```
import { AbilityConstant, UIAbility, Want } from '@kit.AbilityKit';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { window } from '@kit.ArkUI';
import { BreakpointSystem } from '@jellyfin-harmony/core';
import { AvoidAreaSystem } from '@jellyfin-harmony/core';
import { AudioManager } from '../player/AudioManager';

export default class EntryAbility extends UIAbility {

  private breakpointSystem: BreakpointSystem = new BreakpointSystem();

  private avoidAreaSystem: AvoidAreaSystem = new AvoidAreaSystem();

  private audioManager?: AudioManager

  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onCreate');
    this.audioManager = new AudioManager(this.context)
    AppStorage.setOrCreate(AudioManager.AUDIO_MANAGER, this.audioManager)
  }

  onDestroy(): void {
    this.audioManager?.release()
    AppStorage.delete(AudioManager.AUDIO_MANAGER)
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onDestroy');
  }

  onWindowStageCreate(windowStage: window.WindowStage): void {
    // Main window is created, set main page for this ability
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onWindowStageCreate');
    windowStage.getMainWindowSync().setImmersiveModeEnabledState(true)
    this.breakpointSystem.register();
    this.avoidAreaSystem.register(windowStage.getMainWindowSync())
    windowStage.loadContent('pages/splash/SplashPage', (err) => {
      if (err.code) {
        hilog.error(0x0000, 'testTag', 'Failed to load the content. Cause: %{public}s',
```

```
JSON.stringify(err) ?? '');
            return;
          }
          hilog.info(0x0000, 'testTag', 'Succeeded in loading the content.');
        });
    }

    onWindowStageDestroy(): void {
        this.avoidAreaSystem.unregister()
        this.breakpointSystem.unregister()
        // Main window is destroyed, release UI related resources
        hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onWindowStageDestroy');
    }

    onForeground(): void {
        // Ability has brought to foreground
        hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onForeground');
    }

    onBackground(): void {
        // Ability has back to background
        hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onBackground');
    }
}

import { AbilityStage, Configuration, ConfigurationConstant } from "@kit.AbilityKit";
import { AppPrefer } from "../prefer/AppPrefer";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 9:35
 * @Version V1.0
 * @Description
 */
export class App extends AbilityStage {
```

```
  onCreate(): void {
    //
https://developer.huawei.com/consumer/cn/doc/harmonyos-guides-V5/arkts-light-dark-color-adaptation-V5#section1421172621111

this.context.getApplicationContext().setColorMode(ConfigurationConstant.ColorMode.COLOR_MODE_NOT_SET);
    AppStorage.setOrCreate(AppPrefer.PREFER, new AppPrefer(this.context.getApplicationContext()))
  }

  onConfigurationUpdate(newConfig: Configuration): void {
  }

}
/**
 * Common constants for common component.
 */
export class CommonConstants {
  // Font family
  static readonly HARMONY_HEI_TI_FONT_FAMILY = 'HarmonyHeiTi';
  static readonly HARMONY_HEITI_MEDIUM_FONT_FAMILY = 'HarmonyHeiTi-Medium';
  static readonly HARMONY_HEITI_BOLD_FONT_FAMILY = 'HarmonyHeiTi-Bold';
  // Font weight
  static readonly DIALOG_TITLE_FONT_WEIGHT: number = 700;
  static readonly DIALOG_BUTTON_FONT_WEIGHT: number = 500;
  static readonly NORMAL_FONT_WEIGHT: number = 400;
  // Opacity
  static readonly FIRST_LEVEL_OPACITY: number = 0.9;
  static readonly SECOND_LEVEL_OPACITY: number = 0.6;
  static readonly HALF_OPACITY: number = 0.5;
  static readonly THIRD_LEVEL_OPACITY: number = 0.3;
  static readonly Divider_OPACITY: number = 0.05;
  // Blur
```

```
static readonly REGULAR_BLUR: number = 250;
// Space
static readonly SPACE_4: number = 4;
static readonly SPACE_8: number = 8;
static readonly SPACE_12: number = 12;
static readonly SPACE_16: number = 16;
// MaxLines
static readonly MAX_LINE_TWO: number = 2;

// Column count
static readonly SM_COLUMN_COUNT: number = 1;
static readonly MD_COLUMN_COUNT: number = 2;
static readonly LG_COLUMN_COUNT: number = 3;

// Swiper duration
static readonly SWIPER_DURATION: number = 1000;
// Percent
static readonly FULL_PERCENT: string = '100%';
static readonly HALF_PERCENT: string = '50%';
static readonly NAVI_BAR_WIDTH: string = '40%';
// Skeleton animation config
static readonly SKELETON_ANIMATION: AnimateParam = {
  duration: 400,
  tempo: 0.6,
  curve: Curve.EaseInOut,
  delay: 200,
  iterations: -1,
  playMode: PlayMode.Alternate
}

// item
static readonly ITEM_WIDTH = 140

static readonly ITEM_RATIO = 0.67

static readonly ITEM_HEIGHT = 240
```

```
}
import { deviceInfo } from "@kit.BasicServicesKit";
import { BaseItemDto, BaseItemKind,
   BaseItemPerson,
   ClientInfo,
   CollectionType,
   DeviceInfo, ImageType,
   ItemFields,
   ItemFilter,
   ItemSortBy,
   Jellyfin,
   PlaybackInfoResponse,
   SortOrder,
   UserItemDataDto} from "@jellyfin-harmony/sdk";
import { ServerInfo } from "@jellyfin-harmony/core";
import { UserInfo } from "@jellyfin-harmony/core";
import { DataStore } from "@jellyfin-harmony/core";
import { AddressInfo } from "@jellyfin-harmony/core";
import { ActiveInfo } from "@jellyfin-harmony/core";
import { MD5 } from "@jellyfin-harmony/core";
import { uri } from "@kit.ArkTS";
import BuildProfile from "BuildProfile";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/9 11:50
 * @Version V1.0
 * @Description Repository
 */
export class Repository {

   public static readonly REPOSITORY = "Repository"
```

```
/**
 * Client info
 */
private readonly clientInfo: ClientInfo = {
    name: "FinMusic",
    version: BuildProfile.VERSION_NAME
}

/**
 * Device info
 */
private readonly deviceInfo: DeviceInfo = {
    id: deviceInfo.ODID + '-music',
    name: deviceInfo.marketName
}

/**
 * Jellyfin api
 */
private jellyfin: Jellyfin = new Jellyfin({
    clientInfo: this.clientInfo,
    deviceInfo: this.deviceInfo,
})

private jellyfinTmp: Jellyfin = new Jellyfin({
    clientInfo: this.clientInfo,
    deviceInfo: this.deviceInfo,
})

private context: Context

private dataStore: DataStore

private activeInfo?: ActiveInfo

constructor(context: Context) {
```

```
        this.context = context
        this.dataStore = new DataStore(context)
    }

    /**
     * Init active server and user
     * @returns
     */
    public async init(): Promise<void> {
        let activeInfo = await this.dataStore.queryActive()
        if (activeInfo) {
            this.initActiveInfo(activeInfo)
        }
    }

    private initActiveInfo(activeInfo: ActiveInfo): void {
        this.jellyfin.apiClient.updateServerInfo({address: activeInfo.addressInfo.address})
        this.jellyfin.apiClient.updateUserInfo({id: activeInfo.userInfo.userId, token:
activeInfo.userInfo.accessToken})
        this.activeInfo = activeInfo
    }

    public getActiveInfo(): ActiveInfo | undefined {
        return this.activeInfo
    }

    public setActiveInfo(activeInfo: ActiveInfo) {
        this.initActiveInfo(activeInfo)
        this.dataStore.insertActive(activeInfo)
            .then(() => {
                // save success
            })
            .catch((error: Error) => {
                // save fail
            })
    }
```

```
/**
  * get all server
  * @returns
  */
public async getServers(): Promise<Array<ServerInfo>> {
    await this.dataStore.queryActive()
    return this.dataStore.queryAllServer()
}

/**
  * get all address
  * @returns
  */
public async getAddresses(): Promise<Array<AddressInfo>> {
    if (this.activeInfo == undefined) {
        return []
    }
    return this.dataStore.queryAllAddress(this.activeInfo.serverInfo.serverId)
}

/**
  * get all user
  * @returns
  */
public async getUsers(): Promise<Array<UserInfo>> {
    if (this.activeInfo == undefined) {
        return []
    }
    return this.dataStore.queryAllUser(this.activeInfo.serverInfo.serverId)
}

/**
  * connect server
  * @returns
  */
```

```
public async connectServer(url: string): Promise<ServerInfo> {
    this.jellyfinTmp.apiClient.updateServerInfo({address: url})
    let systemInfo = await this.jellyfinTmp.getSystemApi().getPublicSystemInfo()

    // save server info
    let serverInfo: ServerInfo = {
        serverId: systemInfo.Id!,
        serverName: systemInfo.ServerName!
