



Tareas asíncronas `AsyncTask`



AsyncTask

- ▶ **AsyncTask** permite ejecutar operaciones en segundo plano y definir los resultados en el UI Thread sin tener que quebrarnos la cabeza manipulando Threads y Handlers.
- ▶ **AsyncTask** permite realizar tareas asíncronas en la interfaz de usuario.
- ▶ Para ello, **utiliza un hilo worker que se encarga de realizar las tareas de larga duración cuyos resultados publica en el hilo principal, sin que sea necesario gestionar directamente ni hilos ni handlers.**



AsyncTask

- ▶ Para utilizar **AsyncTask** debemos:
 - ▶ **Crear una subclase de AsyncTask**, comúnmente como una clase interna privada dentro de la actividad en la que estemos trabajando
 - ▶ **Sobrescribir uno o más métodos de AsyncTask** para poder realizar el trabajo en segundo plano, además de cualquier otra tarea asociada para poder mostrar alguna actualización en el UI Thread.
 - ▶ Cuando sea necesario, **crear una instancia de AsyncTask y llamar al método execute()** para que empiece a realizar su trabajo.

AsyncTask

- La clase **AsyncTask** es una **clase abstracta**, con lo cual para poder utilizarla **deberemos de implementarla**.

AsyncTask

```
public abstract class AsyncTask
```

```
extends Object
```

```
java.lang.Object
```

```
↳ android.os.AsyncTask<Params, Progress, Result>
```

AsyncTask

- AsyncTask utiliza **tipos genéricos**, por lo que **resulta necesario definir tres tipos de datos cada vez que necesitemos crear una instancia de esta clase**:
 - **Parámetros (Params)**: El tipo de información que se necesita para procesar la tarea. Tipo de parámetros enviados a la tarea en ejecución. Se usará como parámetro al invocar **doInBackground()**. Por ejemplo la duración de una canción, o la URL donde está ubicada una imagen que nos interesa.
 - **Progreso (Progress)**: El tipo de información (unidades) que se pasa dentro de la tarea para indicar el progreso de la tarea que se está ejecutando en background. Se usará como parámetro al invocar **onProgressUpdate()**. Por ejemplo el progreso en segundos de la barra de progreso.
 - **Resultado (Result)**: El tipo de información que se pasa cuando la tarea ha sido completada. Tipo de resultado obtenido de la tarea en background. Se usará como parámetro al invocar **onPostExecute()**. Por ejemplo el total de KBytes descargados de algún servidor web, un boolean que nos diga si todo ha ido correctamente, o la imagen que hemos recuperado de una URL (Params).
- Hay que tener en cuenta que existen casos en los que algún tipo genérico no es utilizado por la AsyncTask, para ello utilizamos el tipo Void como se muestra a continuación:

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

AsyncTask

■ Ejemplo:

```
extends AsyncTask<String,Void,Bitmap>
```

- El primer parámetro (String) define el tipo de variable que usamos como parámetro **(Params)** al invocar doInBackground().
- El segundo parámetro (Void) define el tipo de variable que pasamos como parámetro **(Progress)** al invocar onProgressUpdate(), que es llamado después de llamar a publishProgress().
- Finalmente, el último parámetro (Bitmap) define el tipo de variable que pasamos como parámetro **(Result)** al invocar onPostExecute(). Este valor es devuelto por doInBackground().

AsyncTask

- Si no se desea declarar uno de los tipos genéricos, se utilizará Void.

```
public class Task extends AsyncTask<Long,Void,String>
```

- En la declaración anterior, se define como parámetro del método doInBackground() el tipo Long, no se utilizará el método onProgressUpdate(), por lo que no se define su tipo (Void) y se define un tipo String como el tipo de resultado obtenido de la tarea en background.
- Existe una característica adicional de los tipos genéricos Params y Progress: los métodos de la tarea asíncrona que los utilizan pueden recibir múltiples parámetros del tipo que definan los tipos genéricos. Por ejemplo, según se ha definido Task previamente, El método doInBackground(Long... miliseconds) podría ser invocado a través del método execute (Long.. miliseconds) con múltiples parámetros Long:

```
new Task().execute(1000,2000,500);
```



Etapas del AsyncTask

- Cada vez que una tarea asíncrona se ejecuta, se pasa a través de 4+1 etapas():
 - `onPreExecute();`
 - `doInBackground(Params...)`
 - `onProgressUpdate(Progress...)`
 - `onPostExecute(Result)`
 - `onCancelled()`



Etapas del AsyncTask

■ **onPreExecute()**

- Se invoca en el UI Thread inmediatamente después de que la tarea es ejecutada/iniciada.
- Este paso se utiliza normalmente para configurar/inicializar la tarea.
- El ejemplo básico es cuando se muestra un progressBar en la interfaz de usuario, con la barra de progreso inicializada a cero por ejemplo.

Etapas del AsyncTask

➤ **doInBackground(Params...)**

- Se invoca en el hilo background inmediatamente después de que `onPreExecute()` termine de ejecutarse.
- Es donde **se realizarán las tareas de larga duración.**
- En esta etapa se calcula el tiempo estimado que tomará la realización de la tarea, que deberá pasarse a la última etapa de todo el proceso.
- **Recibe los N parámetros de tipo genérico Params que se han pasado al método `execute(Params...)`** (Método llamado para lanzar a ejecución la tarea asíncrona desde el hilo principal).
- **Devolverá como resultado un objeto cuyo tipo genérico será results y que será recibido por el método `onPostExecute(Results)`.**
- **Dentro de esta etapa podemos utilizar el método `publishProgress(Progress...)` para publicar una o más unidades de progreso en el UI Thread, de esta manera se pasará a la siguiente etapa que es `onProgressUpdate(Progress...)`.**



Etapas del AsyncTask

➤ **onProgressUpdate(Progress...)**

- Se invoca en el UI thread cada vez que se invoque o después de una llamada a `publishProgress(Progress...)`.
- El momento de la ejecución es indefinido
- Este método es **utilizado para mostrar cualquier tipo de progreso visual en la interfaz de usuario (Hilo principal) mientras la tarea en segundo plano o background sigue ejecutándose.**
- Por ejemplo, se puede utilizar para mostrar una barra de progreso o para actualizar mensajes tipo log en un `TextView`.



Etapas del AsyncTask

■ onPostExecute(Result)


- Se invoca en el UI Thread después de que el proceso en segundo plano o background haya terminado.
- El resultado devuelto por todo el proceso se pasa a este método como parámetro, es decir, **recibe como parámetro el resultado que devuelva el método doInBackground(Params...) y que será un objeto de tipo genérico Result.**

Etapas del AsyncTask

- onCancelled()



- Será llamado cuando se llame al método cancel() asociado a alguna instancia de AsyncTask.

```
@Override  
protected void onCancelled() {  
    Toast.makeText(MainHilos.this, "Tarea cancelada!",  
        Toast.LENGTH_SHORT).show();  
}
```



AsyncTask

- Para poder trabajar sin problemas con la clase AsyncTask, existen algunas reglas acerca de los threads que debes tomar en cuenta:
 - La instancia de la tarea debe crearse en el UI Thread.
 - El método execute(Params) debe invocarse en el UI Thread.
 - Los métodos onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...), onCancelled() No deben llamarse de forma manual.
 - La tarea debe ser ejecutada sólo una vez (con excepción de que la ejecución corresponda a un segundo intento).

- 
- 
- <https://github.com/rononunsys/AsyncTask.git>
 - <https://github.com/rononunsys/Api.git>