# CMoney Exercise

## 如何不用第三方來抓資料 & 顯示圖片

**Jimmy 2020/05/03**

https://github.com/KuoPingL-Android/JsonPlaceholder

# 題目


Carrier
12:35 PM
JSON Placeholder
Request API

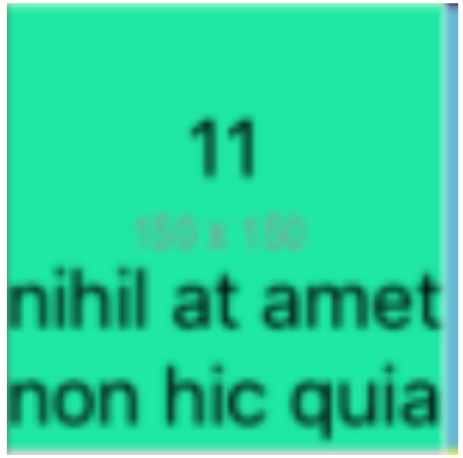

Carrier
12:35 PM
Back
1 accusamus beatae ad
2 reprehende rit est
3 officia porro iure quia
4 culpa odio esse rerum
5 natus nisi omnis
6 accusamus ea aliquid et
7 officia delectus
8 aut porro officiis
9 qui eius qui autem sed
10 beatae et provident et
11 nihil at amet non hic quia
12 mollitia soluta ut
13 repudianda e iusto
14 est necessitatib
15 harum dicta similique
16 iusto sunt nobis quasi
17 natus doloribus
18 laboriosam odit nam
19 perferendis nesciunt
20 assumenda voluptatem
21 ad et natus qui
22 et ea illo et sit voluptas
23 harum velit vero totam
24 beatae officiis ut
25
26
27
28

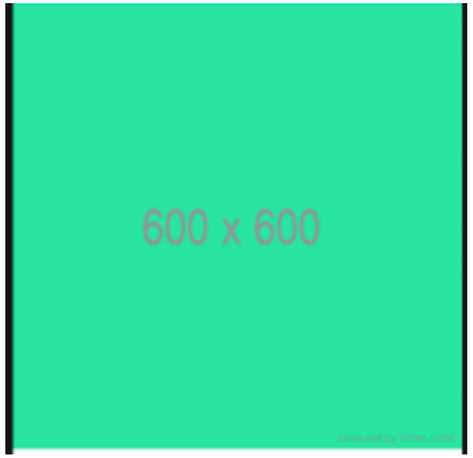

4:52
Back
600 x 600
id: 11
title: nihil at amet non hic quia qui

# API

base API: https://jsonplaceholder.typicode.com
end point: /photos


```
[
 {
      "albumId": 1,
      "id": 1,
      "title": "accusamus beatae ad facilis cum similique qui sunt",
      "url": "https://via.placeholder.com/600/92c952",
      "thumbnailUrl": "https://via.placeholder.com/150/92c952"
 }, …
]
```