    }
    await this.dataStore.insertServer(serverInfo)

    // save address info
    let addressInfo: AddressInfo = {
        addressId: MD5.digestSync(url),
        address: url,
        serverId: systemInfo.Id!
    }
    await this.dataStore.insertAddress(addressInfo)

    return serverInfo
}

public async addServer(url: string): Promise<ServerInfo> {
    throw new Error("Not impl")
}

/**
  * auth account
  * @param name
  * @param psw
  * @returns
  */
public async authAccount(url: string, name: string, psw?: string): Promise<ActiveInfo> {
    this.jellyfinTmp.apiClient.updateServerInfo({address: url})
    let authResult = await this.jellyfinTmp.getUserApi().authenticateUserByName({
        Username: name,
```

```
        Pw: psw
    })

    let serverInfo = await this.dataStore.queryServer(authResult.ServerId!)

    let addressInfo: AddressInfo = {
        addressId: MD5.digestSync(url),
        address: url,
        serverId: authResult.ServerId!
    }

    let userInfo: UserInfo = {
        userId: authResult.User!.Id!,
        userName: authResult.User!.Name!,
        serverId: authResult.ServerId!,
        accessToken: authResult.AccessToken!
    }
    return { serverInfo: serverInfo!, addressInfo: addressInfo, userInfo: userInfo}
}

public async addAccount(name: string, psw?: string): Promise<UserInfo> {
    throw new Error("Not impl")
}

private filterItem(items?: Array<BaseItemDto> | null): Array<BaseItemDto> | undefined {
    let newItems: Array<BaseItemDto> | undefined
    items?.forEach((item) => {
        let type = item.CollectionType
        if (type == CollectionType.Music) {
            if (newItems == undefined) {
                newItems = new Array()
            }
            newItems.push(item)
        }
    }
    )
```

```
    return newItems
}

public buildImage(dto: BaseItemDto, type: ImageType): string {
    let imageUri = new uri.URI(this.jellyfin.apiClient.getServerInfo()!.address)
    imageUri = imageUri.addEncodedSegment(`items/${dto.Id!}/Images/${type}`)
    return imageUri.toString()
}

public buildPersonImage(person: BaseItemPerson, type: ImageType): string {
    let imageUri = new uri.URI(this.jellyfin.apiClient.getServerInfo()!.address)
    imageUri = imageUri.addEncodedSegment(`items/${person.Id!}/Images/${type}`)
    return imageUri.toString()
}

/**
  * 查询媒体库
  * @returns
  */
public async getMediaList(): Promise<Array<BaseItemDto>> {
    let result = await this.jellyfin.getUserViewsApi().getUserViews({
        userId: this.activeInfo!.userInfo.userId,
        presetViews: [CollectionType.Music]
    })
    let groups = this.filterItem(result.Items)
    if (!groups) {
        return []
    }
    return groups
}

public async getLatestMedia(id?: string): Promise<Array<BaseItemDto>> {
    let result = await this.jellyfin.getUserLibraryApi().getLatestMedia({
        userId: this.activeInfo!.userInfo.userId,
        parentId: id,
        limit: 8
```

```
        })
        if (result) {
            return result
        }
        return []
    }

    /**
     * 查询
     * @returns
     */
    public async loadMedia(): Promise<Array<BaseItemDto>> {
        let result = await this.jellyfin.getItemsApi().getItems({
            userId: this.activeInfo!.userInfo.userId
        })
        let items = this.filterItem(result.Items)
        if (items) {
            return items
        }
        return []
    }

    public async loadFavourite(includeItemTypes?: Array<BaseItemKind>):
Promise<Array<BaseItemDto>> {
        let result = await this.jellyfin.getItemsApi().getItems({
            userId: this.activeInfo!.userInfo.userId,
            filters: [ItemFilter.IsFavorite],
            includeItemTypes: includeItemTypes,
            recursive: true
        })
        if (result.Items) {
            return result.Items
        }
        return []
    }
```

```
    public async loadMediaList(id: string, types: Array<BaseItemKind>
        , recursive: boolean, sortBy?: Array<ItemSortBy>, sortOrder?: Array<SortOrder>):
Promise<Array<BaseItemDto>> {
        let result = await this.jellyfin.getItemsApi().getItems({
            parentId: id,
            includeItemTypes: types,
            recursive: recursive,
            sortBy: sortBy,
            sortOrder: sortOrder
        })
        return result.Items!
