

ExoPlayer под задачи VK: как настройка эстиматора, вьюпорта и буферов сократила задержки



О себе



Егор Баженов, старший Android-разработчик в команде VK.

Опыт разработки более 6 лет



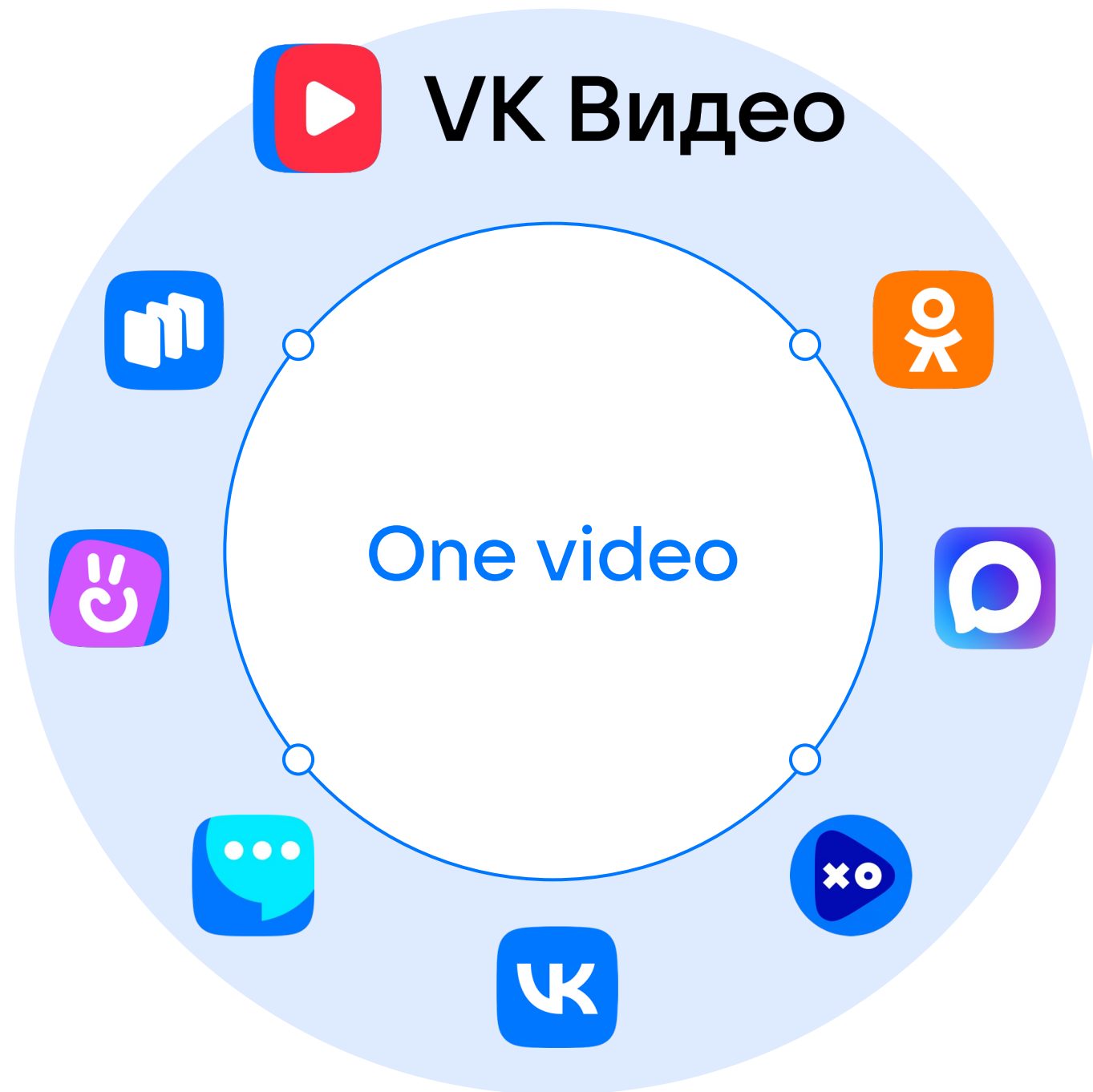
Аспирант Университета ИТМО по программе Искусственный интеллект и машинное обучение. Сотрудник лаборатории ИТМО



Преподаватель VK Education



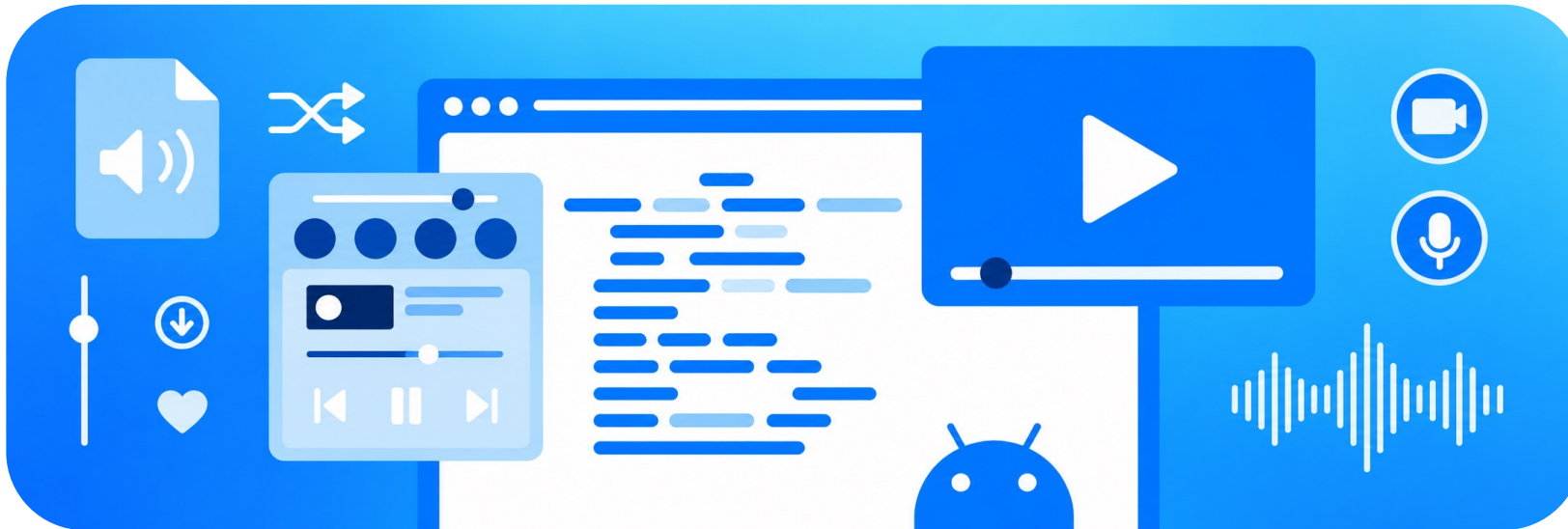
Библиотека *one video*



Технические аспекты реализации

- Основной язык — kotlin
- Базовое решение — [Exo Player \(Media 3\)](#)
- Оптимизации проигрывания
- В закрытом доступе (внутри VK)

- Покрыт единой статистикой
- Поддержка оффлайна
- Практически полное покрытие автотестами
- Почти без compose



Нестабильное воспроизведение

С чем сталкиваются пользователи



Воспроизведение на нестабильных сетях

Проблема

Зависания вливают на пользовательский UX





Цель

**Улучшить пользовательский UX
при просмотре видео при помощи
минимизации количества
и длительности зависаний**



Задачи



Собрать
информацию
о зависании



Изучить
существующие
средства ExoPlayer
для борьбы
со столлами



Реализовать
и настроить
правильные
подходы
в соответствии
со спецификой
продукта

Итоги



**Уменьшили количество
зависаний на 30%
средствами ExoPlayer'a**

Что будем
считать
замедлением?

Какие бывают столлы



Какие бывают столлы



Столл при старте

У плеера нет данных

Какие бывают столлы



Столл при старте

У плеера нет данных



Столл при проигрывании

Плеер не успевает качать данные

Какие бывают столлы



Столл при старте

У плеера нет данных



Столл при проигрывании

Плеер не успевает
качать данные



Столл при перемотке

Плеер перешел
в область, где нет
данных

Кто отвечает за сбор



```
androidx.media3.common.Player.Listener {  
  
    //...  
  
    @IntDef({STATE_IDLE, STATE_BUFFERING, STATE_READY, STATE_ENDED})  
    @interface State {}  
  
    /**  
     * Called when the value returned from getPlaybackState() changes.  
     * @param playbackState The new playback {@link State}.  
     */  
    default void onPlaybackStateChanged(@State int playbackState) {}  
  
}
```

Покрытие статистикой



```
val listener = object : Player.Listener {  
  
    var bufferingStartTimeMs: Long = -1  
  
    override fun onPlaybackStateChanged(playbackState: Int) {  
        when{  
            playbackState == STATE_BUFFERING && bufferingStartTimeMs == -1L -> {  
                bufferingStartTimeMs = System.currentTimeMillis()  
            }  
            playbackState != STATE_BUFFERING && bufferingStartTimeMs != -1L -> {  
                bufferingStartTimeMs = -1  
                //send analytics  
            }  
        }  
    }  
}
```

