4.4. Определение местоположения устройства

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 4.4. Определение местоположения устройства

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:55

Оглавление

1. Сервис определения местоположения.

Android устройства могут предоставить данные по текущему местоположению. Основные примеры использования геолокации - определить собственное местонахождение, определить координаты какого-нибудь объекта и т.д. Это вовсю используется для, например, пользования картой, получения актуальной для вашей местности информации (например, прогноз погоды).

При разработке приложений, использующих геолокационные сервисы, необходимо помнить следующее. Во-первых местонахождение пользователя конфиденциально, т.е. вы всегда должны явно указать пользователю, что собираетесь использовать эти данные и запросить согласие на подобное использование. Во-вторых такие сервисы достаточно ресурсоёмки как по части потребления энергии, так и по части передачи объемов информации, следовательно необходимо грамотно подходить к их применению и предусмотреть возможность отключения.

Для определения места положения в Android существует специальный класс Location. Местоположение может состоять из широты, долготы, отметки времени и другой информации, такой как азимут, высота и скорость. Получить местоположение можно с помощью класса LocationManager. Этот класс предоставляет доступ к системным службам определения местоположения. Эти службы позволяют приложениям получать обновления положения устройства или получать уведомления, когда устройство приближается к заданному объекту.

Первым шагом настройки доступа к обновлению местоположения является объявление разрешений в манифесте. В файл *AndroidManifest.xml* необходимо добавить

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

для определения местоположения по GPS или

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

для определения местоположения через Интернет. Вам могут потребоваться оба способа, если вы хотите создать более универсальное приложение. В случае, если объявления отсутствуют, приложение сгенерируется исключение `SecurityException` .

В случае, если приложению требуется доступ к местоположению пользователя, когда приложение работает в фоновом режиме, нам необходимо добавить следующие разрешения вместе с указанными выше:

```
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
```

`LocationManager` - это класс, через который приложение может получать доступ к службам определения местоположения на Android. Как и в случае с другими системными службами, ссылку на него можно получить, вызвав метод getSystemService (). Если ваше необходимо получать обновления местоположения в рамках Activity, следует выполнить этот шаг в методе onCreate().

```
locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
```

Далее в приложении следует проверить доступ к соответствующим сервисам. Сделать это можно следующим образом:

```
if ((ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED)) {
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), locationPermissionCode)
        }
```

Для определения расположения пользователя используется несколько технологий. Используемое оборудование зависит от типа поставщика сведений о расположении, выбранного для сбора данных. Android использует три поставщика сведений о расположении.

- Поставщик данных GPS – технология GPS обеспечивает наиболее точные сведения о расположении, потребляет больше всего электроэнергии и лучше всего подходит для туризма. Этот поставщик использует сочетание технологии GPS и технологии aGPS, которая возвращает данные GPS, собранные вышками сотовой связи.
- Поставщик данных сети – предоставляет сочетание данных Wi-Fi и сотовой связи, включая данные aGPS, собранные вышками сотовой связи. Он использует меньше электроэнергии, чем поставщик данных GPS, но возвращает данные о расположении переменной точности.
- Пассивный поставщик – дополнительный вариант, при котором другие приложения или службы запрашивают у поставщика создание данных расположения в приложении. Это менее надежный, но более энергосберегающий вариант, который идеально подходит для приложений, для работы которых не требуется постоянное обновление данных расположения.

Когда приложение получит ссылку на `LocationManager`, ему нужно указать требуемый тип сведений о расположении и частоту обновления данных. Для этого вызовите `RequestLocationUpdates` для объекта `LocationManager` и передайте в него какие-либо критерии для данных об изменении и обратный вызов для получения данных об изменении расположения. Этот обратный вызов является типом, который должен реализовывать интерфейс `LocationListener` (более подробно это описывается далее в этой главе). Метод `requestLocationUpdates` сообщает системной службе расположения, что приложение должно начать получать данные об изменении расположения. Этот метод позволяет указать поставщик, а также пороги времени и расстояния для управления частотой обновления. Например, приведенный ниже метод запрашивает данные об изменении расположения от поставщика сведений о расположении GPS каждые 5000 миллисекунд и только при изменении расположения более чем на 5 метров.

