序言

在過去幾年中，「可觀測性」（observability）這個詞從系統工程社群的小眾邊緣，逐漸成為軟體工程社群的日常用語。隨著這個詞彙的重要性日益提升，它也不可避免地與另一個詞彙「監控」（monitoring）混用，並出現一定的相似之處。

接下來的情況就像預料中的那樣令人混淆：監控工具和供應商開始誤用相同的語言和詞彙，這些詞彙原本是用來區分可觀測性的哲學、技術，和融合技術與人文的系統模型基礎和監控之間差異的。這樣的混淆不太有幫助，也可以這樣說，這麼做可能會把「可觀測性」和「監控」混為一談，使人們更難以就它們之間的差異展開有意義、細緻的對話。

僅把監控和可觀測性之間的區別視為語義上的不同可謂愚蠢。可觀測性並不僅僅是一個可以透過購買的「可觀測性工具」（不管供應商怎麼說），或採用當前開放標準來實現的技術概念，更有甚者，「可觀測性」是一個融合技術和人文的系統概念。要成功實施可觀測性，並在軟體的開發、部署、除錯和維護方面得到適當的支持，也要建立起相同重要的文化，而這並不僅僅取決於擁有合適的工具。

在大多數，甚至可以說是所有的情況下，團隊需要同時利用監控和可觀測性，才能成功構建和操作其服務。然而，想要這樣成功實施，絕對需要從業人員先了解這兩者之間的哲學差異。

監控和可觀測性之間的區別在於系統行為的狀態空間，更重要的是，我們想從這狀態空間去探索出什麼層次的細節。這裡所謂的「狀態空間」，指的是系統在各個階段可能出現的所有未預期行為：從系統設計開始，到系統開發、測試、部署，再到系統開放給使用者使用，以及在生命週期中除錯等等。系統越複雜，其狀態空間就越複雜且多變。

可觀測性允許耐心而細緻地對狀態空間進行映射和探索。這種仔細的探索通常是為了更深入了解系統行為中的不可預測、長尾或多模態分布；監控則提供系統整體健康狀態的近似值。

因此，從蒐集資料到如何儲存資料，再到如何探索資料以求更了解系統行為，這些都與監控和可觀測性的目的有關。

在過去幾十年中，監控的概念影響了無數工具、系統、流程和實踐的發展，其中許多已成為業界事實標準。由於這些工具、系統、流程和實踐是專為監控目的而設計的，因此在此方面表現出色。然而，它們不能，也不應該重新命名或包裝成「可觀測性」工具或流程，只為了銷售給毫不知情的客戶。這樣做沒有明顯的好處，還可能為客戶帶來花費眾多時間、精力和金錢投入的風險。

此外，工具只是解決問題的一部分。構建或採用在其他公司已證實成功的可觀測性工具和實踐，並不一定能解決組織面臨的所有問題，因為成品無法講述工具和相應流程背後演變的故事，無法闡明其旨在解決的總體問題、產品中潛在的假設等等。

如果沒有為團隊建立一個有助於成功的文化框架，建立或購買適當的觀測工具無法成為萬靈丹。以監控為核心的觀念和文化，如儀表板、警報、靜態閾值，對於發揮可觀測性的全部潛力沒有什麼幫助。可觀測性工具可能有大量又非常精細的海量資料，是該工具整體可行性和實用性的最終裁判者，甚至可以說是可觀測性本身的裁判者；想要成功理解這些海量資料，需要一個基於假設驅動、迭代除錯的思維方式。

僅擁有最先進的工具並不能自動培養從業人員具備這種思維；此外，僅僅闡述監控和可觀測性之間模糊的哲學區別，也無法幫助實踐者培養這種思維，必須將這些理念提煉成具體的解決方案。例如，本書中有些章節對將日誌（logs）、指標（metrics）和追蹤（traces）視為「可觀測性的三大支柱」持懷疑態度。雖然這些懷疑有一定的道理，但事實上長期以來，日誌、指標和追蹤一直是真實世界運行系統時所擁有的唯一具體遙測示例，因此「三大支柱」的敘述仍然無可避免地會出現。

對於實踐者而言，最重要的是提供一個可行的藍圖，以解決他們所面臨的技術和文化問題，而不僅僅是提出抽象、虛幻的想法。本書成功地填補了可觀測性的哲學原則和實際應用之間的差距，提供了一個具體藍圖，雖然有些主觀，但也展示了如何將這些想法實踐的可能樣貌。

本書深入淺出地介紹將可觀測性理念轉化為實際行動的方法，並提供解決重要技術和文化問題的方案。不僅關注協議、標準或低層級的遙測訊號表示方式，本書還把可觀測性的三大支柱定義為結構化事件（或沒有前後關係的追蹤）、假設驗證的迭代（或以假設驅動來進行除錯），和「核心分析循環」。僅僅依賴遙測訊號本身或使用工具，並不能將可觀測性最大化系統，需要以第一原則重新構建可觀測性的基礎。本書探討了在組織中建立可觀測性文化時可能面臨的挑戰，並提供有價值的指導，確保實踐者從長期成功中獲益。

— *Cindy Sridharan*
基礎架構工程師
San Francisco
April 26, 2022

前言

感謝你閱讀這本主要介紹針對現代化軟體系統來設計可觀測性的書籍，我們的目的是幫助你在工程團隊中發展可觀測性的實踐。本書總結我們作為可觀測性實踐者，和為希望改進自己可觀測性實踐的使用者開發可觀測性工具的經驗。

