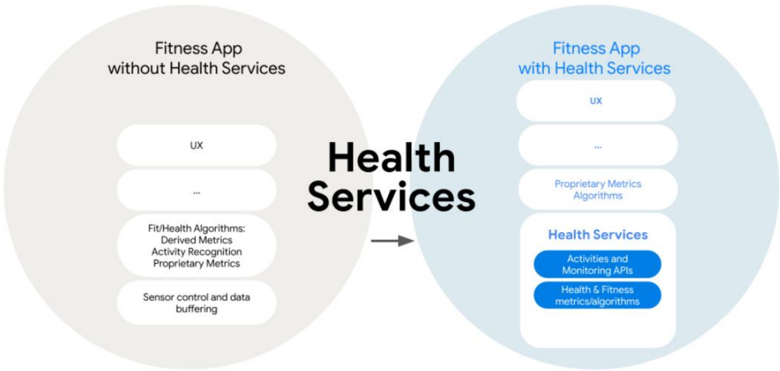
适配指导 | Wear Health Services

Google 在今年四月最新发布的 Wear OS 库中，推出了一项新功能：Health Services。通过 Health Services 即可利用系统能力，简化应用开发者的重复工作，使其更专注于开发独特功能来提升用户体验。

三星和 Google 在技术创新方面一直保持着深度的合作，三星最新一代 Galaxy Watch4 系列将首次搭载和 Google 深度定制的 Wear OS 系统，兼容 Wear OS 应用。包括这次的新功能 Health Services，也是三星和 Google 从软硬件共同打造。我们希望在三星设备上有更多中国应用使用 Health Services 功能，扩大应用的适配范围，为用户带来更可靠、更省电的使用体验。

1. 什么是 Health Services

Health Services 是各种传感器和相关算法的中介，为 APP 提供活动、锻炼和健康相关的高质量数据。Health Services 会自动配置所有健身相关的传感器，收集传感器数据，并计算心率、距离、卡路里、海拔、楼层、速度、配速等指标。 APP 可以直接从 Health Services 得到这些数据，简化每个 APP 自己连接传感器、配置传感器、接收原始传感器数据、使用自己的算法来获取信息的繁琐过程。



- Health Services 使用优化电源效率的传感器配置，节省电量。
- 统一计算方法，确保同一设备上所有 APP 的数据一致性。
- APP 可以利用在系统本地的强大算法，简化工作量。

2. 如何使用 Health Services

Google 提供了 Jetpack 软件包：androidx.health:health-services-client:1.0.0-alpha01，为开发者提供了 Health Services 库。目前，该软件包支持 Wear OS 3 模拟器和即将推出的设备，未来也会推广到更多平台。

2.1 声明依赖项

在应用或模块的 build.gradle 文件中，将 Google Maven 代码库添加到项目中，添加依赖项：

```
dependencies {
  implementation 'androidx.health:health-services-client:1.0.0-alpha01'
  // For Kotlin, this library helps bridge between Futures and coroutines.
  implementation "com.google.guava:guava:30.1.1-android"
  implementation "androidx.concurrent:concurrent-futures-ktx:1.1.0"
}
```

为了与 Health Services 进行交互，在应用 AndroidManifest.xml 文件中添加以下内

容：

```
<queries>
    <package android:name="com.google.android.wearable.healthservices" />
</queries>
```

2.2 按场景提供 API

根据使用场景，分为被动追踪和主动追踪。

2.2.1 被动追踪

如果您的 APP 需要跟踪用户全天的活动，那么 PassiveMonitoringClient 是一个绝佳选择。

1. 了解设备可以跟踪哪种数据类型：

```
val healthClient = HealthServices.getClient(this /*context*/)
val passiveMonitoringClient = healthClient.passiveMonitoringClient
lifecycleScope.launchWhenCreated {
    val capabilities = passiveMonitoringClient.capabilities.await()
    supportsHeartRate =
        DataType.HEART_RATE_BPM in capabilities.supportedDataTypesPassiveMonitoring
    supportsStepsEvent =
        DataType.AGGREGATE_STEP_COUNT in capabilities.supportedDataTypesEvents
}
```

