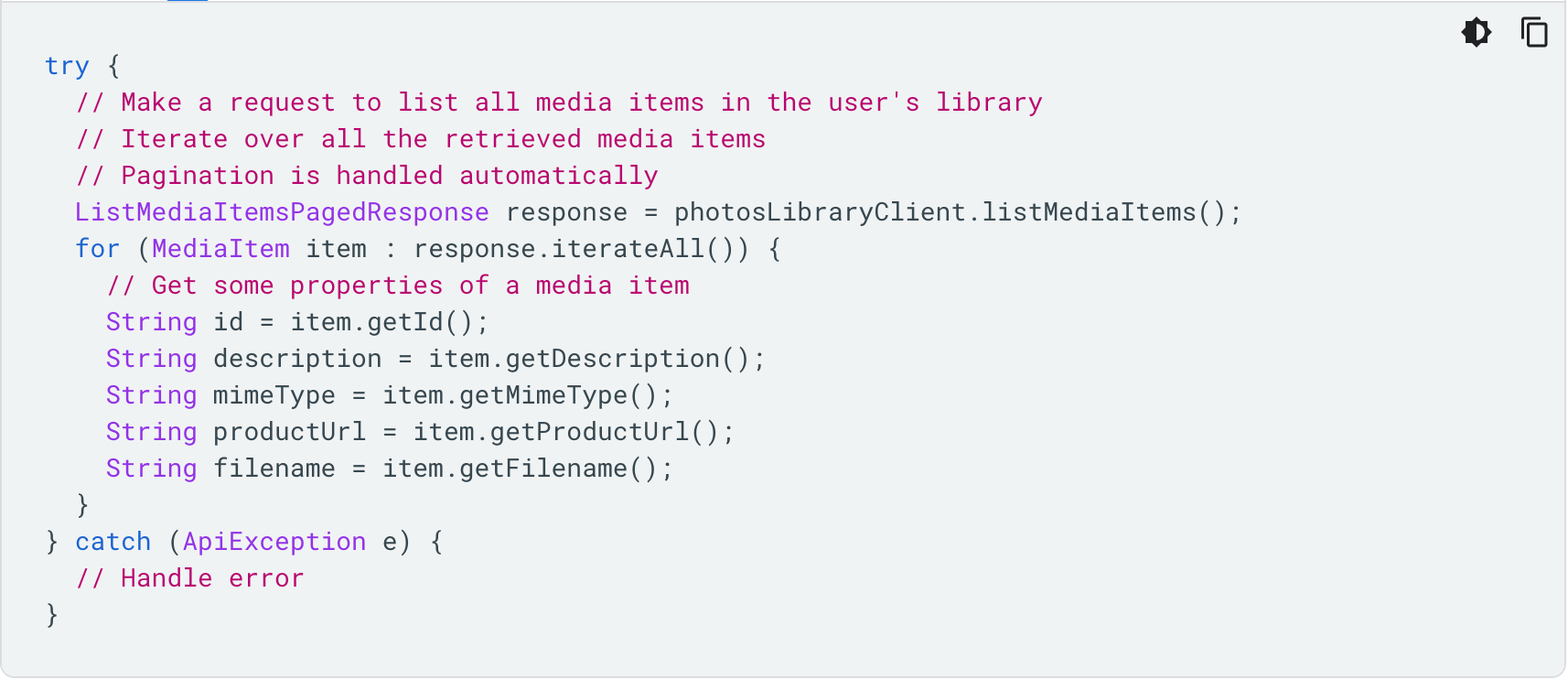
1. 閱讀英文Google Photos API文件，並回答問題 <https://vivipic.notion.site/Google-Photos-API-208ff38d22ec473ca0a49d729d8400b4>

