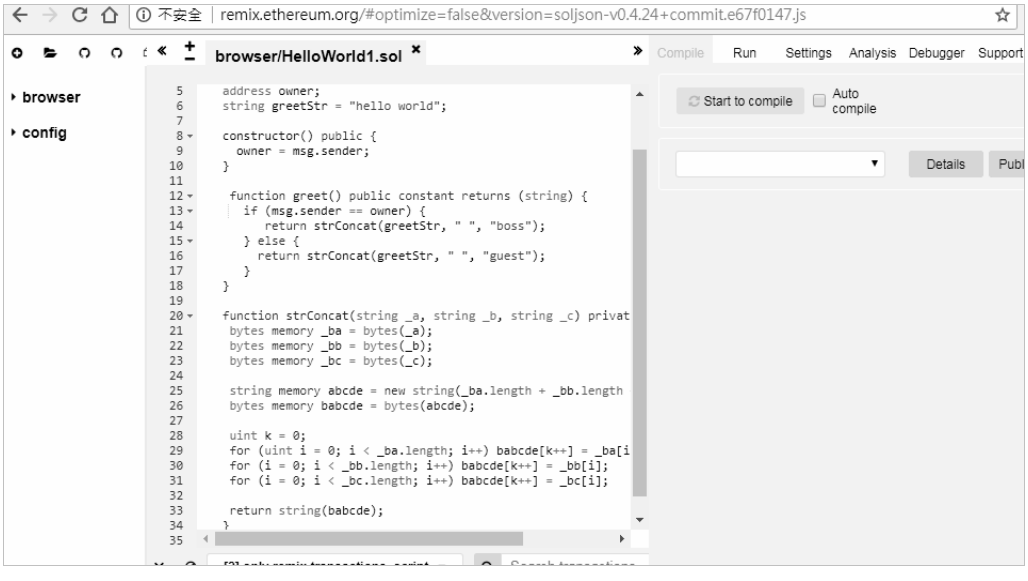
greet 函數使用 if 敘述語法，比對「呼叫 greet 函數的用戶帳號」和「智能合約上鏈的帳號」是否為同一個，若比對結果相同將回傳「hello world boss」字串，反之則回傳「hello world guest」字串。

在此順便一提，以太坊智能合約因先天上的限制，對於字串處理相較其它程式語言較為繁瑣，以本節示範的智能合約 function strConcat 函數為例，若想進行

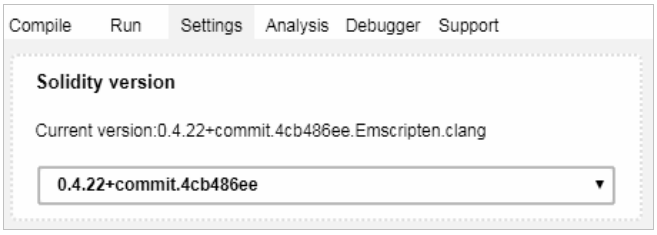
字串合併時，須先將字串轉換成位元組型別，再透過對位元組中每個位元進行複製的方式來達到字串合併的目的，strConcat 函數內容應十分容易理解，因此不再詳細說明。

撰寫合約後與傳統程式無異，接著須對它進行編譯的工作。讀者雖然可在自己電腦安裝適當的編譯軟體，然更簡單的方式是直接透過 solidity 語言的線上編譯器進行編譯工作，請拜訪下列網址：http://remix.ethereum.org/。

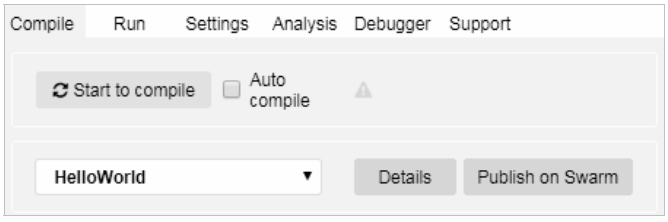
進入網址後，點選編譯器上方的+號，新增一個空白的程式編寫區，並將剛剛寫好的智能合約複製貼上到該編輯區中，即如下所示：



