103. Sensores

103.1 Información sobre Sensores en Android

La plataforma de Android admite tres amplias categorías de sensores:

- **Sensores de movimiento**
  Estos sensores miden las fuerzas de aceleración y las fuerzas de rotación en tres ejes. Esta categoría incluye acelerómetros, sensores de gravedad, giroscopios y sensores del vector de rotación.
- **Sensores ambientales**
  Estos sensores miden varios parámetros ambientales, como la temperatura y la presión del aire ambiental, la iluminación y la humedad. Esta categoría incluye barómetros, fotómetros y termómetros.
- **Sensores de posición**
  Estos sensores miden la posición física de un dispositivo. Esta categoría incluye sensores de orientación y magnetómetros.

[TODO]

El marco de trabajo del sensor de Android permite acceder a muchos tipos de sensores. Algunos de estos sensores se basan en hardware y otros en software. Los sensores basados en hardware son componentes físicos integrados en un dispositivo de mano o tablet. Para obtener los datos, miden directamente propiedades ambientales específicas, como la aceleración, la intensidad del campo geomagnético o el cambio angular. Los sensores basados en software no son dispositivos físicos, aunque imitan los sensores basados en hardware. Los sensores basados en software derivan sus datos de uno o más de los sensores basados en hardware y, a veces, se denominan sensores virtuales o sensores sintéticos. El sensor de aceleración lineal y el sensor de gravedad son ejemplos de sensores basados en software. En la tabla 1, se resumen los sensores que son compatibles con la plataforma de Android.

Tabla de sensores:
Ver en Android Developer

103.1.1 Marco de Trabajo de los Sensores

Puedes acceder a estos sensores y adquirir datos del sensor sin procesar con el marco de trabajo del sensor de Android. El marco de trabajo del sensor es parte del paquete android.hardware e incluye las siguientes clases e interfaces:

SensorManager
Puedes usar esta clase para crear una instancia del servicio del sensor. Esta clase proporciona varios métodos para acceder a sensores y escucharlos, registrar y cancelar el registro de objetos de escucha de sensores de eventos y adquirir información de orientación. También proporciona varias constantes del sensor que se usan para informar la exactitud del sensor, definir las velocidades de adquisición de datos y calibrar sensores.

Sensor
Puedes usar esta clase para crear una instancia de un sensor específico. Esta clase proporciona varios métodos que permiten determinar las capacidades de un sensor.

SensorEvent
El sistema usa esta clase para crear un objeto de evento de sensor, que proporciona información sobre un evento del sensor. Un objeto de evento de sensor incluye la siguiente información: los datos sin procesar del sensor, el tipo de sensor que generó el evento, la exactitud de los datos y la marca de tiempo del evento.

SensorEventListener
Puedes usar esta interfaz para crear dos métodos de devolución de llamada que reciben notificaciones (eventos del sensor) cuando cambian los valores del sensor o cuando cambia la exactitud del sensor.

En una aplicación típica, estas API relacionadas con sensores se usan para realizar dos tareas básicas:

Identificación de sensores y capacidades de los sensores
La identificación de sensores y capacidades de sensores en el tiempo de ejecución es útil si la aplicación tiene características que dependen de tipos o capacidades de sensores específicos. Por ejemplo, es posible que quieras identificar todos los sensores que están presentes en un dispositivo e inhabilitar las funciones de la aplicación que dependen de sensores que no están presentes. Del mismo modo, es posible que quieras identificar todos los sensores de un tipo determinado para que puedas elegir la implementación del sensor que tenga el rendimiento óptimo para tu aplicación.

Supervisión de los eventos del sensor
La supervisión de los eventos del sensor es la manera en la que adquieres los datos sin procesar del sensor. Un evento de sensor ocurre cada vez que un sensor detecta un cambio en los parámetros que está midiendo. Un evento de sensor proporciona cuatro tipos de datos: el nombre del sensor que activó el evento, la marca de tiempo del evento, la exactitud del evento y los datos sin procesar del sensor que activaron el evento.

103.1.1.1 Disponibilidad del sensor

En la tabla se resume la disponibilidad de los sensores en cada una de las plataformas. Solo se enumeran cuatro plataformas porque son las plataformas que involucraron cambios en los sensores. Los sensores enumerados como obsoletos seguirán estando disponibles en plataformas posteriores (siempre que el sensor esté presente en un dispositivo), en concordancia con la política de compatibilidad de versiones anteriores de Android.