1. 某位使用者在Google Photos儲存的所有照片

列出程式庫內容

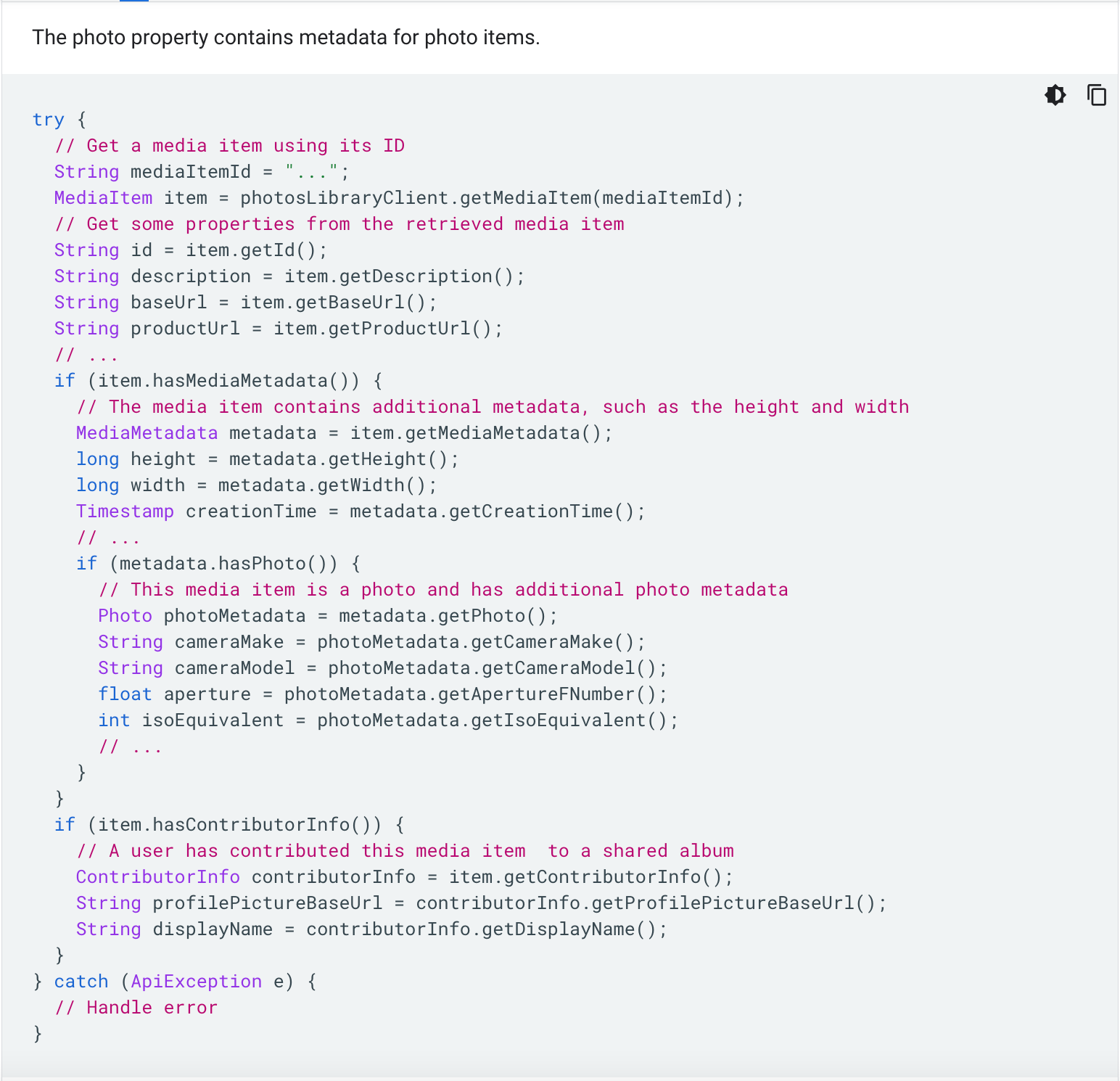
<https://developers.google.com/photos/library/guides/list#listing-library-contents>



* 使用 photosLibraryClient 呼叫 listMediaItems 方法，它返回一個 ListMediaItemsPagedResponse 物件。
* 使用 for 迴圈遍歷 response 中的所有媒體項目，item.getId() 取得媒體項目的唯一識別碼

取得媒體項目

<https://developers.google.com/photos/library/guides/access-media-items#get-media-item>