智能合約宣告須使用 0.4.22 版本的編譯器，故請切換到 Setting 頁籤選擇編譯器的版本，請透過下拉式選單選取符合需求的版本。



再切回至 Compile 頁籤點選「Start to compile」鈕，若無顯示任何錯誤訊息則代表編譯工作已順利完成，最後請點選「Details」鈕。



捲動彈跳出來的對話框至 WEB3DEPLOY 區段，此區段中的文字內容即為智能合約的中介碼，可點選複製鈕取得文字內容，再將其內容儲存至.js 結尾的文字檔，例如 HelloWorld.js。

接著再捲動對話框至 ABI 區段，此文字內容就是智能合約的 ABI，它是一段描述如何存取智能合約的介面說明，讀者同樣可點旁邊的複製鈕取得 ABI 文字內容，再將其儲存到.abi 結尾的檔案中，例如 HelloWorld.abi。

下方即為範例合約的 ABI 內容，原先複製所得的 ABI 是以多行且方便閱讀的方式呈現，但為了稍後執行指令方便，請手動調整為單行呈現方式。

```
[{"constant": true,"inputs": [],"name": "greet","outputs": [{"name": "","type":
"string"}],"payable": false,"stateMutability": "view","type":
"function"},{"inputs": [],"payable": false,"stateMutability":
"nonpayable","type": "constructor"}]
```

萬事俱備後便可準備將智能合約上鏈，請將所取得的 HelloWorld.js 檔置於與 geth.exe 相同的目錄中。

在啟動私有鏈之前，為了能透過礦工節點的控制台將智能合約上鏈，需在啟動節點 1 的指令中加入--unlock 0 參數，用來將礦工帳號解除鎖定，因此在啟動節點 1 的過程中，將會多一道詢問礦工帳號密碼的手續。

```
geth --identity "Node1" --networkid 168 --nodiscover --maxpeers 5 --rpc --rpcapi
"web3" --rpcport "8080" --datadir "c:\MyGeth\node01" --port "30303" --mine
--minerthreads=1 --cache=1024 --unlock 0
```

確認具三個節點的私有鏈已正常運作，且具礦工帳號的節點也正常挖礦時，請再開啟另一個 DOS 視窗執行 geth attach 指令來進入 geth 控制台。

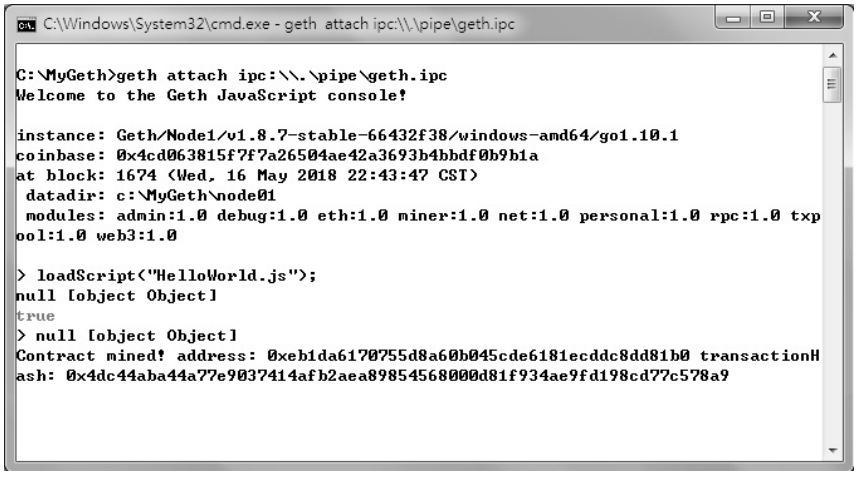
```
geth attach ipc:\\.\pipe\geth.ipc
```

geth 控制台所提供的命令列介面是一種採用 javascript 語法的執行環境，其所使用的函式庫稱為 web3.js。geth 控制台可讓使用者連接到本地端或遠端的以太坊

