CS 646 Android Mobile Application Development
Spring Semester, 2015
Doc 11 Volley, Handlers, Loopers
March 5, 2015

Android Programming: Big Nerd Ranch

Chapter 26 HTTP & Background Tasks

Covers HTTP & AsyncTask

Chapter 27 Loopers, Handlers & HandlerThread

Volley

Volley

Networking Library for Android

Google I/O 2013 Presentation

http://www.youtube.com/watch?v=yhv8l9F44qo

Download

git clone https://android.googlesource.com/platform/frameworks/volley

Designed to improve network performance

Some Tutorials

https://www.captechconsulting.com/blog/clinton-teegarden/android-volley-library-tutorial

http://tinyurl.com/jwth6g5

http://arnab.ch/blog/2013/08/asynchronous-http-requests-in-android-using-volley/

http://tinyurl.com/kd6td6l

Basic Parts

RequestQueue

- Used to make network requests
- Processes them off UI thread
- Handles multiple requests at same time

Requests need

- URL
- Response.Listener
- Response.ErrorListener

Request Types

- JsonObjectRequest
- JsonArrayRequest
- StringRequest
- ImageRequest

Basic Flow

Create RequestQueue

Create Request

Add request to queue

- Request is handled on separate thread

Proper listener is called when request is done

- Listeners are called on UI thread

Simple Example

```
RequestQueue queue = Volley.newRequestQueue(this);  // this = current context

String url = "http://www.eli.sdsu.edu/courses/fall09/cs696/examples/names.json";


Response.Listener<JSONArray> success = new Response.Listener<JSONArray>() {
            public void onResponse(JSONArray response) {
             Log.d("rew", response.toString());
            }
        };
Response.ErrorListener failure = new Response.ErrorListener() {
        public void onErrorResponse(VolleyError error) {
              Log.d("rew", error.toString());
          }
        };
JsonArrayRequest getRequest = new JsonArrayRequest( url, success, failure);
queue.add(getRequest);
```

Caching Network Data

When going back to an Activity

When recreating activity

When restarting activity

When don't have network access

Volley Data Cache

Default behavior

Cache data on disk

- Up to 5M default

Cache contains the network response

Cache persists after app quits

Can add in memory cache

Simple Example

```
RequestQueue queue = Volley.newRequestQueue(this);  // this = current context

String url = "http://www.eli.sdsu.edu/courses/fall09/cs696/examples/names.json";

 Cache.Entry cachedData = queue.getCache().get(url);

 if (cachedData != null ) {
      try {
           JSONArray dataAgain = new JSONArray(new String(cachedData.data, "UTF8"));
         } catch (UnsupportedEncodingException e) {
                  e.printStackTrace();
         } catch (JSONException e) {
                  e.printStackTrace();
         }
} else {
     request the data for the first time
}
```

Expired & Needs Refreshing

```
Cache.Entry cachedData = queue.getCache().get(url);

cachedData.isExpired()

cachedData.refreshNeeded()
```

Downloading Images

```
ImageRequest(
    String url,
    Response.Listener<Bitmap> listener,
    int maxWidth,                                   //0 for no max
    int maxHeight,
    Config decodeConfig,                            // null works for us
    Response.ErrorListener errorListener)
```

Sample Example

```
RequestQueue queue = Volley.newRequestQueue(this);
String url = "http://blahblah";
ImageView image =(ImageView) this.findViewById(R.id.photo);


ImageRequest ir = new ImageRequest(url,
        new Response.Listener<Bitmap>() {
                public void onResponse(Bitmap response) {
                    image.setImageBitmap(response);
    }
}, 0, 0, null, anErrorListener);

queue.add(ir);
```

Volley Issues

RequestQueue Detail

Contains network resources

If multiple activities use network share same queue

RequestQueue Sharing - 1

Make it static

```
public class MainActivity extends Activity {
    static RequestQueue queue;

    public static RequestQueue getQueue() {
        if (queue == null)
            queue = Volley.newRequestQueue(this);
        return queue;
    }

    public static void stopQueue() {
        if (queue != null)  {
            queue.stop();
            queue = null;
        }
    }
```

RequestQueue Sharing - 1 Issues

When to call stopQueue?