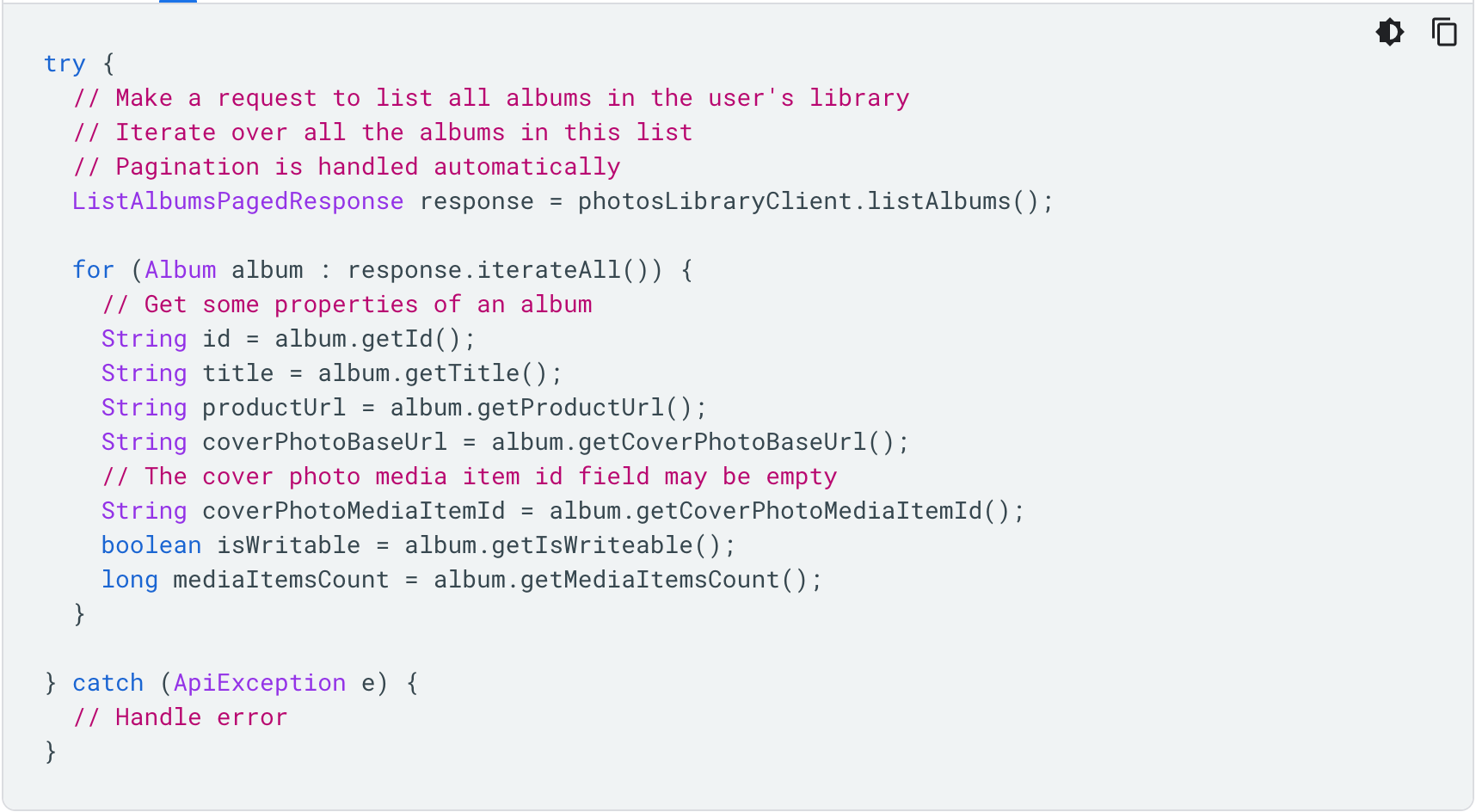


* 使用 photosLibraryClient 呼叫 getMediaItem 方法，傳入 mediaItemId 作為參數。
* 使用 item.hasMediaMetadata() 檢查是否存在媒體項目的附加資料metadata，使用 metadata.hasPhoto() 檢查媒體項目是否是一張照片，如果是照片，則可以從 metadata 中取得照片相關資訊。