節點，並允許使用者透過命令列方式佈署、呼叫與使用智能合約。順利進入 geth 控制台後，請輸入下列指令來將智能合約佈署到私有鏈。

```
loadScript("HelloWorld.js");
```

如下即為智能合約成功上鏈的執行結果。從執行結果的訊息中可看到「address: 0x…」字樣，此位址便是智能合約在私有鏈中的位址。



若欲呼叫已經上鏈的智能合約則需由配合剛才所取得的 ABI 介面描述，下列指令是在 geth 控制台使用智能合約的範例，請將 ABI 的內容及智能合約位址分別複製到下列指令的相對位置。

```
var greeter = eth.contract(ABI 的內容).at(智能合約的位址);
```

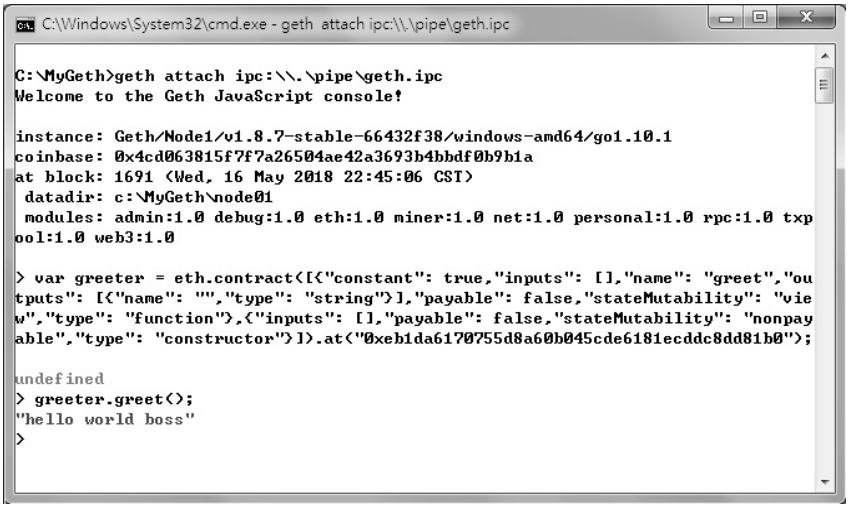
置換後的新指令即如下所示：

```
var greeter = eth.contract([{"constant": true,"inputs": [],"name": "greet",
"outputs": [{"name": "","type": "string"}],"payable": false,"stateMutability":
"view","type": "function"},{"inputs": [],"payable": false,"stateMutability":
"nonpayable","type": "constructor"}]).at("0xeb1da6170755d8a60b045cde6181ecdd
c8dd81b0");
```

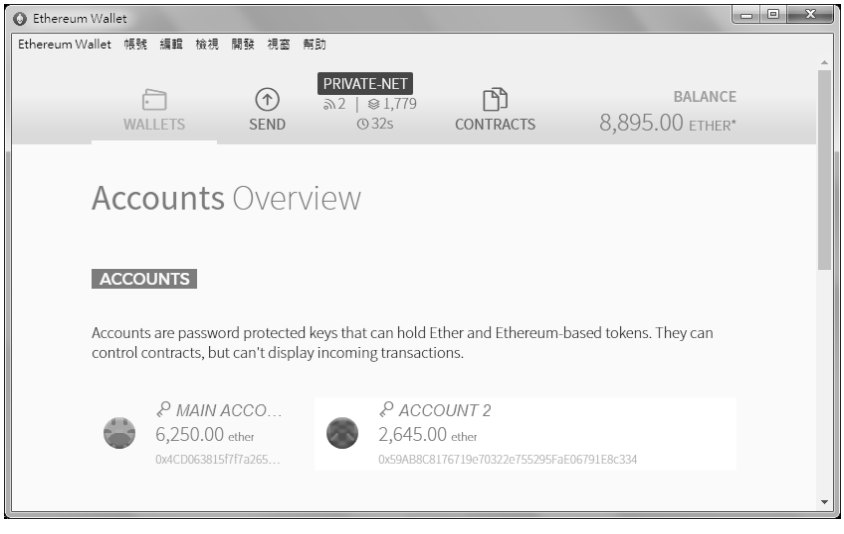
順利執行指令後，變數 greeter 便會對映到區塊鏈中的智能合約，接著即可透過變數 greeter 使用智能合約中的函數，例如：

```
greeter.greet();
```

