一文了解Nginx server 和location 塊選擇算法

聆聽世界的魚 Linux公社 2021-12-17 08:20

收錄於話題
#Nginx 5 #服務器 3

點擊上方藍字 ● 關注Linux公社

介紹

Nginx 是世界上最受歡迎的Web 服務器之一。它可以成功處理具有許多並發客戶端連接的高負載，並且可以用作Web 服務器、郵件服務器或反向代理服務器。

在本指南中，我們將討論一些決定Nginx 如何處理客戶端請求的幕後細節。理解這些想法有助於在設計server 和location 塊時進行預測，並使請求處理看起來不那麼不可預測。

Nginx 塊配置

Nginx 在邏輯上將旨在為不同內容提供服務的配置劃分為塊，這些塊以分層結構存在。每次發出客戶端請求時，Nginx 都會開始一個過程，以確定應該使用哪些配置塊來處理請求。我們將在本指南中討論這個決策過程。

我們將討論的主要塊是server 塊和location 塊。

server 塊是Nginx 配置的一個子集，它定義了一個虛擬服務器，用於處理已定義類型的請求。管理員通常配置多個服務器塊，並根據請求的域名、端口和IP 地址決定哪個塊應該處理哪個連接。

location 塊位於server 塊中，用於定義Nginx 應如何處理對父服務器的不同資源和URI 的請求。URI 空間可以按照管理員喜歡使用這些塊的任何方式進行細分。這是一個非常靈活的模型。

Nginx 如何決定哪個server 塊將處理請求

由於Nginx 允許管理員定義多個server 塊作為單獨的虛擬Web 服務器實例，因此它需要一個過程來確定將使用這些server 塊中的哪些來滿足請求。

它通過定義的檢查系統來實現這一點，這些檢查系統用於找到可能的最佳匹配。Nginx 在此過程中關注的主要server 塊指令是`listen`指令和`server_name`指令。

解析`listen`指令以查找可能的匹配項

首先，Nginx 查看請求的IP 地址和端口。它將這與`listen`每個服務器的指令相匹配，以構建可能解析請求的服務器塊列表。

該`listen`指令通常定義server 塊將響應的IP 地址和端口。默認情況下，任何不包含`listen`指令的server 塊都被賦予監聽參數`0.0.0.0:80`（或者`0.0.0.0:8080`如果Nginx 由普通的非**root**用戶運行）。這允許這些塊響應端口80 上任何接口上的請求，但此默認值在服務器選擇過程中沒有太大影響。

該`listen`指令可以設置為：

- IP 地址/端口組合。
- 一個單獨的IP 地址，然後將偵聽默認端口80。
- 一個單獨的端口，它將偵聽該端口上的每個接口。
- Unix 套接字的路徑。

最後一個選項通常只會在不同服務器之間傳遞請求時產生影響。

在嘗試確定將請求發送到哪個服務器塊時，Nginx 將首先嘗試`listen`使用以下規則根據指令的特殊性來決定：

- Nginx`listen`通過用默認值替換缺失值來轉換所有“不完整”指令，以便每個塊都可以通過其IP 地址和端口進行評估。一些例子是：
  - 沒有`listen`指令的塊使用value `0.0.0.0:80`。
  - 設置為`111.111.111.111`沒有端口的IP 地址的塊變為`111.111.111.111:80`
  - 設置為`8888`沒有IP 地址的端口的塊變為`0.0.0.0:8888`
- 然後，Nginx 會嘗試根據IP 地址和端口收集與請求最匹配的server 塊列表。這意味著`0.0.0.0`如果存在列出特定IP 地址的匹配塊，則不會選擇任何功能上用作其IP 地址（以匹配任何接口）的塊。在任何情況下，端口都必須完全匹配。
- 如果只有一個最具體的匹配項，則該server 塊將用於為請求提供服務。如果有多個server 塊具有相同級別的特異性匹配，Nginx 然後開始評估`server_name`每個server 塊的指令。

了解Nginx 僅`server_name`在需要區分與`listen`指令中相同級別的特定性匹配的server 塊時才會評估指令很重要。例如，如果`example.com`託管在的端口`80`上`192.168.1.10`，`example.com`則在此示例中，儘管`server_name`第二個塊中有指令，請求將始終由第一個塊提供服務。

```
server {
    listen 192.168.1.10;

    . . .
```

```
}

server {
    listen 80;
    server_name example.com;

    . . .

}
```