作為軟體工程中可觀測性概念的倡導者，我們希望透過本書清晰地介紹現代化軟體系統中的「可觀測性」，而這個詞彙在軟體開發生態系統中已廣泛使用。本書目標是透過深入分析以下內容，幫助你區分真相和炒作：

- 可觀測性在軟體交付和運營中的含義
- 構建有助於實現可觀測性基本組件的方法
- 可觀測性對團隊動力的影響
- 擴充可觀測性系統時需要考慮的事項
- 在組織中建立可觀測性文化的實踐方法

誰適合閱讀本書？

本書主要聚焦於軟體工程師在實際環境中開發應用程式的設計和實作，因此對於負責開發應用程式的軟體工程師來說非常有用。不過，任何支援軟體在實際營運環境中運作的人員，都可以從本書的內容中獲益良多。

此外，對於軟體交付和營運團隊的經理來說，如果有興趣了解觀測實踐如何有益於他們的組織，特別是關注團隊動態、文化和規模的章節，本書對他們來說也能派上用場。

此外，任何協助團隊交付和操作營運環境軟體的人，例如產品經理、支援工程師和利益相關者等，或是對於這個新興名詞「可觀測性」（observability）感到好奇，並想知道為什麼大家都在討論它的人，也可以從本書中獲得幫助。

我們為什麼寫這本書？

「可觀測性」（observability）現在已經成為一個熱門的話題，但是不幸的是，總有人錯誤地把它當成「監控」（monitoring）或「系統遙測」（system telemetry）的同義詞。實際上，「可觀測性」是軟體系統的一個特性。此外，只有當團隊採用支持其持續發展的新實踐時，才能有效地利用它。因此，將可觀測性引入你的系統不僅是技術上的挑戰，還是一個文化上的挑戰。

我們對於可觀測性這個議題十分關注，並且希望能大力推廣，因此創立了一家公司，唯一目的就是將可觀測性的優勢帶給所有管理軟體產品的團隊。我們也在推動一個新的可觀測性工具類別，其他供應商也紛紛效仿。

雖然我們都在 Honeycomb[1] 工作，但目的不是要向你推銷我們的工具；相反的，撰寫這本書旨在解釋適應原始可觀測性概念管理現代化軟體系統的方法和原因。你可以使用不同的工具和方式來實現可觀測性，但我們也相信自己有資格撰寫一本有關可觀測性實踐指南的書籍，以推進軟體行業，並詳細說明常見的挑戰和有效的解決方案。無論你選擇什麼工具，你都可以應用本書中的概念，以實踐使用可觀測性構建生產軟體系統的功能。

本書旨在讓你了解，當團隊使用可觀測性來管理其生產軟體系統時，會涉及到各種考慮因素、能力和挑戰。有時，我們可能會提供 Honeycomb 的做法以說明解決共同挑戰的方法，這並不是要推銷 Honeycomb，而是將抽象概念轉化為實際情境，目標仍然是在向你展示如何在其他環境中應用相同原則，不論工具為何。

1 *https://honeycomb.io*

透過本書，你能學到的是……

你將學到「可觀測性」的定義，如何識別可觀測系統，以及「可觀測性」最適合用於管理現代化軟體系統的原因。你將了解「可觀測性」與「監控」之間的區別，以及不同情境下需要使用不同方法的理由。此外，我們也會探討為什麼行業趨勢推動了對「可觀測性」的需求，以及它如何適應新興的領域，例如雲端原生（cloud native）生態系統。

接下來，我們將深入探討可觀測性的基礎知識，確認結構化事件是可觀測系統構成要素的原因，以及如何將這些事件串連成追蹤。事件由軟體內置的遙測產生，你將學習有關開源倡議，例如 OpenTelemetry，它可以幫助開始檢測過程。此外，你還將了解在可觀測系統中用於定位問題源的基於資料的調查流程，以及它與傳統監控中基於直覺的調查流程有何不同。最後，你還將了解可觀測性與監控的區別，以及如何在實踐中結合這兩個概念。

在介紹這些基本的技術概念之後，你將了解到使用可觀測性的社會科技。在營運環境中，管理軟體產品是一種團隊活動。你將學習如何使用可觀測性來加強團隊動力。你還將學習可觀測性如何融入商業流程中，影響軟體供應鏈，並揭示潛在的風險。你也將學習如何實踐這些技術和社會概念，例如使用服務級別目標進行更有效的警報，並深入技術細節，了解為什麼使用可觀測性資料可以使警報更具可行動性和可調試性。

然後，你將學習到推出可觀測性解決方案時面臨的固有挑戰，這部分將從你決定購買還是建立可觀測性解決方案時應考慮的因素開始。可觀測性解決方案的一個必要屬性是，在持續排查期間能夠快速提供答案，因此，我們將向你展示如何處理管理大數據集時高效儲存和檢索的固有挑戰。你還將學習何時引入像事件採樣這樣的解決方案，以及如何判斷其利弊，找到適合你需求的正確方法。此外，你還將學習如何使用遙測流水線來管理大量資料。

最後，本書將探討採用可觀測性文化的組織方法。除了向團隊推廣可觀測性，你還將學習在整個組織中推廣可觀測性實踐的實用辦法，也將學習如何識別和與關鍵利益相關者合作，以技術方法贏得支持，並為採用可觀測性實踐做出商業案例。

