4.10. Сенсоры. Обнаружение датчиков

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 4.10. Сенсоры. Обнаружение датчиков

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:57

Оглавление

1. Использование датчиков в мобильных устройствах

Каждое современное мобильное устройство оснащено датчиками, определяющими физическое состояние самого устройства. Операционная система Android с помощью датчиков реагирует на пользовательское управление (сенсорный экран); определяет критические состояния устройства и вовремя принимает меры (если есть такая функциональность) или сигнализирует пользователю о необходимости их принятия (перегрев, повышенный расход заряда батареи); управляет отображением контента (поворот экрана, наклоны в игровом процессе); запускает приложения (счётчик шагов, пульс, активность человека).

Сенсоры по реализации делятся на два типа:

- аппаратные (hardware-based), реализованные как составляющая единица аппаратного обеспечения устройства. Эти датчики получают данные с отслеживаемого параметра путём непосредственного измерения величин: температура воздуха, влажность, магнитное поле;
- программные (software-based), использующие данные с аппаратных датчиков и интегрирующие их в одно измерение: сила тяжести, гравитация, ускорение, перемещение устройства в пространстве. Такие датчики называют *виртуальными*.

Все датчики, поддерживаемые системой Android, можно условно разделить на

три категории

- **датчики окружающей среды.** Аппаратные сенсоры для измерения температуры, влажности, давления, освещённости и других параметров окружающей атмосферы;
- **датчики положения:** Аппаратные датчики: магнитометры, датчики положения устройства;
- **датчики движения.** Виртуальные датчики, читающие данные с физических датчиков, размещённых по трём осям направлений: гироскопы, акселерометры, сенсоры центра тяжести.

Набор датчиков зависит от конкретного устройства: даже на двух одинаковых моделях может оказаться различный набор датчиков. При этом на устройстве могут как отсутствовать некоторые датчики, так и присутствовать несколько одинаковых сенсоров с (возможно) отличающимися характеристиками. Следует учитывать, что программное управление датчиком так же зависит от используемого API: на устройстве может присутствовать некоторый сенсор, но если платформа его не поддерживает, то он доступен не будет. Узнать возможность программной работы с сенсорами в зависимости от используемой версии операционной системы можно в списке доступных датчиков.

Android в составе пакета `android.hardware` располагает собственным набором классов обработки данных, поступающих с датчиков. Sensor Framework позволяет

- опросить устройство по факту наличия датчиков и считать их характеристики;
- собрать сведения о всех установленных на устройстве датчиках;
- получить данные с датчика;
- создать слушателя изменений показателей датчика.

При отсутствии на устройстве датчика, используемого приложением, данное приложение не должно устанавливаться на устройство. В параметрах Google Play имеется функционал, позволяющий скрывать в маркете приложение, работающее с датчиком, отсутствующим на устройстве. Для этого датчики регистрируются в файле манифеста:

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

Приложения, имеющие данное ограничение, не будут включаться в список поиска GooglePlay, выполняемого на устройстве без геомагнитного датчика.

Свойство `android:required` позволяет регулировать видимость приложения в зависимости от его работы с указанным датчиком:

- значение `android:required ="true"` устанавливается для приложений, которые без указанного датчика работать не смогут. Такие приложения на устройство установлены не будут;
- значение `android:required ="false"` устанавливается для приложений, в которых датчик в случае его отсутствия на устройстве программно отключается. Такое приложение будет установлено на устройство, но функционал работы с отсутствующим датчиком реализован не будет.

Полный список имён датчиков и их описание содержится в документации разработчиков.

1.1. Создание объекта сенсора

Взаимодействие программного обеспечения с физическими и виртуальными датчиками регулируется набором классов, входящих в отдельный фреймворк.

SensorManager

Класс, открывающий доступ к сенсорам устройства через создание объекта из системной службы `SENSOR_SERVICE`.

```
val sm = getSystemService(SENSOR_SERVICE) as SensorManager
```

Объект `SensorManager`позволяет

- 1. Получить список установленных датчиков с их параметрами, вызовом метода getSensorList(typeSensor: Int);

```
sm.getSensorList(Sensor.TYPE_ALL)
```

Во входном параметре передаётся константа, определяющая тип сенсора. Начально значение константы было целочисленным, но с Android версии API 20.0 в набор были введены текстовые константы. Оба вида констант обращаются к одному и тому же сенсору, использование определённой константы зависит от минимальной версии операционной системы, на которую ориентируется приложение. Описание методик расчёта величин, получаемых с датчиков, размещено в документации по использованию свойства SensorEvent.values. Сами типы датчиков являются константами класса Sensor