    }

    public loadMovie(id: string): Promise<BaseItemDto> {
        return this.jellyfin.getUserLibraryApi().getItem({
            userId: this.activeInfo!.userInfo.userId,
            itemId: id
        })
    }

    public loadShow(showId: string): Promise<BaseItemDto> {
        return this.jellyfin.getUserLibraryApi().getItem({
            userId: this.activeInfo!.userInfo.userId,
            itemId: showId
        })
    }

    public getResumeItems(): Promise<Array<BaseItemDto>> {
        return this.jellyfin.getItemsApi().getResumeItems({
            userId: this.activeInfo!.userInfo.userId,
            limit: 8,
            includeItemTypes: [BaseItemKind.Movie, BaseItemKind.Episode]
        }).then((result) => {
            if (result.Items) {
                return result.Items
            }
```

```
            return []
        })
    }

    public getNextUp(seriesId?: string): Promise<Array<BaseItemDto>> {
        return this.jellyfin.getTvShowsApi().getNextUp({
            userId: this.activeInfo!.userInfo.userId,
            seriesId: seriesId,
            enableResumable: false,
            limit: 8
        })
            .then((result) => {
                if (result.Items) {
                    return result.Items
                }
                return []
            })
    }

    public getSeasons(seriesId: string): Promise<Array<BaseItemDto>> {
        return this.jellyfin.getTvShowsApi().getSeasons({
            userId: this.activeInfo!.userInfo.userId,
            seriesId: seriesId
        }).then((result) => {
            if (result.Items) {
                return result.Items
            }
            return []
            })
}

    public getEpisodes(seriesId: string, seasonId: string): Promise<Array<BaseItemDto>> {
        return this.jellyfin.getTvShowsApi().getEpisodes({
            userId: this.activeInfo!.userInfo.userId,
            seriesId: seriesId,
            seasonId: seasonId,
```

```
        fields: [ItemFields.Overview]
    }).then((result) => {
        if (result.Items) {
            return result.Items
        }
        return []
    })
}

public getEpisode(episodeId: string): Promise<BaseItemDto> {
    return this.jellyfin.getUserLibraryApi().getItem({
        userId: this.activeInfo!.userInfo.userId,
        itemId: episodeId
    })
}

public loadMediaSource(id: string): Promise<PlaybackInfoResponse> {
    return this.jellyfin.getMediaInfoApi().getPlaybackInfo({
        userId: this.activeInfo!.userInfo.userId,
        itemId: id
    })
}

public loadStreamUrl(id: string, sourceId: string): Promise<string> {
    return this.jellyfin.getAudioApi().getAudioStream({
        itemId: id,
        static: true,
        mediaSourceId: sourceId
    })
}

public searchMedia(keyword: string): Promise<Array<BaseItemDto>> {
    return Promise.resolve([])
}

public getUserItem(id: string): Promise<UserItemDataDto> {
```

```
        return this.jellyfin.getItemsApi().getItemUserData({
            itemId: id,
            userId: this.activeInfo!.userInfo.userId
        })
    }

    public markFavorite(id: string): Promise<UserItemDataDto> {
        return this.jellyfin.getUserLibraryApi().markFavoriteItem({
            itemId: id,
            userId: this.activeInfo!.userInfo.userId
        })
    }

    public unmarkFavorite(id: string): Promise<UserItemDataDto> {
        return this.jellyfin.getUserLibraryApi().unmarkFavoriteItem({
            itemId: id,
            userId: this.activeInfo!.userInfo.userId
        })
    }

    public markPlayed(id: string): Promise<UserItemDataDto> {
        return this.jellyfin.getPlayStateApi().markPlayedItem({
            itemId: id,
            userId: this.activeInfo!.userInfo.userId
        })
    }

    public unmarkPlayed(id: string): Promise<UserItemDataDto> {
        return this.jellyfin.getPlayStateApi().markUnplayedItem({
            itemId: id,
            userId: this.activeInfo!.userInfo.userId
        })
    }

}
import { BaseItemDto } from "@jellyfin-harmony/sdk";
```

```
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/11 22:28
 * @Version V1.0
 * @Description
 */
export interface GroupInfo {

    group?: BaseItemDto

    items?: Array<BaseItemDto>

}
import { ViewModel} from '@jellyfin-harmony/core'
import { Repository } from '../data/Repository'

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/6 23:28
 * @Version V1.0
 * @Description
 */
export class AppViewModel extends ViewModel {

    public readonly repository: Repository = AppStorage.get(Repository.REPOSITORY)!

    constructor(context: Context) {
        super(context)
    }

}
import { RefreshDataSource } from "@abner/refresh";
import { LoadType } from "@jellyfin-harmony/core";
```

```
import { RefreshRepository } from "@jellyfin-harmony/core";
import { AppViewModel } from './AppViewModel'
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/27 21:36
 * @Version V1.0
 * @Description
 */
export abstract class ListViewModel extends AppViewModel implements RefreshRepository {

  protected readonly dataSource = new RefreshDataSource();

  getDataSource(): RefreshDataSource {
    return this.dataSource
  }

  abstract loadData(type: LoadType): Promise<void>

}
import { ToolBar, WebTool } from '@jellyfin-harmony/core'
import { CommonConstants } from '../../common/CommonConstants'
import { Repository } from '../../data/Repository'

@Entry
@Component
struct AboutPage {

  private repository: Repository = AppStorage.get(Repository.REPOSITORY)!

  @Builder
  preferItemBuilder(icon: Resource, title: ResourceStr, callback: Callback<void>) {
    Row({space: CommonConstants.SPACE_12}) {
      SymbolGlyph(icon)
        .fontSize(24)
```

```
            .fontColor([$r('sys.color.ohos_id_color_text_primary')])
        Text(title)
            .fontSize(18)
            .fontColor($r('sys.color.ohos_id_color_text_primary'))
        Blank()
    }
    .width(CommonConstants.FULL_PERCENT)
    .height(48)
    .padding({
        left: CommonConstants.SPACE_16,
        right: CommonConstants.SPACE_16
    })
    .onClick(() => {
        callback()
    })
}

build() {
    Column({space: CommonConstants.SPACE_8}) {
        ToolBar({
            title: $r('app.string.about')
        })

        Column({ space: CommonConstants.SPACE_12}) {
            Row() {
                Text($r('app.string.prefer_current_server'))
                Text(this.repository?.getActiveInfo()?.serverInfo.serverName)
            }
            Row() {
                Text($r('app.string.prefer_current_address'))
                Text(this.repository?.getActiveInfo()?.addressInfo.address)
            }
            Row() {
                Text($r('app.string.prefer_current_user'))
                Text(this.repository?.getActiveInfo()?.userInfo.userName)
            }
```

```
        }
        .alignItems(HorizontalAlign.Start)
        .padding({
          left: CommonConstants.SPACE_16,
          right: CommonConstants.SPACE_16
        })

        this.preferItemBuilder($r('sys.symbol.doc_plaintext'), $r('app.string.prefer_privacy'),
() => {
          WebTool.openBrowser(getContext(),
"https://agreement-drcn.hispace.dbankcloud.cn/index.html?lang=zh&agreementId=15719
36775037801408")
        })
      }
      .alignItems(HorizontalAlign.Start)
      .width(CommonConstants.FULL_PERCENT)
      .height(CommonConstants.FULL_PERCENT)
  }
}
import { DetailViewModel } from './DetailViewModel'
import { ScrollRefreshView, ToolBar } from '@jellyfin-harmony/core'
import { BaseItemDto, ImageType } from '@jellyfin-harmony/sdk'
import { CommonConstants } from '../../common/CommonConstants'
import { MediaArgs } from './MediaArgs'
import router from '@ohos.router'
import { window } from '@kit.ArkUI'
import { Repository } from '../../data/Repository'
import { GroupInfo } from '../../entity/GroupInfo'
import { PlayerArgs } from '../player/PlayerArgs'
import { AudioManager } from '../../player/AudioManager'

@Entry
@Component
struct DetailPage {
  private args: MediaArgs = router.getParams() as MediaArgs
  private audioManager: AudioManager =
```

```
AppStorage.get(AudioManager.AUDIO_MANAGER)!
  public readonly repository: Repository = AppStorage.get(Repository.REPOSITORY)!
  private scroller = new Scroller()
  private viewModel = new DetailViewModel(getContext(), this.args)
  @StorageProp("currentAvoidArea") avoidArea?: window.AvoidArea = undefined

  @Builder
  movieScrollView(group?: GroupInfo) {
    Scroll(this.scroller) {
      Column({ space: CommonConstants.SPACE_12 }) {
        Image(this.repository?.buildImage(group!.group!, ImageType.Primary))
          .alt($r('app.media.alt'))
          .objectFit(ImageFit.Cover)
          .autoResize(true)
          .width(CommonConstants.FULL_PERCENT)
          .aspectRatio(1.5)

        Column({ space: CommonConstants.SPACE_12 }) {

          Text(group?.group?.Name)
            .fontSize($r('sys.float.Title_M'))

          Text(group?.group?.AlbumArtist)
            .fontSize($r('sys.float.Title_S'))

          Row({ space: CommonConstants.SPACE_8 }) {
            Text(group?.group?.ProductionYear?.toString())

            Text(Math.round(((group?.group?.RunTimeTicks!) / 600000000)) + "min")

            Text(group?.group?.OfficialRating)

            Text(group?.group?.CommunityRating?.toString())

          }
```

```
    // list
    if (group?.items) {
      List() {
        ForEach(group.items, (item: BaseItemDto, index) => {
          ListItem() {
            Text(`${index + 1} ${item.Name}`)
              .textAlign(TextAlign.Start)
              .width(CommonConstants.FULL_PERCENT)
              .height(60)
          }
          .onClick(() => {
            let queueManager = this.audioManager.getAudioQueue()
            queueManager.resetItem(this.viewModel.audios())
            this.audioManager.getAudioControl().playItem(item.Id!)