1. 某位使用者在Google Photos儲存的相簿名稱，以及各相簿裏面的照片

列出專輯

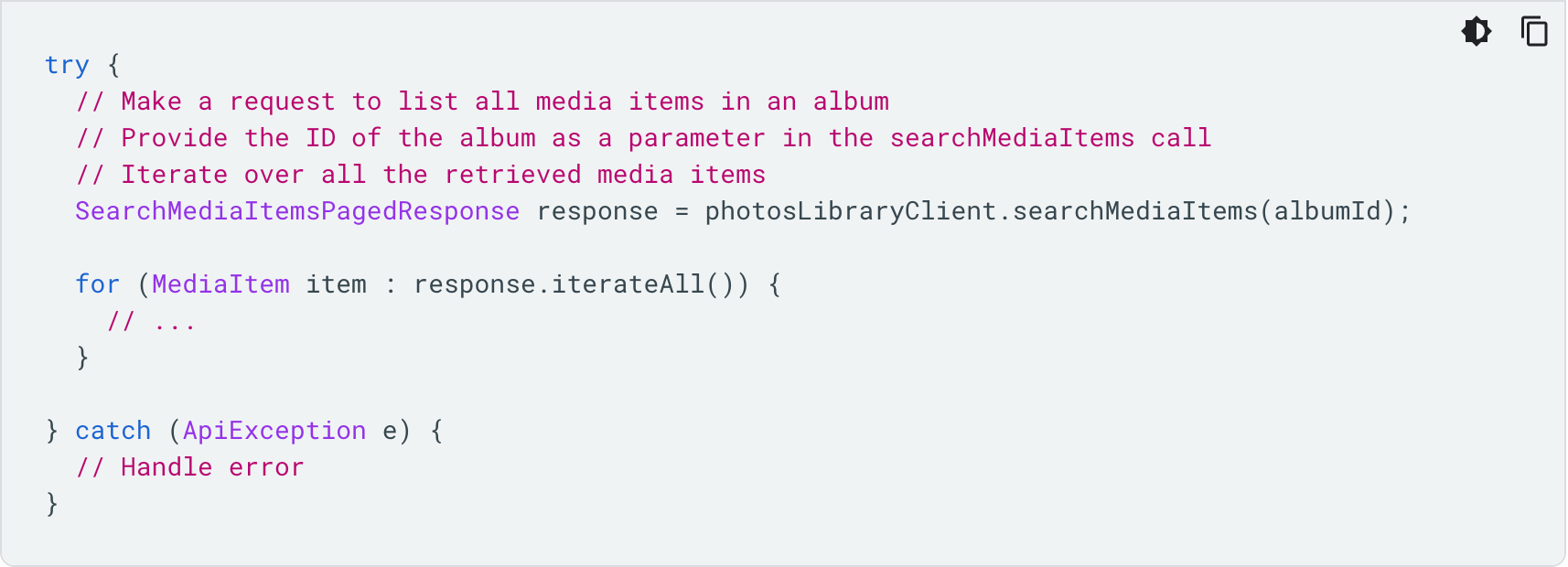
<https://developers.google.com/photos/library/guides/list#listing-albums>



* 使用 photosLibraryClient 呼叫 listAlbums 方法，它返回一個 ListAlbumsPagedResponse 物件。
* 使用 for 迴圈遍歷 response 中的所有專輯項目，album.getId() 取得專輯項目的唯一識別碼。