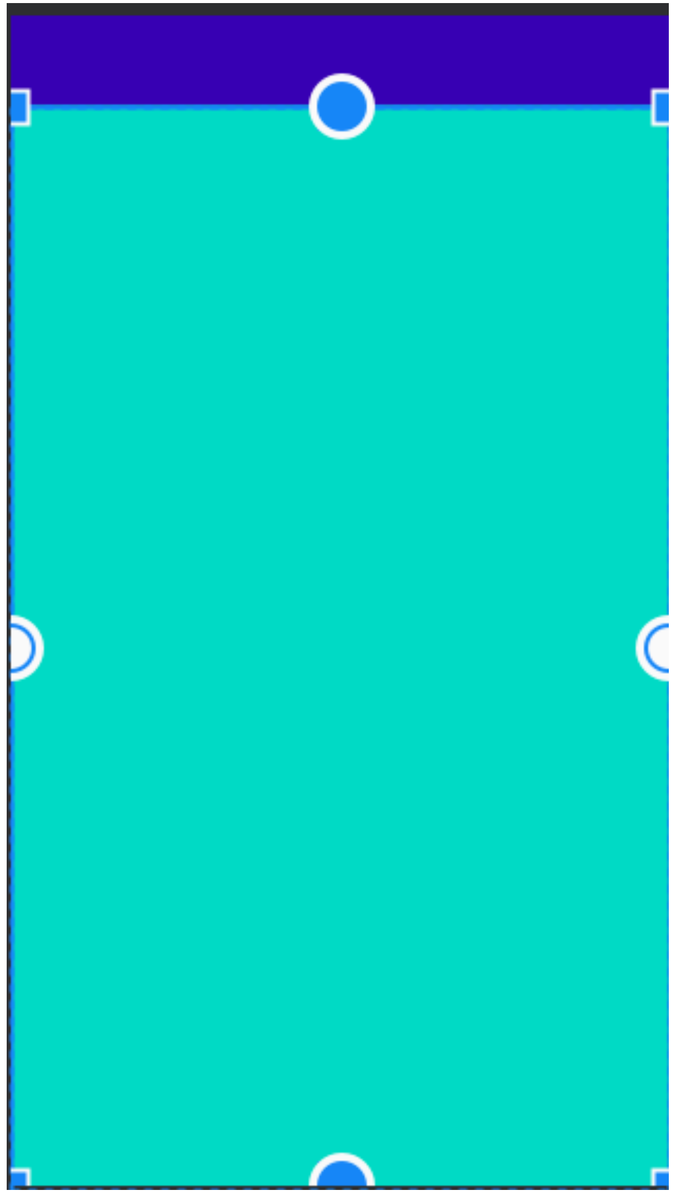



# Fragment

# Fragments - https://developer.android.com/guide/components/fragments#Adding

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/layout_tabcontainer"
        app:layout_constraintTop_toTopOf="parent">
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_constraintTop_toTopOf="parent"
            android:background="@color/colorPrimaryDark"/>
        <TextView
            android:id="@+id/textview_title"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_constraintTop_toTopOf="parent"
            />

    </androidx.constraintlayout.widget.ConstraintLayout>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintTop_toBottomOf="@id/layout_tabcontainer"
        app:layout_constraintBottom_toBottomOf="parent"
        android:id="@+id/fragment_container"
        tools:background="@color/colorAccent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```



# Fragments

Start a series of edit operations on the Fragments associated with this FragmentManager.

```
val transaction : FragmentTransaction = supportFragmentManager.beginTransaction()
transaction.add(R.id.fragment_container, EntryPageFragment(), it.id)
transaction.commit()
viewModel.doneNavigateToPage()
```

ContainerID, Fragment, Tag of Fragment

Schedule a commit of this transaction

# Fragments

Replace an existing fragment that was added to a container

```
val transaction : FragmentTransaction  = supportFragmentManager.beginTransaction()
val fragment = SelectImageFragment()
val bundle : Bundle  = Bundle().apply { this: Bundle
    putParcelable(SelectImageFragment.KEY_PHOTO, photo)
}
fragment.arguments = bundle
transaction.replace(R.id.fragment_container, fragment, it.id)
transaction.addToBackStack(it.id)
transaction.commit()
viewModel.doneNavigateToSelectedPhoto()
```

Add this transaction to the back stack

# Fetch Data From Server

# Response

base API: https://jsonplaceholder.typicode.com
end point: /photos


```
[
 {
      "albumId": 1,
      "id": 1,
      "title": "accusamus beatae ad facilis cum similique qui sunt",
      "url": "https://via.placeholder.com/600/92c952",
      "thumbnailUrl": "https://via.placeholder.com/150/92c952"
 }, …
]
```

```kotlin
override suspend fun getPhotos(urlString: String): List<RemotePhoto> {
    return  withContext(Dispatchers.IO) { this: CoroutineScope
        val inputStream: InputStream
        val jsonString: String
        val jsonArray: JSONArray
        var list : List<RemotePhoto>  = listOf<RemotePhoto>()

        val url = URL(urlString)

        val conn : HttpsURLConnection  = url.openConnection() as HttpsURLConnection

        try {
            // make GET request to the given URL
            conn.connect()
            // receive response as inputStream
            inputStream = conn.inputStream

            if (inputStream != null) {
                jsonString = inputStream.readTextAndClose()
                jsonArray = JSONArray(jsonString)
                list = jsonArray.toRemotePhotos()
            } else {
                // TODO: Log ERROR
                println("inputStream=null")
            }
        } catch (e: IOException) {
            e.stackTrace
        } finally {
            try {
                conn.disconnect()
            } catch (e: IOException) {
                e.stackTrace
            }
        }

        return@withContext list
    }
}
```