Физическое положение устройства

STRING_TYPE_ACCELEROMETER	"android.sensor.accelerometer"	строковая константа для датчика акселерометра. Распознаётся системой Android 5.0 и выше
TYPE_ACCELEROMETER	1	целочисленная константа для датчика акселерометра. Распознаётся более ранними версиями операционной системы Android (от API 3.0 и выше)
STRING_TYPE_ACCELEROMETER_UNCALIBRATED TYPE_ACCELEROMETER_UNCALIBRATED	"android.sensor.accelerometer" 35	константы для акселерометра без проведения калибровки, введены с API 26.0
STRING_TYPE_ROTATION_VECTOR TYPE_ROTATION_VECTOR	"android.sensor.rotation_vector" 11	датчик вектора вращения. Ориентирует устройство в пространстве в декатровой системе координат. показывает угол отклонения по каждой из трёх осей координат.
STRING_TYPE_GAME_ROTATION_VECTOR TYPE_GAME_ROTATION_VECTOR	"android.sensor.game_rotation_vector" 15	датчик вектора вращения. Не учитывает геомагнитное поле, то есть не привязан к сторонам света
STRING_TYPE_GEOMAGNETIC_ROTATION_VECTOR TYPE_GEOMAGNETIC_ROTATION_VECTOR	"android.sensor.geomagnetic_rotation_vector" 20	датчик вектора вращения с привязкой к сторонам света.
STRING_TYPE_GRAVITY TYPE_GRAVITY	"android.sensor.gravity" 9	датчик силы притяжения.
STRING_TYPE_GYROSCOPE STRING_TYPE_GYROSCOPE_UNCALIBRATED TYPE_GYROSCOPE TYPE_GYROSCOPE_UNCALIBRATED	"android.sensor.gyroscope" "android.sensor.gyroscope_uncalibrated" 4 16	гироскоп, определяет колебательные движения устройства. Если разрешена погрешность измерения, то используется некалиброванный маятник
STRING_TYPE_HINGE_ANGLE TYPE_HINGE_ANGLE	"android.sensor.hinge_angle" 36	измерение угла поворота устройства.
STRING_TYPE_MOTION_DETECT TYPE_MOTION_DETECT	"android.sensor.motion_detect" 30	датчик движения

STRING_TYPE_PROXIMITY TYPE_PROXIMITY	"android.sensor.proximity" 8	датчик сближения. Используется в сигнализациях при приближении к заданной локации и выводит устройство из состояния сна . Некоторые датчики приближения поддерживают только двоичное измерение ближнего или дальнего расстояния. В этом случае датчик должен сообщить свое максимальное и минимальное значение при удалении и приближении соответственно.
STRING_TYPE_SIGNIFICANT_MOTION TYPE_SIGNIFICANT_MOTION	"android.sensor.significant_motion" 17	датчик "активного" движения. Регистрирует только значительные колебания устройства в пространстве. Выводит устройство из состояния сна .
STRING_TYPE_STATIONARY_DETECT TYPE_STATIONARY_DETECT	"android.sensor.stationary_detect" 29	датчик режима покоя. Срабатывает, если устройство более 5 секунд не используется.
STRING_TYPE_STEP_COUNTER TYPE_STEP_COUNTER	"android.sensor.step_counter" 19	счётчик шагов. Определяет количество колебаний от момента последней перезагрузки устройства. Возвращает значение типа `float`, но без дробной части. Сенсор не работает при дезактивации приложения, по этому рекомендуется использование датчика в фоновом потоке. Для регистрации датчика требуется разрешение в манифесте *android.permission.ACTIVITY_ RECOGNITION*
STRING_TYPE_STEP_DETECTOR TYPE_STEP_DETECTOR	"android.sensor.step_detector" 18	датчик одиночного шага. Определяет ровно один шаг, не ведёт подсчёта общего количества шагов. Для регистрации датчика требуется разрешение в манифесте *android.permission.ACTIVITY_ RECOGNITION*
STRING_TYPE_LINEAR_ACCELERATION TYPE_LINEAR_ACCELERATION	"android.sensor.linear_acceleration" 10	датчик ускорения при прямолинейном движении без вращений
Окружающая среда		
STRING_TYPE_AMBIENT_TEMPERATURE	"android.sensor.ambient_temperature"	строковая константа для датчика температуры. Распознаётся системой Android 5.0 и выше
TYPE_AMBIENT_TEMPERATURE	13	константа для датчика температуры. Введена с API 14.0
STRING_TYPE_LIGHT TYPE_LIGHT	"android.sensor.light" 5	датчик освещённости окружающей среды
STRING_TYPE_MAGNETIC_FIELD STRING_TYPE_MAGNETIC_FIELD_UNCALIBRATED TYPE_MAGNETIC_FIELD TYPE_MAGNETIC_FIELD_UNCALIBRATED	"android.sensor.magnetic_field" "android.sensor.magnetic_field_uncalibrated" 2 14	датчик уровня магнитного поля. В случае некалиброванного датчика магнитное поле твёрдых металлов не определяется и используются заводские показания, записанные при сборке устройства.
STRING_TYPE_PRESSURE TYPE_PRESSURE	"android.sensor.pressure" 6	датчик давления в окружающей среде
STRING_TYPE_RELATIVE_HUMIDITY TYPE_RELATIVE_HUMIDITY	"android.sensor.relative_humidity" 12	датчик уровня относительной влажности воздуха
Датчики состояния человека		
STRING_TYPE_LOW_LATENCY_OFFBODY_DETECT TYPE_LOW_LATENCY_OFFBODY_DETECT	""android.sensor.low_latency_offbody_detect" 34	датчик удаления устройства от тела человека. Например, снятие часов или фитнес - браслета с руки.
STRING_TYPE_HEART_BEAT TYPE_HEART_BEAT	"android.sensor.heart_beat" 31	датчик пульса. Считает частоту сердечных сокращений.