Покрытие статистикой

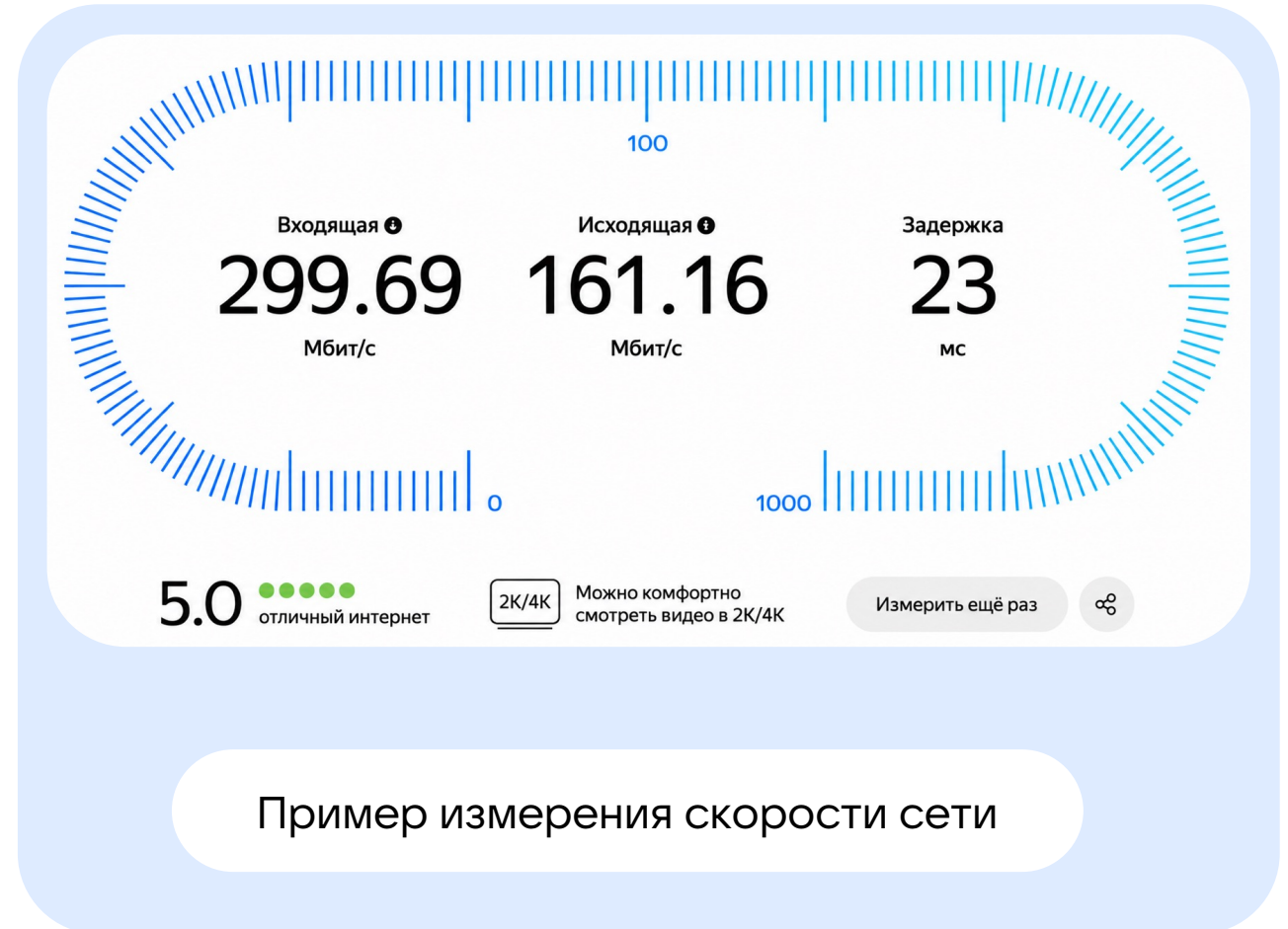


```
val listener = object : Player.Listener {  
  
    var bufferingStartTimeMs: Long = -1  
  
    override fun onPlaybackStateChanged(playbackState: Int) {  
        when{  
            playbackState == STATE_BUFFERING && bufferingStartTimeMs == -1L -> {  
                bufferingStartTimeMs = System.currentTimeMillis()  
            }  
            playbackState != STATE_BUFFERING && bufferingStartTimeMs != -1L -> {  
                bufferingStartTimeMs = -1  
                //send analytics  
            }  
        }  
    }  
}  
  
val player = ExoPlayer.Builder(context).build()  
player.addListener(listener)
```

BandwidthMeter (Estimator)

Немного определений

- **Бэндвиз (bandwidth)** — это максимальный объем данных, который может передаваться по сети или каналу связи за определенное время
- **Эстиматор (estimator)** — это правило, формула или функция, используемая для вычисления приближенного значения параметра генеральной совокупности на основе имеющихся данных. Он же BandwidthMeter



Базовая реализация

```
/**
 * Estimates bandwidth by listening to data transfers.
 *
 * The bandwidth estimate is calculated using a SlidingPercentile
 * and is updated each time a transfer ends. The initial estimate is
 * based on the current operator's network country code or the locale
 * of the user, as well as the network connection type. This can be
 * configured in the Builder.
 */

@UnstableApi
public final class DefaultBandwidthMeter implements
BandwidthMeter, TransferListener {...}
```

Базовая реализация

```
/**
 * Estimates bandwidth by listening to data transfers.
 *
 * The bandwidth estimate is calculated using a SlidingPercentile
 * and is updated each time a transfer ends. The initial estimate is
 * based on the current operator's network country code or the locale
 * of the user, as well as the network connection type. This can be
 * configured in the Builder.
 */

@UnstableApi
public final class DefaultBandwidthMeter implements
BandwidthMeter, TransferListener {...}
```

Базовая реализация

```
@UnstableApi
public class SlidingPercentile {

    //Adds a new weighted value.
    public void addSample(int weight, float value) {...}

    //Computes a percentile by integration.
    public float getPercentile(float percentile) {...}

    //Resets the sliding percentile.
    public void reset() {...}

}
```

Взвешенная скользящая средняя

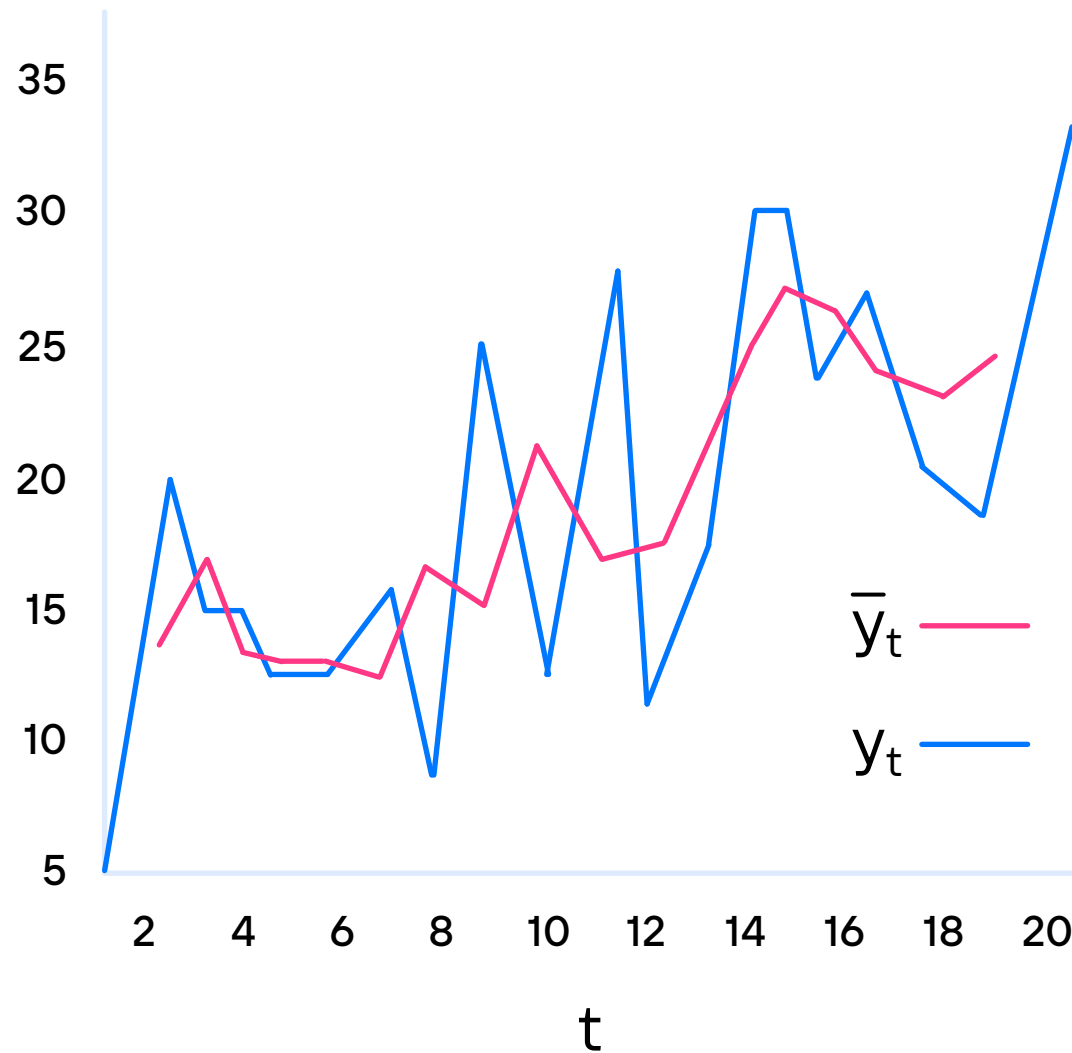


График исходной функции (синий)
и его скользящая средняя (красная)
с шириной окна $n = 2$

Базовая реализация

```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация

```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```

Базовая реализация



```
@Override
public synchronized void onTransferEnd(DataSource source, DataSpec dataSpec, boolean isNetwork) {
    long nowMs = clock.elapsedRealtime();
    int sampleElapsedTimeMs = nowMs - sampleStartTimeMs;
    totalElapsedTimeMs += sampleElapsedTimeMs;
    totalBytesTransferred += sampleBytesTransferred;
    if (sampleElapsedTimeMs > 0) {
        float bitsPerSecond = (sampleBytesTransferred * 8000f) / sampleElapsedTimeMs;
        slidingPercentile.addSample((int) Math.sqrt(sampleBytesTransferred), bitsPerSecond);
        if (totalElapsedTimeMs >= ELAPSED_MILLIS_FOR_ESTIMATE
            || totalBytesTransferred >= BYTES_TRANSFERRED_FOR_ESTIMATE) {
            bitrateEstimate = (long) slidingPercentile.getPercentile(0.5f);
        }
        maybeNotifyBandwidthSample(sampleElapsedTimeMs, sampleBytesTransferred, bitrateEstimate);
        sampleStartTimeMs = nowMs;
        sampleBytesTransferred = 0;
    }
}
```



Этого достаточно?

Этого достаточно?



Куда смотреть дальше?



Куда смотреть дальше?

```
/**  
 * Estimates bandwidth by listening to data transfers.  
 *  
 * The bandwidth estimate is calculated using a SlidingPercentile and is updated  
 * each time a transfer ends. The initial estimate is based on the current  
 * operator's network country code or the locale of the user, as well as the network  
 * connection type. This can be configured in the Builder.  
 */  
  
@UnstableApi  
public final class DefaultBandwidthMeter implements BandwidthMeter,  
TransferListener {...}
```

Куда смотреть дальше?



```
/**  
 * Estimates bandwidth by listening to data transfers.  
 *  
 * The bandwidth estimate is calculated using a SlidingPercentile and is updated  
 * each time a transfer ends. The initial estimate is based on the current  
 * operator's network country code or the locale of the user, as well as the network  
 * connection type. This can be configured in the Builder.  
 */  
  
@UnstableApi  
public final class DefaultBandwidthMeter implements BandwidthMeter,  
TransferListener {...}
```

Кто еще наследует BandwidthMeter





Кто еще наследует BandwidthMeter

```
public final class ExperimentalBandwidthMeter implements
BandwidthMeter, TransferListener {

    @Override
    public synchronized void onTransferEnd(DataSource source,
DataSpec dataSpec, boolean isNetwork) {
        if (!isTransferAtFullNetworkSpeed(dataSpec, isNetwork)) {
            return;
        }
        bandwidthEstimator.onTransferEnd(source);
    }

}
```



Кто еще наследует BandwidthMeter

```
public final class ExperimentalBandwidthMeter implements
BandwidthMeter, TransferListener {

    @Override
    public synchronized void onTransferEnd(DataSource source,
    DataSpec dataSpec, boolean isNetwork) {
        if (!isTransferAtFullNetworkSpeed(dataSpec, isNetwork)) {
            return;
        }
        bandwidthEstimator.onTransferEnd(source);
    }

    boolean isNetwork
```

Что такое BandwidthEstimator?





Что такое BandwidthEstimator?

```
/** The interface for different bandwidth estimation strategies. */  
@UnstableApi  
public interface BandwidthEstimator {  
    void onTransferInitializing(DataSource source);  
    void onTransferStart(DataSource source);  
    void onBytesTransferred(DataSource source, int bytesTransferred);  
    void onTransferEnd(DataSource source);  
    long getBandwidthEstimate();  
    void onNetworkTypeChange(long newBandwidthEstimate);  
}
```



Что такое BandwidthEstimator?

```
/** The interface for different bandwidth estimation strategies. */
```

```
@UnstableApi
```

```
public interface BandwidthEstimator {
```

```
    void onTransferInitializing(DataSource source);
```

```
    void onTransferStart(DataSource source);
```

```
    void onBytesTransferred(DataSource source, int bytesTransferred);
```

```
    void onTransferEnd(DataSource source);
```

```
    long getBandwidthEstimate();
```

```
    void onNetworkTypeChange(long newBandwidthEstimate);
```

TransferListener

BandwidthMeter

```
}
```



Какие есть варианты BandwidthEstimator?