              .catch((error: Error) => {

              })
            let args: PlayerArgs = {
              index: index,
              id: item.Id!,
              name: item.Name!
            }
            router.pushUrl({url: 'pages/player/PlayerPage', params: args})
          })
        })
      }
      .width(CommonConstants.FULL_PERCENT)
      .divider({
        color: $r('sys.color.ohos_id_divider_color'),
        strokeWidth: 1
      })
    }

  }
  .padding({
    left: CommonConstants.SPACE_16,
```

```
            right: CommonConstants.SPACE_16
          })
          .width(CommonConstants.FULL_PERCENT)
          .alignItems(HorizontalAlign.Start)
      }
      .align(Alignment.Top)
      .padding({
        bottom: px2vp(this.avoidArea?.bottomRect.height)
      })
      .width(CommonConstants.FULL_PERCENT)
    }
      .height(CommonConstants.FULL_PERCENT)
      .width(CommonConstants.FULL_PERCENT)
}

build() {

    Stack() {

      ScrollRefreshView({
        source: this.viewModel,
        itemLayout: (info: ESObject) => {
          this.movieScrollView(info)
        }
      })
        .width(CommonConstants.FULL_PERCENT)
        .height(CommonConstants.FULL_PERCENT)

      ToolBar({
        color: $r('sys.color.font_on_primary')
      })
    }
    .alignContent(Alignment.Top)
    .width(CommonConstants.FULL_PERCENT)
    .height(CommonConstants.FULL_PERCENT)
```

```
    }
}
import { LoadType, SimpleRepository } from "@jellyfin-harmony/core";
import { BaseItemDto, BaseItemKind, ImageType } from "@jellyfin-harmony/sdk";
import { GroupInfo } from "../../entity/GroupInfo";
import { AppViewModel } from "../../lifecycle/AppViewModel";
import { AudioItem } from "../../player/AudioItem";
import { MediaArgs } from "./MediaArgs";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 11:54
 * @Version V1.0
 * @Description
 */
export class DetailViewModel extends AppViewModel implements SimpleRepository {

    private readonly args: MediaArgs

    private readonly audioItems = new Array<AudioItem>()

    constructor(context: Context, args: MediaArgs) {
        super(context)
        this.args = args
    }

    public audios(): Array<AudioItem> {
        return this.audioItems
    }

    async loadData(type: LoadType): Promise<Object> {
        let album = await this.repository.loadMovie(this.args.id)
        let audioArray = await this.repository.loadMediaList(album.Id!, [BaseItemKind.Audio],
true)
        audioArray.sort((left: BaseItemDto, right: BaseItemDto) => {
```

```
            return left.IndexNumber! - right.IndexNumber!
        })
        let groupInfo: GroupInfo = {
            group: album,
            items: audioArray
        }
        this.audioItems.splice(0)
        for (let item of audioArray) {
            this.audioItems.push({
                id: item.Id!,
                name: item.Name!,
                art: this.repository.buildImage(item, ImageType.Primary)!,
                url: await this.repository.loadStreamUrl(item.Id!, item.Id!),
                extra: item
            })
        }
        return groupInfo
    }

}
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/19 22:32
 * @Version V1.0
 * @Description Media args
 */
export interface MediaArgs {

    id: string,

    name: string,

}
import { BreakpointTypeEnum, GridRefreshView } from '@jellyfin-harmony/core'
import { CommonConstants } from '../../common/CommonConstants'
```

```
import { HomeToolBar } from '../../widget/bar/HomeToolBar'
import { HomeViewModel } from './HomeViewModel'
import { router, window } from '@kit.ArkUI'
import { BaseItemDto, ImageType } from '@jellyfin-harmony/sdk'
import { Repository } from '../../data/Repository'
import { MediaArgs } from '../detail/MediaArgs'

@Entry
@Component
struct HomePage {

  private viewModel = new HomeViewModel(getContext())

  @StorageProp('currentBreakpoint') currentBreakpoint: string = BreakpointTypeEnum.MD;

  @StorageProp('currentAvoidArea') currentAvoidArea?: window.AvoidArea = undefined

  aboutToAppear(): void {
    this.viewModel.loadData('Init')
  }

  @Builder
  mediaListView(info: BaseItemDto, index: number) {
    MediaItem({item: info})
  }

  build() {

    Column() {
      HomeToolBar({
        title: $r('app.string.app_name')
      })

      GridRefreshView({
        source: this.viewModel,
```

```
            gridAttribute: (attr) => {
              attr.padding = {
                left: CommonConstants.SPACE_16,
                right: CommonConstants.SPACE_16,
                bottom: px2vp(this.currentAvoidArea?.bottomRect.height)
              }
              attr.rowsGap = CommonConstants.SPACE_12
              attr.columnsGap = CommonConstants.SPACE_12
              attr.columnsTemplate = ('1fr 1fr')
              attr.width = CommonConstants.FULL_PERCENT
              attr.height = CommonConstants.FULL_PERCENT
            },
            itemLayout: this.mediaListView
          })
            .width(CommonConstants.FULL_PERCENT)
            .layoutWeight(1)

      }

    }

}


@Reusable
@Component
struct MediaItem {

  @StorageProp("Repository") repository?: Repository = undefined

  @Require item?: BaseItemDto

  aboutToReuse(params: BaseItemDto): void {
    this.item = params
    console.log("MediaList item reuse: pre = " + this.item?.Name + ", new = " +
params.Name)
```

```
  }

  build() {
    Column() {
      Image(this.repository?.buildImage(this.item!, ImageType.Primary))
        .alt($r('app.media.alt'))
        .objectFit(ImageFit.Cover)
        .autoResize(true)
        .borderRadius($r('app.float.lg_border_radius'))
        .width('100%')
        .aspectRatio(1)

      Text(this.item?.Name)
        .width('100%')
        .maxLines(1)
      Text(this.item?.AlbumArtist)
        .width('100%')
        .maxLines(1)
    }
    .onClick(() => {
      let args: MediaArgs = {
        id: this.item?.Id!,
        name: this.item?.Name!
      }
      router.pushUrl({url: 'pages/detail/DetailPage', params: args})
    })
  }

  aboutToRecycle(): void {
    console.log("MediaList item recycle: " + this.item?.Name)
  }

}
import { LoadType } from "@jellyfin-harmony/core";
import { BaseItemKind } from "@jellyfin-harmony/sdk";
import { ListViewModel } from "../../lifecycle/ListViewModel";
```

```
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 11:28
 * @Version V1.0
 * @Description
 */
export class HomeViewModel extends ListViewModel {

  async loadData(type: LoadType): Promise<void> {
    let mediaArray = await this.repository.getMediaList()
    if (mediaArray.length > 0) {
      let media = mediaArray[0]
      let albums = await this.repository.loadMediaList(media.Id!,
[BaseItemKind.MusicAlbum], true)
      this.dataSource.initData(albums)
    }
  }

}
import { promptAction, router } from '@kit.ArkUI'
import { Repository } from '../../data/Repository'
import { ProgressDialog } from '@jellyfin-harmony/core'
import { common } from '@kit.AbilityKit'

@Entry
@Component
struct LoginPage {

  private repository: Repository = AppStorage.get(Repository.REPOSITORY)!