STRING_TYPE_HEART_RATE TYPE_HEART_RATE	"android.sensor.heart_rate" 21	монитор пульса. Периодическое измерение частоты сердечных сокращений. Для чтения показаний с этого датчика приложение должно иметь разрешение в манифесте *android.permission.BODY_SENSORS*.

Метод возвращает массив объектов типа `Sensor`, помещённых в список `MutableList<Sensor>`. Для получения списка сенсоров всех типов во входном параметре передаётся тип Sensor.TYPE_ALL или константа `-1`.

```
sm.getSensorList(Sensor.TYPE_ALL).forEach { Log.d("info_sensor", it.name + " " + it.power + " " + it.type) }
```

У элементов списка можно прочитать только некоторые поля (как в примере выше), тогда в протокол логирования выведутся только наименование, мощность и тип (целочисленное значение) всех датчиков, установленных на устройстве:

```
K6DS3TR Acceleration Sensor 0.25 1
SAMSUNG Step Counter Sensor 0.3 19
SContext 0.001 65586
Rear Ambient Light 1.0 65577
Orientation Sensor 6.0 3
 и т.д.
```

Можно просмотреть все сведения об объекте:

```
 Log.d("info_sensor", ""+sm.getSensorList(Sensor.TYPE_GRAVITY))
```

В этом случае просматривается полное описание сенсора.

```
{Sensor name="Gravity Sensor", vendor="Samsung Electronics", version=3, type=9, maxRange=19.6133, resolution=5.9604645E-8,
power=6.0, minDelay=10000}
```

- 2. Получить сведения о датчиках, используемых по умолчанию, можно методом getDefaultSensor(Int), возвращающим объект Sensor

```
var g =  sm.getDefaultSensor(1)
Log.d("info_sensor", ""+ g)

Результат:
{Sensor name="K6DS3TR Acceleration Sensor", vendor="STM", version=1, type=1, maxRange=78.4532, resolution=0.0023942017,
power=0.25, minDelay=2000}
```

Датчик, используемый по умолчанию, может оказаться виртуальным и показывать усреднённые данные с нескольких сенсоров, из которых он состоит. По этой причине данные с датчиков по умолчанию не могут считаться точными, однако бывают случаи, когда такое выравнивание необходимо для корректной работы приложения: при ходьбе, при попадании на устройство прямых солнечных лучей и т.п.