1. URL
2. openConnection()
3. connect()
4. InputStream
5. To String
6. To Data class
7. Disconnect

# URL

```
public URL(URL context, String spec, URLStreamHandler handler)
    throws MalformedURLException
```

```
const val BASE_URL = “https://jsonplaceholder.typicode.com/photos"

val url = URL(urlString)
```

# URL.openConnection()

```
public URLConnection openConnection() throws java.io.IOException {
    return handler.openConnection(this);
}
```

URL’s protocol

- HTTP : HttpURLConnection
- HTTPS: HttpsURLConnection
- JAR : JarURLConnection

```
val conn = url.openConnection()
                        as HttpsURLConnection
```

# URL.openConnection()

URLConnection 主要有兩個階段

- Create
- Connect

Optional 階段

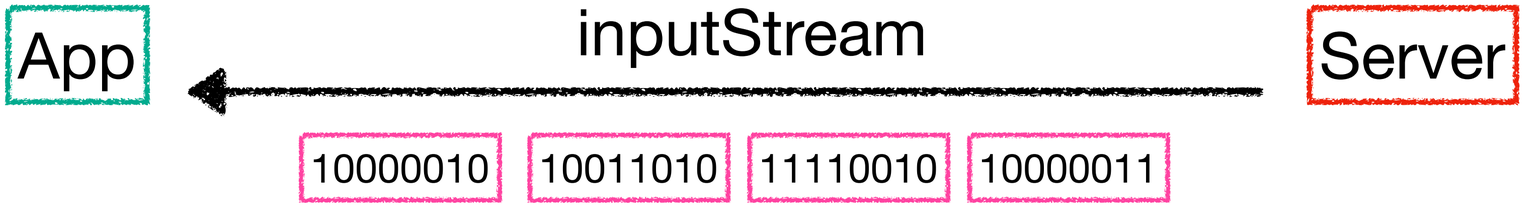
- boolean :
  - doInput, doOutput
- Time out
- Request Property

```
val conn :HttpsURLConnection = url.openConnection() as HttpsURLConnection

// DEFAULT values
conn.doInput = true
conn.doOutput = false

// make GET request to the given URL
conn.connect()
```

# URLConnection.connect()

# URLConnection.connect()


# URLConnection.inputStream




```
public InputStream getInputStream() throws IOException {
    throw new UnknownServiceException("protocol doesn't support input");
}
```

# Decode InputSteam

```
fun InputStream.readTextAndClose(charset: Charset = Charsets.UTF_8): String {
    return this.bufferedReader(charset).use {bufferReader -> bufferReader.readText() }
}
```

**BufferedReader** is a **class in Java** that reads text from a character-input stream, **buffering** characters so as to provide for the efficient reading of characters, lines and arrays.