Sensor	Android 4.0 (API nivel 14)	Android 2.3 (API nivel 9)	Android 2.2 (API nivel 8)	Android 1.5 (API nivel 3)
TYPE_ACCELEROMETER	Sí	Sí	Sí	Sí
TYPE_AMBIENT_TEMPERATURE	Sí	n/a	n/a	n/a
TYPE_GRAVITY	Sí	Sí	n/a	n/a

Sensor	Android 4.0 (API nivel 14)	Android 2.3 (API nivel 9)	Android 2.2 (API nivel 8)	Android 1.5 (API nivel 3)
TYPE_GYROSCOPE	Sí	Sí	n/a1	n/a1
TYPE_LIGHT	Sí	Sí	Sí	Sí
TYPE_LINEAR_ACCELERATION	Sí	Sí	n/a	n/a
TYPE_MAGNETIC_FIELD	Sí	Sí	Sí	Sí
TYPE_ORIENTATION	Sí2	Sí2	Sí2	Sí
TYPE_PRESSURE	Sí	Sí	n/a1	n/a1
TYPE_PROXIMITY	Sí	Sí	Sí	Sí
TYPE_RELATIVE_HUMIDITY	Sí	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Sí	Sí	n/a	n/a
TYPE_TEMPERATURE	Sí2	Sí	Sí	Sí

103.1.1.2 Identificación de sensores y capacidades de sensores

Para identificar los sensores que están en un dispositivo, primero debes obtener una referencia al servicio del sensor. Para ello, crea una instancia de la clase SensorManager llamando al método getSystemService() y pasando el argumento SENSOR_SERVICE. Por ejemplo:

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

Luego, puedes obtener una lista de cada sensor en un dispositivo llamando al método getSensorList() y utilizando la constante TYPE_ALL. Por ejemplo:

```
val deviceSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_ALL)
```

Si quieres enumerar todos los sensores de un tipo determinado, puedes usar otra constante en lugar de TYPE_ALL, como TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION o TYPE_GRAVITY.

También puedes determinar si un tipo específico de sensor existe en un dispositivo mediante el método getDefaultSensor() y pasar la constante de tipo para un sensor específico. Si un dispositivo tiene más de un sensor de un tipo determinado, uno de los sensores se debe designar como el sensor predeterminado. Si no existe un sensor predeterminado para un tipo de sensor dado, la llamada al método mostrará un

valor nulo, lo que significa que el dispositivo no tiene ese tipo de sensor. Por ejemplo, en el siguiente código, se comprueba si hay un magnetómetro en un dispositivo:

```
    private lateinit var sensorManager: SensorManager
    ...
    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
        // Success! There's a magnetometer.
    } else {
        // Failure! No magnetometer.
    }
```

Además de enumerar los sensores que están en un dispositivo, puedes usar los métodos públicos de la clase Sensor para determinar las capacidades y los atributos de los sensores individuales. Esto es útil si quieres que tu aplicación se comporte de manera diferente según los sensores o las capacidades de los sensores disponibles en un dispositivo. Por ejemplo, puedes usar los métodos getResolution() y getMaximumRange() para obtener la resolución de un sensor y el rango máximo de medición. También puedes usar el método getPower() para obtener los requisitos de energía de un sensor.

Dos de los métodos públicos son particularmente útiles si quieres optimizar tu aplicación para diferentes sensores del fabricante o diferentes versiones de un sensor. Por ejemplo, si tu aplicación necesita supervisar gestos del usuario, como inclinar y agitar, puedes crear un conjunto de reglas y optimizaciones para dispositivos más nuevos, que tengan un sensor de gravedad específico, y otro conjunto de reglas de filtrado de datos y optimizaciones para dispositivos que no tengan sensor de gravedad y solo tengan un acelerómetro. En el siguiente ejemplo de código, se muestra cómo puedes usar los métodos getVendor() y getVersion() para hacer esto. En esta muestra, buscamos un sensor de gravedad que enumere a Google LLC como proveedor y que tenga el número de versión 3. Si ese sensor en particular no está presente en el dispositivo, intentamos usar el acelerómetro.

```
    private lateinit var sensorManager: SensorManager
    private var mSensor: Sensor? = null

    ...

    sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

    if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null) {
        val gravSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_GRAVITY)
        // Use the version 3 gravity sensor.
        mSensor = gravSensors.firstOrNull { it.vendor.contains("Google LLC") &&
it.version == 3 }
    }
    if (mSensor == null) {
        // Use the accelerometer.
        mSensor = if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null)
{
            sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
        } else {
            // Sorry, there are no accelerometers on your device.
            // You can't play this game.
            null
        }
    }
```