Так же датчика определённого типа может не оказаться в устройстве, тогда метод возвратит `null`. Чтобы избежать исключения обращения к несуществующему объекту, необходимо получение датчика размещать в условном операторе

```
var g =  sm.getDefaultSensor(Sensor.TYPE_LIGHT);
        if(g!= null) Log.d("info_sensor", ""+ g)
        else Log.d("info_sensor", "no sensors")
```

- 3. Получить список динамических датчиков (на сегодняшний день в смартфонах и планшетах не используются) можно методом getDynamicSensorList(type: Int). Метод добавлен в класс с версии Android 24 , что обосновано широким распространением других "умных" устройств помимо смартфона, которые в основном ориентированы на взаимодействие с сенсорами (AndroidThings). Динамические датчики - это сенсоры, которые могут быть добавлены в устройство или изъяты из него аппаратно самим пользователем. Для взаимодействия с динамическими датчиками предусмотрен интерфейс DynamicSensorCallback

При отсутствии на устройстве подключаемых сенсоров запрос списка динамических датчиков возвращает пустой список:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            Log.d("info_sensor_dyn",""+sm.getDynamicSensorList(Sensor.TYPE_ALL))
        }

Результат:
[]
```

- 4. Характеристики датчика можно получить при помощи методов класса Sensor

Наиболее употребимые методы представлены в таблице

метод	описание	вовращаемое значение
getFifoMaxEventCount() getFifoReservedEventCount()	количество событий в массиве показателей. Максимально возможное / минимально возможное значение	целое число в типе данных Int
getMaxDelay() getMinDelay()	максимальное/минимальное время задержки между двумя измерениями (в микросекундах). Определяется только для датчиков, снимающих более одного показателя в одной серии. Может быть использован для определения времени заполнения массива данных показателя	положительное целое число. отрицательные значения указывают на неподдерживаемый датчик, нулевое значение - на датчик с одиночным показателем. Неположительное значение не является информативным и не используется
getPower()	потребляемая мощность датчика во время работы (в миллиамперах). Используется при расчёте расхода заряда батареи	положительное целое число
getReportingMode()	режим работы датчика: постоянные измерения с определённой скоростью (REPORTING_MODE_CONTINUOUS); измерение показателя при смене состояния (REPORTING_MODE_ON_CHANGE); одноразовое измерение, отключающее датчик сразу после срабатывания (REPORTING_MODE_ONE_SHOT) измерения по правилу, описанному в слушателе датчика (REPORTING_MODE_SPECIAL_TRIGGER)	целое число из набора {0, 1, 2, 3}, соответствующее каждой константе режимов {REPORTING_MODE_CONTINUOUS, REPORTING_MODE_ON_CHANGE, REPORTING_MODE_ONE_SHOT, REPORTING_MODE_SPECIAL_TRIGGER}
getType() getStringType()	тип датчика	целое число, соответствующее типу строка с типом данных, если система поддерживает строковое именование
getVendor()	производитель сенсора	строковое описание фирмы-производителя
getVersion()	версия установленного модуля датчика	целочисленное значение
isDirectChannelTypeSupported(sharedMemType: Int)	проверка на возможность прямого подключения к датчику	логическое значение
isDynamicSensor()	проверка на динамичность сенсора	логическое значение
isWakeUpSensor()	проверка на возможность вывода устройства из спящего режима. Использование датчика с возможностью пробуждения устройства может привести к 10-кратному повышению расхода заряда батареи.	логическое значение

1.2. Упражнение 4.10.1 Получение списка датчиков

1. Создайте новый проект, назовите его CheckedSensors. Замените корневую разметку на линейную в вертикальной ориентации, добавьте под текстовым полем кнопку Button и список ListView.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

   android:orientation="vertical"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textSize="15sp"
        android:text="@string/discription" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Вывести список датчиков"
        android:onClick="readSensor" />

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/sensorlist"/>
</LinearLayout>
```

2. Создайте в файле строковых ресурсов элемент с именем discription и запишите в него назначение приложения

```
<string name="discription">Система индикации датчиков устройства\nНажмите кнопку, чтобы получить список установленных
сенсоров</string>
```

3. В коде объявите переменные для SensorManager и ListView

```
lateinit var sm: SensorManager
lateinit var list: ListView
```

и в методе onCreate() создайте менеджера датчиков

```
sm = getSystemService(SENSOR_SERVICE) as SensorManager
```

4. Напишите метод `readSensor(View)` для считывания сведений о датчиках и вывода их наименований в список ListView, установленный в качестве обработчика на кнопку.