2. 两种操作方式：数据收集和事件跟踪。数据收集适用于数据更新相对不频繁且随时间分布的长期体验。事件跟踪是指设置某种触发条件，例如达到特定步数、达到特定心率。

   - **方式一：数据收集：**

   被动数据收集适用于需要后台监视 Health Services 数据的应用。 APP 可以选择接收所有数据点，或者只有与特定数据类型相关的数据点。

   需要使用 PassiveMonitoringClient 在后台接收数据更新。Health

Services 批处理更新，数据点和用户活动状态可能反映过去发生的事件，应用需要利用对象中包含的时间戳正确评估数据。

1.使用 BroadcastReceiver 接收数据更新，在 onReceive 中，使用 PassiveMonitoringUpdate.fromIntent 解析数据。

```
class BackgroundDataReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val state = PassiveMonitoringUpdate.fromIntent(intent) ?: return
        // Log data points. In a production app, store these to a database, etc.
        state.dataPoints.forEach {
            Log.i(TAG, "Data point: $it")
        }
        val latestActivity = state.userActivityInfoUpdates
            .maxByOrNull { it.stateChangeTime }
            ?.userActivityState
        Log.i(TAG, "latest activity state: $latestActivity")
    }
}
```

2.可以使用一组 DataType 注册一个 PendingIntent，用来接收数据。

```
// To register for data
val intent = Intent(this /*context*/, BackgroundDataReceiver::class.java)
val pendingIntent = PendingIntent.getBroadcast(
    this /*context*/, 1, intent, PendingIntent.FLAG_UPDATE_CURRENT
)
val dataTypes = setOf(DataType.HEART_RATE_BPM, DataType.AGGREGATE_STEP_COUNT)
lifecycleScope.launch {
   HealthServices.getClient(context)
       .passiveMonitoringClient
       .registerDataCallback(dataTypes, pendingIntent)
       .await()
}

// To unregister
lifecycleScope.launch {
   HealthServices.getClient(context)
       .passiveMonitoringClient
       .unregisterDataCallback()
       .await()
}
```

- **方式二：事件追踪**

注册事件和注册数据的流程类似，需要使用 passiveClient.registerEventCallback 进行回调注册。但有一点不同，每个事件都是单独注册，并可注册多个事件。如果需要处理多个 event，则可以为每

个 event 使用不同的 BroadcastReceiver，或者使用不同的 PendingIntent 而使用相同的 BroadcastReceiver。

1.定义触发的事件：

```
const val OVER_1M_STEPS = "com.example.passivedata.OVER_1M_STEPS"
const val FASTER_THAN_SOUND = "com.example.passivedata.FASTER_THAN_SOUND"

val millionStepsEvent by lazy {
    val condition = DataTypeCondition(
        dataType = DataType.AGGREGATE_STEP_COUNT,
        threshold = Value.ofLong(1_000_000),
        comparisonType = ComparisonType.GREATER_THAN_OR_EQUAL
    )
    Event(condition, Event.TriggerType.ONCE) // only the next occurrence
}

val speedOfSoundEvent by lazy {
    val speedOfSound = 343.0 // meters per second
    val condition = DataTypeCondition.builder()
        dataType = DataType.SPEED,
        threshold = Value.ofDouble(speedOfSound),
        comparisonType = ComparisonType.GREATER_THAN_OR_EQUAL
    )
    Event(condition, Event.TriggerType.REPEATED) // every occurrence
}
```

2.注册不同的 PendingIntent，使用相同的 BroadcastReceiver：

```
val stepsIntent = Intent(this /*context*/, BackgroundDataReceiver::class.java).apply {
    action = OVER_1M_STEPS
}
val stepsPendingIntent = PendingIntent.getBroadcast(
    this /*context*/, 1, stepsIntent, PendingIntent.FLAG_UPDATE_CURRENT
)
val speedIntent = Intent(this /*context*/, BackgroundDataReceiver::class.java).apply {
   action = FASTER_THAN_SOUND
}
val speedPendingIntent = PendingIntent.getBroadcast(
    this /*context*/, 1, speedIntent, PendingIntent.FLAG_UPDATE_CURRENT
)
lifecycleScope.launch {
    val passiveClient = HealthServices.getClient(this /*context*/)
        .passiveMonitoringClient
    passiveClient
        .registerEventCallback(millionStepsEvent, stepsPendingIntent)
        .await()
    passiveClient
        .registerEventCallback(speedOfSoundEvent, speedPendingIntent)
        .await()
}
```