# Decode InputStream

```kotlin
@InlineOnly
@RequireKotlin( version: "1.2", versionKind = RequireKotlinVersionKind.COMPILER_VERSION, message = "Requires newer compiler
public inline fun <T : Closeable?, R> T.use(block: (T) -> R): R {
    var exception: Throwable? = null
    try {
        return block(this)
    } catch (e: Throwable) {
        exception = e
        throw e
    } finally {
        when {
            apiVersionIsAtLeast( major: 1,  minor: 1,  patch: 0) -> this.closeFinally(exception)
            this == null -> {}
            exception == null -> close()
            else ->
                try {
                    close()
                } catch (closeException: Throwable) {
                    // cause.addSuppressed(closeException) // ignored here
                }
        }
    }
}
```

# Decode InputStream
## JSON

jsonString = inputStream.*readTextAndClose*()

# Decode InputStream
## JSON

jsonString = inputStream.*readTextAndClose*()

```
{...} jsonString = "[\n  {\n    "albumId": 1,\n    "id": 1,\n    "title": "accusamus beatae ad facilis cum similique qui sunt",\n    "u
```

jsonArray = JSONArray(jsonString)

```
{...} jsonArray = {JSONArray@10496} "[{"albumId":1,"id":1,"title":"accusamus beatae ad facilis cum similique qui sunt",
```

# Decode JSON

## Convert to Data Class Objects

val list: List<RemotePhoto> = jsonArray.*toRemotePhotos*()

```
fun JSONArray.toRemotePhotos(): List<RemotePhoto> {
    return List(length()) { it: Int
        val obj : JSONObject! = getJSONObject(it)
        RemotePhoto(
            albumId = obj.optInt(RemotePhoto.KEYS.ALBUM_ID.value),
            id = obj.optInt(RemotePhoto.KEYS.ID.value),
            title = obj.optString(RemotePhoto.KEYS.TITLE.value),
            url = obj.optString(RemotePhoto.KEYS.URL.value),
            thumbnailUrl = obj.optString(RemotePhoto.KEYS.THUMBNAIL_URL.value)
        ) ^List
    }
}
```

# Close Connection

```
try {
    // receive response as inputStream
    inputStream = conn.inputStream

    if (inputStream != null) {
        jsonString = inputStream.readTextAndClose()
        jsonArray = JSONArray(jsonString)
        list = jsonArray.toRemotePhotos()
    } else {
        // TODO: Log ERROR
        println("inputStream=null")
    }
} catch (e: IOException) {
    e.stackTrace
} finally {
    try {
        conn.disconnect()
    } catch (e: IOException) {
        e.stackTrace
    }
}
```

# Decode Image & Cache

# 定義 Singleton Loader

```
typealias ImageLoadCompleteHandler = ((Bitmap?) -> Unit)
object ImageLoader {
    interface ImageCache {
        fun getBitmap(url: String): Bitmap?
        fun putBitmap(url: String, bitmap: Bitmap)
    }

    private const val TIMEOUT = 10000 //ms

    private val bitmapCache: ImageCache = LruBitmapCache()
```

# LruCache 快存

```
class LruBitmapCache constructor(sizeInKB: Int = defaultLruCacheSize):
    LruCache<String, Bitmap>(sizeInKB), ImageLoader.ImageCache {

    override fun sizeOf(key: String, value: Bitmap): Int = value.byteCount / 1024

    override fun getBitmap(url: String): Bitmap? {
        return get(url)
    }

    override fun putBitmap(url: String, bitmap: Bitmap) {
        put(url, bitmap)
    }

    companion object {
        val defaultLruCacheSize: Int
            get() {
                val maxMemory :Int  = (Runtime.getRuntime().maxMemory() / 1024).toInt()
                // Use 1/8th of the available memory for this memory cache.
                return maxMemory / 8
            }
    }
}
```