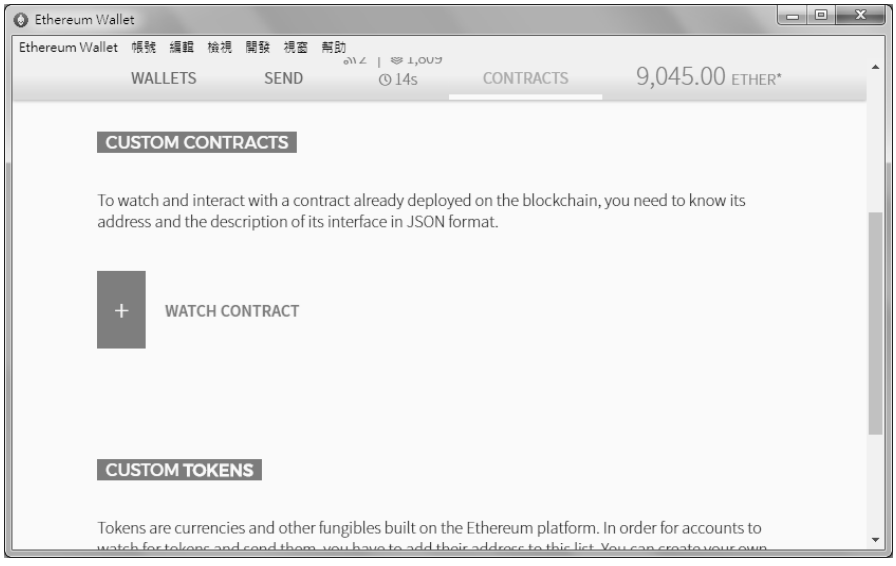
由於目前將智能合約上鏈的帳號和使用智能合約的帳號是同一個，因此執行結果即顯示「hello world boss」。