   4.1 получите список сенсоров

```
val sensorList = sm.getSensorList(-1)
```

4.2 в отдельный список запишите имена найденных устройств

```
var sensorName = ArrayList<String>()
sensorList.forEach { sensorName.add(it.name) }
```

4.3 найдите ListView по его id, создайте стандартный адаптер для списка с данными из списка имён датчиков и установите адаптер на ListView

```
list = findViewById(R.id.sensorlist)
var adapterName = ArrayAdapter(applicationContext, android.R.layout.simple_list_item_1, sensorName)
list.adapter = adapterName
```

5. Определите обработчик нажатия по пункту списка так, чтобы у выбранного датчика считывались его параметры

```
list.onItemClickListener = AdapterView.OnItemClickListener { adapt, view, index, id ->
    val rSensor = sensorList.get(index)
    var sensor_info = "name: " + rSensor.name +
            "\ntype: " + rSensor.type +
            "\npower: " + rSensor.power +
            "\nversion: " + rSensor.version +
            "\nvendor: " + rSensor.vendor +
            "\nresolution: " + rSensor.resolution +
          "\nDynamic: " + rSensor.isDynamicSensor +
           "\nWakeUp: " + rSensor.isWakeUpSensor
}
```

и выведите эту строку в Toast

```
Toast.makeText(applicationContext,sensor_info,Toast.LENGTH_LONG).show()
```

Поскольку некоторые характеристики (в данном случае проверка на динамический сенсор и возможность выводить устройство из режима сна) были добавлены в SensorFramework для более поздних версий API, то среда программирования предложит аннотировать метод для определения версии платформы, на которой эти методы будут работать

```
@RequiresApi(Build.VERSION_CODES.N)
```

6. Поскольку отображение информации в Toast не совсем удобна в виду его непродолжительного отображения, выполните вывод сведений о датчике на новую активность.

   6.1 Создайте в проекте новую активность `SensorInfo` с корневой разметкой FrameLayout и единственным экранным элементов TextView

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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"
    tools:context=".SensorInfo">
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:textSize="20dp"
    android:id="@+id/info"/>
</FrameLayout>
```

6.2 В методе readSensor(View) класса MainActivity создайте намерение и передайте через него в класс SensorInfo строку с параметрами выбранного датчика

```
var intent = Intent(this@MainActivity, SensorInfo::class.java)
intent.putExtra("sd", sensor_info)
startActivity(intent)
```

6.3 В классе SensorInfo получите данные о сенсоре и выведите их в текстовое поле

```
val intent = getIntent()
(findViewById<TextView>(R.id.info)).text = intent.getStringExtra("sd")
```

Проверьте правильность выполнения упражнения сравнением с данным приложением

1.3. Чтение данных с датчика

Для чтения данных с датчика необходимо для него зарегистрировать слушателя методами registerListener(listener: SensorEventListener!, sensor: Sensor!, samplingPeriodUs: Int) или registerListener(listener: SensorEventListener!, sensor: Sensor!, samplingPeriodUs: Int, maxReportLatencyUs: Int)

```
sm = getSystemService(SENSOR_SERVICE) as SensorManager
sensor = sm.getDefaultSensor(Sensor.TYPE_GYROSCOPE)

sm.registerListener(this, sensor, SensorManager.SENSOR_DELAY_UI)
```

Нужно обязательно предусмотреть отсутствие датчика на устройстве:

а) прописать в манифесте требование на наличие датчика (при отсутствии сенсора приложение, устанавливаемое из .apk-файла не будет найдено и установлено)

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

или

б) в момент регистрации слушателя сделать проверку на непустой объект сенсора