第一部分

邁向可觀測性之路

這章節將定義本書中反覆提到的概念，讓讀者學習何謂可觀測性，辨識可觀測系統的方法；以及基於可觀測性的除錯技術，比基於監控的除錯技術更適用於現代化軟體系統管理的原因。

第 1 章探討「Observability」一詞的起源，介紹如何將這個概念應用於軟體系統，並提供一些具體問題，讓讀者得以判斷他們是否擁有一個可觀測的系統。

第 2 章介紹工程師以傳統監控方法進行故障排除和問題定位的做法，並和基於可觀測性的系統所使用的方法比較。本章的描述方式比較理論性，但第二部分會具體呈現技術和工作流程的實現。

第 3 章是由合著者 Charity Majors 撰寫的案例研究，從她的角度講述。本章將第一、二章的概念帶入實際案例研究中，說明轉向可觀測性的時機以及絕對必要性。

第 4 章說明產業趨勢促進對可觀測性需求的原因，以及它如何適應新興領域，例如雲端原生生態系統。

第 1 章

什麼是可觀測性？

在軟體開發產業中，「可觀測性」主題引起許多人的興趣，並經常出現在熱門新主題的清單中。但是，但當熱門話題受到關注時，也就往往容易誤解，需要更深入了解相關細節。本章將探討「可觀測性」一詞的數學起源，並檢視軟體開發從業人員適應它的方式，以描述實際營運軟體系統的特性。

我們還將探討將可觀測性適應到實際營運軟體系統中的必要性原因。傳統的軟體應用程式內部狀態的除錯方法，是針對相對簡單的舊系統而設計的，但隨著系統架構、基礎架構平台和用戶期望的不斷演進，用於理解這些組件的工具並未改變。大體上，許多工程團隊今天仍然在使用幾十年前開發的監控工具和除錯方法，即使他們管理的系統已複雜許多。當傳統工具和除錯方法無法快速找到深藏不露的問題時，可觀測性工具便應運而生。

這一章會幫你了解「可觀測性」的意思、如何判斷一個軟體系統是否具有可觀測性，可觀測性的必要性原因，以及當其他方法都沒辦法的時候，如何用可觀測性方式來找出問題。

可觀測性的數學定義

「可觀測性」（Observability）這個詞最初是由工程師 Rudolf E. Kálmán 於 1960 年提出，自此之後，它在不同領域內就有了許多不同的含義。在談論現代化軟體系統之前，讓我們先了解一下不同領域內對這個概念的應用。

Kálmán 在 1960 年的論文介紹可觀測性的特質，用以描述數學控制系統中的概念[1]。在控制理論[2]中，可觀測性的定義為，從系統的外部輸出中推斷其內部狀態[3]的能力。

按照這個定義，可觀測性和可控制性是數學上的相對概念，與傳感器、線性代數方程和形式方法密切相關。這種傳統的可觀測性定義主要適用於機械工程師，和管理具有特定目標狀態的物理系統人員。

但如果你正在尋找以數學和工程過程為導向的教科書，那你來錯地方了。這些書籍確實存在，任何機械工程師或控制系統工程師都會告訴你，而且會充滿熱情且詳細地講述可觀測性在傳統系統工程術語中具有的正式含義。然而，當這個概念運用至更為靈活的軟體系統時，它開啟了一種截然不同的方式，讓你與你開發的程式碼互動和理解。

將可觀測性應用於軟體系統

Kálmán 的可觀測性定義可以應用於現代化軟體系統。不過，應用可觀測性的概念時必須考慮軟體工程領域的特定因素。為了使軟體應用具有可觀測性，必須能夠完成以下事項：

- 理解應用程式的內部運作方式
- 理解應用程式可能出現的任何系統狀態，甚至是前所未見過且無法預測的新狀態
- 只需透過觀察和使用外部工具，就能了解內部運作和系統狀態
- 理解內部狀態，而不需要編寫任何新的自定義程式碼來處理它；因為這意味著需要知道才能解釋

1 Rudolf E. K.lm.n, "On the General Theory of Control Systems"（*https://oreil.ly/u7BM4*）, *IFAC Proceedings Volumes* 1, no. 1 (August 1960): 491–502.

2 *https://w.wiki/4wHw*

3 *https://w.wiki/55Pc*

以下問題可作為判斷上述條件是否成立的標準：

- 你能夠不斷地回答有關你的應用程式內部運作的開放式問題，以解釋任何異常情況，而不會遇到調查死胡同的情況嗎？即問題可能在某些東西中，但你無法進一步分解以確認
- 你能夠了解你的軟體的任何特定用戶，在任何給定時間正在經歷的情況嗎？
- 你是否能夠迅速查看你關心的系統性能的任何橫截面，從整體視圖到可能導致緩慢的單個確切用戶請求以及其中間的任何層次？
- 你是否可以比較任意的用戶請求組，通過識別所有遇到應用程式中異常行為的用戶所共享的屬性，來找出可疑的行為模式？
- 在找到某個特定用戶請求中的可疑屬性後，你是否可以在所有用戶請求中搜索，以識別類似的行為模式來證實或排除你的懷疑？
- 你能否確定哪個系統用戶產生的負載最大，因此大幅降低應用程式性能？以及第 2、3 甚至是第 100 個產生最大負載的用戶？
- 你能否確定哪些用戶的負載最大且最近開始影響性能？
- 如果第 142 個速度最慢的用戶抱怨性能速度，你是否可以隔離他們的請求，以了解為什麼特定用戶的速度如此之慢？
- 如果用戶抱怨發生超時，但你的圖表顯示 99th、99.9th、甚至 99.99th 百分位的請求速度很快，你是否可以找到隱藏的超時？
- 你是否可以在不需要先預測將來某天可能會被詢問這類問題（需要提前設置特定監控來匯總必要資料）的情況下，回答類似前面的問題？
- 即使你從未遇到或調試過此類特定問題，你是否仍然可以回答關於應用系統的這些問題？
- 你是否可以快速獲取前面那些問題的答案，以便你反覆提出一個又一個的新問題，直到找到問題的正確來源，而過程中不會中斷思路，通常意味著在幾秒鐘內而非幾分鐘內獲得答案？
- 即使以前從未發生過這種特定問題，你是否仍然可以回答前面那些問題？
- 你在除錯或調試後的結果，是否經常讓你感到驚訝，因為揭示了新的、令人困惑的和奇怪的發現，還是你通常只找到預期可能發現的問題？