Otro método útil es el método getMinDelay(), que muestra el intervalo de tiempo mínimo (en microsegundos) que un sensor puede usar para detectar datos. Cualquier sensor que muestre un valor

distinto de cero para el método getMinDelay() es un sensor de transmisión. Los sensores de transmisión detectan datos a intervalos regulares y se introdujeron en Android 2.3 (API nivel 9). Si un sensor muestra cero cuando llamas al método getMinDelay(), significa que el sensor no es un sensor de transmisión, porque informa datos solo cuando hay un cambio en los parámetros que está detectando.

El método getMinDelay() es útil porque te permite determinar la velocidad máxima con la que un sensor puede adquirir datos. Si ciertas características de la aplicación requieren altas velocidades de adquisición de datos o un sensor de transmisión, puedes usar este método para determinar si un sensor cumple con esos requisitos y luego habilitar o inhabilitar las funciones relevantes en tu aplicación según corresponda.

103.1.1.3 Supervisar los eventos del sensor

Para supervisar los **datos sin procesar** del sensor, debes implementar dos métodos de devolución de llamada expuestos a través de la interfaz SensorEventListener: `onAccuracyChanged()` y `onSensorChanged()`. El sistema Android llama a estos métodos siempre que ocurre lo siguiente:

La exactitud de un sensor cambia.
En este caso, el sistema invoca el método `onAccuracyChanged()`, que proporciona una referencia al objeto Sensor que cambió y la nueva exactitud del sensor. La exactitud está representada por una de las cuatro constantes de estado: `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH` o `SENSOR_STATUS_UNRELIABLE`.

Un sensor informa un valor nuevo.**
En este caso, el sistema invoca el método `onSensorChanged()`, que proporciona un objeto **SensorEvent. Un objeto SensorEvent contiene información sobre los nuevos datos del sensor, por ejemplo, la exactitud de los datos, el sensor que generó los datos, la marca de tiempo en la que se generaron los datos y los nuevos datos que registró el sensor.

En el siguiente código, se muestra cómo usar el método onSensorChanged() para supervisar los datos del sensor de luz. En este ejemplo, se muestran los datos sin procesar del sensor en un TextView que se define en el archivo main.xml como sensor_data.

```
    class SensorActivity : Activity(), SensorEventListener {
        private lateinit var sensorManager: SensorManager
        private var mLight: Sensor? = null

        public override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.main)

            sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
            mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
        }

        override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
            // Do something here if sensor accuracy changes.
        }

        override fun onSensorChanged(event: SensorEvent) {
            // The light sensor returns a single value.
            // Many sensors return 3 values, one for each axis.
            val lux = event.values[0]
            // Do something with this sensor value.
        }

        override fun onResume() {
```

```
            super.onResume()
            mLight?.also { light ->
                sensorManager.registerListener(this, light,
SensorManager.SENSOR_DELAY_NORMAL)
            }
        }

        override fun onPause() {
            super.onPause()
            sensorManager.unregisterListener(this)
        }
    }
```

[TODO REDUCIR] En este ejemplo, el retraso de los datos predeterminado (SENSOR_DELAY_NORMAL) se especifica cuando se invoca el método registerListener(). El retraso de los datos (o la tasa de muestreo) controla el intervalo en el que los eventos del sensor se envían a la aplicación a través del método de devolución de llamada onSensorChanged(). El retraso de datos predeterminado es adecuado para supervisar los cambios típicos de orientación de la pantalla y utiliza un retraso de 200,000 microsegundos. Puedes especificar otros retrasos de datos, como SENSOR_DELAY_GAME (retraso de 20,000 microsegundos), SENSOR_DELAY_UI (retraso de 60,000 microsegundos) o SENSOR_DELAY_FASTEST (retraso de 0 microsegundos). A partir de Android 3.0 (API nivel 11), también puedes especificar el retraso como un valor absoluto (en microsegundos).

El retraso que especificas es solo un retraso sugerido. El sistema Android y otras aplicaciones pueden modificar este retraso. Como práctica recomendada, debes evitar especificar del retraso más largo posible, porque el sistema en general usa un retraso menor que el que especificas (es decir, debes elegir la tasa de muestreo más baja que cumpla con las necesidades de tu aplicación). El uso de un retraso mayor impone una carga menor en el procesador y, por lo tanto, usa menos alimentación.