  private progressDialog?: ProgressDialog

  private url?: string
```

```
private userName?: string

private psw?: string

public aboutToAppear(): void {
  this.url = (router.getParams() as object)["url"]
  this.progressDialog = new ProgressDialog(this.getUIContext())
}

build() {

  RelativeContainer() {

    TextInput({ placeholder: $r('app.string.login_user_name') })
      .id("login_name")
      .constraintSize({
        maxWidth: 300
      })
      .alignRules({
        middle: { anchor: "__container__", align: HorizontalAlign.Center },
        center: { anchor: "__container__", align: VerticalAlign.Center },
      })
      .onChange((value) => {
        this.userName = value
      })

    Image($r('app.media.foreground'))
      .size({
        width: 100,
        height: 100
      })
      .alignRules({
        middle: { anchor: "__container__", align: HorizontalAlign.Center },
        top: { anchor: "__container__", align: VerticalAlign.Top },
        bottom: { anchor: "login_name", align: VerticalAlign.Top },
      })
```

```
    TextInput({ placeholder: $r('app.string.login_psw') })
      .id("id_psw")
      .type(InputType.Password)
      .constraintSize({
        maxWidth: 300
      })
      .margin({
        top: 15
      })
      .alignRules({
        middle: { anchor: "__container__", align: HorizontalAlign.Center },
        top: { anchor: "login_name", align: VerticalAlign.Bottom },
      })
      .onChange((value) => {
        this.psw = value
      })

    Button($r('app.string.login_auth'))
      .id("login_login")
      .constraintSize({
        minWidth: 100
      })
      .margin({
        top: 15
      })
      .alignRules({
        middle: { anchor: "__container__", align: HorizontalAlign.Center },
        top: { anchor: "id_psw", align: VerticalAlign.Bottom },
      })
      .onClick(() => {
        this.auth()
      })

  }
```

```
  }

  private auth() {
    if (this.url == undefined) {
      promptAction.showToast({message: $r('app.string.error_url_not_validate')})
      return
    }
    if (this.userName == undefined || this.userName.length <= 0) {
      promptAction.showToast({message:
$r('app.string.error_user_name_not_validate')})
      return
    }
    this.progressDialog?.show()
    this.repository?.authAccount(this.url, this.userName, this.psw)
      .then((activeInfo) => {
        this.progressDialog?.dismiss()
        this.repository!.setActiveInfo(activeInfo)
        let state = router.getStateByIndex(0)
        if (state?.name == "pages/home/HomePage") {
          router.back(0)
        } else {
          (getContext()                                                                    as
common.UIAbilityContext).windowStage.loadContent("pages/home/HomePage")
        }
      })
      .catch((error: Error) => {
        this.progressDialog?.dismiss()
        promptAction.showToast({message: error.message})
      })
  }

}
import { CommonConstants } from "../../../common/CommonConstants";
import { AudioItem } from "../../../player/AudioItem";
import { AudioManager } from "../../../player/AudioManager";
```

```
@Component
export struct PlaylistPage {

  @Require callback?: Callback<void> = undefined

  private audioManager: AudioManager = AppStorage.get(AudioManager.AUDIO_MANAGER)!

  aboutToAppear(): void {
    this.audioManager.getAudioQueue().getQueue()
  }

  build() {
    Column() {
      List() {
        ForEach(this.audioManager.getAudioQueue().getQueue(), (item: AudioItem, index) => {
          ListItem() {
            Text(`${index + 1} ${item.name}`)
              .height(60)
              .textAlign(TextAlign.Start)
              .width(CommonConstants.FULL_PERCENT)
          }
          .padding({
            left: CommonConstants.SPACE_16,
            right: CommonConstants.SPACE_16
          })
          .onClick(() => {
            this.audioManager.getAudioControl().playItem(item.id)
            this.callback?.()
          })
        })
      }
      .width(CommonConstants.FULL_PERCENT)
      .divider({
        color: $r('sys.color.ohos_id_divider_color'),
```

```
            strokeWidth: 1
          })
          Blank(CommonConstants.SPACE_16)
        }
        .padding(CommonConstants.SPACE_16)
        .width(CommonConstants.FULL_PERCENT)
    }

    aboutToDisappear(): void {

    }

}
import { MediaArgs } from "../detail/MediaArgs";
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 13:59
 * @Version V1.0
 * @Description
 */
export interface PlayerArgs extends MediaArgs {

    index: number

}
import router from '@ohos.router'
import { PlayerArgs } from './PlayerArgs'
import { CommonConstants } from '../../common/CommonConstants'
import { TimeTool, ToolBar } from '@jellyfin-harmony/core'
import { window } from '@kit.ArkUI'
import { AudioManager } from '../../player/AudioManager'
import { avSession } from '@kit.AVSessionKit'
import { image } from '@kit.ImageKit'
import { PlayListSheet } from './PlayListSheet'
```

```
@Entry
@Component
struct PlayerPage {
  private args: PlayerArgs = router.getParams() as PlayerArgs
  private audioManager: AudioManager = AppStorage.get(AudioManager.AUDIO_MANAGER)!
  private playlistSheet: PlayListSheet = new PlayListSheet(this.getUIContext())
  @StorageProp("currentAvoidArea") avoidArea?: window.AvoidArea = undefined
  @State state?: avSession.PlaybackState = avSession.PlaybackState.PLAYBACK_STATE_INITIAL
  @State modeRes: Resource = $r('sys.symbol.repeat')
  @State playRes: Resource = $r('sys.symbol.play_fill')
  @State favourite: boolean = false
  @State progress: number = 0
  @State duration: number = 100
  @State title?: string = undefined
  @State album?: string = undefined
  @State artist?: string = undefined
  @State art?: image.PixelMap | string = undefined

  private metadataCallback: ((data: avSession.AVMetadata) => void) = (metadata) => {
    this.updateMetadata(metadata)
  }

  private stateCallback: ((data: avSession.AVPlaybackState) => void) = (state) => {
    this.updatePlaybackState(state)
  }

  aboutToAppear(): void {
    this.title = this.args.name
    let metadata = this.audioManager.getAudioControl().getMetadata()
    if (metadata) {
      this.updateMetadata(metadata)
    }
    let playbackState = this.audioManager.getAudioControl().getPlaybackState()
    if (playbackState) {
```

```
      this.updatePlaybackState(playbackState)
    }
    this.audioManager.getAudioControl().onMetadataChange(this.metadataCallback)
    this.audioManager.getAudioControl().onPlaybackStateChange(this.stateCallback)
  }

  build() {
    Stack() {
      // title bar
      ToolBar()

      // content
      Column({ space: CommonConstants.SPACE_16 }) {
        // cover
        Stack() {

          Image(this.art)
            .width(CommonConstants.FULL_PERCENT)
            .height(CommonConstants.FULL_PERCENT)
            .alt($r('app.media.alt'))
            .objectFit(ImageFit.Cover)
            .autoResize(true)
            .borderRadius($r('app.float.lg_border_radius'))
            .aspectRatio(1)
            .margin(CommonConstants.SPACE_16)

        }
        .width(CommonConstants.FULL_PERCENT)
        .layoutWeight(1)

        // info
        Stack() {
          Column({space: CommonConstants.SPACE_8}) {
            Text(this.title)
              .fontSize($r('sys.float.Title_M'))
              .maxLines(1)
```

```
            Text(this.album)
              .fontSize($r('sys.float.Title_S'))
              .maxLines(1)
            Text(this.artist)
              .fontSize($r('sys.float.Title_S'))
              .maxLines(1)
          }
          .alignItems(HorizontalAlign.Start)
          .width(CommonConstants.FULL_PERCENT)

          SymbolGlyph(this.favourite ? $r('sys.symbol.heart_fill') : $r('sys.symbol.heart'))
            .fontSize(32)
            .fontColor([this.favourite ? $r('sys.color.ohos_id_color_badge_red') :
$r('sys.color.ohos_id_color_text_primary')])
            .onClick(() => {