我們除了透過 geth 控制台使用與呼叫智能合約的外，亦可透過以太錢包軟體調用合約。請開啟錢包軟體後，點選上方的「Contracts」鈕。

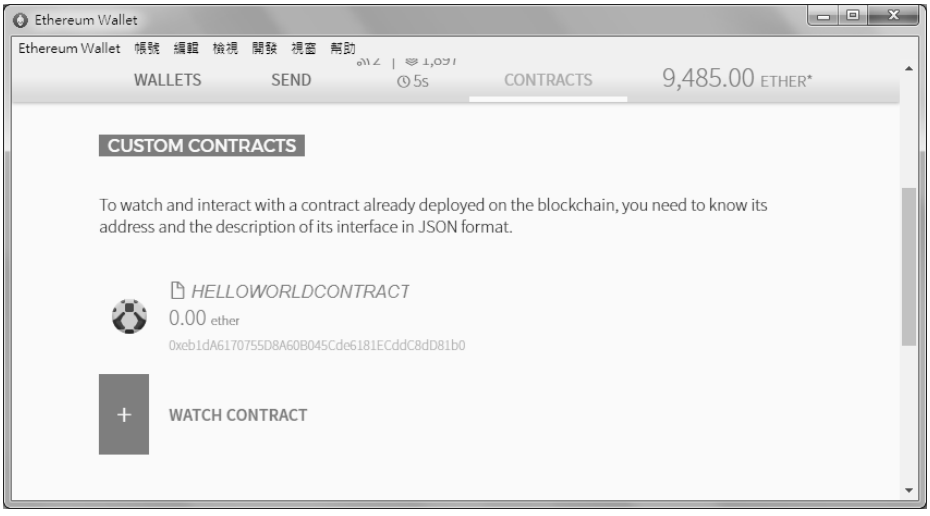


切換至「Contracts」頁面後，請點選下方「Custom Contracts」區段中的「Watch Contract」鈕。

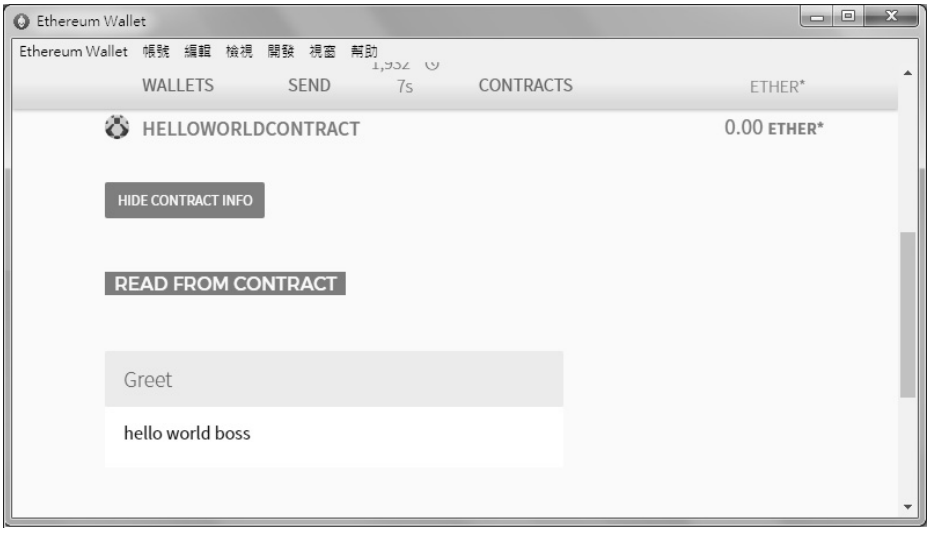


請在彈出的對話框中分別輸入適當的參數：Contract Address 欄位請填入智能合約的位址；Contract Name 請填入任意的合約識別名稱（例如 HelloWorldContract）；Json Interface 欄位請填入 ABI 的內容。全部填妥後請點選「Ok」鈕。

經由剛才一系列的動作，我們已在錢包軟體建立可以與智能合約對映的選項。如下所示，請點選合約名稱以準備使用該合約。



進入合約細節畫面後，請捲軸至「Read From Contract」區段即可看到智能合約提供唯一的 greet 函數。由於 greet 函數宣告為 constant 且同時也毋須填入任何參數，因此在 greet 函數下方便會直接顯示函數執行結果，也就是「hello world boss」字串。



3-3 JSON-RPC 遠端存取智能合約

前一節示範了如何以手動方式利用命令列指令呼叫與使用智能合約，然而在大多數情況下，現代化的應用軟體系統皆會提供友善的操作介面給終端用戶，如同前一節介紹的錢包軟體即為一種讓終端用戶可透過圖形介面使用智能合約的方式。因此學會鏈外應用軟體系統連接存取智能合約是十分重要的，此即為本節所要介紹的重點。

首先請撰寫如下的一個新智能合約，並在該合約中提供一個命名為 doMultiply 的函數，其將傳入的數值進行乘法運算並回傳運算後的乘積，由於 doMultiply 函數並不會更動區塊鏈的狀態，因此可宣告成 constant。

```
pragma solidity 0.4.22;

contract Multiply {

  function doMultiply(uint in01, uint in02) public constant returns (uint) {
   return in01 * in02;
  }
}
```