- 無論問題有多複雜、深埋或隱藏在你的技術堆疊中，你是否能在很短的時間內，也就是幾分鐘之內找出並隔離系統中的任何故障？

對許多軟體工程組織而言，滿足上述所有標準是一個非常高的要求。如果你能達到這個標準，毫無疑問，你就能明白為什麼可觀測性已經成為軟體工程團隊熱門話題。

簡單來說，我們定義的「可觀測性」，就是一個評估你是否能理解和解釋系統可能出現的千奇百怪狀態標準。在一個臨時的反覆調查中，你必須能夠在系統狀態資料的所有維度以及維度組合中，相對地去調試那些千奇百怪狀態，而不需要事先定義或預測這些調試需求。如果你在**不需要發布新程式碼的情況下**，就能理解任何千奇百怪狀態，就表示你具有可觀測性。

我們認為，以這種方式將傳統的可觀測性概念應用於軟體系統是一種獨特的方法，其中包含了值得探討的額外細節。對於現代化軟體系統來說，可觀測性與資料類型或輸入資料無關，也與數學方程式無關。它關乎的是人們如何與複雜系統互動並試圖理解它們。因此，可觀測性需要認識到人與技術之間的互動，以理解這些複雜系統如何協同工作。

如果你接受這個定義，還有很多其他問題需要解答：

- 如何蒐集這些資料並將其整理以供檢查？
- 處理這些資料的技術要求是什麼？
- 需要哪些團隊能力才能從這些資料中受益？

在閱讀本書的過程中，我們將討論這些問題以及更多相關內容。現在，讓我們為在軟體領域應用可觀測性增加一些額外的背景知識。

將可觀測性應用於軟體系統與控制理論有很多共通之處。但可觀測性的實用性較高，而數學應用較少。部分原因是軟體工程是一個比其前身，已成熟的機械工程更新，發展也更迅速的學科。營運用的軟體系統通常不像數學理論那樣經過嚴謹證明，這種不夠嚴謹的狀況，某種程度上是因為這一行業在實際運行自己寫的軟體時遇過的各種問題和挑戰，讓我們「吃足苦頭」。

作為軟體工程師，我們試圖理解如何彌補理論和實踐之間的差距，尤其是當程式運行規模成長時。不用尋找一個新的術語、定義或功能來描述如何辦到這點；相反的，是因為迫於管理系統和團隊的現況，而不得不從像監控這樣不再具有效率的概念中重新發展。身在業內，我們需要超越當前的工具和術語差距，克服由於中斷和缺乏更主動解決方案而帶來的痛苦和折磨。

可觀測性是解決這一差距的解決方案。現今的系統很複雜，有很多技術和社會因素造成了混亂，因此需要社會和技術方面的解決方案以幫忙擺脫困境。可觀測性無法解決所有軟體工程問題，但它的確可以幫助你清楚地看到軟體的晦澀角落發生什麼事，否則你只能摸黑摸索，試圖找出問題。

假設你要在星期六早上為家人做一頓豐盛的早午餐，你計畫多道菜餚，包括複雜的起司奶酥蛋食譜，和考慮到每個人的過敏源與不喜歡食物的清單，加上時間緊迫，因為奶奶必須在中午前趕到機場。這本身就是一個不小的挑戰，然後你發現和我們一樣嚴重近視的你，現在找不到眼鏡。想要解決軟體工程中的實際和時間敏感問題時，可觀測性是一個非常好的起點。

關於軟體可觀測性的錯誤認知

在繼續之前，需要解決另一個關於「可觀測性」的定義，這是軟體即服務（SaaS）開發者工具供應商普遍宣傳的定義。這些供應商堅稱「可觀測性」沒有任何特殊含義，它僅僅是「遙測」的另一個同義詞，與「監控」無異。這個定義的支持者認為可觀測性不過就是了解軟體運作方式的另一個通用術語，你會聽到這個觀點的支持者，將可觀測性解釋為他們今日通用以販售給你的「三大支柱」：指標、日誌和追蹤[4]。

這個定義是否有問題？其實我們也難以下定論。或許這個詞彙是多餘的，畢竟，到底為什麼會需要另一個「遙測」的同義詞？或者可能會造成認知上的混亂，為什麼要將不同資料類型混合在一起，並照時間順序視覺化？不管怎樣，這個定義的邏輯缺陷在於，它的支持者有一個既得利益，他們想要推銷集中蒐集和儲存資料的工具和心態，並利用他們現有的指標、日誌和追蹤工具套件。這個定義的支持者讓他們的商業模式限制對未來可能性的想法。