Две реализации в media3:

```
/**
 * A BandwidthEstimator that captures a transfer sample each time all parallel transfers end.
 */
public class CombinedParallelSampleBandwidthEstimator implements BandwidthEstimator
```

```
/**
 * A BandwidthEstimator that captures a transfer sample each time a transfer ends. When
 * parallel transfers are happening at the same time, the transferred bytes are aggregated in
 * a single sample.
 */
public class SplitParallelSampleBandwidthEstimator implements BandwidthEstimator
```

Взглянем на метод onTransferEnd



Взглянем на метод onTransferEnd

```
@Override
public void onTransferEnd(DataSource source) {
    //...
    long nowMs = clock.elapsedRealtime();
    long sampleElapsedTimeMs = (int) (nowMs - sampleStartTimeMs);
    if (sampleElapsedTimeMs > 0) {
        bandwidthStatistic.addSample(sampleBytesTransferred, sampleElapsedTimeMs * 1000);
        totalSamplesAdded++;
        if (totalSamplesAdded > minSamples && totalBytesTransferred > minBytesTransferred) {
            bandwidthEstimate = bandwidthStatistic.getBandwidthEstimate();
        }
        maybeNotifyBandwidthSample(
            (int) sampleElapsedTimeMs, sampleBytesTransferred, bandwidthEstimate);
        sampleBytesTransferred = 0;
    }
    //...
}
```



Взглянем на метод onTransferEnd

```
@Override
public void onTransferEnd(DataSource source) {
    //...
    long nowMs = clock.elapsedRealtime();
    long sampleElapsedTimeMs = (int) (nowMs - sampleStartTimeMs);
    if (sampleElapsedTimeMs > 0) {
        bandwidthStatistic.addSample(sampleBytesTransferred, sampleElapsedTimeMs * 1000);
        totalSamplesAdded++;
        if (totalSamplesAdded > minSamples && totalBytesTransferred > minBytesTransferred) {
            bandwidthEstimate = bandwidthStatistic.getBandwidthEstimate();
        }
        maybeNotifyBandwidthSample(
            (int) sampleElapsedTimeMs, sampleBytesTransferred, bandwidthEstimate);
        sampleBytesTransferred = 0;
    }
    //...
}
```

Взглянем на метод onTransferEnd

```
@Override
public void onTransferEnd(DataSource source) {
    //...
    long nowMs = clock.elapsedRealtime();
    long sampleElapsedTimeMs = (int) (nowMs - sampleStartTimeMs);
    if (sampleElapsedTimeMs > 0) {
        bandwidthStatistic.addSample(sampleBytesTransferred, sampleElapsedTimeMs * 1000);
        totalSamplesAdded++;
        if (totalSamplesAdded > minSamples && totalBytesTransferred > minBytesTransferred) {
            bandwidthEstimate = bandwidthStatistic.getBandwidthEstimate();
        }
        maybeNotifyBandwidthSample(
            (int) sampleElapsedTimeMs, sampleBytesTransferred, bandwidthEstimate);
        sampleBytesTransferred = 0;
    }
    //...
}
```

Взглянем на метод onTransferEnd

```
@Override
public void onTransferEnd(DataSource source) {
    //...
    long nowMs = clock.elapsedRealtime();
    long sampleElapsedTimeMs = (int) (nowMs - sampleStartTimeMs);
    if (sampleElapsedTimeMs > 0) {
        bandwidthStatistic.addSample(sampleBytesTransferred, sampleElapsedTimeMs * 1000);
        totalSamplesAdded++;
        if (totalSamplesAdded > minSamples && totalBytesTransferred > minBytesTransferred) {
            bandwidthEstimate = bandwidthStatistic.getBandwidthEstimate();
        }
        maybeNotifyBandwidthSample(
            (int) sampleElapsedTimeMs, sampleBytesTransferred, bandwidthEstimate);
        sampleBytesTransferred = 0;
    }
    //...
}
```

А BandwidthStatistic кто такой?



A BandwidthStatistic кто такой?

```

/** The interface for different bandwidth estimation statistics. */
public interface BandwidthStatistic {

    /**
     * Adds a transfer sample to the statistic.
     */
    void addSample(long bytes, long durationUs);

    /**
     * Returns the bandwidth estimate in bits per second, or {@link
     * BandwidthEstimator#ESTIMATE_NOT_AVAILABLE} if there is no estimate available yet.
     */
    long getBandwidthEstimate();

    /**
     * Resets the statistic. The statistic should drop all samples and reset to its initial state,
     * similar to right after construction.
     */
    void reset();
}

```

A BandwidthStatistic кто такой?

```
/** A BandwidthStatistic that calculates estimates using an exponential weighted average. */  
public class ExponentialWeightedAverageStatistic implements BandwidthStatistic
```

```
/**  
* A BandwidthStatistic that calculates estimates based on a sliding window weighted  
* percentile.  
*/  
public class SlidingPercentileBandwidthStatistic implements BandwidthStatistic {
```

```
/**  
* A BandwidthStatistic that calculates estimates based on a sliding window weighted  
* average.  
*/  
public class SlidingWeightedAverageBandwidthStatistic implements BandwidthStatistic
```