2.2.2 主动追踪

- **MeasureClient**

有时，用户需要查看当前的某个指标（例如心率），而不是锻炼时或全天的指标。

在这些时候，MeasureClient 是合适的选择。APP 只需通过注册支持

的 DataType 回调来接收数据流即可，当不再需要回调时，将其取消注册。

1. 了解设备可以跟踪哪种数据类型：

```
val healthClient = HealthServices.getClient(this /*context*/)
val measureClient = healthClient.measureClient
lifecycleScope.launch {
    val capabilities = measureClient.capabilities.await()
    supportsHeartRate =
        DataType.HEART_RATE_BPM in capabilities.supportedDataTypesMeasure
}
```

2. 注册数据

注册的每个回调是用于单个数据类型。

```
val heartRateCallback = object : MeasureCallback {
    override fun onAvailabilityChanged(type: DataType, availability: Availability) {
        // Handle availability change.
    }

    override fun onData(dataPoints: List<DataPoint>) {
        // Inspect data points.
    }
}
val healthClient = HealthServices.getClient(this /*context*/)

// Register the callback.
lifecycleScope.launch {
    healthClient.measureClient
        .registerCallback(DataType.HEART_RATE_BPM, heartRateCallback)
        .await()
}

// Unregister the callback.
lifecycleScope.launch {
    healthClient.measureClient.unregisterCallback(heartRateCallback).await()
}
```

- **ExerciseClient**

ExerciseClient 是专为用于跟踪锻炼的应用而设计的，可提供多达 82 种不同

的 ExerciseType，例如步行、跑步、舞蹈和水球等。在跟踪这些锻炼时，有 50 种不同的 DataType 可供选择，具体取决于锻炼类型和设备上可用的硬件。如需开始使用，只需执行以下操作即可：在 ExerciseConfig 中指定相关信息，调用 exerciseClient.startExercise 并通过更新监听器监听进度。

1. 了解设备可以跟踪哪种数据类型：

```
val healthClient = HealthServices.getClient(this /*context*/)
val exerciseClient = healthClient.exerciseClient
lifecycleScope.launch {
    val capabilities = exerciseClient.capabilities.await()
    if (ExerciseType.RUNNING in capabilities.supportedExerciseTypes) {
        runningCapabilities =
            capabilities.getExerciseTypeCapabilities(ExerciseType.RUNNING)
    }
}
```

2. 配置锻炼跟踪

```
const val CALORIES_THRESHOLD = 250f
const val DISTANCE_THRESHOLD = 1_000f // meters

suspend fun startExercise() {
    // Types for which we want to receive metrics.
    val dataTypes = setOf(
        DataType.HEART_RATE_BPM,
        DataType.AGGREGATE_CALORIES_EXPENDED,
        DataType.AGGREGATE_DISTANCE
    )
    // Create a one-time goal.
    val calorieGoal = ExerciseGoal.createOneTimeGoal(
        condition = DataTypeCondition(
            dataType = DataType.AGGREGATE_CALORIES_EXPENDED,
            threshold = Value.ofFloat(CALORIES_THRESHOLD),
            comparisonType = ComparisonType.GREATER_THAN_OR_EQUAL
        )
    )
    // Create a milestone goal. To make a milestone for every kilometer, set the initial
    // threshold to 1km and the period to 1km.
    val distanceGoal = ExerciseGoal.createMilestone(
        condition = DataTypeCondition(
            dataType = DataType.AGGREGATE_DISTANCE,
            threshold = Value.ofDouble(DISTANCE_THRESHOLD),
            comparisonType = ComparisonType.GREATER_THAN_OR_EQUAL
        ),
        period = Value.ofDouble(DISTANCE_THRESHOLD)
    )
```

```
    val config = ExerciseConfig.builder()
        .setExerciseType(ExerciseType.RUNNING)
        .setDataTypes(dataTypes)
        .setExerciseGoals(listOf(calorieGoal, distanceGoal))
        .setAutoPauseAndResume(false)
        .build()
    HealthServices.getClient(this /*context*/)
        .exerciseClient
        .startExercise(config)
        .await()
}
```