同樣地請透過線上工具編譯智能合約，取得其中介碼並儲存為 Multiply.js 文字檔。

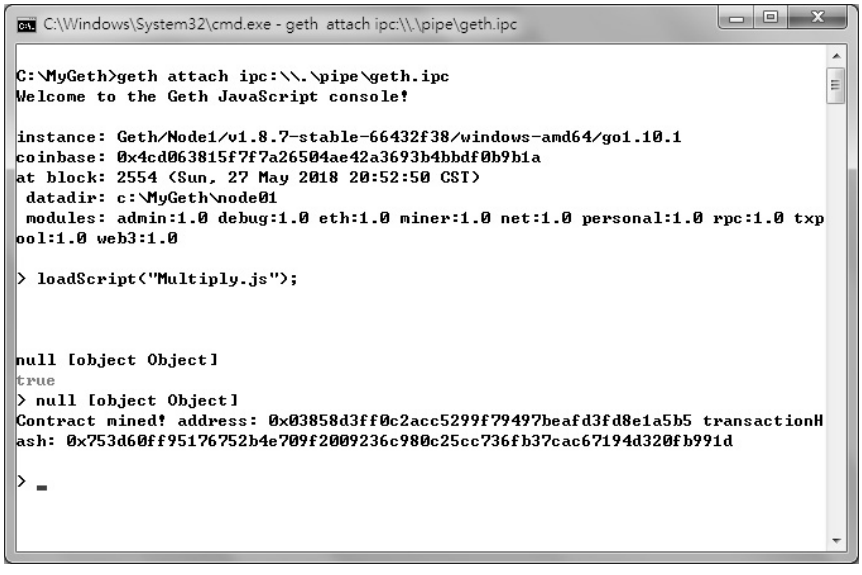
```
var multiplyContract = web3.eth.contract([{"constant":true,"inputs":[{"name":
"in01","type":"uint256"},{"name":"in02","type":"uint256"}],"name":"doMultiply"
,"outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"v
iew","type":"function"}]);
var multiply = multiplyContract.new(
   {
     from: web3.eth.accounts[0],
     data: '0x6080604052348015610010576000080fd5b5060c58061001f6000396000f30060
80604052600436106103f576000357c0100000000000000000000000000000000000000000000000
0000000000900463ffffffff168063648146a2146044575b600080fd5b348015604f57600080fd
5b506076600480360381019080803590602001909291908035906020019092919050505050608c56
5b604051808281526020019150506040518091039f35b60008183029050929150505600a16562
```

```
7a7a723058203b839f390dbccb99f5903002bbc0d82039b5bba59b954e754e71f84828b16cf600
29',
     gas: '4700000'
   }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
         console.log('Contract mined! address: ' + contract.address + '
transactionHash: ' + contract.transactionHash);
    }
 })
```

同時請將所取得的 ABI 儲存為 Multiply.abi。

```
[{"constant": true,"inputs": [{"name": "in01","type": "uint256"},{"name":
"in02","type": "uint256"}],"name": "doMultiply","outputs": [{"name": "","type":
"uint256"}],"payable": false,"stateMutability": "view","type": "function"}]
```

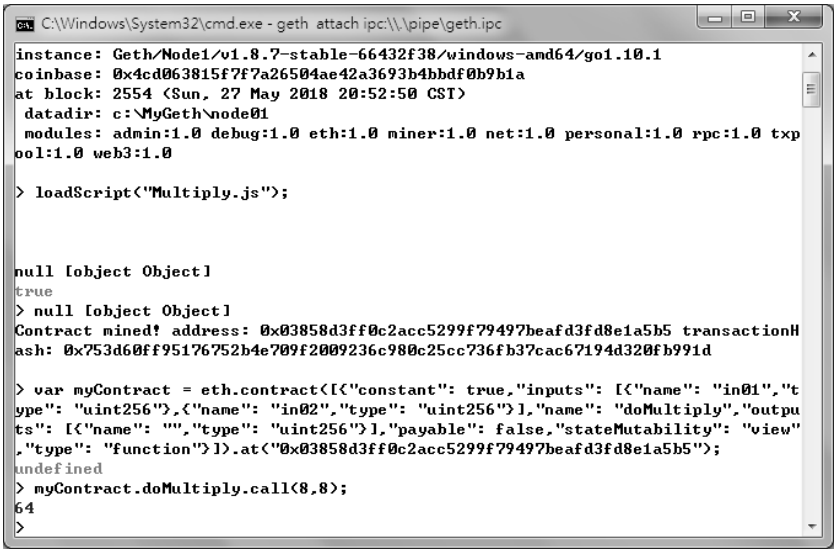
請參考前一節所介紹的步驟，將編譯後的智能合約上鏈。為求慎重，讀者可透過 geth 控制台驗證智能合約是否能正確執行。如下所示，智能合約上鏈後得到的合約位址是 0x03858d3ff0c2acc5299f79497beafd3fd8e1a5b5。