              this.audioManager.getAudioControl().toggleFavourite()
            })
        }
        .alignContent(Alignment.TopEnd)

        // progress
        Row() {
          Text(TimeTool.formatMillionSecondsToHMS(this.progress))
          Blank()
            .layoutWeight(1)
          Text(TimeTool.formatMillionSecondsToHMS(this.duration))
        }

        Slider({
          value: this.progress >= 0 ? this.progress : 0,
          max: this.duration > 0 ? this.duration : 100
        })
          .enabled(this.state === avSession.PlaybackState.PLAYBACK_STATE_PLAY
            || this.state === avSession.PlaybackState.PLAYBACK_STATE_PAUSE)
          .onChange((num, mode) => {
            if (this.duration > 0 && mode === SliderChangeMode.End) {
```

```
        this.audioManager.getAudioControl().seek(num)
      }
    })


// operate
Row() {

  SymbolGlyph(this.modeRes)
    .fontSize(32)
    .fontColor([$r('sys.color.ohos_id_color_text_primary')])
    .onClick(() => {
      this.audioManager.getAudioControl().switchMode()
    })

  SymbolGlyph($r('sys.symbol.backward_end_fill'))
    .fontSize(32)
    .fontColor([$r('sys.color.ohos_id_color_text_primary')])
    .onClick(() => {
      this.audioManager.getAudioControl().pre()
    })

  SymbolGlyph(this.playRes)
    .fontSize(48)
    .fontColor([$r('sys.color.ohos_id_color_text_primary')])
    .onClick(() => {
      if (this.state === avSession.PlaybackState.PLAYBACK_STATE_PLAY) {
        this.audioManager.getAudioControl().pause()
      } else {
        this.audioManager.getAudioControl().play()
      }
    })

  SymbolGlyph($r('sys.symbol.forward_end_fill'))
    .fontSize(32)
    .fontColor([$r('sys.color.ohos_id_color_text_primary')])
```

```
            .onClick(() => {
              this.audioManager.getAudioControl().next()
            })

          SymbolGlyph($r('sys.symbol.music_note_list_fill'))
            .fontSize(32)
            .fontColor([$r('sys.color.ohos_id_color_text_primary')])
            .onClick(() => {
              this.playlistSheet.show()
            })
        }
        .alignItems(VerticalAlign.Center)
        .width(CommonConstants.FULL_PERCENT)
        .justifyContent(FlexAlign.SpaceBetween)

        Blank(CommonConstants.SPACE_16)
      }
      .width(CommonConstants.FULL_PERCENT)
      .height(CommonConstants.FULL_PERCENT)
      .padding({
        top: px2vp(this.avoidArea?.topRect.height),
        left: CommonConstants.SPACE_16,
        right: CommonConstants.SPACE_16,
        bottom: px2vp(this.avoidArea?.bottomRect.height)
      })
    }
    .alignContent(Alignment.Top)
    .width(CommonConstants.FULL_PERCENT)
    .height(CommonConstants.FULL_PERCENT)
  }

  private updateMetadata(metadata: avSession.AVMetadata) {
    this.title = metadata.title
    this.album = metadata.album
    this.artist = metadata.artist
    this.art = metadata.mediaImage
```

```
  }

  private updatePlaybackState(state: avSession.AVPlaybackState) {
    this.duration = state.duration !== undefined ? state.duration : -1
    this.progress = state.position !== undefined ? state.position.elapsedTime : -1
    this.state = state.state
    this.favourite = state.isFavorite ? true : false
    this.playRes = state.state === avSession.PlaybackState.PLAYBACK_STATE_PLAY ?
$r('sys.symbol.pause_fill') : $r('sys.symbol.play_fill')
    switch (state.loopMode) {
      case avSession.LoopMode.LOOP_MODE_SEQUENCE:
        this.modeRes = $r('sys.symbol.order_play')
        break
      case avSession.LoopMode.LOOP_MODE_SINGLE:
        this.modeRes = $r('sys.symbol.repeat_1')
        break
      case avSession.LoopMode.LOOP_MODE_LIST:
        this.modeRes = $r('sys.symbol.repeat')
        break
      case avSession.LoopMode.LOOP_MODE_SHUFFLE:
        this.modeRes = $r('sys.symbol.shuffle')
        break
      default :
        this.modeRes = $r('sys.symbol.repeat')
        break
    }
  }

  aboutToDisappear(): void {
    this.audioManager.getAudioControl().offMetadataChange(this.metadataCallback)
    this.audioManager.getAudioControl().offPlaybackStateChange(this.stateCallback)
  }

}
import { AppViewModel } from "../../lifecycle/AppViewModel";
```

```
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 12:56
 * @Version V1.0
 * @Description
 */
export class PlayerViewModel extends AppViewModel {

}
import { Sheet } from "@jellyfin-harmony/core";
import { ComponentContent, UIContext } from "@kit.ArkUI";
import { PlaylistPage } from "./list/PlaylistPage";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 19:43
 * @Version V1.0
 * @Description
 */
export class PlayListSheet extends Sheet {

  private callback: Callback<void> = () => {
    this.dismiss(true)
  }

  constructor(context: UIContext) {
    super(context)
  }

  protected onCreateContent(): ComponentContent<object> {
    return new ComponentContent(this.uiContext, wrapBuilder(playListComponent),
this.callback)
  }
```

```
}

@Builder
function playListComponent(callback: Callback<void>) {
  PlaylistPage({ callback: callback })
}
import { Repository } from '../../data/Repository'
import { ProgressDialog } from '@jellyfin-harmony/core'
import { promptAction, router } from '@kit.ArkUI'

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/7 21:21
 * @Version V1.0
 * @Description
 */
@Entry
@Component
struct ServerPage {

  private url: string = ""

  private progressDialog?: ProgressDialog

  private repository: Repository = AppStorage.get(Repository.REPOSITORY)!

  aboutToAppear(): void {
    this.progressDialog = new ProgressDialog(this.getUIContext())
  }

  build() {
    RelativeContainer() {
      TextInput({ placeholder: $r('app.string.server_address') })
        .id("server_address")
        .constraintSize({
```

```
      maxWidth: 300
    })
    .alignRules({
      middle: {anchor: "__container__", align: HorizontalAlign.Center},
      center: { anchor: "__container__", align: VerticalAlign.Center}
    })
    .onChange((value) => {
      this.url = value
    })

  Image($r('app.media.foreground'))
    .width(120)
    .height(120)
    .alignRules({
      middle: {anchor: "__container__", align: HorizontalAlign.Center},
      top: { anchor: "__container__", align: VerticalAlign.Top},
      bottom: { anchor: "server_address", align: VerticalAlign.Top}
    })


  Button($r('app.string.server_connect'))
    .id("server_connect")
    .width(120)
    .margin(15)
    .constraintSize({
      minWidth: 100
    })
    .alignRules({
      middle: {anchor: "__container__", align: HorizontalAlign.Center},
      top: { anchor: "server_address", align: VerticalAlign.Bottom}
    })
    .onClick(() => {
      this.connectServer()
    })

  LoadingProgress()
```

```
            .alignRules({
              start: {anchor: "server_connect", align: HorizontalAlign.Start},
              top: { anchor: "server_connect", align: VerticalAlign.Top},
              bottom: { anchor: "server_connect", align: VerticalAlign.Bottom}
            })
            .visibility(Visibility.Hidden)

        }
    }

    private connectServer() {
        this.progressDialog?.show()
        this.repository.connectServer(this.url)
          .then((serverInfo) => {
            this.progressDialog?.dismiss()
            let obj: Record<string, string> = {
              "url": this.url
            }
            router.pushUrl({url: "pages/login/LoginPage", params: obj})
          })
          .catch((error: Error) => {
            this.progressDialog?.dismiss()
            promptAction.showToast({message: error.message})
          })
    }

    aboutToDisappear(): void {
        this.progressDialog?.dismiss()
        this.progressDialog = undefined
    }

}
import { PrivacyDialog } from '@jellyfin-harmony/core'
import { router } from '@kit.ArkUI'
import { Repository } from '../../data/Repository'
import { AppPrefer } from '../../prefer/AppPrefer'
```

```
@Entry
@Component
struct SplashPage {

  private repository: Repository = AppStorage.get(Repository.REPOSITORY)!