Формируем АБ



BandwidthMeter

Формируем АБ

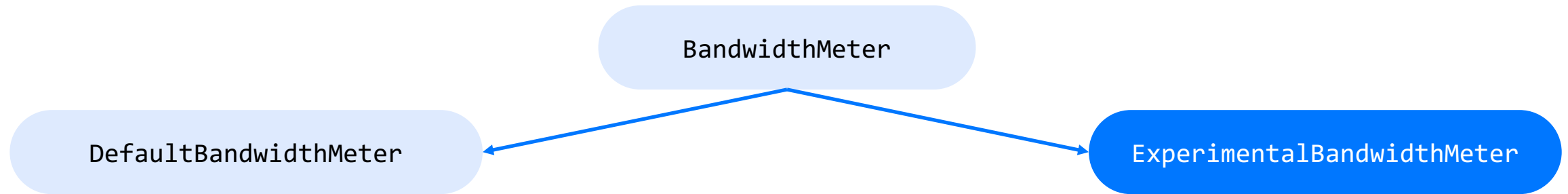


BandwidthMeter

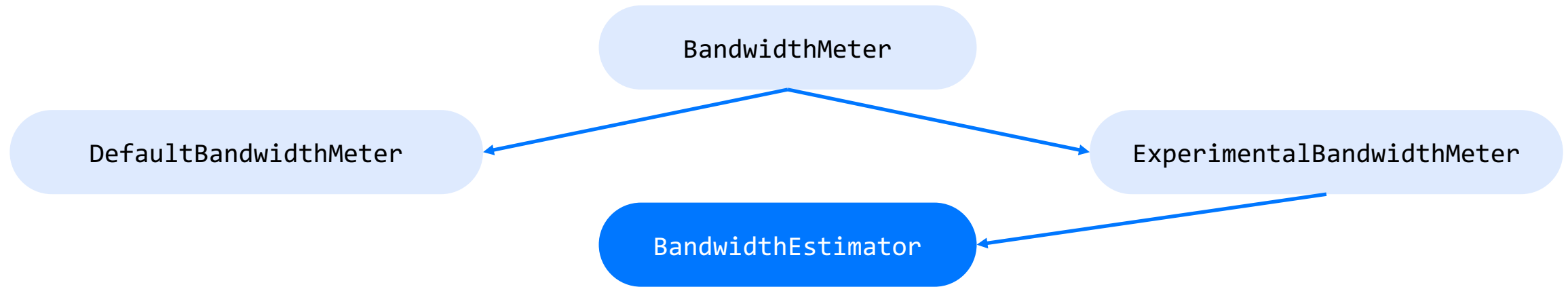
DefaultBandwidthMeter



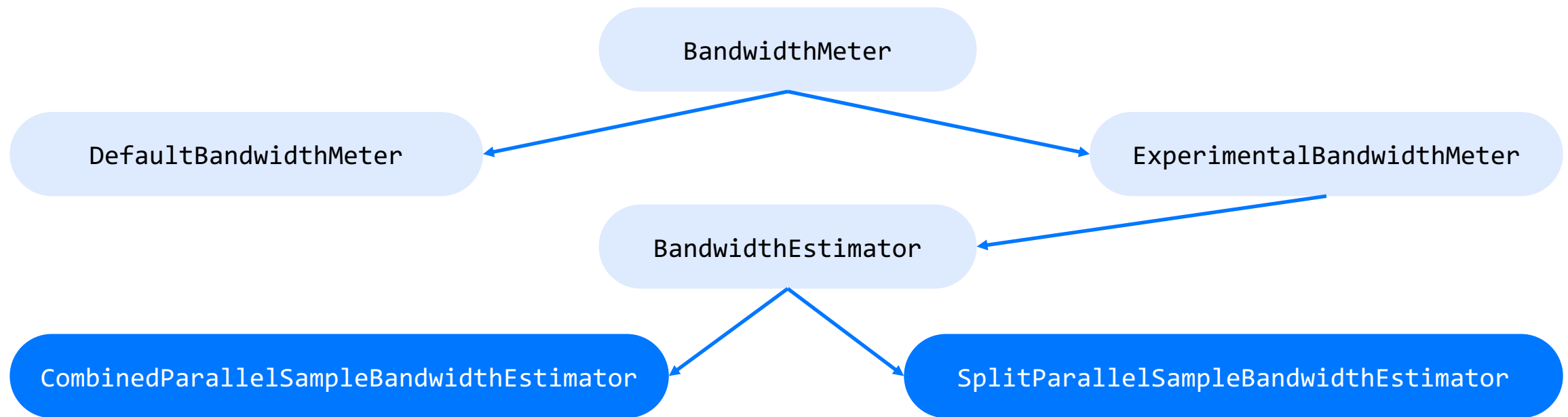
Формируем АБ



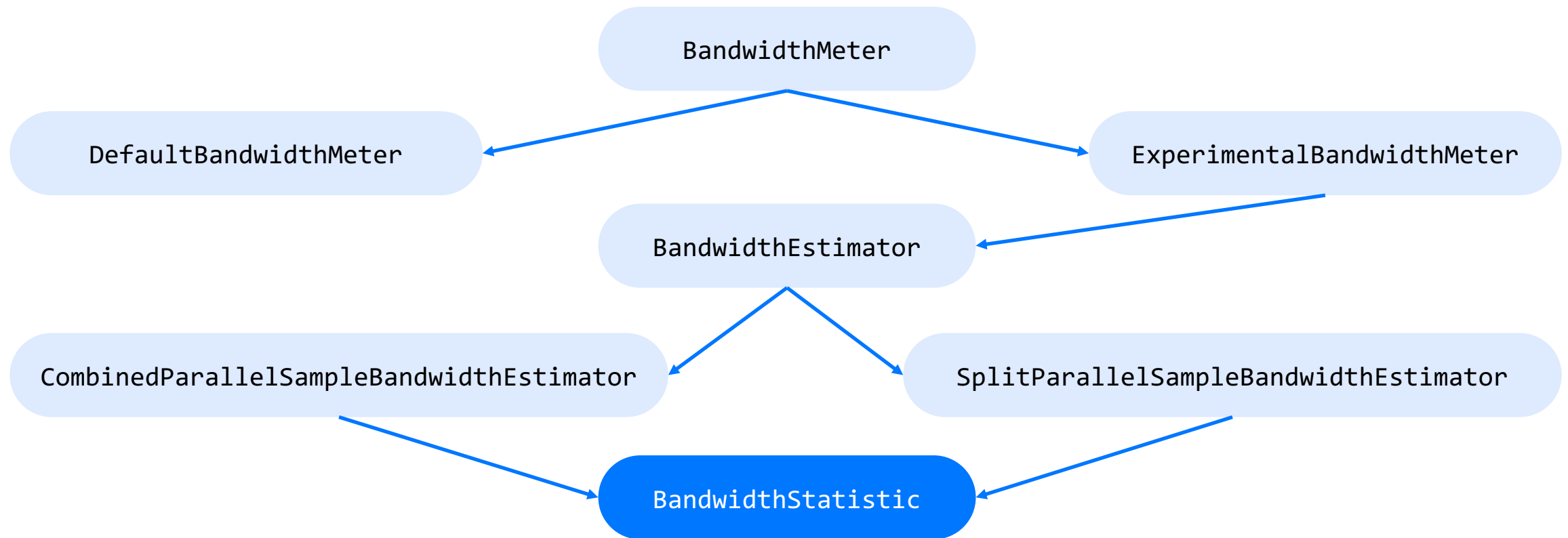
Формируем АБ



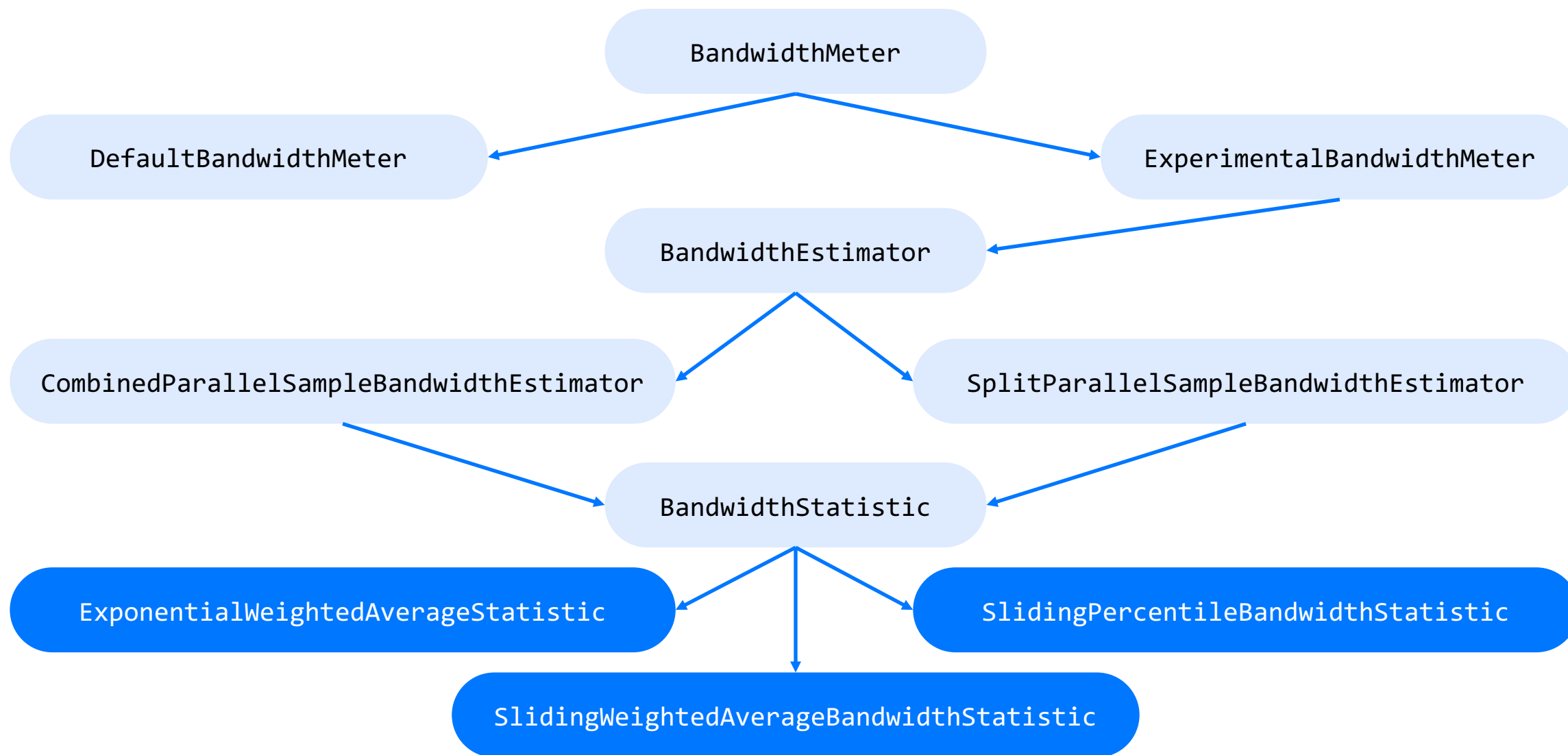
Формируем АБ



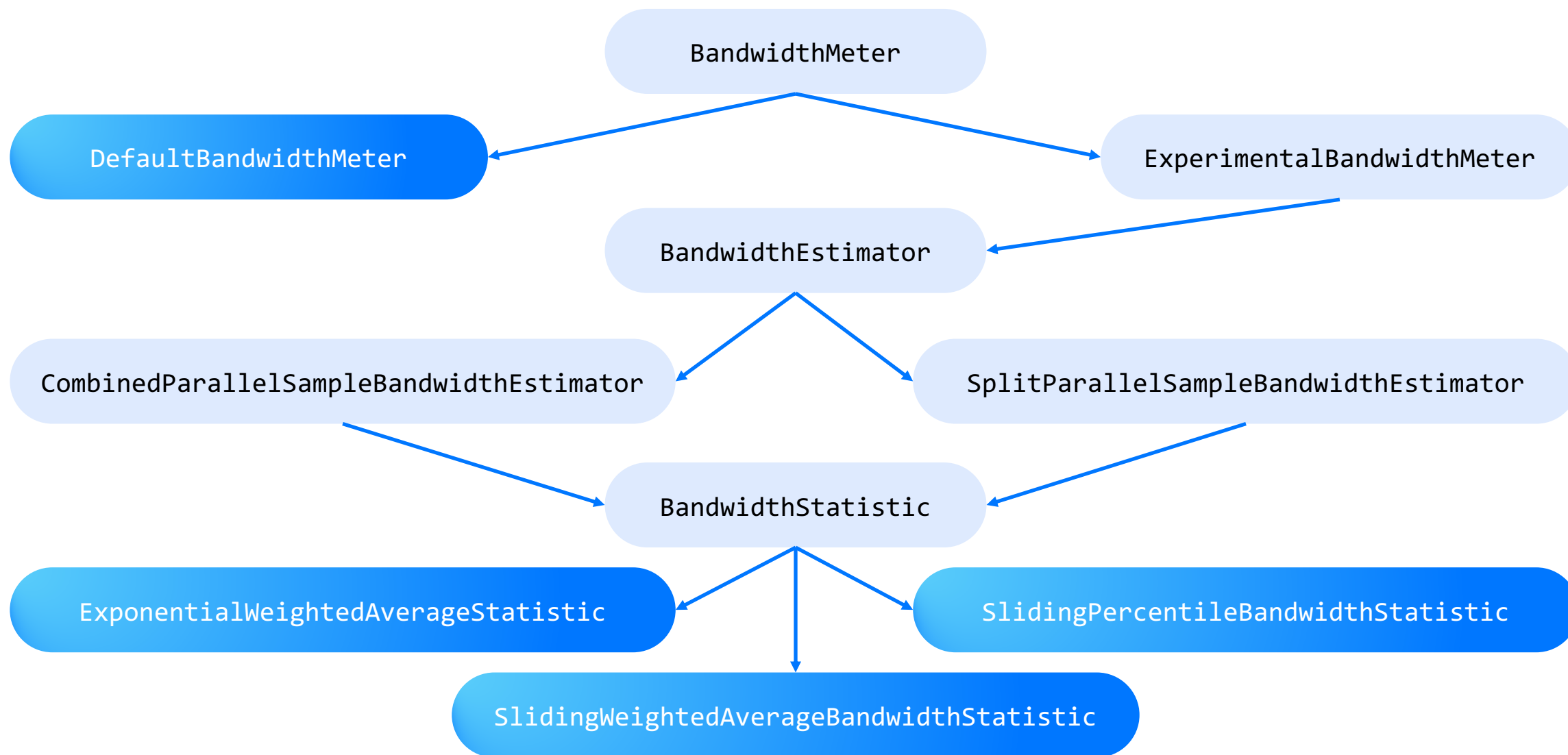
Формируем АБ



Формируем АБ



Формируем АБ



Что еще можно настроить



Для ExperimentalBandwidthMeter

1. TimeToFirstByteEstimator — алгоритм для подсчета эстиматора на основе первых байт
2. InitialBitrateEstimate — начальное значение битрейта для определенного типа сети



Для BandwidthEstimator

1. MinBytesTransferred — минимальное кол-во байт для начала расчёта битрейта
2. MinSamples — минимальное кол-во сэмплов для начала расчёта битрейта

Добавляем в проект



Добавляем в проект



```
val bandwidthStatistic = SlidingPercentileBandwidthStatistic(  
    BANDWIDTH_MAX_SAMPLES_COUNT,  
    BANDWIDTH_PERCENTILE.coerceIn(0.0, 1.0)  
)
```



Добавляем в проект

```
val bandwidthStatistic = SlidingPercentileBandwidthStatistic(  
    BANDWIDTH_MAX_SAMPLES_COUNT,  
    BANDWIDTH_PERCENTILE.coerceIn(0.0, 1.0)  
)
```

```
val bandwidthEstimator = SplitParallelSampleBandwidthEstimator.Builder()  
    .setBandwidthStatistic(bandwidthStatistic)  
    .setMinBytesTransferred(MIN_BYTES_TRANSFERRED)  
    .setMinSamples(MIN_SAMPLE_COUNT)  
    .build()
```



Добавляем в проект

```
val bandwidthStatistic = SlidingPercentileBandwidthStatistic(  
    BANDWIDTH_MAX_SAMPLES_COUNT,  
    BANDWIDTH_PERCENTILE.coerceIn(0.0, 1.0)  
)
```

```
val bandwidthEstimator = SplitParallelSampleBandwidthEstimator.Builder()  
    .setBandwidthStatistic(bandwidthStatistic)  
    .setMinBytesTransferred(MIN_BYTES_TRANSFERRED)  
    .setMinSamples(MIN_SAMPLE_COUNT)  
    .build()
```

```
val bandwidthMeter = ExperimentalBandwidthMeter.Builder(context)  
    .setBandwidthEstimator(bandwidthEstimator)  
    .setTimeToFirstByteEstimator(PercentileTimeToFirstByteEstimator())
```