No existe un método público para determinar la velocidad a la que el marco de trabajo del sensor envía eventos del sensor a tu aplicación; sin embargo, puedes usar las marcas de tiempo asociadas con cada evento del sensor para calcular la tasa de muestreo en varios eventos. No es necesario cambiar la tasa de muestreo (retraso) una vez definida. Si por algún motivo necesitas cambiar el retraso, deberás cancelar el registro y volver a registrar el objeto de escucha.

También es importante tener en cuenta que este ejemplo utiliza los métodos de devolución de llamada onResume() y onPause() para registrar el evento y cancelar el registro del objeto de escucha del sensor. Como práctica recomendada, siempre debes desactivar los sensores que no necesitas, en especial, cuando la actividad está en pausa. De lo contrario, es posible que la batería se agote en unas horas, ya que algunos sensores tienen requisitos de alimentación intensa y pueden agotar la batería con rapidez. El sistema no inhabilitará los sensores automáticamente cuando la pantalla se apague.

103.1.1.4 Controlar diferentes configuraciones del sensor

103.1.1.4.1 DETECTA LOS SENSORES EN EL TIEMPO DE EJECUCIÓN

Si tu aplicación usa un tipo específico de sensor, pero no depende de él, puedes usar el marco de trabajo del sensor para detectar el sensor en el tiempo de ejecución y luego inhabilitar o habilitar las características de la aplicación según corresponda. Por ejemplo, una aplicación de navegación podría usar el sensor de temperatura, el sensor de presión, el sensor GPS y el sensor del campo geomagnético para mostrar la temperatura, la presión barométrica, la ubicación y la dirección de la brújula. Si un dispositivo no tiene sensor de presión, puedes usar el marco de trabajo del sensor para detectar la ausencia del sensor de presión en el tiempo de ejecución y luego inhabilitar la parte de la IU de la

aplicación que muestra la presión. Por ejemplo, en el siguiente código, se comprueba si hay un sensor de presión en un dispositivo:

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

if (sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null) {
    // Success! There's a pressure sensor.
} else {
    // Failure! No pressure sensor.
}
```

103.1.1.4.2 USA LOS FILTROS DE GOOGLE PLAY PARA ORIENTAR CONFIGURACIONES ESPECÍFICAS DE SENSORES

Si estás publicando tu aplicación en Google Play, puedes usar el elemento en el archivo de manifiesto para filtrar la aplicación en los dispositivos que no tienen la configuración de sensor adecuada para tu aplicación. El elemento tiene varios descriptores de hardware que te permiten filtrar aplicaciones según la presencia de sensores específicos. Algunos de los sensores que puedes incluir son los siguientes: acelerómetro, barómetro, brújula (campo geomagnético), giroscopio, luz y proximidad. La siguiente es una entrada de manifiesto de ejemplo que filtra las apps que no tienen un acelerómetro:

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
              android:required="true" />
```

Si agregas este elemento y descriptor al manifiesto de tu aplicación, los usuarios verán tu aplicación en Google Play solo si el dispositivo tiene un acelerómetro.

Debes establecer el descriptor en android:required="true" solo si tu aplicación se basa completamente en un sensor específico. Si tu aplicación utiliza un sensor para alguna funcionalidad, pero aún se ejecuta sin el sensor, debes incluir el sensor en el elemento , pero establecer el descriptor en android:required="false". Esto ayuda a garantizar que los dispositivos puedan instalar la app incluso si no tienen ese sensor en particular. Esta es también una práctica recomendada de administración de proyectos que te ayuda a hacer un seguimiento de las funciones que usa tu app. Ten en cuenta que, si la aplicación usa un sensor en particular, pero aún se ejecuta sin el sensor, debes detectar el sensor en el tiempo de ejecución y habilitar o inhabilitar las características de la aplicación según corresponda.

103.1.1.5 Sistema de coordenadas de sensores

Seguir en AD

103.1.1.6 Prácticas recomendadas para usar e sensores

Seguir en AD

103.2 Apéndice

Enlaces:
* Documentación en Android Developer
* https://info448-s17.github.io/lecture-notes/sensors.html *
https://guides.codepath.com/android/Listening-to-Sensors-using-SensorManager * Ejemplos de codigo

en github de sensores
* Ejemplo de sensor de proximidad

Versión 0.5 12-12-23
Version 0.8 7-1-24

¿Fue útil esta página?

☺ ☹