  private appPrefer: AppPrefer = AppStorage.get(AppPrefer.PREFER)!

  private privacyDialog: PrivacyDialog = new PrivacyDialog(this.getUIContext())

  aboutToAppear(): void {
    if (this.appPrefer.isPrivacyGrant()) {
      setTimeout(() => {
        this.enterHomeOrAddServer()
      }, 1000)
    } else {
      let resourceManager = getContext().resourceManager
      let privacyUrl = resourceManager.getStringSync($r('app.string.privacy'))
      this.privacyDialog.setAppInfo(privacyUrl)
      this.privacyDialog.setPrivacyCallback(() => {
        this.appPrefer.setPrivacyGrant()
        this.privacyDialog.dismiss(true)
        setTimeout(() => {
          this.enterHomeOrAddServer()
        }, 1000)
      })
      this.privacyDialog.show()
    }
  }

  build() {

    Stack() {
      Image($r('app.media.foreground'))
        .size({
```

```
                width: 100,
                height: 100
            })
        }
        .width('100%')
        .height('100%')

    }

    onBackPress(): boolean | void {
        return true
    }

    private enterHomeOrAddServer() {
        let path = 'pages/server/ServerPage'
        if (this.repository?.getActiveInfo()) {
            path = 'pages/home/HomePage'
        }
        router.replaceUrl({url: path})
    }

}
import { AudioSession } from "./AudioSession";
import { avSession } from "@kit.AVSessionKit";
import { AudioQueue } from "./AudioQueue";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 17:27
 * @Version V1.0
 * @Description
 */
export class AudioControl {

    private readonly context: Context
```

```
    private readonly queue: AudioQueue

    private readonly session: AudioSession

    private sessionControl?: avSession.AVSessionController

    private metadata?: avSession.AVMetadata

    private playbackState?: avSession.AVPlaybackState

    private metadataChangeSet = new Set<(data: avSession.AVMetadata) => void>()

    private playbackStateChangeSet = new Set<(data: avSession.AVPlaybackState) =>
void>()

    constructor(context: Context, queue: AudioQueue, session: AudioSession) {
        this.context = context
        this.queue = queue
        this.session = session
    }

    public getMetadata(): avSession.AVMetadata | undefined {
        return this.metadata
    }

    public getPlaybackState(): avSession.AVPlaybackState | undefined {
        return this.playbackState
    }

    private getSession(): Promise<avSession.AVSession> {
        return this.session.getSession()
    }

    private async getSessionControl(): Promise<avSession.AVSessionController> {
        if (!this.sessionControl) {
```

```
        this.sessionControl = await this.session.getSessionControl()
        this.sessionControl.on('metadataChange', 'all', (metadata) => {
            this.metadata = metadata
            this.metadataChangeSet.forEach((callback) =>{
                callback(metadata)
            })
        })
        this.sessionControl.on('playbackStateChange', 'all', (state) => {
            this.playbackState = state
            this.playbackStateChangeSet.forEach((callback) =>{
                callback(state)
            })
        })
        this.sessionControl.on('sessionDestroy', () => {
            this.sessionControl = undefined
        })
    }
    return this.sessionControl
}

onMetadataChange(callback: (data: avSession.AVMetadata) => void) {
    this.metadataChangeSet.add(callback)
}

offMetadataChange(callback?: (data: avSession.AVMetadata) => void) {
    if (callback) {
        this.metadataChangeSet.delete(callback)
    } else {
        this.metadataChangeSet.clear()
    }
}

onPlaybackStateChange(callback: (data: avSession.AVPlaybackState) => void) {
    this.playbackStateChangeSet.add(callback)
}
```

```
  offPlaybackStateChange(callback?: (data: avSession.AVPlaybackState) => void) {
    if (callback) {
      this.playbackStateChangeSet.delete(callback)
    } else {
      this.playbackStateChangeSet.clear()
    }
  }

  public async switchMode(): Promise<void> {
    let control = await this.getSessionControl()
    return control.sendControlCommand({command: 'setLoopMode', parameter:
this.queue.getMode()})
  }

  public async toggleFavourite() {
    let control = await this.getSessionControl()
    control.sendControlCommand({command: "toggleFavorite", parameter:
this.metadata?.assetId})
  }

  public async playItem(id: string) {
    await this.getSessionControl()
    this.session.playFromAssetId(id)
  }

  public async play(): Promise<void> {
    let control = await this.getSessionControl()
    return control.sendControlCommand({command: 'play'})
  }

  public async pause(): Promise<void> {
    let control = await this.getSessionControl()
    return control.sendControlCommand({command: 'pause'})
  }

  public async seek(pos: number): Promise<void> {
```

```
        let control = await this.getSessionControl()
        return control.sendControlCommand({command: 'seek', parameter: pos})
    }

    public async next(): Promise<void> {
        let control = await this.getSessionControl()
        return control.sendControlCommand({command: 'playNext'})
    }

    public async pre(): Promise<void> {
        let control = await this.getSessionControl()
        return control.sendControlCommand({command: 'playPrevious'})
    }

}
import { BaseItemDto } from "@jellyfin-harmony/sdk"

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 17:09
 * @Version V1.0
 * @Description
 */
export interface AudioItem {

    id: string

    name: string

    url: string

    art: string

    extra: BaseItemDto
```

```
}
import { AudioControl } from './AudioControl'
import { AudioPlayer } from './AudioPlayer'
import { AudioQueue } from './AudioQueue'
import { AudioSession } from './AudioSession'
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 17:15
 * @Version V1.0
 * @Description
 */
export class AudioManager {

    public static readonly AUDIO_MANAGER = 'audioManager'

    private readonly audioQueue: AudioQueue

    private readonly audioControl: AudioControl

    private readonly audioSession: AudioSession

    private readonly audioPlayer: AudioPlayer

    constructor(context: Context) {
        this.audioPlayer = new AudioPlayer(context)
        this.audioQueue = new AudioQueue()
        this.audioSession = new AudioSession(context, this.audioQueue, this.audioPlayer)
        this.audioControl = new AudioControl(context, this.audioQueue, this.audioSession)
    }

    public getAudioQueue(): AudioQueue {
        return this.audioQueue
    }

    public getAudioSession(): AudioSession {
```

```
        return this.audioSession
    }

    public getAudioControl(): AudioControl {
        return this.audioControl
    }

    public release() {
        this.audioSession.release()
    }

}
import { media } from "@kit.MediaKit";
import { Repository } from "../data/Repository";
import { ErrorCallback } from "@ohos.base";
import { AudioItem } from "./AudioItem";
import { MediaProtocol, MediaSourceInfo } from "@jellyfin-harmony/sdk";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 17:00
 * @Version V1.0
 * @Description
 */
export class AudioPlayer {

    private readonly context: Context

    private readonly repository: Repository = AppStorage.get(Repository.REPOSITORY)!