如果有多個server 塊以相同的特異性匹配，下一步是檢查server_name指令。

解析server_name指令以選擇匹配

接下來，為了進一步評估具有同樣特定listen指令的請求，Nginx 檢查請求的Host標頭。該值包含客戶端實際嘗試訪問的域或IP 地址。

Nginx 試圖通過查看server_name仍然是候選的每個server 塊中的指令來找到它找到的值的最佳匹配。Nginx 使用以下公式評估這些：

- Nginx 將首先嘗試找到一個與請求頭server_name中的值*完全*匹配的server 塊。如果找到，將使用關聯的塊來為請求提供服務。如果找到多個完全匹配，則使用**第**一個。Host
- 如果沒有找到完全匹配，Nginx 將嘗試server_name使用前導通配符（由*配置中名稱開頭的a 表示）查找匹配的服務器塊。如果找到，則該塊將用於為請求提供服務。如果找到多個匹配項，將使用**最長的**匹配項來為請求提供服務。
- 如果使用前導通配符未找到匹配項，Nginx 將server_name使用尾隨通配符（由*配置中以a 結尾的服務器名稱表示）查找具有匹配項的server 塊。如果找到，則該塊用於為請求提供服務。如果找到多個匹配項，將使用**最長的**匹配項來為請求提供服務。
- 如果使用尾隨通配符未找到匹配項，Nginx 將評估定義server_nameusing 正則表達式（由~名稱前的a 表示）的server 塊。第**一個** server_name帶有與“Host”標頭匹配的正則表達式的將用於為請求提供服務。
- 如果沒有找到匹配的正則表達式，Nginx 會為該IP 地址和端口選擇默認的server 塊。

每個IP 地址/端口組合都有一個默認server 塊，當無法通過上述方法確定操作過程時，將使用該塊。對於IP 地址/端口組合，這將是配置中的第一個塊，或者是包含

default_server作為listen指令一部分的選項的塊（這將覆蓋首先找到的算法）。default_server每個IP 地址/端口組合只能有一個聲明。

例子

如果有一個server_name與Host標頭值完全匹配的定義，則選擇該server 塊來處理請求。

在此示例中，如果Host請求的標頭設置為host1.example.com，則將選擇第二個服務器：

```
server {
    listen 80;
    server_name *.example.com;

    . . .

}

server {
    listen 80;
    server_name host1.example.com;

    . . .

}
```

如果沒有找到完全匹配，Nginx 然後檢查是否有server_name適合的起始通配符。將選擇以通配符開頭的最長匹配項來滿足請求。

在此示例中，如果請求的Host標頭為www.example.org，則將選擇第二個server 塊：

```
server {
    listen 80;
    server_name www.example.*;

    . . .

}

server {
```

```
    listen 80;
    server_name *.example.org;

    . . .

}

server {
    listen 80;
    server_name *.org;

    . . .

}
```

如果沒有找到帶有起始通配符的匹配項，Nginx 將使用表達式末尾的通配符查看是否存在匹配項。此時，將選擇以通配符結尾的最長匹配項來為請求提供服務。

例如，如果請求的Host標頭設置為www.example.com，則將選擇第三個server 塊：

```
server {
    listen 80;
    server_name host1.example.com;

    . . .

}

server {
    listen 80;
    server_name example.com;

    . . .

}

server {
    listen 80;
    server_name www.example.*;
```

```
    . . .

}
```

如果找不到通配符匹配，Nginx 將繼續嘗試匹配server_name使用正則表達式的指令。所述*第*一匹配正則表達式將被選擇，以響應該請求。

例如，如果Host請求的標頭設置為www.example.com，則將選擇第二個server 塊來滿足請求：

```
server {
    listen 80;
    server_name example.com;

    . . .

}

server {
    listen 80;
    server_name ~^(www|host1).*\.example\.com$;

    . . .

}

server {
    listen 80;
    server_name ~^(subdomain|set|www|host1).*\.example\.com$;

    . . .

}
```

如果以上步驟都不能滿足請求，則請求將傳遞給匹配的IP 地址和端口的*默認*服務器。

匹配location 塊

類似於Nginx 用來選擇將處理請求的服務器塊的過程，Nginx 也有一個既定的算法來決定服務器中的哪個location 塊用於處理請求。

location 塊語法

在我們介紹Nginx 如何決定使用哪個location 塊來處理請求之前，讓我們回顧一下您可能在location 塊定義中看到的一些語法。location 塊位於server 塊（或其他location 塊）中，用於決定如何處理請求URI（請求的一部分，位於域名或IP 地址/端口之後）。

位置塊通常採用以下形式：

```
location optional_modifier location_match {

    . . .

}
```

在location_match上面定義了Nginx的應檢查請求URI反對。上例中修飾符的存在與否影響了Nginx 嘗試匹配location 塊的方式。下面的修飾符將導致相關的location 塊被解釋如下：

- **(none)**：如果不存在修飾符，則該位置被解釋為*前綴*匹配。這意味著給定的位置將與請求URI 的開頭匹配以確定匹配。
- =: 如果使用等號，如果請求URI 與給定的位置完全匹配，則此塊將被視為匹配。
- ~：如果存在波浪號修飾符，則此位置將被解釋為區分大小寫的正則表達式匹配。
- ~*：如果使用波浪號和星號修飾符，則位置塊將被解釋為不區分大小寫的正則表達式匹配。
- ^~: 如果存在carat 和tilde 修飾符，並且如果此塊被選為最佳非正則表達式匹配，則不會進行正則表達式匹配。