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5f, this)
```

Приложение должно запрашивать данные об изменении расположения только так часто, как это необходимо для его эффективной работы. Это увеличивает время работы аккумулятора и повышает удобство работы пользователя.

Для получения место положения через интернет необходимо задействовать `LocationManager.NETWORK_PROVIDER`.

После того как приложение запросило данные об изменении из `LocationManager`, оно может получить сведения от службы, реализовав интерфейс LocationListener.

Интерфейс содержит четыре метода:

- `onLocationChanged(location: Location)` - вызывается при изменении местоположения.
- `onProviderDisabled(provider: String)` - вызывается, когда поставщик отключен пользователем.
- `onProviderEnabled(provider: String)` - вызывается, когда поставщик активирован пользователем.
- `onStatusChanged(provider: String!, status: Int, extras: Bundle!)` - вызывается при изменении статуса провайдера (на это есть множество причин).

Чтобы обеспечить экономию системных ресурсов, приложение должно как можно скорее отменить подписку на данные об изменении расположения. Метод `removeUpdates` указывает `LocationManager` остановить отправку данных об изменении в наше приложение. Например, действие может вызвать `removeUpdates` в методе `OnPause`, чтобы снизить энергопотребление, если приложение не нуждается в данных об изменении расположения, пока это действие не отображается на экране.

2. Класс Geocoder

Геокодирование - это процесс перевода описания местоположения в GPS-координаты (широта, долгота, высота над уровнем моря) и обратный процесс. Естественно, устройство должно иметь соответствующее оснащение и поддерживать связь с сетью.

В Android для геокодирования есть специальный класс Geocoder. С его помощью вы можете трансформировать адрес или другое описание местности в координаты и наоборот.

Количество параметров в описании местоположения с обратным геокодированием может разниться, например, одно может содержать полный адрес ближайшего здания, а другое может содержать только название города и почтовый индекс.

В классе существует два конструктора:

- `Geocoder(context: Context!, locale: Locale!)` - геокодер, с локализацией для данной местности.
- `Geocoder(context: Context!)` - геокодер, с локализацией по умолчанию.

Класс содержит несколько публичных методов

- `getFromLocation()`. Метод возвращает массив адресов, которые, как известно, описывают область, непосредственно окружающую заданные широту и долготу. Результаты являются приблизительными и не гарантируют значимости или правильности. Параметрами метода являются: `latitude` - широта, `longitude` - долгота, `maxResults` - максимальное количество адресов. Синтаксис метода следующий:

```
fun getFromLocation(
    latitude: Double,
    longitude: Double,
    maxResults: Int
): MutableList<Address!>!
```

- `getFromLocationName()`. Метод возвращает массив адресов, которые описывают местоположение. Параметрами метода являются: `locationName` - предоставленное пользователем описание местоположения (например, название места, например "Dalvik, Iceland", адрес "1600 Amphitheatre Parkway, Mountain View, CA", код аэропорта, например "SFO"), `maxResults` - максимальное количество адресов. Синтаксис метода следующий:

```
fun getFromLocationName(
    locationName: String!,
    maxResults: Int
): MutableList<Address!>!
```

Еще один вариант функции:

```
fun getFromLocationName
Added in API level 1
fun getFromLocationName(
    locationName: String!,
    maxResults: Int,
    lowerLeftLatitude: Double,
    lowerLeftLongitude: Double,
    upperRightLatitude: Double,
    upperRightLongitude: Double
): MutableList<Address!>!
```

Здесь `lowerLeftLatitude` - широта нижнего левого угла ограничивающей рамки, `lowerLeftLongitude` - долгота нижнего левого угла ограничительной рамки, `upperRightLatitude` - широта правого верхнего угла ограничивающей рамки, `upperRightLongitude` - долгота правого верхнего угла ограничивающей рамки.

В приложении функция получения адреса местоположения по координатам широты и долготы может выглядеть следующим образом:

```
private fun getAddress(lat: Double, lng: Double): String {
        val geocoder = Geocoder(this, Locale.getDefault())
        val list = geocoder.getFromLocation(lat, lng, 1)
        return list[0].getAddressLine(0)
}
```

3. Настройка обновления информации о местоположении

Для определения местоположения вы можете использовать как стандартный `requestLocationUpdate`, так и Fused Location Provider, который является частью api Google play services. Плюсами решения на основе Fused Location Provider является то, что вам не надо самому подыскивать наиболее точный источник координат, он сделает это за вас. Если запускать его с критериями для определения наиболее точного местоположения, то он будет работать по следующему принципу:

- Если GPS доступен и включен, то выбирается `GPS_PROVIDER` в качестве источника.
- Если GPS недоступен то выбирается `NETWORK_PROVIDER`. В данном случаи координаты определяются при помощи вышек сотовой связи и WI-FI.
- Если не того не другого нет то используется `PASSIVE_PROVIDER`. Система пытается использовать любой способ получить хоть какое-то местоположение в том числе и при использовании акселерометра. Этот провайдер наименее точный.

Fused Location Provider включает в себя все три и переключается между ними в зависимости от ситуации для получения наиболее точного место положения, так же при его использовании заряд батареи жрется медленнее. Его главными минусами является то, что для его работы необходим Google play services и то, что вы не сможете отследить какой именно тип провайдера используется в данный момент.

Что касается стандартного `LocationManager`, то вам придется самому определять какой провайдер использовать и переключаться между ними в ручную. Плюсом является то, что вы полностью контролируете весь процесс.

Прежде чем запрашивать обновления местоположения, ваше приложение должно подключиться к службам определения местоположения и сделать запрос местоположения. В зависимости запроса провайдер местоположения либо вызывает метод обратного вызова `LocationCallback.onLocationResult()` и передает ему список объектов `Location`, либо выдает `PendingIntent`, который содержит местоположение в его расширенных данных.

В этом разделе показано, как получить обновление местоположения с помощью метода `LocationCallback`. Вызовем метод `requestLocationUpdates()`, передав ему экземпляр объекта `LocationRequest` и `LocationCallback`. Далее, определим метод `startLocationUpdates()`, как показано ниже:

```
override fun onResume() {
    super.onResume()
    if (requestingLocationUpdates) startLocationUpdates()
}

private fun startLocationUpdates() {
    fusedLocationClient.requestLocationUpdates(locationRequest,
            locationCallback, Looper.getMainLooper())
}
```

Обратите внимание, что приведенный выше фрагмент кода содержит логическую переменную `requestingLocationUpdates`, которая необходима для отслеживания того, включил пользователь обновления местоположения или нет.

Fused Location Provider вызывает метод `LocationCallback.onLocationResult()`. В качестве аргумента выступает список `Location`, содержащий широту и долготу местоположения. В следующем фрагменте показано, как реализовать интерфейс `LocationCallback`, определить метод, а затем получить временную метку обновления местоположения:

```
private lateinit var locationCallback: LocationCallback

// ...

override fun onCreate(savedInstanceState: Bundle?) {
    // ...

    locationCallback = object : LocationCallback() {
        override fun onLocationResult(locationResult: LocationResult?) {
            locationResult ?: return
            for (location in locationResult.locations){
                // Update UI with location data
                // ...
            }
        }
    }
}
```

Чтобы остановить обновление местоположения, вызовите метод `removeLocationUpdates()`, передав ему `LocationCallback`, как показано в следующем примере кода:

```
override fun onPause() {
    super.onPause()
    stopLocationUpdates()
}

private fun stopLocationUpdates() {
    fusedLocationClient.removeLocationUpdates(locationCallback)
}
```

Чтобы отслеживать, включены ли в настоящее время обновления местоположения можно использовать логическую переменную `requestingLocationUpdates`. В методе `onResume()` необходимо проверить, активны ли в настоящее время обновления местоположения, и если нет активировать их:

```
override fun onResume() {
    super.onResume()
    if (requestingLocationUpdates) startLocationUpdates()
}
```

4. Пример 4.4

Разработаем приложение, в котором отобразим текущее местоположение и выведем его адрес.

Для начала в файл манифеста проекта необходимо добавить необходимые разрешения:

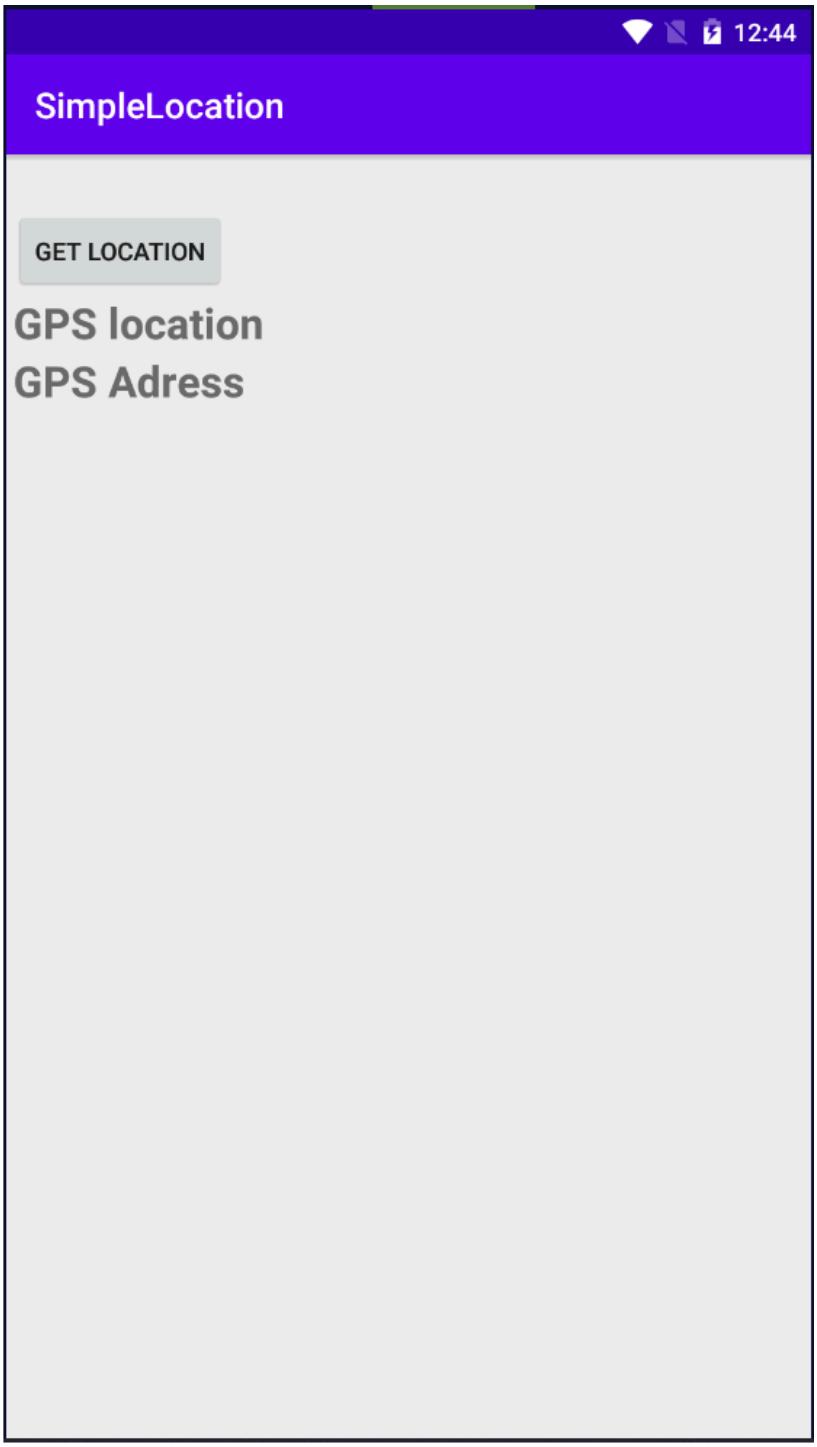
```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Разметка активности будет содержать одну кнопку и два текстовых представления для размещения координат и адреса местоположения устройства.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="4dp" tools:context=".MainActivity">

    <Button
        android:id="@+id/getLocation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView"
        android:layout_marginTop="26dp"
        android:text="Get location" />