再以智能合約的 ABI 與合約位址做為 eth.contract 指令的輸入值，並以 myContract 變數接收 eth.contract 的回傳值，自此 myContract 變數便會指向該合約。

```
var myContract = eth.contract([{"constant": true,"inputs": [{"name":
"in01","type": "uint256"},{"name": "in02","type": "uint256"}],"name":
"doMultiply","outputs": [{"name": "","type": "uint256"}],"payable":
false,"stateMutability": "view","type":
"function"}]).at("0x03858d3ff0c2acc5299f79497beafd3fd8e1a5b5");
```

接著請在 geth 控制台呼叫合約的 doMultiply 函數。如下所示，輸入的測試資料為 8 * 8，並觀察是否可得到正確的乘積（即 64）。



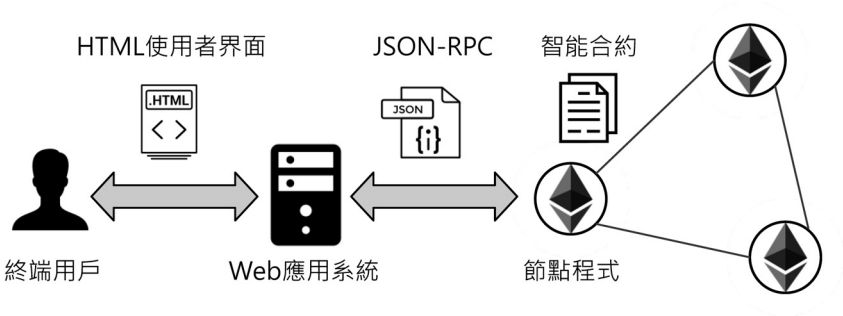
在驗證可正確執行該智能合約後，即可準備開始與鏈外的應用程式進行整合。該如何進行呢？您還記得啟動節點的指令嗎？

```
geth --identity "Node1" --networkid 168 --nodiscover --maxpeers 5 --rpc --rpcapi
"web3" --rpcport "8080" --datadir "c:\MyGeth\node01" --port "30303" --mine
--minerthreads=1 --cache=1024 --unlock 0
```

啟動指令有幾個關鍵的參數，其中 rpc 參數為啟用節點程式的 RPC API 功能；rpcapi 參數設定所要開放的 API 種類；最後 rpcport 參數則用來設定 RPC API 所使用的埠號。

請參考下圖典型的區塊鏈整合系統架構圖，位於中央的電腦主機會動態產製以 HTML 實作的使用者操作界面，同時電腦主機再藉由區塊鏈節點所提供的

JSON-RPC，一種具有無狀態（stateless）、輕量（light-weight）的遠端程序呼叫（remote procedure call, RPC）通訊協訂使用節點的功能，並與區塊鏈進行相互運作，例如佈署與調用智能合約、建立帳號、傳輸加密貨幣等。



以太坊的 RPC API 部份遵循 JSON-RPC 2.0，同時亦具有專屬之規範，例如：

- 數字型態採用 16 進制編碼表示。為支援某些無法表示極大化數字，或有限制條件的程式語言避免數字呈現上的誤差，因此採用 16 進制表示法，並將數字轉換的工作留給應用系統所使用的程式語言自行處理。
- 在共識尚在進行的過程中存取區塊鏈資訊時，無法非常明確地給定區塊編號，因此在以太坊 RPC API 規範中會以字串列舉的方式，以預設區塊編號（default block number）來方便程式開發人員調用智能合約，例如「earliest」代表最早的區塊、「latest」代表最新挖到的區塊、「pending」代表待處理的狀態或交易。影響的 RPC API 包括 eth_getBalance、eth_getCode、eth_getTransactionCount、eth_getStorageAt、eth_call。

雖然，本書所示範的 RPC 埠號為 8080，但其實不同程式語言開發的以太坊節點程式預設埠號有其慣例可循，例如：

程式語言	埠號
Go	8545
C++	8545
Py	4000
Parity	8545