展示location 塊語法的示例

作為前綴匹配的一個例子，以下location 塊可以被選擇為響應於請求的URI的樣子/site，/site/page1/index.html或/site/index.html：

```
location /site {

    . . .

}
```

為了演示精確的請求URI 匹配，此塊將始終用於響應類似於/page1. 它**不會**用於響應/page1/index.html請求URI。請記住，如果選擇此塊並使用索引頁完成請求，則內部重定向將發生到另一個位置，該位置將成為請求的實際處理程序：

```
location = /page1 {

    . . .

}
```

作為應解釋為區分大小寫的正則表達式的location 示例，此塊可用於處理對的請求/tortoise.jpg，但**不能**用於處理/FLOWER.PNG：

```
location ~ \.(jpe?g|png|gif|ico)$ {

    . . .

}
```

下面顯示了一個允許與上述類似的不區分大小寫匹配的塊。在這裡，/tortoise.jpg *和* /FLOWER.PNG都可以由這個塊處理：

```
location ~* \.(jpe?g|png|gif|ico)$ {

    . . .

}
```

最後，如果該塊被確定為最佳非正則表達式匹配，則該塊將阻止正則表達式匹配發生。它可以處理以下請求/costumes/ninja.html：

```
location ^~ /costumes {

    . . .

}
```

如您所見，修飾符指示應如何解釋location 塊。然而，這並*沒有*告訴我們Nginx 用來決定將請求發送到哪個location 塊的算法。我們接下來會討論這個。

Nginx 如何選擇使用哪個location 來處理請求

Nginx 選擇將用於為請求提供server的location，其方式與它選擇server 塊的方式類似。它通過一個過程來確定任何給定請求的最佳location 塊。了解這個過程是能夠可靠、準確地配置Nginx 的關鍵要求。

記住我們上面描述的位置聲明的類型，Nginx 通過將請求URI 與每個位置進行比較來評估可能的location 上下文。它使用以下算法執行此操作：

- Nginx 首先檢查所有基於前綴的location 匹配（所有不涉及正則表達式的location 類型）。它根據完整的請求URI 檢查每個位置。
- 首先，Nginx 尋找完全匹配。如果=發現使用修飾符的location 塊與請求URI 完全匹配，則立即選擇該location 塊來為請求提供服務。
- 如果沒有找到精確的（使用=修飾符）位置塊匹配，Nginx 然後繼續評估非精確前綴。它發現給定請求URI 的最長匹配前綴位置，然後按如下方式評估：
  - 如果最長匹配的前綴location 有^~修飾符，則Nginx 將立即結束其搜索並選擇此location 來為請求提供服務。
  - 如果最長匹配前綴location*沒有*使用^~修飾符，則匹配由Nginx 暫時存儲，以便可以移動搜索的焦點。
- 在確定並存儲最長匹配前綴位置後，Nginx 繼續評估正則表達式位置（區分大小寫和不區分大小寫）。如果有任何的正則表達式的location*內*的最長前綴匹配的location ，Nginx的將這些移動到它的正則表達式的location 列表的頂部進行檢查。然後Nginx 嘗試按順序匹配正則表達式location。所述**第一**所述請求URI匹配正則表達式的location 被立即選擇來服務該請求。
- 如果找不到與請求URI 匹配的正則表達式location，則選擇先前存儲的前綴location 來為請求提供服務。

重要的是要理解，默認情況下，Nginx 將優先於前綴匹配提供正則表達式匹配。但是，它首先*評估*前綴位置，允許管理員通過使用=和^~修飾符指定位置來覆蓋這種趨勢。

同樣重要的是要注意，雖然前綴位置通常根據最長、最具體的匹配進行選擇，但在找到第一個匹配location時會停止正則表達式評估。這意味著配置中的定位對正則表達式location 有很大的影響。

最後，要明白，正則表達式匹配它是非常重要的*內*最長前綴匹配將"跳線"，當Nginx的正則表達式的計算結果的location。在考慮任何其他正則表達式匹配之前，將按順序對這些進行評估。

location 塊評估何時跳轉到其他location?