由 JVM 决定

# 實作 Load Image

1. 讀取快取
2. 取得 inputStream
3. 轉換成 Bitmap
4. 顯示

```kotlin
fun loadBitmap(urlString: String, reqWidth: Int, reqHeight: Int, completeHandler: ImageLoadCompleteHandler) {
    getBitmapFromMemCache(urlString)?.also { it: Bitmap
        completeHandler(it)
    } ?: run { this: ImageLoader
        CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
            withContext(Dispatchers.IO) { this: CoroutineScope
                val url = URL(urlString)
                val conn : HttpURLConnection = url.openConnection() as HttpURLConnection
                try {
                    conn.connectTimeout = TIMEOUT
                    // Cannot use Dalvik as the User-Agent for via.placeholder.com
                    conn.requestMethod = "GET"
                    conn.setRequestProperty("User-Agent", "Dalvic")
                    conn.setRequestProperty("Content-Type", "image/png")
                    conn.connect()

                    var inputStream: InputStream
                    if (conn.responseCode != HttpURLConnection.HTTP_OK) {
                        inputStream = conn.errorStream
                    } else {
                        BitmapFactory.Options().run { this: BitmapFactory.Options
                            // First decode with inJustDecodeBounds=true to check dimensions
                            inJustDecodeBounds = true
                            inputStream = conn.inputStream
                            val bytes : ByteArray = inputStream.toByteArray()
                            inputStream.close()
                            BitmapFactory.decodeByteArray(bytes, offset: 0,
                                bytes.size, opts: this)

                            // Calculate inSampleSize
                            inSampleSize = calculateInSampleSize( options: this,
                                reqWidth, reqHeight)

                            // Decode bitmap with inSampleSize set
                            inJustDecodeBounds = false
                            BitmapFactory.decodeByteArray(bytes, offset: 0,
                                bytes.size, opts: this) ^run
                        }?.let { it: Bitmap
                            bitmapCache.putBitmap(urlString, it)
                    } ^withContext
```

Kotlin

# 讀取快存

```
getBitmapFromMemCache(urlString)?.also {
    completeHandler(it)
} ?: run { ... }
```

```
private fun getBitmapFromMemCache(key: String): Bitmap? {
    return bitmapCache.getBitmap(key)
}
```

# 取得 InputStream

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

Headers | Hide auto-generated headers

| KEY | VALUE |
| --- | --- |
| Cookie | __cfduid=df809bf48199c70567d99d4cea1a809891587434949 |
| Cache-Control | no-cache |
| Postman-Token | <calculated when request is sent> |
| Host | <calculated when request is sent> |
| User-Agent | PostmanRuntime/7.24.1 |
| Accept | */* |
| Accept-Encoding | gzip, deflate, br |
| Connection | keep-alive |

# 取得 InputStream

```
CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
    withContext(Dispatchers.IO) { this: CoroutineScope
        val url = URL(urlString)
        val conn : HttpURLConnection  = url.openConnection() as HttpURLConnection
        try {
            conn.connectTimeout = TIMEOUT
            // Cannot use Dalvik as the User-Agent for via.placeholder.com
            conn.requestMethod = "GET"
            conn.setRequestProperty("User-Agent", "Dalvic")
            conn.setRequestProperty("Content-Type", "image/png")
            conn.connect()
```

https://via.placeholder.com/ 不喜歡 Dalvik

# 取得 InputStream

```
var inputStream: InputStream
if (conn.responseCode != HttpURLConnection.HTTP_OK) {
    inputStream = conn.errorStream
} else {
    BitmapFactory.Options().run { this: BitmapFactory.Options
        // First decode with inJustDecodeBounds=true to check dimensions
        inJustDecodeBounds = true
        inputStream = conn.inputStream
        val bytes : ByteArray  = inputStream.toByteArray()
        inputStream.close()
        BitmapFactory.decodeByteArray(bytes,  offset: 0,
            bytes.size,  opts: this)

        // Calculate inSampleSize
        inSampleSize = calculateInSampleSize( options: this,
            reqWidth, reqHeight)

        // Decode bitmap with inSampleSize set
        inJustDecodeBounds = false
        BitmapFactory.decodeByteArray(bytes,  offset: 0,
            bytes.size,  opts: this) ^run
    }?.let { it: Bitmap
        bitmapCache.putBitmap(urlString, it)
    } ^withContext
}
```