列出專輯內容

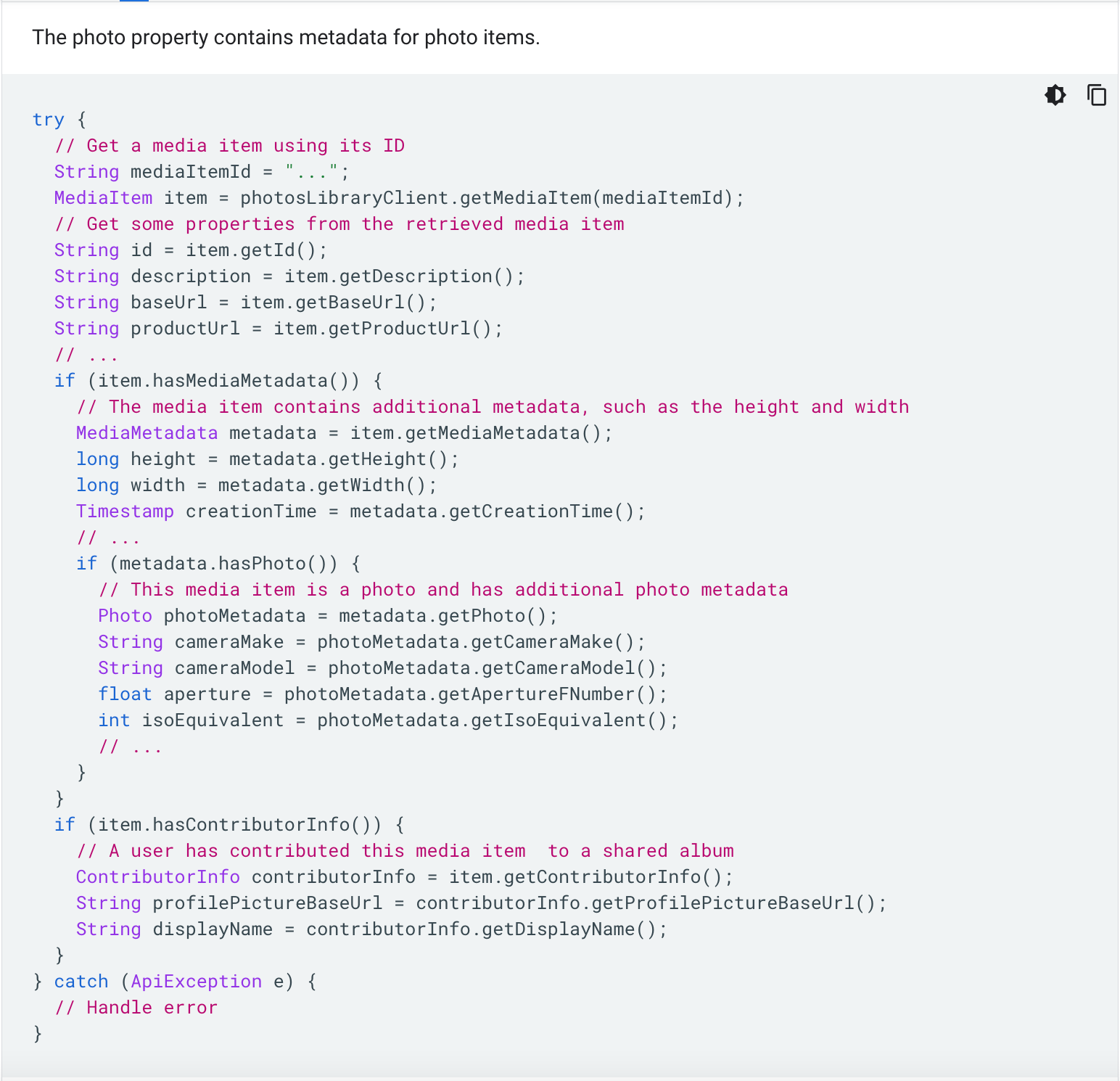
<https://developers.google.com/photos/library/guides/list#listing-album-contents>



* 使用 photosLibraryClient 呼叫searchMediaItems 方法，它返回一個 SearchMediaItemsPagedResponse 物件。
* 使用 for 迴圈遍歷 response 中的所有媒體項目，item.getId() 取得媒體項目的唯一識別碼

取得媒體項目

<https://developers.google.com/photos/library/guides/access-media-items#get-media-item>



* 使用 photosLibraryClient 呼叫 getMediaItem 方法，傳入 mediaItemId 作為參數。
* 使用 item.hasMediaMetadata() 檢查是否存在媒體項目的附加資料metadata，使用 metadata.hasPhoto() 檢查媒體項目是否是一張照片，如果是照片，則可以從 metadata 中取得照片相關資訊。

2. 影像處理佇列程式虛擬碼<https://vivipic.notion.site/ddbb8ce1732b42b49d9563faf3f6ca89>

在影像處理軟體中處理複數個工作項目時，如果每個項目的運算量或記憶體用量較大，又可能會在運算過程中動態添加/移除工作項目的時候，其中一種做法是，將所有的工作項目排入佇列 (queue) 中，固定只處理佇列中的第一個項目，處理完成的項目就會從佇列中被移除，而新增的工作項目則會附加到佇列的最後，以此來達成先加入的工作項目就先被處理 (first-in-first-out) 的效果。 請使用「照片處理function」以及「array」段落定義的array，實作一個class來完成上述的佇列結構。處理照片佇列的過程中，必須要確保同一時間，只能夠對一張照片進行處理;此外，先被選擇的照片必須先被處理。在這個佇列結構對應的array中，每個element主要的內容，是待處理照片的路徑，而不是待處理的影像處理程序，至於是否需要其他內容，可以自行決定;此佇列結構的class需要有以下公開的方法 (method)，私有的屬性 (property) 或方法可自行添加：

isEmpty(): Boolean

檢查佇列是否爲空（表示已經沒有任何待處理 的照片，背景執行緒也沒有處理任何照片了）。

addToQueue(imagePath: String)