4 有時會加入時間跨度來表示「變化的離散事件」。這可以被視為遙測的一個擴充面向或第四大支柱。

公平地說，我們這些本書作者也是可觀測性領域的供應商。但是，撰寫這本書不是為了推銷我們的工具，而是為了解釋將可觀測性的原始概念應用於管理現代化軟體系統的方式和理由。無論你選擇哪種工具，都可以應用本書中的概念來實踐具有可觀測性的營運軟體系統建設，不用藉由營銷將不同工具拼湊在一起，才能實現可觀測性；不需要使用某個特定工具，才能在軟體系統中實現可觀測性。*相反的，我們認為可觀測性需要改變蒐集有效調試或除錯所需資料的方式*。我們相信，這一行是時候改進用來管理現代化軟體系統的實踐了。

為什麼可觀測性現在很重要？

既然已經知道現代軟體系統中「可觀測性」的意義和限制，就來談談為什麼現在轉變到這種方法較為重要。簡單來說，使用傳統方法，即通過軟體指標和監控方式來了解系統正在做什麼的方法，已經遠遠不夠了。這種方法只是被動地回應問題，過去可能足以滿足產業需求，但現代化系統需要更好的方法論。

在過去的二、三十年中，硬體、系統操作者和機器之間的交互，受一組大多稱為「監控」的工具和慣例所規範，從事此領域的人員大多已經接受這一組工具和慣例，並將其視為理解虛擬空間和他們的程式碼之間的最佳方法。即使在很多情況下，這種方法固有的局限性讓他們在深夜感到困擾而無法入睡，他們仍然對它產生了信任感，甚至可能有些喜愛，因為那是他們所擁有的最佳工具。

使用傳統監控方法讓現在的軟體開發者無法完全了解他們的系統運作，必須不斷地盯著系統，試圖預測和避免所有可能的錯誤和故障，並設置效能門檻，根據自己的主觀判斷宣布它們為「好」或「壞」。他們還會部署一些代表自己檢查和重新檢查這些閾值的機器人，以機器人為中心，將自己的發現蒐集到儀表板中，然後組織成團隊、輪班和逐級上報；當這些機器人告訴他們表現不佳時，他們會發出警報。隨著時間的推移，這種方法就像是花園裡的園丁，需要不斷地修剪、微調和處理它們生長的噪音訊號。

這真的是最好的方式嗎？

數十年來，開發人員和運維人員一直都使用監控來了解系統。監控已經成為了理解系統的實際做法，以至於他們傾向於認為這是了解系統的*唯一方法*，而不僅僅是*其中一種方法*。監控變得非常普遍和常規，以至於人們可能不太注意到它的存

在，因為它已經成為了日常生活中的一部分。處在這一行中，我們通常不會質疑*應該*做的事，而是做的*方法*。

監控實踐的基礎建立在許多未明確提及的系統假設之上，後面將會詳細介紹。但隨著系統不斷演進，它變得更加抽象和複雜，加上底層組件的重要性逐漸降低，讓這些假設變得越來越不成立。隨著開發人員和運維人員繼續採用更現代化的方法，來部署和運行軟體系統，例如 SaaS 依賴、容器編排平台或分散式系統等，這些假設中的漏洞也就越來越明顯。

有越來越多人發現，傳統的監控方法已經遇到很多限制，對於理解現代世界的複雜系統已經無效且不實用。過去所使用的監控指標和假設已經過時，需要使用新的方法來了解系統。要理解它們失敗的原因，可以從歷史和應用背景起源來分析。

為什麼指標和監控不夠用？

1988 年通過的簡單網絡管理協議（Simple Network Management Protocol, SNMPv1，如 RFC 1157 中定義），催生了監控的基礎：指標（metric），這是一個可以選擇性地加標籤的數字，方便分組和搜索，而且成本低廉。它們具有可預測的儲存空間占用，易於沿著規律的時間序列進行聚合。因此，指標成為此後遙測資料的基本單位，從遠程端點蒐集用於自動傳輸，到監控系統的資料都是。

許多複雜的設備都以指標為基礎構建：時序資料庫（TSDB）、統計分析、繪圖庫、炫麗的儀表板、值班輪換、運營團隊、升級策略，以及大量方法來消化和回應那些小型機器人軍團告訴你的訊息。

然而，使用指標和監控工具理解系統複雜性有限；一旦越過界限，變化就會突然出現。上個月還能好好運作的方法，現在說無效就無效，只能退回到使用低級命令，如 strace、tcpdump 和數百個 print 語句，回答有關系統表現的問題。

很難精確計預測什麼時候系統會達到臨界點。最終，系統可能會產生太多狀態，超過你團隊能夠用過去的經驗來分析故障的數量。這些獨特的狀態需要不斷理解，你的團隊已經不知道要創建多少個儀表板來顯示故障模式了。

監控和基於指標的工具，在設計時對你的架構和組織有一定的假設，這些假設在實踐中充當了複雜性的上限。通常情況下，這些假設在你超過它們之前都是隱藏的；而一旦超過，它們就會變成你理解系統問題的痛苦來源。這些假設可能包括：