Добавляем в проект

```
val bandwidthStatistic = SlidingPercentileBandwidthStatistic(  
    BANDWIDTH_MAX_SAMPLES_COUNT,  
    BANDWIDTH_PERCENTILE.coerceIn(0.0, 1.0)  
)
```

```
val bandwidthEstimator = SplitParallelSampleBandwidthEstimator.Builder()  
    .setBandwidthStatistic(bandwidthStatistic)  
    .setMinBytesTransferred(MIN_BYTES_TRANSFERRED)  
    .setMinSamples(MIN_SAMPLE_COUNT)  
    .build()
```

```
val bandwidthMeter = ExperimentalBandwidthMeter.Builder(context)  
    .setBandwidthEstimator(bandwidthEstimator)  
    .setTimeToFirstByteEstimator(PercentileTimeToFirstByteEstimator())
```

```
val player = ExoPlayer.Builder(context)  
    .setBandwidthMeter(bandwidthMeter)  
    .build()
```

Результаты интеграции



на **8%**

СНИЗИЛОСЬ КОЛИЧЕСТВО
ЗАВИСАНИЙ

на **6%**

УПАЛО КОЛИЧЕСТВО ВЫХОДОВ
НА СТОЛЛАХ

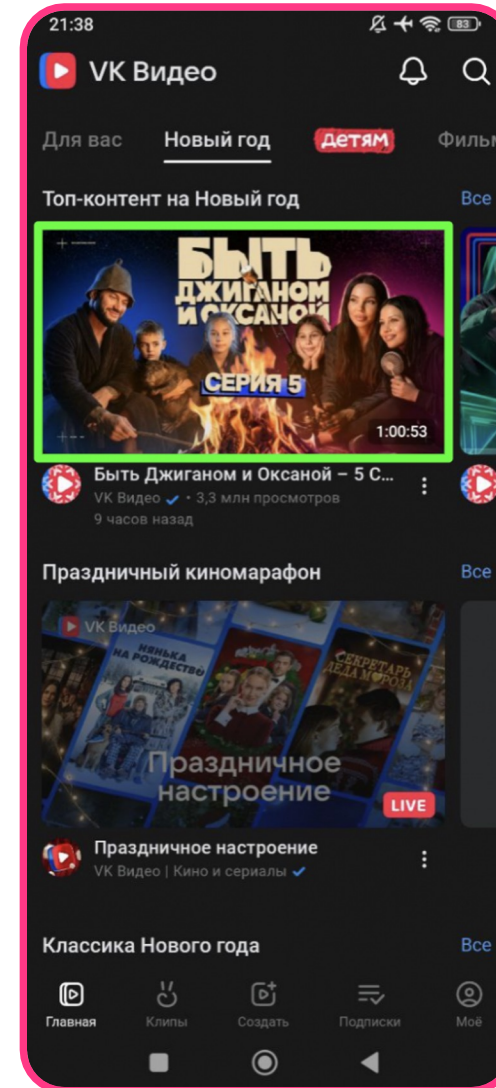
Viewport

Что такое viewport

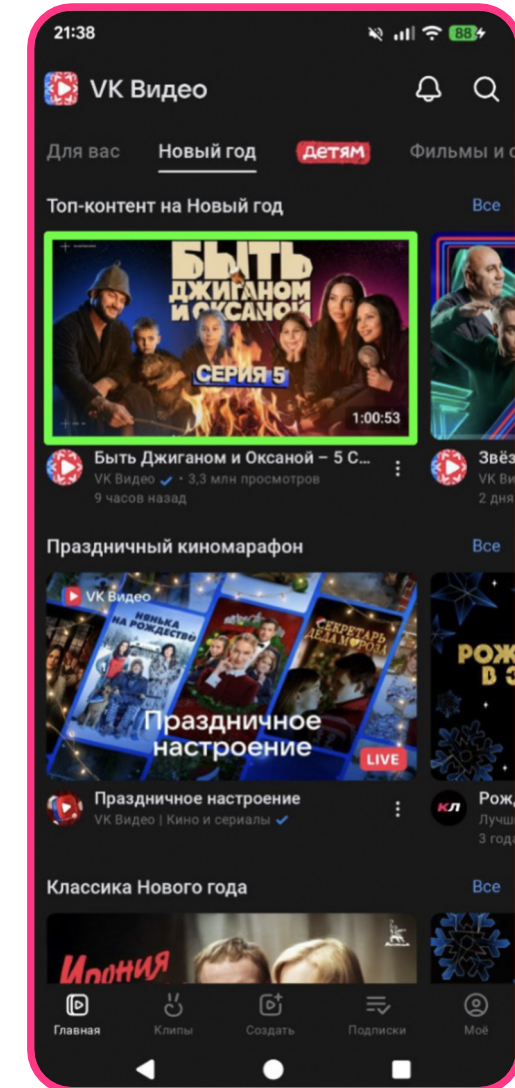
Viewport (вьюпорт) — часть сайта или приложения, которую пользователь видит сразу, без прокрутки. В контексте видео вьюпорт — область видимости видео на экране устройства.

— Viewport телефона

— Viewport видео



Xiaomi 12 Pro



Google Pixel 7A

Качества видео



Качество видео	Соотношение пикселей (ширина x высота)	Количество пикселей
144p	256 x 144	36 864
240p	426 x 240	102 240
360p	640 x 360	230 400
480p (SD)	854 x 480	409 920
720p (HD)	1 280 x 720	921 600
1080p (Full HD)	1 920 x 1 080	2 073 600
1440p (Quad HD)	2 560 x 1 440	3 686 400
2160p (4k Ultra HD)	3 840 x 2 160	8 294 400
4320p (8k Ultra HD)	7 680 x 4 320	33 177 600

Связь качества видео и viewport

Формат видео	Пример кадра	Формат видео	Пример кадра
240p		720p	
360p		1080p	
480p		1440p	

Связь качества видео и viewport



Формат
видео

Пример
кадра

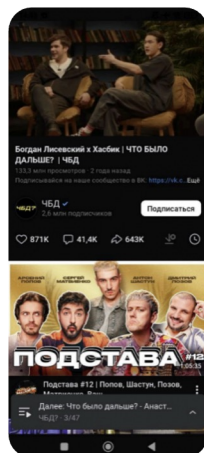
Формат
видео

Пример
кадра

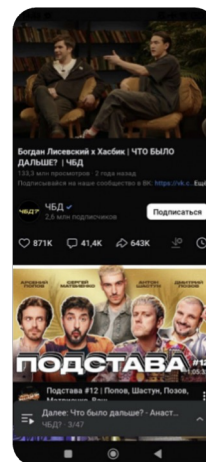
Формат
видео

Пример
кадра

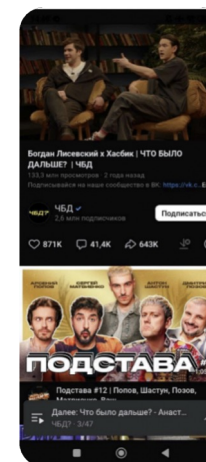
240p



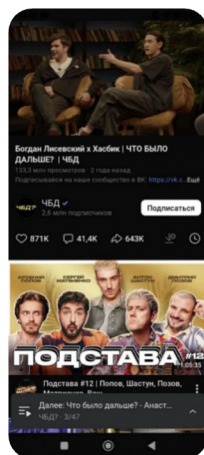
480p



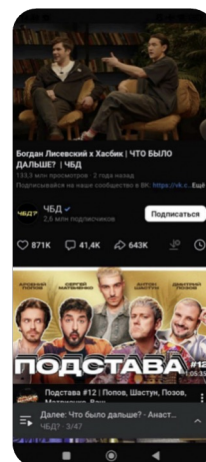
1080p



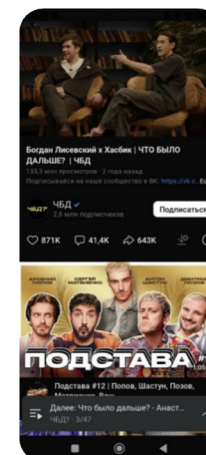
360p



720p



1440p



Связь качества видео и viewport



Чем меньше вьюпорт,
тем хуже глаз человека
может различать качество
картинок в высоких
разрешения

Связь качества видео и viewport

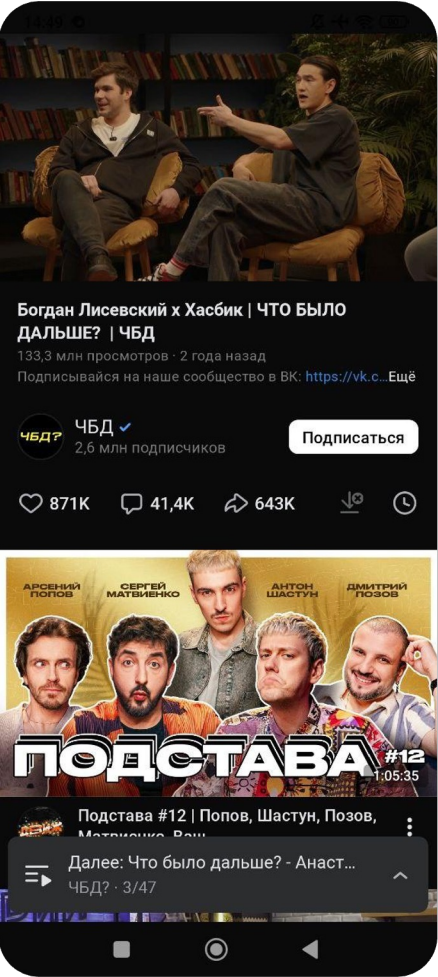


Чем меньше вьюпорт,
тем хуже глаз человека
может различать качество
картинок в высоких
разрешения