使用者選取了某張照片時，會呼叫此方法，並傳入該照片的路徑 (imagePath)。此照片將會被排入待處理的佇列中

removeFromQueue(imagePath: String)

使用者反選取了某張照片時，會呼叫此方法，並傳入該照片的路徑(imagePath)。此照片將從待處理的佇列中被移除

請注意，如果即將被移除的照片己經進行影像處理的程序，從佇列中移除照片，並不會中止對應的處理程序

在程式中，外部的程式碼只會透過addToQueue/removeFromQueue來對佇列的內容進行修改，並根據isEmpty的值來判斷，是否所有選擇的照片都已經處理完成。因此，必須自行決定在 class中觸發影像處理的時機。

- 撰寫的pseudo code中，可以使用一般程式語言通用的if/else、switch/case、while或for loop 語法，也可以自定義function來將邏輯判斷或運算式包裝起來。

- 宣告變數時請用let，如果變數值足以表明該變數型別，就不必特別宣告型別。宣告class， 可參考「array」段落的範例

- 未列在「照片處理function」以及「array」中的function，除非是自定義，否則不能使用。

- 假設只有影像後處理會在背景執行緒執行，其餘程式碼皆在主執行緒上，因此不必考慮 class中的不同方法對於共同的變數會發生競爭情況 (race condition)。

「照片處理function」：

無回傳值

function imageProcess(imagePath: String, callback: function)

在主執行緒呼叫此function後，影像將會在背景執行緒中進行處理。 imagePath爲待處理的影像路徑。

callback爲背景執行緒完成影像處理後觸發的function，會回到主執行緒來執行。

imageProcess爲全域函數，可在程式中的任意地方直接呼叫。

範例：

// 範例中的x是用來說明主執行緒與背景執行緒的執行順序

let imagePath = "path/to/image";

let x = 0;

// 主執行緒呼叫 imageProcess，背景執行緒開始進行影像處理

ImageProcess(imagePath, function() {

// 背景執行緒執行完後，觸發callback function

x=x+1; // x=2

})

// 主執行緒完成imageProcess呼叫後，立刻執行底下程式碼

x=x+1; // x=1

「array」：

與C/C++或Java語言中嚴格的定義不同，此處的array長度可以在程式執行過程中動態增加或減少，不必考量初始化就需指定array長度的限制，但若存取的index超過array的長度，仍然會出錯。假設array有以下的屬性/方法可使用：

屬性：length: int，意義：array長度

方法：insertAt(element: Element, index: int)，意義：在 array 中指定的 index 插入新的元素

方法：removeFrom(index: int)，意義：從 array 中指定的 index 移除元素

範例：

//宣告SampleObject，作爲下述陣列的元素型別

Class SampleObject {

let data: int;

constructor(data: int) {

this.data = data;

}

}

// 宣告空陣列，每個元素的型別爲SampleObject

let sampleArray = [SampleObject];

// sampleArray.length = 0

// 在index = 0 插入第1個元素

sampleArray.insertAt(new SampleObject(“path/to/image1.jpg”), 0);

// sampleArray.length = 1

// sampleArray[0] = {“data”: “path/to/image1.jpg”}

//在index = 0 插入第2個元素

sampleArray.insertAt(new SampleObject(“path/to/image2.jpg”), 0);

// sampleArray.length = 2

// sampleArray[0] = {“data”: “path/to/image2.jpg”}

// sampleArray[1] = {“data”: “path/to/image1.jpg”}

//在index = 1 插入第3個元素

sampleArray.insertAt(new SampleObject(“path/to/image3.jpg”), 1);

// sampleArray.length = 3

// sampleArray[0] = {“data”: “path/to/image2.jpg”}

// sampleArray[1] = {“data”: “path/to/image3.jpg”}

// sampleArray[2] = {“data”: “path/to/image1.jpg”}

//移除index = 1的元素

sampleArray.removeFrom(1);