- 你的應用程式是一個單體應用程式。
- 你運行一個有狀態的資料儲存系統（即資料庫）。
- 提供許多低層系統指標，例如常駐記憶體和 CPU 負載平均值。
- 應用程式運行在由你控制的容器、虛擬機器（VM）或實體機器上。
- 系統指標和檢測指標是調試程式碼的主要資訊來源。
- 你擁有一組相對穩定且長時間運行的服務節點、容器或主機，以長期監控。
- 工程師只會在問題發生後，才想到去檢查系統中的問題。
- 儀表板和遙測資料主要為運維工程師提供服務。
- 監控將「黑盒」應用程式視為與本地應用程式相同的方式檢查。
- 監控的重點是正常運行時間和故障預防。
- 相關性檢查僅涉及有限的維度或維度較少。

相較於現代化系統，傳統的監控方法存在多個缺點。現代化系統通常具有以下特點：

- 應用程式架構包含許多服務。
- 存在多樣化的持久性儲存，包括多種資料庫和儲存系統。
- 基礎設施非常靈活，可以彈性地增減。
- 需要管理許多分散且鬆散耦合（loose coupled）的服務，其中很多並不直接受控制。
- 工程師會主動檢查營運環境程式碼的變更情況，以便在微小問題對用戶產生影響之前及早發現。
- 儀表板無法理解複雜系統中發生的情況。

- 軟體工程師在實際營運環境中的部署有自己開發的程式碼，並激勵工程師主動為程式碼進行儀表板監控，並在部署新變更時檢查性能。
- 可靠性的重點是如何容忍持續不斷的劣化，並藉由使用像錯誤預算、服務質量和用戶體驗等構造，來增強對影響用戶的失敗的恢復力。
- 關連性檢查需要在幾乎無限數量的維度上進行。

最後一點非常重要，因為它描述了在現代化系統架構現實和相關知識的極限之間的差距。發現性能問題背後的基本關連性變得如此複雜，以至於沒有人類的大腦可以理解，甚至沒有任何模式可以包含所有的維度。

在可觀測性中，高維度和高基數資料是發現複雜系統架構中潛在問題的關鍵組件。

以指標和可觀測性除錯的區別

當系統變得複雜到超出我們的記憶能力與理解能力時，也就無法再用直覺和過去的經驗來解決問題了。

作為一名工程師，你可能習慣於以直覺來調試除錯。為了找到問題的根源，你可能會相信直覺，或者依靠對過去故障的瞬間回憶，來展開調查。然而，在這樣的複雜系統中，過去的技能可能已經不再適用。直覺方法只有在遇到的大部分問題，都是過去遇到的那幾個可預測內容變化時才能發揮作用[5]。

同樣地，基於指標的監控方法非常依賴於過去遇到的已知故障模式。監控有助於檢測當系統超過或低於之前認定為異常的可預測閾值時；但是，當你可能甚至不知道這種異常存在時，會發生什麼事？

過往，軟體工程師所面對的大多數問題，都是某種程度上可預測的故障模式的變體。也許你不知道你的服務會以這種方式失敗，但如果你對這個情況及其組件進行推敲，發現一個新的錯誤或故障模式並不會讓人措手不及。事實上，大多數軟體工程師很少遇到真正無法預測的邏輯跳躍，因為他們通常不必處理使這些跳躍結果變得普遍的複雜性；直到現在，大多數工程師所面臨的複雜性，仍集中在單體應用程式內。

5 更深入的分析請參閱 Pete Hodgson 的部落格文章〈Why Intuitive Troubleshooting Has Stopped Working for You〉（*https://oreil.ly/JXx0c*）。

> 每個應用程式都具有一定程度的不可簡化複雜性。唯一的問題是：誰會需要處理它？使用者、應用程式工程師還是平台開發工程師？
>
> — *Larry Tesler*

現今分散式系統的架構，常常以無人能預測、之前也無人經歷過的方式出現故障[6]。這種情況經常發生，以至於創造出一整套的語句，用以描述初涉分散式架構的工程師常常會做出的錯誤假設。現代分散式系統通常透過抽象的基礎設施平台提供給應用程式開發人員，但作為這些平台的用戶，應用程式工程師現在必須面對不可化簡的複雜性，因為這些複雜性直接落在他們的工作上。

以前，模組與函式之間的複雜互動，隱藏在單個物理機器的隨機存取記憶體中，但現在已表面化為主機之間的服務請求。這些新的複雜性需求跨越多個服務，在單一函式的過程中，多次穿越不可預測的網絡。當現代化架構開始將單體解構為微服務時，軟體工程師失去了使用傳統除錯器逐步追蹤他們的程式碼的能力。但與此同時，他們手上的工具尚未適應這種劇變。

簡而言之：我們將單體應用拆分，現在每個請求都必須在網絡上多次跳躍，每個軟體開發人員都需要更熟練地掌握系統和操作，才能完成日常工作。

這種劇變的例子包括容器化趨勢、容器編排平台的興起、轉向微服務架構、多語言的普遍存在、服務網格的引入、瞬態的自動擴展實例、無服務器計算、Lambd函式和其他無數的 SaaS 應用。軟體工程師工具箱內的工具逐漸多樣化。將這些各種工具串聯成現代化系統架構，意味著在請求到達你可以控制的邊緣之後，它可能會經過 20 到 30 次跳轉（如果請求包括資料庫查詢，次數可能乘以 2）。