利用 responseCode

測 conn 是否由拿到圖

! HTTP 200

inputStream = conn.*errorStream*

HTTP 200

inputStream = conn.*inputStream*

# 轉換成圖

```
BitmapFactory.Options().run { this: BitmapFactory.Options
    // First decode with inJustDecodeBounds=true to check dimensions
    inJustDecodeBounds = true
    inputStream = conn.inputStream
    val bytes : ByteArray  = inputStream.toByteArray()
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this)

    // Calculate inSampleSize
    inSampleSize = calculateInSampleSize( options: this,
        reqWidth, reqHeight)

    // Decode bitmap with inSampleSize set
    inJustDecodeBounds = false
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this) ^run
}?.let { it: Bitmap
    bitmapCache.putBitmap(urlString, it)
} ^withContext
```

1. 定義 BitmapFactory
2. 將 inputStream 轉成 Bytes
3. 解讀原始圖
4. 計算出合適大小
5. 依照合適大小解析圖片並回傳

# 轉換成圖 - BitmapFactory

```
BitmapFactory.Options().run { this: BitmapFactory.Options
    // First decode with inJustDecodeBounds=true to check dimensions
    inJustDecodeBounds = true
    inputStream = conn.inputStream
    val bytes : ByteArray  = inputStream.toByteArray()
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this)

    // Calculate inSampleSize
    inSampleSize = calculateInSampleSize( options: this,
        reqWidth, reqHeight)

    // Decode bitmap with inSampleSize set
    inJustDecodeBounds = false
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this) ^run
}?.let { it: Bitmap
    bitmapCache.putBitmap(urlString, it)
} ^withContext
```

*Creates Bitmap objects from various sources, including files, streams, and byte-arrays.*

我們想知道的值

outWidth, outHeight

https://developer.android.com/topic/performance/graphics/load-bitmap

# 轉換成圖 - convert InputStream to bytes / Buffering

```kotlin
fun InputStream.toByteArray (): ByteArray {
    val os = ByteArrayOutputStream()
    val buffer = ByteArray( size: 1024)
    var len: Int

    try {
        while (this.read(buffer).also { len = it } != -1) {
            // write bytes from the buffer into output stream
            os.write(buffer,  off: 0, len)
        }
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        os.close()
        this.close()
    }

    return os.toByteArray()
}
```

```java
public int read(byte b[]) throws IOException {
    return read(b, 0, b.length);
}
```

buffer：可以想像成是個小的暫存

# 轉換成圖 - convert to Bitmap

```
BitmapFactory.Options().run { this: BitmapFactory.Options
    // First decode with inJustDecodeBounds=true to check dimensions
    inJustDecodeBounds = true
    inputStream = conn.inputStream
    val bytes : ByteArray  = inputStream.toByteArray()
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this)

    // Calculate inSampleSize
    inSampleSize = calculateInSampleSize( options: this,
        reqWidth, reqHeight)

    // Decode bitmap with inSampleSize set
    inJustDecodeBounds = false
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this) ^run
}?.let { it: Bitmap
    bitmapCache.putBitmap(urlString, it)
} ^withContext
```

# 轉換成圖 - convert to Bitmap

BitmapFactory.decodeByteArray(bytes, 0, bytes.size, this)

```
/**
 * Decode an immutable bitmap from the specified byte array.
 *
 * @param data byte array of compressed image data
 * @param offset offset into imageData for where the decoder should begin
 *               parsing.
 * @param length the number of bytes, beginning at offset, to parse
 * @param opts null-ok; Options that control downsampling and whether the
 *             image should be completely decoded, or just is size returned.
 * @return The decoded bitmap, or null if the image data could not be
 *         decoded, or, if opts is non-null, if opts requested only the
 *         size be returned (in opts.outWidth and opts.outHeight)
 * @throws IllegalArgumentException if {@link BitmapFactory.Options#inPreferredConfig}
 *         is {@link android.graphics.Bitmap.Config#HARDWARE}
 *         and {@link BitmapFactory.Options#inMutable} is set, if the specified color space
 *         is not {@link ColorSpace.Model#RGB RGB}, or if the specified color space's transfer
 *         function is not an {@link ColorSpace.Rgb.TransferParameters ICC parametric curve}
 */
```