    <TextView
        android:id="@+id/textGPS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="GPS location"
        android:textSize="24sp"
        android:textStyle="bold" />
    <TextView
        android:id="@+id/textAdress"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="GPS Adress"
        android:textSize="24sp"
        android:textStyle="bold" />
</LinearLayout>
```



Сначала инициализируем представления и в методе `onCreate()` добавим слушателя для кнопки:

```
private lateinit var locationManager: LocationManager
private lateinit var tvGpsLocation: TextView
private lateinit var tvGpsAdress: TextView
private val locationPermissionCode = 2
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    //title = "KotlinApp"
    val button: Button = findViewById(R.id.getLocation)
    button.setOnClickListener {
        getLocation()
    }
}
```

В методе `getLocation()` создадим объект LocationManager, проверим доступ к соответствующим сервисам и вызовем метод requestLocationUpdates() для обновления местоположения с использованием GPS провайдера как показано ниже:

```
 private fun getLocation() {
        locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
        if ((ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED)) {
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), locationPermissionCode)
        }
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5f, this)

    }
```

Реализуем функцию `getAddress` для перевода координат в адрес. Для этого используем объект типа `Geocoder`:

```
private fun getAddress(lat: Double, lng: Double): String {
        val geocoder = Geocoder(this, Locale.getDefault())
        val list = geocoder.getFromLocation(lat, lng, 1)
        return list[0].getAddressLine(0)
    }
```

Далее реализуем метод `onLocationChanged` интерфейса `LocationListener`, который вызывается при изменении местоположения. В нем установим в текстовые поля необходимые значения:

```
override fun onLocationChanged(location: Location) {
        tvGpsLocation = findViewById(R.id.textGPS)
        tvGpsLocation.text = "Latitude: " + location.latitude + " \nLongitude: " + location.longitude
        tvGpsAdress = findViewById(R.id.textAdress)
        tvGpsAdress.text = getAddress(location.latitude, location.longitude)
    }
```

Наконец, реализуем метод - `onRequestPermissionsResult` обратный вызов для результата запроса разрешений:

```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {
        if (requestCode == locationPermissionCode) {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission Granted", Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "Permission Denied", Toast.LENGTH_SHORT).show()
            }
        }
    }
```

Полностью код *MainActivity.kt* будет выглядеть следующим образом:

```
package ru.samsung.itacademy.mdev.simplelocation

import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.location.Geocoder
import android.location.Location
import android.location.LocationListener
import android.location.LocationManager
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import java.util.*

class MainActivity : AppCompatActivity(), LocationListener {
    private lateinit var locationManager: LocationManager
    private lateinit var tvGpsLocation: TextView
    private lateinit var tvGpsAdress: TextView
    private val locationPermissionCode = 2
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //title = "KotlinApp"
        val button: Button = findViewById(R.id.getLocation)
        button.setOnClickListener {
            getLocation()
        }
    }
    private fun getLocation() {
        locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
        if ((ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED)) {
            ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), locationPermissionCode)
        }
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000, 5f, this)

    }
    override fun onLocationChanged(location: Location) {
        tvGpsLocation = findViewById(R.id.textGPS)
        tvGpsLocation.text = "Latitude: " + location.latitude + " \nLongitude: " + location.longitude
        tvGpsAdress = findViewById(R.id.textAdress)
        tvGpsAdress.text = getAddress(location.latitude, location.longitude)
    }
    override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {
        if (requestCode == locationPermissionCode) {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission Granted", Toast.LENGTH_SHORT).show()
            } else {
                Toast.makeText(this, "Permission Denied", Toast.LENGTH_SHORT).show()
            }
        }
    }

    private fun getAddress(lat: Double, lng: Double): String {
        val geocoder = Geocoder(this, Locale.getDefault())
        val list = geocoder.getFromLocation(lat, lng, 1)
        return list[0].getAddressLine(0)
    }

}
```

После запуска приложения и нажатия на кнопку на экране появятся широта и долгота устройства, а также адрес местоположения:

В качестве самостоятельной работы, попробуйте исправить код приложения так, чтобы для получения координат использовался провайдер данных сети (`NETWORK_PROVIDER`)

Начать тур для пользователя на этой странице