// sampleArray.length = 2

// sampleArray[0] = {“data”: “path/to/image2.jpg”}

// sampleArray[1] = {“data”: “path/to/image1.jpg”}

3. 伺服器溝通

<https://vivipic.notion.site/4e94cfb2475b4bb0be16170ffebe7cf9>

一個典型的客戶端與伺服器api的互動，通常包含客戶端發出請求給伺服器api、伺服器api回覆結果給客戶端，以及錯誤重試的動作。假設希望達成的流程如下：

1. 客戶端上傳照片至伺服器(id爲9999，image爲img\_file，type爲3)。伺服器收到影像後就會回覆客戶端，接着會開始對影像進行處理。

2. 伺服器上傳照片api若無法順利執行，或者api回覆失敗，則進行重試。

3. 若api回覆成功，則開始向伺服器端進行確認照片處理狀態的請求。

4. 由於伺服器處理影像需要一段時間，因此確認照片狀態時，伺服器回覆的結果可能是處理中、完成、或者失敗。

5. 若照片仍在處理中或者api若無法順利執行，則等待2秒後再重新確認狀態。

6. 爲了避免客戶端等待過久，若進行照片狀態的請求進行20次後，照片仍處於處理中的狀態時，則顯示訊息詢問客戶端是否繼續等待。若客戶端選擇是，則重新進行步驟4的確認照片狀態請求；若客戶端選擇否，則判定爲照片處理失敗。

7. 照片處理完成或失敗，則呼叫照片處理完成function並傳入對應的狀態值。

請使用「function列表」中定義的function，以及「伺服器api列表」中定義的api，來完成上述指定的客戶端的流程。

- 撰寫的pseudo code中，可以使用一般程式語言通用的if/else、switch/case、while或for loop語法，也可以自定義function來將邏輯判斷或運算式包裝起來。

- 宣告變數時請用let，如果變數值足以表明該變數型別，就不必特別宣告型別。

- 未列在「function列表」中的function，除非是自定義，否則不能使用；此外，不一定所有在「function列表」中的function都需要使用才能完成指定流程。

「function列表」：

function request(url, params, callback, fallback)

無回傳值

呼叫伺服器api時使用。

ur代表api的網址。

params代表api所需的參數，以JSON格式傳入。

callback爲api回應時所觸發之function。

fallback爲呼叫api過程發生錯誤時所觸發之function。

範例，確認id為’123’之照片的處理狀態：

// 傳給確認照片狀態api之參數

let params = { 'id': 123 };

request (URL\_POLLING, params, function (response) {

//自定義之caltback function内容

// response為api之回傳內容

let flag = response.status;

if (flag === 0) {

// 處理中 }

else if (flag === 1) {

// 成功

} else {

//失敗

}}, function(error) {

//自定義之fallback function内容

// error爲錯誤訊息

});

function setDelay(callback, delay)

id (配合clearDelay使用，見下)

延遲觸發某個function時使用。

callback爲欲延遲觸發的function。

delay為延遲的時間，單位爲秒數。

範例，延遲3秒觸發某function：

setDelay(function(){ // 某function之内容 }, 3);

function clearDelay (id)

無回傳值

取消某個透過setDealy排定的延遲觸發動作。

id爲setDelay之回傳值。

範例：

// 排定某function延遲5秒後執行

let delayID = setDelay(function () { … }, 5);

// 取消排定之動作

clearDelay(delayID);

function setTimer(callback, interval)

id (配合clearTimer使用，見下）

排定某個function每隔固定時間間隔執行一次。

callback爲欲執行的function。

interval爲每次執行的時間間隔，單位爲秒數。

範例，排定某function每隔3秒執行一次：

setTimer(function() { //某function之内容 }, 3);

function clearTimer (id)

無回傅值

取消某個透過setTimer排定之動作。

id爲setTimer之回傳值。

範例：

// 排定某function每隔5秒執行一次

let timerID = setTimer(function () { … }, 5);

// 取消排定之動作

clearTimer (timerID);

function showMessage (msg, callback)

無回傳值

用來顯示訊息。

msg爲顯示的訊息內容。

callback爲使用者選擇是或否之後觸發之function。

範例：

// 界面上會顯示訊息提示視窗，訊息内容爲

// ‘是否繼續執行’

showMessage(‘是否總續執行’, function (result) {

if(result === true) {

// 使用者選擇是

} else {

//使用者選擇否

}

});

function finishProcess (isDone)