在現代雲端原生系統中，最具挑戰性的除錯排查不再是理解程式碼如何運行，而是找出系統性能瓶頸，或問題在系統的哪些節點上變得非常具有挑戰性。因為系統中的請求通常會在不同節點之間往返傳送，這些請求可能會在不同的節點之間產生迴圈，進而讓你很難透過儀表板或服務地圖，來判斷哪些節點或服務出現了瓶頸。當系統中的某個部分變慢時，整個系統都會變慢。更具挑戰性的是，由於雲端原生系統通常作為平台運作，程式碼可能位於該團隊甚至不受控制的系統部分中，這使得定位問題變得更加困難。

6 *https://w.wiki/56wa*

在現代化的架構中，透過指標除錯排查，需要連接數十個不相關的指標，這些指標在執行任何一個特定請求的過程中記錄下來，它們跨越任意數量的服務或機器，用來推斷可能發生的情況。這數十個線索的幫助程度完全取決於某人是否能夠預測到那個測量超過或低於閾值，這意味著此操作有助於創建以前未遇到的異常失敗模式。

相比之下，透過可觀測性除錯建立在一個非常不同的基礎上：深入了解在進行此操作時發生的事。使用可觀測性除錯，關乎於盡可能保存有關任何給定請求當前情境的上下文資訊，以便你可以重建觸發導致新型故障模式的錯誤環境和情況。監控是針對已知的未知事物，而可觀測性則針對未知的未知事物。

基數的作用

在資料庫的情境中，**基數**（*cardinality*）指的是一個集合中資料值的唯一性，**低基數**意味著該欄位在其集合中有很多重複值，**高基數**意味著該欄位包含了相當高比例完全不同的值。包含一個值的欄位將始終具有最低可能的基數；而包含唯一 ID 的欄位，將始終具有最高可能的基數。

例如，在一個包含一億個用戶紀錄的集合中，你可以假設任何通用唯一識別碼（UUID）都將具有最高可能的基數。另一個高基數的例子可能是公鑰簽名。名字和姓氏將具有較高的基數，但比 UUID 低，因為有些名字會重複。像性別這樣的欄位在 50 年前的模式下將具有低基數，但考慮到最近對性別的理解，也許已經不再是如此。如果你的所有用戶都是人類，像物種這樣的欄位將具有最低可能的基數。

基數對於可觀測性非常重要，因為**高基數資訊幾乎都是在排查或了解系統資料的最有用資訊**。考慮按照用戶 ID、購物車 ID、請求 ID 或其他眾多 ID，如實例、容器、主機名、編譯序號及跨度等排序的有用性。能夠根據唯一 ID 查詢是在任何給定資料堆中精確定位出單個資料的最佳方式。你可以將高基數值降採樣為低基數值，例如按前綴對姓氏分類，但你永遠無法做反向操作。

不幸的是，在任何合理的規模下，基於指標的工具系統只能處理低基數度的欄位。即使你只需要比較幾百個主機，在基於指標的系統中，你也無法使用主機名稱作為識別標籤，否則就會超過基數空間的限制。

這些固有的限制對資料的查詢方式意外產生了限制。當使用指標排查時，對於可能想要查詢的每個問題，你必須在發生錯誤**之前**，事先決定需要記錄在指標中的資訊。

這造成兩個重要的影響。首先，在調查過程中，如果你決定先提出其他問題以發現潛在問題的來源，那這些問題將無法在事後完成；你必須先設置可能回答該問題的指標，然後等待問題再次發生。其次，因為回答這些額外問題需要另一組指標，大多數基於指標的工具供應商將向你收取記錄該資料的費用，每當你決定以新的方式詢問你的資料以尋找可能無法預測的隱藏問題時，你的成本都將直線上升。

維度的作用

在資料分析中，基數是指資料中值的唯一性，而**維度**（*Dimensionality*）是指資料中的欄位數量。在可觀測系統中，遙測資料以任意廣泛的結構化事件形式生成（詳見第 8 章）。這些事件之所以稱為「廣泛」，因為它們可以並且應該包含成百上千個關鍵字與值的成對（或維度）。事件越廣泛，捕獲事件發生時的情境與背景資訊就越豐富，因此在以後除錯時，可以發現更多關於事件發生的情況。

假設你的事件架構定義了每個事件的 6 個高基數維度：`time`、`host`、`app`、`user`、`endpoint` 和 `status`。有了這 6 個維度，你可以創建任意組合的分析維度查詢，以找出可能導致異常的相關模式。例如，你可以查詢「在過去半小時內主機 `foo` 上發生的所有 502 錯誤」、「由用戶 `bar` 發出的對 `/export` 端點的請求產生的所有 403 錯誤」，或「在應用程式 `baz` 發出的對 `/payments` 端點的請求中發生的所有超時情況，以及它們來自哪個主機」。

有了這 6 個基本維度，你就可以檢查一組有用的條件，以確定你的應用程式系統可能發生的事。現在想像一下，你可以檢查包含數百或數千個維度，這些維度包含任何看起來可能對你的排查和除錯目的有用的細節、值、計數器或字串。例如，你可以包括以下類似的維度：

```
app.api_key
app.batch
app.batch_num_data_sets
app.batch_total_data_points
app.dataset.id
app.dataset.name
app.dataset.partitions
```

```
app.dataset.slug
app.event_handler
app.raw_size_bytes
app.sample_rate
app.team.id
...
response.content_encoding
response.content_type
response.status_code
service_name
trace.span_id
trace.trace_id
```