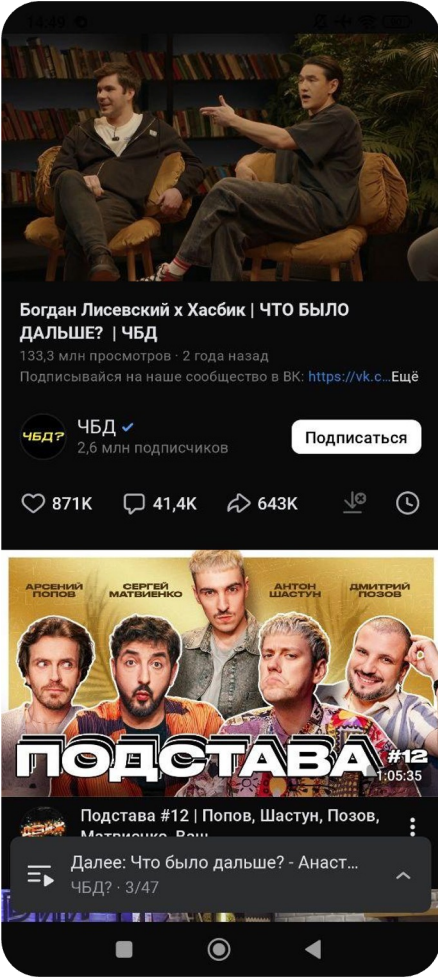


Чем меньше вьюпорт,
тем меньше пикселей
для отображения картинок

Найдите 1 612 800 отличий



1080p

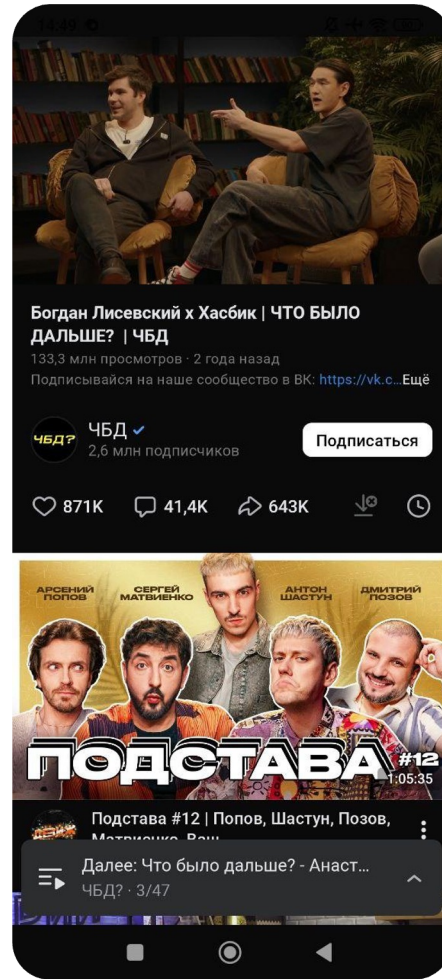


1440p

Найдите 1 612 800 отличий



1080p



1440p

1080p — 2 073 600 пикселей
1440p — 3 686 400 пикселей

Учитываем размер области видео при выборе качества



Решаем проблему «лишних» данных

```
class OneVideoViewHolder {  
  
    constructor(surfaceView: SurfaceView) {  
        attachView(surfaceView)  
    }  
  
    constructor(textureView: TextureView) {  
        attachView(textureView)  
    }  
  
    @Volatile  
    var viewportSize: Size? = null  
        private set  
  
    private fun attachView(view: View) {  
        val setNonEmptyViewport: (View) -> Unit = { v ->  
            viewportSize = if (v.width == 0 || v.height == 0) null else Size(v.width, v.height)  
        }  
        setNonEmptyViewport(view)  
        view.addOnLayoutChangeListener { v, _, _, _, _, _, _, _ ->  
            setNonEmptyViewport(v)  
        }  
    }  
}
```



Решаем проблему «лишних» данных

```
class OneVideoViewHolder {  
  
    constructor(surfaceView: SurfaceView) {  
        attachView(surfaceView)  
    }  
  
    constructor(textureView: TextureView) {  
        attachView(textureView)  
    }  
  
    @Volatile  
    var viewportSize: Size? = null  
        private set  
  
    private fun attachView(view: View) {  
        val setNonEmptyViewport: (View) -> Unit = { v ->  
            viewportSize = if (v.width == 0 || v.height == 0) null else Size(v.width, v.height)  
        }  
        setNonEmptyViewport(view)  
        view.addOnLayoutChangeListener { v, _, _, _, _, _, _, _ ->  
            setNonEmptyViewport(v)  
        }  
    }  
}
```



Решаем проблему «лишних» данных

```
class OneVideoViewHolder {  
  
    constructor(surfaceView: SurfaceView) {  
        attachView(surfaceView)  
    }  
  
    constructor(textureView: TextureView) {  
        attachView(textureView)  
    }  
  
    @Volatile  
    var viewportSize: Size? = null  
    private set  
  
    private fun attachView(view: View) {  
        val setNonEmptyViewport: (View) -> Unit = { v ->  
            viewportSize = if (v.width == 0 || v.height == 0) null else Size(v.width, v.height)  
        }  
        setNonEmptyViewport(view)  
        view.addOnLayoutChangeListener { v, _, _, _, _, _, _, _ ->  
            setNonEmptyViewport(v)  
        }  
    }  
}
```

Решаем проблему «лишних» данных

```
class OneVideoViewHolder {  
  
    constructor(surfaceView: SurfaceView) {  
        attachView(surfaceView)  
    }  
  
    constructor(textureView: TextureView) {  
        attachView(textureView)  
    }  
  
    @Volatile  
    var viewportSize: Size? = null  
    private set  
  
    private fun attachView(view: View) {  
        val setNonEmptyViewport: (View) -> Unit = { v ->  
            viewportSize = if (v.width == 0 || v.height == 0) null else Size(v.width, v.height)  
        }  
        setNonEmptyViewport(view)  
        view.addOnLayoutChangeListener { v, _, _, _, _, _, _, _ ->  
            setNonEmptyViewport(v)  
        }  
    }  
}
```

Решаем проблему «лишних» данных

```
class OneVideoTrackSelection(  
    group: TrackGroup,  
    tracks: IntArray,  
    bandwidthMeter: BandwidthMeter,  
    private val viewportSizeProvider: () -> Size?,  
) : AdaptiveTrackSelection(  
    group,  
    tracks,  
    bandwidthMeter,  
)
```

Решаем проблему «лишних» данных

```
class OneVideoTrackSelection(...) {

    override fun updateSelectedTrack(...) {
        super.updateSelectedTrack(...)
        updateSelectedIndexByViewportSize()
    }

    private fun updateSelectedIndexByViewportSize() {
        val viewportSize = viewportSizeProvider() ?: return
        val needToAdaptToViewport = !fitsFully(viewportSize, videoTracks[selectedIndex])
        if (needToAdaptToViewport) {
            videoTracks.forEachIndexed { index, format ->
                if (fitsFully(viewportSize, format)) {
                    selectedIndex = index
                    return
                }
            }
        }
    }

    private fun fitsFully(viewportSize: Size, format: Format) =
        format.width <= viewportSize.width && format.height <= viewportSize.height
}
```

Решаем проблему «лишних» данных

```
class OneVideoTrackSelection(...) {

    override fun updateSelectedTrack(...) {
        super.updateSelectedTrack(...)
        updateSelectedIndexByViewportSize()
    }

    private fun updateSelectedIndexByViewportSize() {
        val viewportSize = viewportSizeProvider() ?: return
        val needToAdaptToViewport = !fitsFully(viewportSize, videoTracks[selectedIndex])
        if (needToAdaptToViewport) {
            videoTracks.forEachIndexed { index, format ->
                if (fitsFully(viewportSize, format)) {
                    selectedIndex = index
                    return
                }
            }
        }
    }

    private fun fitsFully(viewportSize: Size, format: Format) =
        format.width <= viewportSize.width && format.height <= viewportSize.height
}
```

Решаем проблему «лишних» данных



```
val trackSelectionFactory = OneVideoTrackSelection.Factory(viewportSizeProvider)
```



Решаем проблему «лишних» данных

```
val trackSelectionFactory = OneVideoTrackSelection.Factory(viewportSizeProvider)
```

```
val trackSelector = DefaultTrackSelector(  
    context,  
    trackSelectionFactory  
)
```



Решаем проблему «лишних» данных

```
val trackSelectionFactory = OneVideoTrackSelection.Factory(viewportSizeProvider)
```

```
val trackSelector = DefaultTrackSelector(  
    context,  
    trackSelectionFactory  
)
```

```
val player = ExoPlayer.Builder(context)  
    .setBandwidthMeter(bandwidthMeter)  
    .setTrackSelector(trackSelector)  
    .build
```

Итоги



количество зависаний
снизилось на 13%



приложение стало потреблять
меньше трафика и батареи



время показа первого кадра
ускорилось на 2%



FTR и ANR приложения
сократились

Если немного не хватило данного раздела



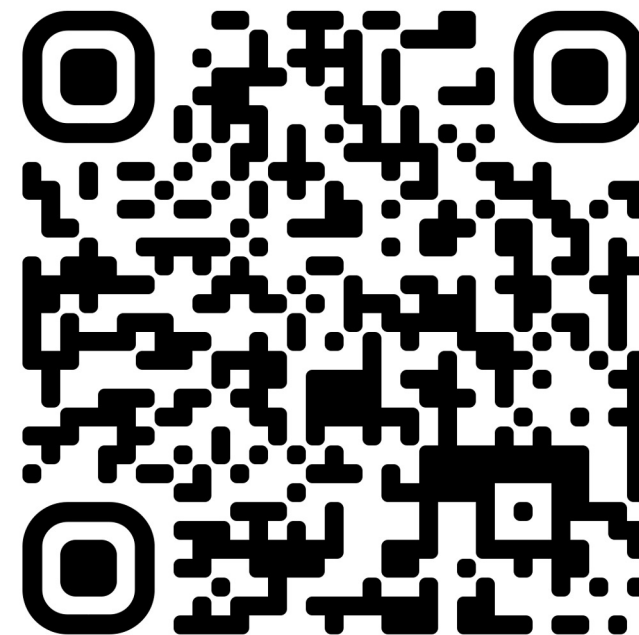
 **EgorBa** 3 фев в 11:28

Уменьшаем зависания и оптимизируем потребление трафика: как команда VK Видео работает с viewport на Android

 Средний  6 мин  9.5K

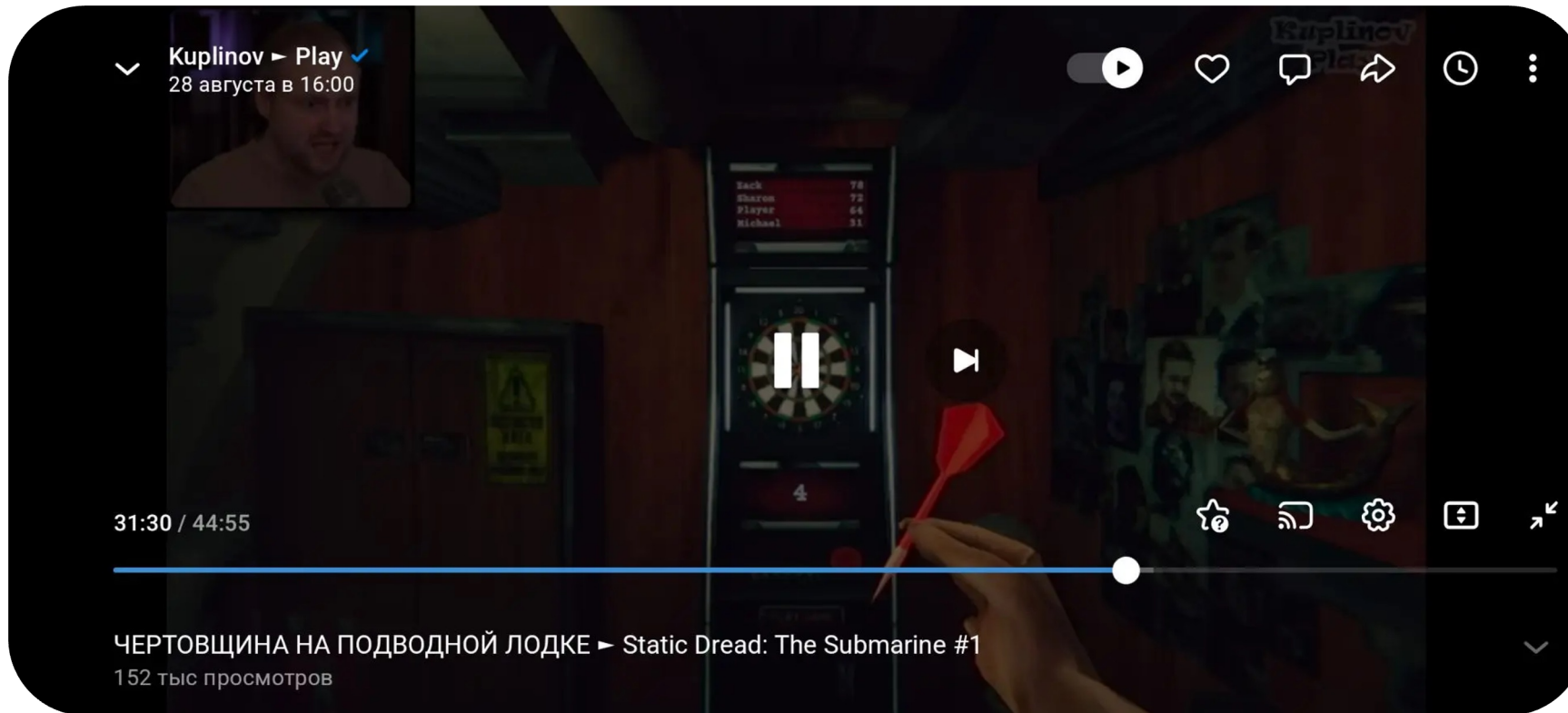
Блог компании VK, Android*, Разработка мобильных приложений*, Работа с видео*