一般來說，當一個location 塊被選擇來服務一個請求時，從那個點開始，請求就完全在那個上下文中處理。只有選定的location 和繼承的指令決定了請求的處理方式，不受同級location 塊的干擾。

儘管這是允許您以可預測的方式設計location 塊的一般規則，但重要的是要意識到有時會由所選位置內的某些指令觸發新的位置搜索。"只有一個location 塊"規則的例外可能會影響請求的實際服務方式，並且可能與您在設計location 塊時的期望不一致。

可能導致這種類型的內部重定向的一些指令是：

- **index**
- **try_files**
- **rewrite**
- **error_page**

讓我們簡單地回顧一下這些。

如果該index指令用於處理請求，則它總是會導致內部重定向。精確的location匹配通常用於通過立即結束算法的執行來加速選擇過程。但是，如果您將精確位置匹配為*目錄*，則很有可能將請求重定向到不同的location 以進行實際處理。

在此示例中，第一個位置與的請求URI 匹配/exact，但為了處理該請求，index塊繼承的指令啟動到第二個塊的內部重定向：

```
index index.html;

location = /exact {

    . . .

}

location / {

    . . .

}
```

在上面的例子中，如果你真的需要執行留在第一個塊中，你將不得不想出一種不同的方法來滿足對目錄的請求。例如，您可以index為該塊設置無效並打開autoindex：

```
location = /exact {
    index nothing_will_match;
    autoindex on;
}

location  / {

    . . .

}
```

這是防止 index切換上下文的一種方法，但它可能對大多數配置沒有用。大多數情況下，目錄上的完全匹配對於重寫請求（這也會導致新的location 搜索）等事情很有幫助。

可以重新評估處理位置的另一個實例是try_files指令。該指令告訴Nginx 檢查是否存在一組命名的文件或目錄。最後一個參數可以是Nginx 將進行內部重定向的URI。

考慮以下配置：

```
root /var/www/main;
location / {
    try_files $uri $uri.html $uri/ /fallback/index.html;
}

location /fallback {
    root /var/www/another;
}
```

在上面的示例中，如果對發出請求/blahblah，則第一個location 最初將收到請求。它將嘗試找到blahblah在/var/www/main目錄中調用的文件。如果找不到，它將通過搜索名為blahblah.html. 然後它會嘗試查看目錄中是否有一個被調用blahblah/的 /var/www/main 目錄。所有這些嘗試都失敗了，它將重定向到/fallback/index.html. 這將觸發另一個location 搜索，該搜索將被第二個location 塊捕獲。這將為文件提供服務/var/www/another/fallback/index.html。

另一個可能導致location 塊傳遞的rewrite指令是指令。當last在rewrite指令中使用參數時，或者根本不使用參數時，Nginx 將根據重寫的結果搜索新的匹配location 。

例如，如果我們修改最後一個示例以包含重寫，我們可以看到請求有時會直接傳遞到第二個location，而不依賴於try_files指令：

```
root /var/www/main;
location / {
    rewrite ^/rewriteme/(.*)$ /$1 last;
    try_files $uri $uri.html $uri/ /fallback/index.html;
}

location /fallback {
    root /var/www/another;
}
```

在上面的例子中，一個請求/rewriteme/hello將首先由第一個location 塊處理。它將被重寫/hello並蒐索一個location 。在這種情況下，它將再次匹配第一個location 並try_files像往常一樣由它處理，/fallback/index.html如果沒有找到，可能會返回（使用try_files我們上面討論的內部重定向）。

但是，如果對發出請求/rewriteme/fallback/hello，則第一個塊將再次匹配。再次應用重寫，這一次導致/fallback/hello. 然後將在第二個location 塊之外提供請求。

return發送301或302狀態代碼時，指令會發生相關情況。這種情況下的不同之處在於，它會以外部可見的重定向形式產生全新的請求。rewrite使用redirectorpermanent標誌時，指令也會發生同樣的情況。但是，這些位置搜索不應出乎意料，因為外部可見的重定向總是會導致新的請求。

該error_page指令可以導致類似於由try_files. 該指令用於定義遇到某些狀態代碼時應該發生的情況。如果try_files設置了，這可能永遠不會執行，因為該指令處理請求的整個生命週期。

考慮這個例子：

```
root /var/www/main;

location / {
    error_page 404 /another/whoops.html;
}

location /another {
    root /var/www;
}
```

每個請求（以開頭的請求除外/another）將由第一個塊處理，該塊將提供/var/www/main. 但是，如果找不到文件（404 狀態），/another/whoops.html則會發生內部重定向，從而導致新的位置搜索最終落在第二個location 上。此文件將從/var/www/another/whoops.html.

如您所見，了解Nginx 觸發新location搜索的情況有助於預測您在發出請求時將看到的行為。

結論

了解Nginx 處理客戶端請求的方式可以使您作為管理員的工作更加輕鬆。您將能夠知道Nginx 將根據每個客戶端請求選擇哪個server 塊。您還可以根據請求URI 判斷如何選擇

location 塊。總的來說，了解Nginx 選擇不同塊的方式將使您能夠跟踪Nginx 將應用的上下文，以便為每個請求提供服務。

來自：Linux迷

鏈接：https://www.linuxmi.com/nginx-server-and-location-block.html