You can restart a stopped queue using method start()

```
queue.start();
```

RequestQueue Sharing - 2

Use Application

Each app has single instance of android.app.Application

Sublcass Application & add queue

Subclassing Application

```
public class NetworkApplication extends Application{
    private RequestQueue queue;

    public void onCreate() {
        queue =Volley.newRequestQueue(this);
    }

    public RequestQueue getQueue() {
        return queue;
    }
```

Telling Android to Use Subclass

In Manifest file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.sdsu.cs.VolleyExample"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />
      <uses-permission android:name="android.permission.INTERNET" />
    <application
→       android:name="edu.sdsu.cs.VolleyExample.NetworkApplication"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

Accessing the Application Object

```
public class MainActivity extends Activity {
    RequestQueue queue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        NetworkApplication app = (NetworkApplication) this.getApplicationContext();
        queue = app.getQueue();
}
```

Releasing Memory when in Background

Application.onTrimMemory(int level)

Called when Android needs more memory

Levels

TRIM_MEMORY_COMPLETE,
TRIM_MEMORY_MODERATE,
TRIM_MEMORY_BACKGROUND,
TRIM_MEMORY_UI_HIDDEN, 
TRIM_MEMORY_RUNNING_CRITICAL,
TRIM_MEMORY_RUNNING_LOW,
TRIM_MEMORY_RUNNING_MODERATE.

Subclassing Application

```
public class NetworkApplication extends Application{
      private RequestQueue queue;

      public void onTrimMemory(int level) {
     if (level == TRIM_MEMORY_UI_HIDDEN) {
         queue.stop();
         queue == null;
      }
```

```
 public RequestQueue getQueue() {
if (queue == null)
     queue = Volley.newRequestQueue(this);
        return queue;
 }
```

Handler

Handler

Used to:

Send message from one thread to another

Execute code in the future

Uses Message object

android.os.Message

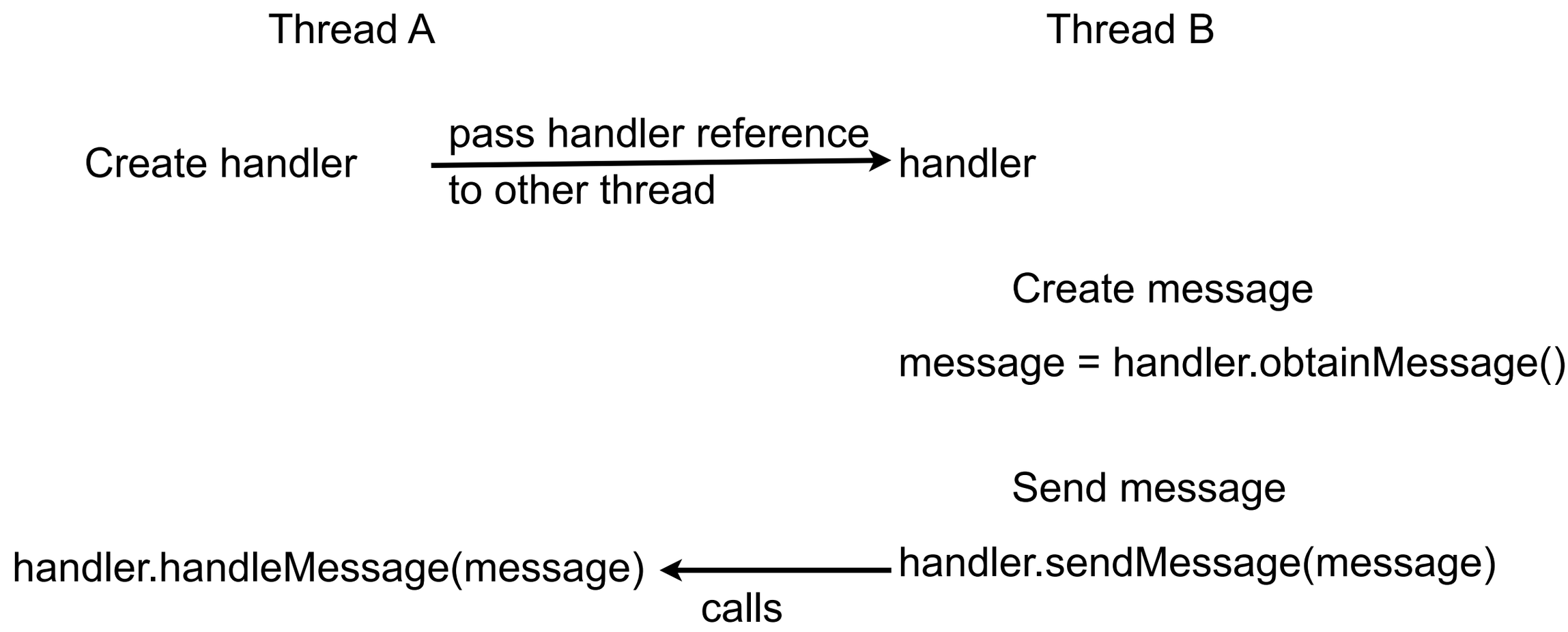
Contains public fields for data

- int arg1
- int arg2
- Object obj
- int what

Data fields mean what ever you want

Can add a bundle for more data

How it Works


Thread A
Thread B
Create handler
pass handler reference
to other thread
handler
Create message
message = handler.obtainMessage()
Send message
handler.sendMessage(message)
calls
handler.handleMessage(message)

Creating Messages - Handler methods

obtainMessage()

obtainMessage(int what)

obtainMessage(int what, Object obj)

obtainMessage(int what, int arg1, int arg2)

obtainMessage(int what, int arg1, int arg2, Object obj)

Handling Messages

Handler subclass must implement

handleMessage(Message aMessage)

This method has to handle messages sent

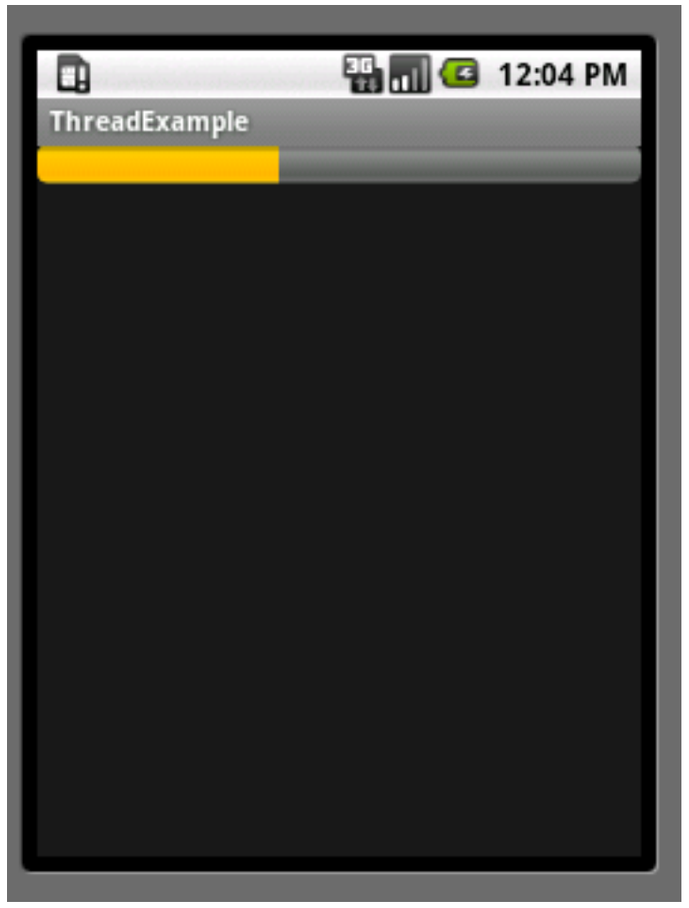
Handler Scheduling

post(Runnable)
postAtTime(Runnable, long)
postDelayed(Runnable, long)

sendEmptyMessage(int)
sendMessage(Message)
sendMessageAtTime(Message, long)
sendMessageDelayed(Message, long)