```
if(sensor !=null)
sensor.also { x -> sm.registerListener(this,x,SensorManager.SENSOR_DELAY_UI) }
        else Toast.makeText(this,"Нет датчика", Toast.LENGTH_LONG).show()
```

При регистрации слушателя передаются следующие аргументы:

- *listener:* сам слушатель в виде непустого объекта SensorEventListener;
- *sensor:* датчик, с которого читается массив значений. Все показатели поступают к слушателю в режиме реального времени. Если датчик имеет настройку вывода устройства из режима сна, то данные сразу же передаются на обработку в приложение. В противном случае данные собираются в слушателе и поступают в приложение только при пробуждении устройства. При этом каждый новый набор данных заменяет данные предыдущего измерения, что приводит к потере данных. То есть для датчиков, не выводящих устройство из режима сна необходимо в приложении предусмотреть либо вывода устройства в активный режим самим приложением (если сохранение всех считанных данных необходимо), либо отключение датчика на время деактивности устройства (если промежуточные данные не нужно учитывать), поскольку зарегистрированный датчик будет тратить заряд батареи. При регистрации слушателя методом registerListener(SensorEventListener!, Sensor!, Int, Int) проблема с потерей данных частично решается за счёт временного размещения считанных данных в очереди до момента их отправки. Время хранения данных в очереди регулируется последним аргументом функции;
- *samplingPeriodUs:* частота дискретизации - задержка данных на датчике перед отправкой в приложение. Под этот параметр в классе SensorManager определены константы

Константа	Числовой десятичный (шестнадцатеричный) эквивалент	Задержка в микросекундах
SENSOR_DELAY_NORMAL	3 (0x00000003)	200 000
SENSOR_DELAY_GAME	1 (0x00000001)	20 000
SENSOR_DELAY_UI	2 (0x00000002)	60 000
SENSOR_DELAY_FASTEST	0 (0x00000000)	0

Можно так же передать любое целое число, указывающее задержку данных в микросекундах.

Указанная частота не является абсолютным параметром. Это значит, что данные будут поступать примерно с указанной частотой, но могут быть получены как раньше, так и позже.

- *maxReportLatencyUs* максимальное время задержки данных (в микросекундах) на датчике перед отправкой в приложение. При указании этого параметра появляется возможность экономии заряда батареи в режиме сна, поскольку при хранении данных на датчике батарея устройства остаётся в режиме энергосбережения. Положительное значение параметра указывает на время задержки данных, значение 0 указывает на необходимость отправки данных сразу после их поступления.

Отменять регистрацию слушателя необходимо сразу же, как только сенсор перестаёт использоваться. В противном случае работа приложения приведёт к очень скорому расходу заряда батареи.

```
sm.unregisterListener(this)
```

Чтение датчиков выполняется от момента регистрации слушателя до снятия регистрации. Если разрегистрация отсутствует, то датчик отслеживается системой в течение активного состояния приложения.

Регистрация и отмена регистрации датчика выполняется в соответствии с жизненным циклом активности:

Регистрация	Разрегистрация
onCreate()	onDestroy()
onStart()	onStop()
onResume()	onPause()

Каждый датчик описывает данные в виде четырёх информационных объектов: имя датчика, время происхождения события, точность данных и необработанные значения показателей. Все данные передаются в виде массивов. Даже если показатель всего один, он всё-равно представляется в виде массива из одного элемента.

Для чтения данных с датчика используются методы обратного вызова из интерфейса SensorEventListener

Таких методов два:

- onSensorChanged(event: SensorEvent!). Вызывается при каждом появлении значения отслеживаемого параметра. Если при очередном считывании значение параметра не изменилось, в массив данных записывается повторное значение.

```
    lateinit  var tw: TextView
...
   override fun onSensorChanged(event: SensorEvent?) {
        tw.text = event!!.values[0].toString()
    }
```

- onAccuracyChanged(sensor: Sensor!, accuracy: Int). Вызывается только при изменении точности отслеживаемого параметра. Вторым аргументом передаётся новая точность в виде константы класса SensorManager:

SENSOR_STATUS_NO_CONTACT	-1	некорректные данные, поскольку датчик не обращается к отслеживаемому объекту
SENSOR_STATUS_UNRELIABLE	0	датчик не откалиброван или недостаточно возможностей для чтения показаний
SENSOR_STATUS_ACCURACY_LOW	1	низкая точность показаний, требуется калибровка окружающей среды
SENSOR_STATUS_ACCURACY_MEDIUM	2	средний уровень показаний. Калибровка может повысить точность, но в большинстве случаев в этом нет необходимости
SENSOR_STATUS_ACCURACY_HIGH	3	максимальная точность показаний

```
override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
        val tw2 = findViewById<TextView>(R.id.temperature)
when(accuracy){
    SensorManager.SENSOR_STATUS_ACCURACY_HIGH -> tw2.text = "good"
    SensorManager.SENSOR_STATUS_ACCURACY_LOW -> tw2.text = "bad"
    SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM -> tw2.text = "average"
    SensorManager.SENSOR_STATUS_NO_CONTACT -> tw2.text = "very bad"
    SensorManager.SENSOR_STATUS_UNRELIABLE -> tw2.text = "oh!"
}
    }
```

Начать тур для пользователя на этой странице