Тutorial



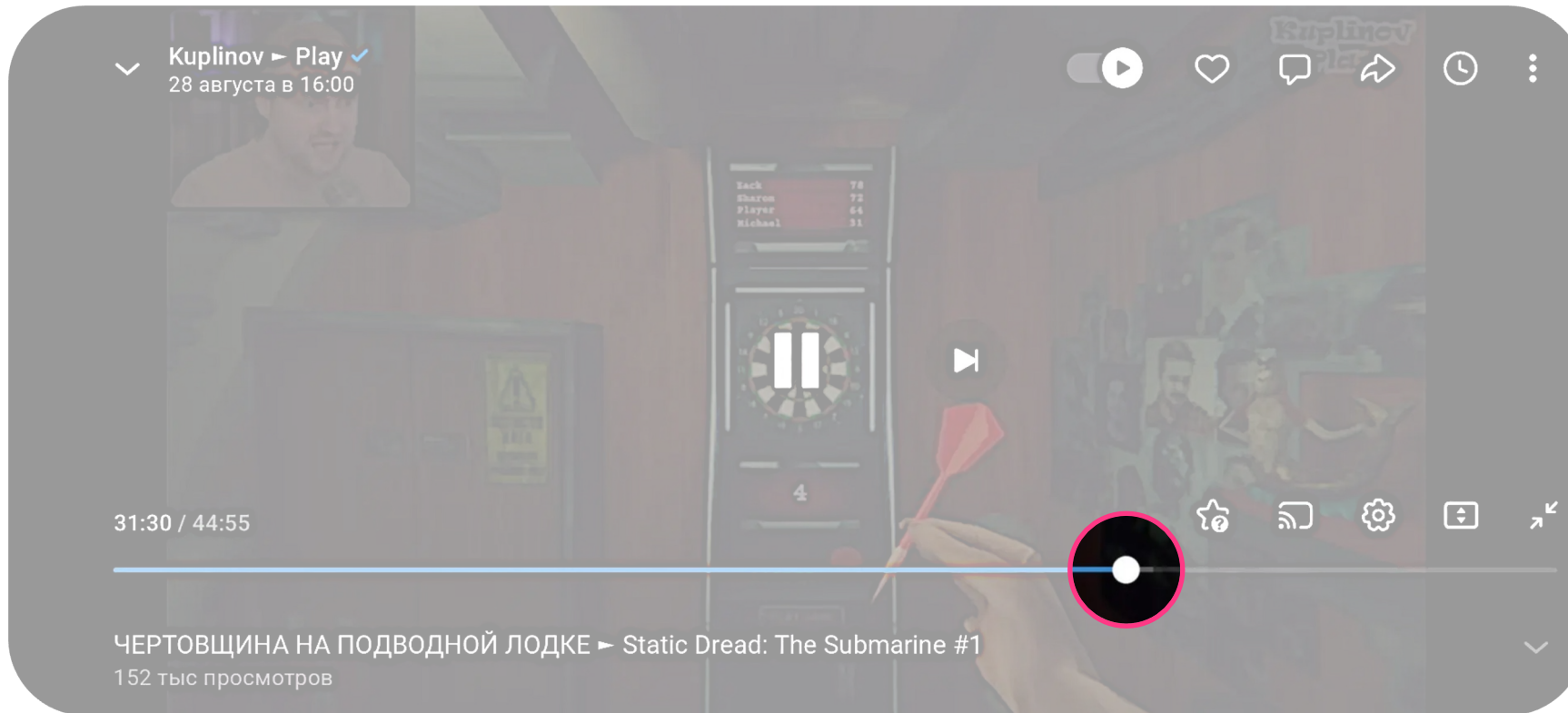
Буфер и не только

Что такое буфер?



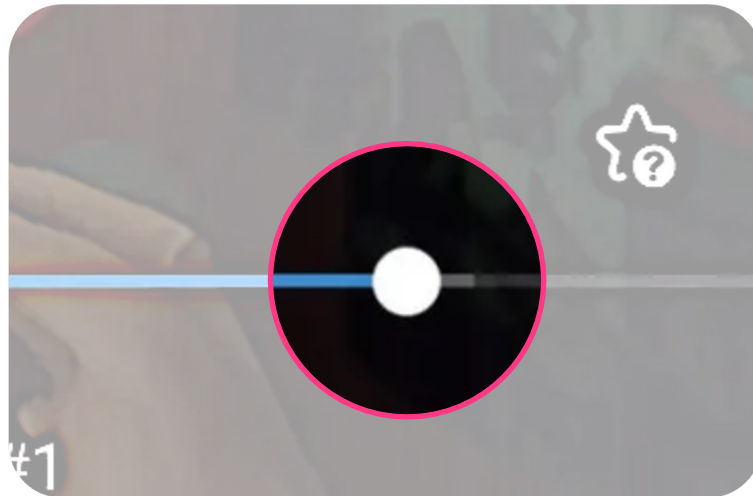
Как обычно выглядит буфер для пользователя

Что такое буфер?



Как обычно выглядит буфер для пользователя

Что такое буфер?



Как обычно выглядит буфер для пользователя

Кто за него отвечает?



Кто за него отвечает?



`androidx.media3.exoplayer.LoadControl`

Кто за него отвечает?



`androidx.media3.exoplayer.LoadControl`



`androidx.media3.exoplayer.DefaultLoadControl`

Из чего состоит



```
public Builder() {  
    playerTargetBufferBytes = new HashMap<>();  
    playerTargetBufferBytes.put(PlayerId.PRELOAD.name, DEFAULT_TARGET_BUFFER_BYTES_FOR_PRELOAD);  
    minBufferMs = DEFAULT_MIN_BUFFER_MS;  
    minBufferForLocalPlaybackMs = DEFAULT_MIN_BUFFER_FOR_LOCAL_PLAYBACK_MS;  
    maxBufferMs = DEFAULT_MAX_BUFFER_MS;  
    maxBufferForLocalPlaybackMs = DEFAULT_MAX_BUFFER_FOR_LOCAL_PLAYBACK_MS;  
    bufferForPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_MS;  
    bufferForPlaybackForLocalPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_FOR_LOCAL_PLAYBACK_MS;  
    bufferForPlaybackAfterRebufferMs = DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS;  
    bufferForPlaybackAfterRebufferForLocalPlaybackMs =  
        DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_FOR_LOCAL_PLAYBACK_MS;  
    targetBufferBytes = DEFAULT_TARGET_BUFFER_BYTES;  
    prioritizeTimeOverSizeThresholds = DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS;  
    prioritizeTimeOverSizeThresholdsForLocalPlayback =  
        DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS_FOR_LOCAL_PLAYBACK;  
    backBufferDurationMs = DEFAULT_BACK_BUFFER_DURATION_MS;  
    retainBackBufferFromKeyframe = DEFAULT_RETAIN_BACK_BUFFER_FROM_KEYFRAME;  
}
```

Из чего состоит



```
public Builder() {
    playerTargetBufferBytes = new HashMap<>();
    playerTargetBufferBytes.put(PlayerId.PRELOAD.name, DEFAULT_TARGET_BUFFER_BYTES_FOR_PRELOAD);
    minBufferMs = DEFAULT_MIN_BUFFER_MS;
    minBufferForLocalPlaybackMs = DEFAULT_MIN_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    maxBufferMs = DEFAULT_MAX_BUFFER_MS;
    maxBufferForLocalPlaybackMs = DEFAULT_MAX_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_MS;
    bufferForPlaybackForLocalPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackAfterRebufferMs = DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS;
    bufferForPlaybackAfterRebufferForLocalPlaybackMs =
        DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_FOR_LOCAL_PLAYBACK_MS;
    targetBufferBytes = DEFAULT_TARGET_BUFFER_BYTES;
    prioritizeTimeOverSizeThresholds = DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS;
    prioritizeTimeOverSizeThresholdsForLocalPlayback =
        DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS_FOR_LOCAL_PLAYBACK;
    backBufferDurationMs = DEFAULT_BACK_BUFFER_DURATION_MS;
    retainBackBufferFromKeyframe = DEFAULT_RETAIN_BACK_BUFFER_FROM_KEYFRAME;
}
```

Из чего состоит



```
public Builder() {
    playerTargetBufferBytes = new HashMap<>();
    playerTargetBufferBytes.put(PlayerId.PRELOAD.name, DEFAULT_TARGET_BUFFER_BYTES_FOR_PRELOAD);
    minBufferMs = DEFAULT_MIN_BUFFER_MS;
    minBufferForLocalPlaybackMs = DEFAULT_MIN_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    maxBufferMs = DEFAULT_MAX_BUFFER_MS;
    maxBufferForLocalPlaybackMs = DEFAULT_MAX_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_MS;
    bufferForPlaybackForLocalPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackAfterRebufferMs = DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS;
    bufferForPlaybackAfterRebufferForLocalPlaybackMs =
        DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_FOR_LOCAL_PLAYBACK_MS;
    targetBufferBytes = DEFAULT_TARGET_BUFFER_BYTES;
    prioritizeTimeOverSizeThresholds = DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS;
    prioritizeTimeOverSizeThresholdsForLocalPlayback =
        DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS_FOR_LOCAL_PLAYBACK;
    backBufferDurationMs = DEFAULT_BACK_BUFFER_DURATION_MS;
    retainBackBufferFromKeyframe = DEFAULT_RETAIN_BACK_BUFFER_FROM_KEYFRAME;
}
```

Из чего состоит



```
public Builder() {
    playerTargetBufferBytes = new HashMap<>();
    playerTargetBufferBytes.put(PlayerId.PRELOAD.name, DEFAULT_TARGET_BUFFER_BYTES_FOR_PRELOAD);
    minBufferMs = DEFAULT_MIN_BUFFER_MS;
    minBufferForLocalPlaybackMs = DEFAULT_MIN_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    maxBufferMs = DEFAULT_MAX_BUFFER_MS;
    maxBufferForLocalPlaybackMs = DEFAULT_MAX_BUFFER_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_MS;
    bufferForPlaybackForLocalPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_FOR_LOCAL_PLAYBACK_MS;
    bufferForPlaybackAfterRebufferMs = DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS;
    bufferForPlaybackAfterRebufferForLocalPlaybackMs =
        DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_FOR_LOCAL_PLAYBACK_MS;
    targetBufferBytes = DEFAULT_TARGET_BUFFER_BYTES;
    prioritizeTimeOverSizeThresholds = DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS;
    prioritizeTimeOverSizeThresholdsForLocalPlayback =
        DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS_FOR_LOCAL_PLAYBACK;
    backBufferDurationMs = DEFAULT_BACK_BUFFER_DURATION_MS;
    retainBackBufferFromKeyframe = DEFAULT_RETAIN_BACK_BUFFER_FROM_KEYFRAME;
}
```



