## Node.js最流行的面試題

#### 1. 請簡要描述一下Node.js框架的概念。

【參考答案】

Node.js是一個基於Chrome JavaScript執行時建立的平臺，用於方便地建立回應速度快、易於擴展的網路應用。Node.js框架使用事件驅動、非阻塞I/O模型而得以輕量和高效率，非常適合在分散式設備上執行資料密集型的即時應用。Node實際上是一個JavaScript執行環境，對Google V8引擎進行了封裝，因為V8引擎執行JavaScript的速度非常快，性能非常好。

#### 2. 請簡要說明幾個Node.js框架的優點。

【參考答案】

* RESTful API：支援Web服務和動態Web應用程式的多層架構，實現再使用性、簡單性、可擴展性和元件可回應性的清晰分離。開發人員可以輕鬆使用Ajax和RESTful Web服務一起建立豐富的介面。
* 單執行緒：Node.js單執行緒是指Node並沒有建立一個執行緒的能力，所有程式碼都是單執行緒執行的。不過Node宿主環境並不是單執行緒的，它會維護一個執行佇列，循環檢測並調度JavaScript執行緒來執行，因此單執行緒執行和併發操作並不衝突。
* 事件輪詢機制：Node.js可以在不新增額外執行緒的情況下，依然可以對任務進行並行處理，它是透過事件輪詢（event loop）來實現平行作業的。
* 非阻塞I/O：由於Node.js是事件驅動的，因此其使用了事件迴圈來解決I/O操作帶來的瓶頸問題。在Node.js中，一個I/O操作通常會帶有一個回呼函數，當I/O操作完成並返回時，就會使用這個回呼函數，而主執行緒則繼續執行接下來的程式碼。
* V8虛擬機器：Node.js是一個基於Google Chrome V8 JavaScript引擎之上的平臺，可用以建立羽量級、快速、可擴展、事件驅動和非阻塞I/O的應用。
* 事件驅動：Node.js框架使用事件驅動模型，即當Web Server接收到請求時，就將其關閉然後進行處理，然後去服務下一個Web請求。當這個請求完成後，它被放回處理佇列中，當到達佇列開頭時，這個結果被返回給用戶端。

#### 3. 簡述Node.js框架適用的應用場景。

【參考答案】

1. 針對服務的架構

針對服務的架構就是做好前、後端的依賴分離，將所有的關鍵業務邏輯都封裝成REST使用，上層只需要考慮如何用這些REST介面來建構具體的應用。這樣，幕後程式員們不需要知道具體資料是如何從一個頁面傳遞到另一個頁面的，也不需要知道使用者資料更新是透過Ajax非同步截取的還是透過刷新頁面截取的。Node.js正適合這樣的開發場景。

1. RESTful API

前面，我們講到了RESTful API是Node.js框架的主要優點，因此也是其最理想的應用場景。應用RESTful API場景可以處理數萬條連接請求，該操作沒有太複雜的邏輯，僅僅就是請求API，將資料進行返回即可。簡而言之，其本質就是從資料庫中查詢一些值並將其組成一個回應，由於這類回應是很小的文字，同時連接請求也是很小的文字，因此整體流量不高，一台伺服器往往也可以處理很繁忙的連接請求。

1. Ajax請求應用

目前，個性化Web應用十分流行，大資料時代針對個人使用者頁面的定制資訊已經成為主流，當暫存失效後需要在頁面載入時發起Ajax請求，此時應用Node.js框架可以回應大量的併發請求。因此Node.js框架適合運用在高併發、I/O密集、少量業務邏輯情況下的Ajax請求應用場景。

#### 4. 簡述Node.js框架不適用的應用場景。

【參考答案】

當然，Node.js框架也不是萬能皆適的，有些應用場景也不是它的強項。下面可以列舉幾種供讀者參考。

1. 即時性要求很高的場景，例如工程交換機、工控機器人和DCS集控系統等。這些場景基本透過垃圾回收機制來管理系統記憶體，因此Node.js框架將會影響回應速度，並且很難進行優化。
2. 計算密集型系統，這些應用基本都是C語言的天下，基於JavaScript語言的Node.js框架在計算性能上是很難與C語言程式碼相比的。
3. 需要單一處理程序控制大記憶體的場景，由於Google V8引擎的設計限制，在32-bit下有1GB 最大記憶體尺寸的限制，在64-bit下是1.7GB。雖然Node.js框架的Buffer分配可以不超過此限制，但是也會帶來垃圾回收機制上性能的退化。

#### 5. 談談Node非同步機制的種類。

【參考答案】

非同步機制方法大致包括回呼函數、pub/sub模式（事件模式）、非同步庫控制庫（async、when等）、promise專案、Generator項目等。

#### 6. 簡述exports與module.exports的區別。

【參考答案】

需要明確的是，module.exports才是真正的介面，而exports只不過是module.exports的一個輔助工具，exports是基於module.exports而實現的。實際上，全部由exports截取的屬性和方法，最後都賦給了module.exports介面，不過有個前提就是module.exports本身不具備任何屬性和方法。換言之，如果module.exports介面已經具備一些屬性和方法，那麼exports截取的屬性和方法將被忽略。