# 轉換成圖 - Calculate Size

```
BitmapFactory.Options().run { this: BitmapFactory.Options
    // First decode with inJustDecodeBounds=true to check dimensions
    inJustDecodeBounds = true
    inputStream = conn.inputStream
    val bytes : ByteArray  = inputStream.toByteArray()
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this)

    // Calculate inSampleSize
    inSampleSize = calculateInSampleSize( options: this,
        reqWidth, reqHeight)

    // Decode bitmap with inSampleSize set
    inJustDecodeBounds = false
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this) ^run
}?.let { it: Bitmap
    bitmapCache.putBitmap(urlString, it)
} ^withContext
```

# 轉換成圖 - Calculate Size

```
private fun calculateInSampleSize(options: BitmapFactory.Options, reqWidth: Int, reqHeight: Int): Int {
    // Raw height and width of image
    val (height: Int, width: Int) = options.run { outHeight to outWidth }
    var inSampleSize = 1

    if (height > reqHeight || width > reqWidth) {

        val halfHeight: Int = height / 2
        val halfWidth: Int = width / 2

        // Calculate the largest inSampleSize value that is a power of 2 and keeps both
        // height and width larger than the requested height and width.
        while (halfHeight / inSampleSize >= reqHeight && halfWidth / inSampleSize >= reqWidth) {
            inSampleSize *= 2
        }
    }

    return inSampleSize
}
```

# 轉換成圖 - 重構 Bitmap 並 Cache 起來

```
BitmapFactory.Options().run { this: BitmapFactory.Options
    // First decode with inJustDecodeBounds=true to check dimensions
    inJustDecodeBounds = true
    inputStream = conn.inputStream
    val bytes : ByteArray  = inputStream.toByteArray()
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this)

    // Calculate inSampleSize
    inSampleSize = calculateInSampleSize( options: this,
        reqWidth, reqHeight)

    // Decode bitmap with inSampleSize set
    inJustDecodeBounds = false
    BitmapFactory.decodeByteArray(bytes,  offset: 0,
        bytes.size,  opts: this) ^run
}?.let { it: Bitmap
    bitmapCache.putBitmap(urlString, it)
} ^withContext
```

# RecyclerView

# ViewHolder - 如何顯示對的圖

```
fun bind(photo: DatabasePhoto) {
    this.photo = photo
    binding.textviewId.text = photo.id
    binding.textviewTitle.text = "Loading"
    binding.prgressBarImage.visibility = View.VISIBLE

    binding.image.tag = photo.thumbnailUrl
    binding.image.setImageResource(R.drawable.image_placeholder)
}
```

# ViewHolder - 如何顯示對的圖

```
fun setImage(bitmap: Bitmap?, urlString: String) {
    if (urlString == binding.image.tag) {
        bitmap?.let { it: Bitmap
            binding.image.apply { this: ImageView
                alpha = 0f
                setImageBitmap(it)
                animate()
                    .alpha( value: 1f).duration = 500.toLong()
            }

        }

        binding.textviewTitle.text = photo?.title
        binding.prgressBarImage.visibility = View.GONE
    }
}
```

# Adapter - 如何取得 ImageView 大小

```
override fun onBindViewHolder(holder: ImageViewHolder, position: Int) {
    val photo : DatabasePhoto! = getItem(position)
    holder.bind(photo)
    holder.itemView.setOnClickListener { it: View!
        if (!holder.isLoading) {
            onClickListener.onClick(photo)
        } else {
            Toast.makeText(JSONPlaceholderApplication.INSTANCE,
                "Loading",
                Toast.LENGTH_SHORT).show()
        }
    }

    holder.itemView.post {
        ImageLoader.loadBitmap(photo.thumbnailUrl,
            holder.itemView.width,
            holder.itemView.height) { it: Bitmap?
            holder.setImage(it, photo.thumbnailUrl)
        }
    }
}
```

# Sites

- https://stuff.mit.edu/afs/sipb/project/android/docs/reference/java/net/HttpURLConnection.html
- https://www.slideshare.net/nhtera888xxx/javaiostream
- https://developer.android.com/topic/performance/graphics/load-bitmap