Из чего состоит

```
public Builder() {  
    playerTargetBufferBytes = new HashMap<>();  
    playerTargetBufferBytes.put(PlayerId.PRELOAD.name, DEFAULT_TARGET_BUFFER_BYTES_FOR_PRELOAD);  
    minBufferMs = DEFAULT_MIN_BUFFER_MS;  
    minBufferForLocalPlaybackMs = DEFAULT_MIN_BUFFER_FOR_LOCAL_PLAYBACK_MS;  
    maxBufferMs = DEFAULT_MAX_BUFFER_MS;  
    maxBufferForLocalPlaybackMs = DEFAULT_MAX_BUFFER_FOR_LOCAL_PLAYBACK_MS;  
    bufferForPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_MS;  
    bufferForPlaybackForLocalPlaybackMs = DEFAULT_BUFFER_FOR_PLAYBACK_FOR_LOCAL_PLAYBACK_MS;  
    bufferForPlaybackAfterRebufferMs = DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS;  
    bufferForPlaybackAfterRebufferForLocalPlaybackMs =  
        DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_FOR_LOCAL_PLAYBACK_MS;  
    targetBufferBytes = DEFAULT_TARGET_BUFFER_BYTES;  
    prioritizeTimeOverSizeThresholds = DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS;  
    prioritizeTimeOverSizeThresholdsForLocalPlayback =  
        DEFAULT_PRIORITIZE_TIME_OVER_SIZE_THRESHOLDS_FOR_LOCAL_PLAYBACK;  
    backBufferDurationMs = DEFAULT_BACK_BUFFER_DURATION_MS;  
    retainBackBufferFromKeyframe = DEFAULT_RETAIN_BACK_BUFFER_FROM_KEYFRAME;  
}
```

Разберем каждый параметр

```
/**  
 * The default minimum duration of media that the player will attempt to  
 * ensure is buffered at all times, in milliseconds.  
 */  
public static final int DEFAULT_MIN_BUFFER_MS = 50_000;
```

Разберем каждый параметр

```
/**  
 * The default maximum duration of media that the player will attempt  
 * to buffer, in milliseconds.  
 */  
public static final int DEFAULT_MAX_BUFFER_MS = 50_000;
```

Разберем каждый параметр

```
/**  
 * The default duration of media that must be buffered for playback to start or resume  
 * following a user action such as a seek, in milliseconds.  
 */  
public static final int DEFAULT_BUFFER_FOR_PLAYBACK_MS = 1000;
```

```
/**  
 * The default duration of media that must be buffered for playback to resume after a  
 * rebuffer, in milliseconds. A rebuffer is defined to be caused by buffer depletion  
 * rather than a user action.  
 */  
public static final int DEFAULT_BUFFER_FOR_PLAYBACK_AFTER_REBUFFER_MS = 2000;
```

Разберем каждый параметр

```
/** The default back buffer duration in milliseconds. */  
public static final int DEFAULT_BACK_BUFFER_DURATION_MS = 0;
```

Как выбрать нужные параметры?



Как выбрать нужные параметры?

1

Провести А/Б
со всеми возможными
параметрами

Как выбрать нужные параметры?

1

Провести А/Б
со всеми возможными
параметрами

2

Выбрать нужные
параметры
в зависимости от типа
и формы потребления
контента на основе
статистики

Ищем баланс



- > Слишком маленький буфер — будут столлы
- > Слишком большой буфер — будут ООМы и лишний трафик

Ищем баланс



- > Слишком маленький буфер — будут столлы
- > Слишком большой буфер — будут ООМы и лишний трафик



Кадр из к/ф "Мстители: Война бесконечности", Правообладатель : Marvel Studios

Формируем АБ



Формируем АБ



1

Разделили LoadControl
для видео и клипов

Формируем АБ



1

Разделили LoadControl
для видео и клипов

2

Разделили LoadControl
для типов девайсов



К чему мы пришли

```
val bufferMs = when {  
    isClip -> 10_000  
    isHigh -> 120_000  
    else -> 70_000  
}
```



К чему мы пришли

```
val bufferMs = when {  
    isClip -> 10_000  
    isHigh -> 120_000  
    else -> 70_000  
}  
val bufferForPlaybackMs = 1000  
val bufferAfterRebufferMs = 2000
```



К чему мы пришли

```
val bufferMs = when {  
    isClip -> 10_000  
    isHigh -> 120_000  
    else -> 70_000  
}  
val bufferForPlaybackMs = 1000  
val bufferAfterRebufferMs = 2000  
val backBufferDurationMs = if (isClip) 0 else 50_000
```



К чему мы пришли

```
val bufferMs = when {  
    isClip -> 10_000  
    isHigh -> 120_000  
    else -> 70_000  
}  
val bufferForPlaybackMs = 1000  
val bufferAfterRebufferMs = 2000  
val backBufferDurationMs = if (isClip) 0 else 50_000  
  
val loadControl = DefaultLoadControl.Builder()  
    .setBufferDurationsMs(bufferMs, bufferMs, bufferForPlaybackMs, bufferAfterRebufferMs)  
    .setBackBuffer(backBufferDurationMs, false)  
    .build()
```



К чему мы пришли

```
val bufferMs = when {  
    isClip -> 10_000  
    isHigh -> 120_000  
    else -> 70_000  
}  
val bufferForPlaybackMs = 1000  
val bufferAfterRebufferMs = 2000  
val backBufferDurationMs = if (isClip) 0 else 50_000  
  
val loadControl = DefaultLoadControl.Builder()  
    .setBufferDurationsMs(bufferMs, bufferMs, bufferForPlaybackMs, bufferAfterRebufferMs)  
    .setBackBuffer(backBufferDurationMs, false)  
    .build()  
  
val player = ExoPlayer.Builder(context)  
    .setBandwidthMeter(bandwidthMeter)  
    .setTrackSelector(trackSelector)  
    .setLoadControl(loadControl)  
    .build()
```

Итоги



количество зависаний
снижено на 8%



количество ошибок
упало на 2%



количество зависаний при
перемотке снизилось на 12%



FTR и ANR приложения
сократились

Чего добились



Итоговый вариант плеера

```
val player = ExoPlayer.Builder(context)
    .setBandwidthMeter(bandwidthMeter)
    .setTrackSelector(trackSelector)
    .setLoadControl(loadControl)
    .build()
```

Итоговый вариант плеера



```
val player = ExoPlayer.Builder(context)
    .setBandwidthMeter(bandwidthMeter)
    .setTrackSelector(trackSelector)
    .setLoadControl(loadControl)
    .build()
```



Итоговый вариант плеера

```
val player = ExoPlayer.Builder(context)
    .setBandwidthMeter(bandwidthMeter)
    .setTrackSelector(trackSelector)
    .setLoadControl(loadControl)
    .build()
```

Итоговый вариант плеера

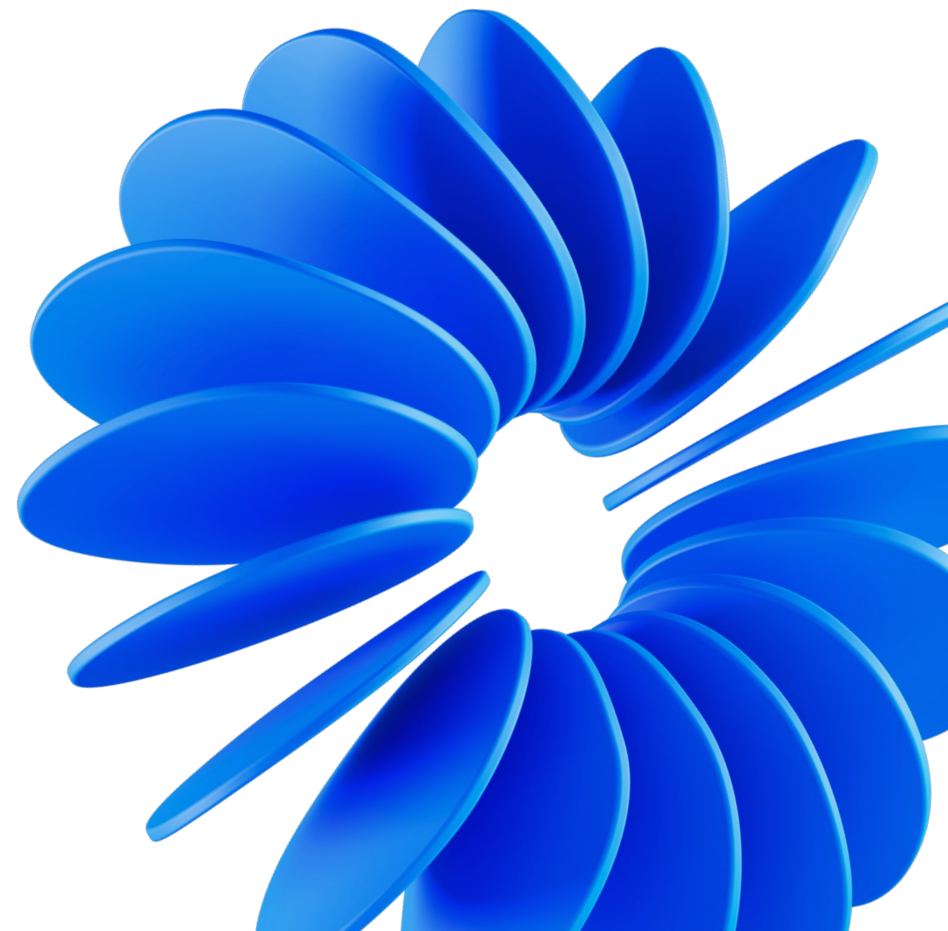


```
val player = ExoPlayer.Builder(context)
    .setBandwidthMeter(bandwidthMeter)
    .setTrackSelector(trackSelector)
    .setLoadControl(loadControl)
    .build()
```

Метрики



- > Снижение столлов на 30%
- > Снижение выходов на столле на 12%
- > Снижение столлов при перемотке на 12%
- > Снижение first byte и first frame
- > Снижение ANR и FTR



А что дальше?



А что дальше?



1 usePlatformDiagnostics

А что дальше?



1 usePlatformDiagnostics

2 stuckBufferingDetectionTimeoutMs

А что дальше?



1 usePlatformDiagnostics

2 stuckBufferingDetectionTimeoutMs

3 seekParameters

А что дальше?

- 1 `usePlatformDiagnostics`
- 2 `stuckBufferingDetectionTimeoutMs`
- 3 `seekParameters`
- 4 `shouldCancelChunkLoad` внутри `TrackSelection`

А что дальше?

- 1 `usePlatformDiagnostics`
- 2 `stuckBufferingDetectionTimeoutMs`
- 3 `seekParameters`
- 4 `shouldCancelChunkLoad` внутри `TrackSelection`
- 5 Дальнейшее изучение экзы...

Заключение



Используйте с умом представленные механизмы.
Реализуйте их под свои нужды и продукты.
Улучшайте работу плеера для пользователей.

Спасибо за внимание!