3. 注册锻炼状态更新

```
val listener = object : ExerciseUpdateListener {
    override fun onStateUpdate(update: ExerciseUpdate) {
        // Process the latest information about the exercise.
        exerciseStatus = update.status // e.g. ACTIVE, USER_PAUSED, etc.
        activeDuration = state.activeDuration // Duration
        latestMetrics = state.latestMetrics // Map<DataType, List<DataPoint>>
        latestGoals = state.latestAchievedGoals // Set<AchievedExerciseGoal>
        latestMilestones = state.latestMilestoneMarkerSummaries // Set<MilestoneMarkerSummary>

    }
    override fun onLapSummary(lapSummary: ExerciseLapSummary) {
        // For ExerciseTypes that support laps, this is called when a lap is marked.
    }
}
val exerciseClient = HealthServices.getClient(this /*context*/).exerciseClient

// Register the listener.
lifecycleScope.launch {
    exerciseClient.setUpdateListener(listener).await()
}

// Unregister the listener.
lifecycleScope.launch {
    exerciseClient.clearUpdateListener(listener).await()
}
```

2.3 使用合成数据进行测试

Google 提供了模拟测试的方法，开发者只需要 adb 命令，就可以验证 APP 能否成功接收 Health Services 的数据更新。

1）启用合成跟踪器：

```
$ adb shell am broadcast \
-a "whs.USE_SYNTHETIC_PROVIDERS" \
com.google.android.wearable.healthservices
```

2）支持的指标：

- 心率
- 每分钟步数
- GPS 位置（使用单一默认路线）
- 活动时长
- 海拔和楼层
- 睡眠状态（睡眠/清醒）

3）开启默认的锻炼

提供如下默认的方式：

- **Walking:** whs.synthetic.user.START_WALKING
- **Running:** whs.synthetic.user.START_RUNNING
- **Hiking:** whs.synthetic.user.START_HIKING
- **Swimming:** whs.synthetic.user.START_SWIMMING
- **Running on treadmill:** whs.synthetic.user.START_RUNNING_TREADMILL

```
$ adb shell am broadcast \
-a "whs.synthetic.user.START_WALKING" \
com.google.android.wearable.healthservices
```

如上举例，表示开启模拟用户散步，平均速度： 每秒 1.4 米，心率：每分钟 120 次，位置：启用。

Activity	Heart rate	Average speed	Elevation change	Use location
Walking	120 bpm	1.4 m/sec	20.0 m/min	true
Running	170 bpm	2.3 m/sec	20.0 m/min	true
Hiking	150 bpm	1.3 m/sec	20.0 m/min	true
Swimming	150 bpm	1.6 m/sec	0.0 m/min	true
Running treadmill	160 bpm	2.3 m/sec	20.0 m/min	false

4）停止锻炼

```
$ adb shell am broadcast \
-a "whs.synthetic.user.STOP_EXERCISE" \
com.google.android.wearable.healthservices
```

5）自定义锻炼

如果需要更精确控制生成的指标，请使用操作字符串

whs.synthetic.user.START_EXERCISE 并提供以下标志的任意组合来启动自定义的锻炼：

```
$ adb shell am broadcast \
-a "whs.synthetic.user.START_EXERCISE" \
--ei exercise_options_heart_rate 90 \
--ef exercise_options_average_speed 1.2 \
--ez exercise_options_use_location true \
com.google.android.wearable.healthservices
```

6）控制睡眠状态

可以模拟用户的睡眠状态。 目前，只支持两种状态：睡眠和清醒。 要在两者之间切换，请使用以下命令。

```
$ adb shell am broadcast \
-a "whs.synthetic.user.START_SLEEPING" \
com.google.android.wearable.healthservices
```

```
$ adb shell am broadcast \
-a "whs.synthetic.user.STOP_SLEEPING" \
com.google.android.wearable.healthservices
```

3. 参考文档

Wear 3 emulator:

https://androidstudio.googleblog.com/2021/05/android-studio-arctic-fox-beta-1.html

Google 官网:

https://developer.android.google.cn/training/wearables/health-services

https://developer.android.google.cn/training/wearables/health-services/passive

https://developer.android.google.cn/training/wearables/health-services/active

Release notes:

https://developer.android.google.cn/jetpack/androidx/releases/health#health-services-client-1.0.0-alpha01

https://developer.android.google.cn/reference/kotlin/androidx/health/services/client/package-summary

参考 Demo：

https://github.com/androidx/androidx/tree/androidx-main/health/health-services-client/src/main/java/androidx/health/services/client

https://github.com/android/health-samples/

4. 联系我们

如果您在适配过程中遇到了技术问题，对 Wear Health Services 适配有任何建议或意见：

欢迎发邮件到：rdtpservice@samsung.com

邮件主题：Wear Health Services 适配+APP 名