無回傳值

當照片處理完成後呼叫。

isDone爲true 或false。

照片成功完成處理的話，傳入true，失敗的話傳入false。

範例：

// 照片成功完成處理

finishProcess (true);

「伺服器api列表」：

名稱上傳照片

網址：URL\_UPLOAD

參數：Id、image、type

回傳：success: 0 失敗，1成功

名稱：確認照片處理狀態

網址URL\_POLLING

參數：id

回傳：status: 0 處理中，1完成，2失敗

一個伺服器api包含該api的網址、所需參數以及回傳值。

客戶端將參數傳給api後，api會在伺服器端執行對應的動作，直到api執行完才將結果回傳給客戶端。

爲了避免客戶端在等待伺服器回傳結果之前無法執行其他動作，一般在客戶端會透過背景執行緒(background thread)來呼叫伺服器api；也就是說，客戶端呼叫伺服器api時，會同時註冊callback function與fallback function，呼叫時客戶端可同時在主要執行緒(main thread)處理其他動作，當伺服器api回應時，就會觸發呼叫api時註冊之callback或fallback function。

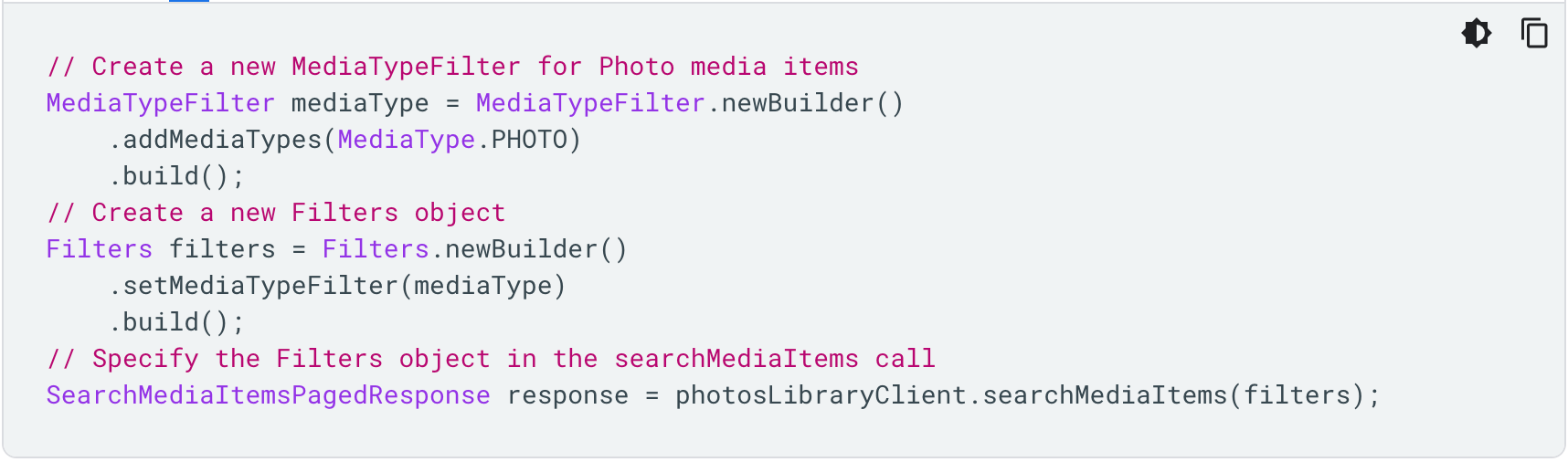
若伺服器正確地回應，就會觸發callback function。

若伺服器api無法被順利執行（例如網路錯誤、網路逾時，或者是api網址錯誤），則會觸發fallback function。

上述兩題，如無需取得檔案中的中繼資料，或處理影片類型的媒體項目，則可以使用媒體類型先進行過濾，獲取照片類型的媒體項目，再進行遍歷取得所有照片項目的相關資訊。

媒體類型

<https://developers.google.com/photos/library/guides/apply-filters#media-types>



* 創建一個MediaTypeFilter，用於篩選媒體類型為照片的媒體項目。
* 創建一個Filters物件，並將此過濾器設定為上述建立的媒體類型過濾器。
* 調用searchMediaItems方法，以搜尋符合過濾條件為照片的媒體項目。