ProgressBar Example

Just shows a progress bar progressing



ThreadExample

```
public class ThreadExample extends Activity {
    ProgressBar progressView;
    boolean isRunning = false;

    Handler handler = new Handler() {
        public void handleMessage(Message empty) {
            progressView.incrementProgressBy(5);
        }
    };

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        progressView = (ProgressBar) findViewById(R.id.progress);
    }

    public void onStop() {
        super.onStop();
        isRunning = false;
    }
```

ThreadExample

```
public void onStart() {
        super.onStart();
        progressView.setProgress(0);

        Thread background = new Thread(new Runnable() {
                public void run() {
                        try {
                                for (int i = 0; i < 20 && isRunning; i++) {
                                        Thread.sleep(1000);
                                        handler.sendMessage(handler.obtainMessage());
                                }
                        } catch (Throwable t) {// just end    }
                }
        });
        isRunning = true;
        background.start();
    }


}
```

Sending Text Messages to the future

Rather than use a thread use sendMessageDelayed

Sends data in the message using Bundle

Sending Text The Hard Way

```
public class ThreadExample extends Activity {
    Handler handler = new Handler() {
        public void handleMessage(Message word) {
            String text = word.getData().getString("key");
            Toast.makeText(ThreadExample.this, text, Toast.LENGTH_SHORT).show();
        }
    };

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onStart() {
        super.onStart();
        String[] text = { "Bat", "cat", "dat", "fat", "hat", "mat" };
        for (int i = 0; i < text.length; i++) {
            Bundle data = new Bundle();
            data.putString("key", text[i]);
            Message word = new Message();
            word.setData(data);
            handler.sendMessageDelayed(word, 1000 * (i + 1));
        }
```


Read the documentation to find the easy way

Progress Dialog

Displays a Progress Dialog
Uses Message what to transmit data

ThreadExample

```
public class ThreadExample extends Activity {
	ProgressDialog waitDialog;
	private static final int WAIT_DIALOG_KEY = 0;
	Handler handler = new Handler() {
		public void handleMessage(Message command) {
			if (command.what == 0)
				showDialog(WAIT_DIALOG_KEY);
			else
				waitDialog.dismiss();
		}
	};

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);
	}
```

ThreadExample

```
protected Dialog onCreateDialog(int id) {
        switch (id) {
        case WAIT_DIALOG_KEY: {
                waitDialog = new ProgressDialog(this);
                waitDialog.setMessage("Please wait while loading...");
                waitDialog.setIndeterminate(true);
                waitDialog.setCancelable(true);
                return waitDialog;
        }
        }
        return null;
}

        public void onStart() {
                super.onStart();
                Message on = new Message();
                on.what = 0;
                handler.sendMessageDelayed(on, 1000);
                Message off = new Message();
                off.what = 1;
                handler.sendMessageDelayed(off, 8000);
        }
}
```

Looper

Basic Idea

Background thread

- Waits for message
- When gets message processes it
- Then waits for another message

Message can be request to fetch data

So one thread can handle multiple requests

Basic Components

Messages

Handlers

HandlerThread

Looper

- Don’t need to see looper directly

HandlerThread - Important Methods

start

Called to start thread

onLooperPrepared

Used to create handler

quitSafely

quit

Used to end thread

HandlerThread Example

```
public class LooperThread extends HandlerThread {
    public Handler simpleHandler;

    public LooperThread() {
        super("Simple looper");
    }

    @Override
    protected void onLooperPrepared() {
        Log.i("rew", "Start Looper");
        simpleHandler = new Handler() {
            public void handleMessage(Message input) {
                Log.i("rew", "Got message");
            }
        };
    }
}
```

Using the LooperThread

```
public class MainActivity extends ActionBarActivity {
    LooperThread sampleLooper;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    protected void onResume() {
        super.onResume();
        sampleLooper = new LooperThread();
        sampleLooper.start();
    }

    protected void onPause() {
        super.onPause();
        sampleLooper.quitSafely();
    }
```

Using the LooperThread part 2

```
    public void runLooper(View source) {
        sampleLooper.simpleHandler.sendEmptyMessage(0);
     }
}
```