有了更多可用的維度，你可以檢查事件，進行高度複雜的相關性分析，以發現在任何一組服務請求之間隱藏或難以捉摸的模式（見第 8 章）。在現代化系統中，可能發生的故障組合無限，因此僅捕獲一些基本維度是不夠的。

你必須蒐集與用戶、程式碼和系統交集相關的所有豐富細節。高維度資料提供了更多關於這些交集如何展開的上下文資訊；在後面的章節中，我們將介紹如何分析高維度資料（通常包含高基數資料），以揭示系統問題發生的位置以及原因。

利用可觀測性排查除錯

相較於監控工具限制監控資料的基數和維度，可觀測性工具鼓勵開發者蒐集每個可能發生事件的豐富遙測資料，傳遞每個請求的完整內容，並將其儲存以供日後使用。可觀測性工具專門設計用於高基數、高維度資料的查詢。

因此，在除錯時，你可以以任意數量的方式查詢事件資料，探索系統的狀態，並提出你未曾預測到的問題，以找到下一個問題的答案或線索，如此不斷，直到找到你所尋找的解決方案。可觀測性系統的一個關鍵功能，是能夠以開放的方式探索系統。

一個系統的可探索性取決於你如何提出任何問題，並檢查其相應的內部狀態。**可探索性**意味著你可以持續地調查，並最終理解系統所處的任何狀態，即使你以前從未見過那個狀態，也不需要提前預測這些狀態的可能性。同樣，可觀測性意味著你可以理解並解釋系統可能遇到的任何狀態，無論多新穎或奇特，而無需發布新程式碼。

監控能夠有效運行這麼長時間，往往是因為系統夠簡單，使工程師可以推理出他們可能需要在哪裡尋找問題，且這些問題可能會如何呈現。例如，當 socket 填滿時，CPU 會超載，解決方案是藉由擴展應用節點實例，或調整資料庫等，來增加更多容量。整體而言，工程師可以預測大部分可能的故障狀態，並在應用程式運行在營運環境中時，以極為艱難的方式發現剩下的問題。

然而，監控導致了系統管理根本上的被動應對。你可以捕捉到預測並知道要檢查的故障狀態。如果你知道會發生，就先檢查它。但對於你不知道要查找的任一狀況，必須先遇到並面對這不愉快的驚喜，盡最大努力調查，可能還會陷入死胡同，需要多次遇到相同狀況才能正確診斷；然後，才能開發檢查它的方法。在這種模式下，工程師往往不願意面對可能導致不可預測故障的情況，這也是為什麼，有些團隊害怕部署新程式碼（關於這個話題，稍後會有更多討論）。

另外一個微妙的觀點是：相對於你的程式碼或用戶產生的問題，硬體和基礎設施問題還比較簡單。從「大多數問題都是組件故障」，轉變為「大多數問題與使用者行為或微妙的程式碼錯誤和交互有關」，這是為什麼即使是擁有單體和簡單架構的人，也可能追求從監控到可觀測性的轉變原因。

現代化系統的可觀測性

在軟體工程領域中，一個系統能夠觀測的程度，取決於是否能夠在不進行任何猜測、預測故障模式或發布新程式碼的情況下，理解內部系統狀態。這種觀測性概念來自控制理論，延伸到軟體工程領域中。藉由這種觀測性概念，可以實現對系統的透明度，並能夠更有效地理解和解決問題，同時也能減少故障修復所需的時間和代價。

在軟體工程的領域中，即使在傳統的架構或單體系統中，可觀測性也是有益的。能夠追蹤程式碼並查看耗時分布，或者從用戶的角度重現行為，總是有所幫助。不管你的架構如何，這種能力確實可以幫助團隊避免在營運環境中發現不可預測的故障模式。但是，在現代化分散式系統中，可觀測性工具提供的手術刀和探照燈絕對不可或缺。

在分散式系統中，相對可預測的故障模式而言，新穎和從未見過的故障模式比例傾向更為奇特和不可預測。這些不可預測的故障模式發生頻率很高，但重複出現的次數很少，超出大多數團隊設置適當且足夠相關的監控儀表板，以便工程團隊確保其開發應用程式持續運行時間、可靠性和良好性能的能力。

本書撰寫時，考慮了這些現代化系統。任何由許多鬆散耦合、具有動態特性且難以理解的組件所構成的系統，都非常適合使用可觀測性來取代傳統的管理方法。如果你負責管理這類型的軟體系統，本書將描述可觀測性對你、你的團隊、客戶和業務的意義。我們還將關注在工程流程的關鍵領域中，發展可觀測性實踐所需的人因素。

總結

儘管「**可觀測性**」一詞已有數十年的定義，但是應用於軟體系統上是一種新的適應方式，並帶來幾個新的考量因素和特徵。與早期較簡單的系統相比，現代化系統架構引入了更多的複雜性，使得故障更加難以預測、檢測和排查。

為了減輕這種複雜性，工程團隊現在必須不斷地以靈活的方式蒐集遙測資料，以允許他們在不需要先預測故障如何發生的情況下排查檢測。可觀測性使工程師能夠以靈活的方式切分和分析遙測資料，以便以前所未有的方式找到任何問題的根源。

很多人誤以為，當你擁有不同類型的 3 種遙測資料時，可觀測性即可實現。然而，如果真的要有 3 個可觀測性的支柱，也應該是支援高基數、高維度和可探索性的工具。接下來，我們將探討可觀測性與傳統系統監控方法的區別。