    private avPlayer?: media.AVPlayer

    private avState: media.AVPlayerState = 'idle';

    private avTimer?: number = 0
```

```
  private avSpeed?: media.PlaybackSpeed

  private avData?: AudioItem = undefined

  private stateChangeCallback?: media.OnAVPlayerStateChangeHandle

  private progressCallback?: Callback<number>

  private errorCallback?: ErrorCallback

  constructor(context: Context) {
    this.context = context
  }

  private async requireAvPlayer(): Promise<media.AVPlayer> {
    if (this.avPlayer) {
      return this.avPlayer
    }
    // player
    this.avPlayer = await media.createAVPlayer()
    this.avPlayer.on('stateChange', (state, reason) => {
      console.log("VideoController: onStateChange state = " + state + ", reason = " +
reason)
      this.avState = state
      if (state === 'initialized') {
        this.avPlayer?.prepare()
      }
      if (state === 'prepared') {
        console.log("VideoController: duration = " + this.avPlayer?.duration)
        this.avPlayer!.videoScaleType                                                =
media.VideoScaleType.VIDEO_SCALE_TYPE_FIT_CROP
        if (this.avSpeed !== undefined) {
          this.avPlayer?.setSpeed(this.avSpeed)
        }
        this.avPlayer?.play()
```

```
        }
        if (state === 'playing') {
          this.startTimer()
        } else {
          this.stopTimer()
        }
        this.stateChangeCallback?.(state, reason)
      })
      this.avPlayer.on('seekDone', (position) => {
        console.log("VideoController: onSeekDone position = " + position)
      })
      this.avPlayer.on('bufferingUpdate', (type, value) => {
        // console.log("VideoController: onBufferUpdate type = " + type + ", value = " +
value)
      })
      this.avPlayer.on('trackChange', (index, selected) => {
        console.log("VideoController: onTrackChange index = " + index + ", selected = " +
selected)
      })
      this.avPlayer.on('durationUpdate', (duration) => {
        console.log("VideoController: onDurationUpdate duration = " + duration)
      })
      this.avPlayer.on('error', (error) => {
        console.log("VideoController: onError error = " + JSON.stringify(error))
        this.errorCallback?.(error)
      })
      return this.avPlayer
    }

    onProgressChange(callback: Callback<number>) {
      this.progressCallback = callback
    }

    offProgressChange() {
      this.progressCallback = undefined
    }
```

```
onStateChange(callback: media.OnAVPlayerStateChangeHandle) {
    this.stateChangeCallback = callback
}

offStateChange() {
    this.stateChangeCallback = undefined
}

onError(callback: ErrorCallback) {
    this.errorCallback = callback
}

offError() {
    this.errorCallback = undefined
}

private async printTrackInfo() {
    let info = await this.avPlayer?.getPlaybackInfo()
    console.log("VideoController: getPlaybackInfo = " + JSON.stringify(info) )
    let selected = await this.avPlayer!.getSelectedTracks()
    console.log("VideoController: getSelectedTracks = " + selected.join(",") )
    let tracks = await this.avPlayer!.getTrackDescription()
    tracks.forEach((info) => {
        console.log("VideoController: getTrackDescription = " + JSON.stringify(info) )
    })
}

private startTimer() {
    this.printTrackInfo()
    if (this.avTimer) return
    this.avTimer = setInterval(() => {
        this.progressCallback?.(this.getPosition())
    }, 1000)
}
```

```
private stopTimer() {
    if (this.avTimer) {
        clearInterval(this.avTimer)
        this.avTimer = undefined
    }
}

public getDuration(): number {
    if (this.avPlayer) {
        return this.avPlayer.duration
    }
    return 0
}

public getPosition(): number {
    if (this.avPlayer) {
        return this.avPlayer.currentTime
    }
    return 0
}

public getData(): AudioItem | undefined {
    return this.avData
}

public async getSource(id: string, sourceInfo: MediaSourceInfo): Promise<string> {
    let url: string | undefined | null
    switch (sourceInfo.Protocol) {
        case MediaProtocol.File:
            url = await this.repository.loadStreamUrl(id, sourceInfo.Id!)
            break
        case MediaProtocol.Http:
            url = sourceInfo.Path
            break
    }
    if (url) {
```

```
        return url
    }
    throw new Error("Source not support.")
}

public async setData(item: AudioItem): Promise<void> {
    let sourceResult = await this.repository.loadMediaSource(item.id)
    // let url = await this.getSource(item.id, (sourceResult!.MediaSources![0])!)
    let url = await this.repository.loadStreamUrl(item.id, item.id)
    console.log("VideoController: setData data 1" )
    let avPlayer = await this.requireAvPlayer()
    console.log("VideoController: setData data 2")
    avPlayer.url = url
    console.log("VideoController: setData data 3 url = " + avPlayer.url)
}

public async prepare(): Promise<void> {
    let avPlayer = await this.requireAvPlayer()
    console.log("VideoController: prepare 1" )
    await avPlayer.prepare()
    console.log("VideoController: prepare 2" )
}

public getState(): media.AVPlayerState {
    return this.avState
}

public isPrepared(): boolean {
    return false
}

public getSpeed(): media.PlaybackSpeed {
    if (this.avSpeed === undefined) {
        return media.PlaybackSpeed.SPEED_FORWARD_1_00_X
    }
    return this.avSpeed
```

```
  }

  public setSpeed(speed: media.PlaybackSpeed) {
    this.avSpeed = speed
    this.avPlayer?.setSpeed(speed)
  }

  public async getTrackInfo(type: media.MediaType):
Promise<Array<media.MediaDescription>> {
    let trackArray = new Array<media.MediaDescription>()
    if (this.avPlayer) {
      let allTracks = await this.avPlayer!.getTrackDescription()
      allTracks.forEach((track) => {
        let trackType = track[media.MediaDescriptionKey.MD_KEY_TRACK_TYPE] as
media.MediaType
        if (trackType === type) {
          trackArray.push(track)
        }
      })
    }
    return trackArray
  }

  public async getCurrentTrack(): Promise<Array<number>> {
    if (this.avPlayer) {
      return this.avPlayer.getSelectedTracks()
    } else {
      return []
    }
  }

  public selectTrack(index: number) {
    this.avPlayer?.selectTrack(index)
  }

  public async start(): Promise<void> {
```

```
        await this.avPlayer?.play()
    }

    public async seek(position: number): Promise<void> {
        console.log("VideoController: seek " + position)
        this.avPlayer?.seek(position, media.SeekMode.SEEK_CLOSEST)
    }

    public async pause(): Promise<void> {
        await this.avPlayer?.pause()
    }

    public async stop(): Promise<void> {
        await this.avPlayer?.stop()
    }

    public async reset(): Promise<void> {
        await this.avPlayer?.reset()
    }

    public async release(): Promise<void> {
        this.progressCallback = undefined
        this.errorCallback = undefined
        this.stateChangeCallback = undefined
        await this.avPlayer?.release()
    }

}
import { AudioItem } from "./AudioItem";
import { avSession } from "@kit.AVSessionKit";

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 17:08
 * @Version V1.0
```

```
 * @Description
 */
export class AudioQueue {

  private loopMode = avSession.LoopMode.LOOP_MODE_SEQUENCE

  private audioQueue = new Array<AudioItem>()

  private audioIndex = -1

  public setCurrentItem(id: string) {
    this.audioIndex = this.audioQueue.findIndex((value) => value.id == id)
  }

  public setMode(mode: avSession.LoopMode) {
    this.loopMode = mode
  }

  public getMode(): avSession.LoopMode {
    return this.loopMode
  }

  public addItem(items: AudioItem[]) {
    this.audioQueue.push(...items)
  }

  public resetItem(items: AudioItem[]) {
    this.audioQueue.splice(0)
    this.audioQueue.push(...items)
  }

  public deleteItem(item: AudioItem) {
    let index = this.audioQueue.indexOf(item)
    if (index >= 0) {
      this.audioQueue.splice(index, 1)
    }
```

```
}

public deleteAll() {
    this.audioQueue.splice(0)
    this.audioIndex = -1
}

public getQueue(): Array<AudioItem> {
    return this.audioQueue
}

public getItem(index: number): AudioItem {
    return this.audioQueue[index]
}

public getItemById(id: string): AudioItem | undefined {
    return this.audioQueue.find((value) => value.id == id)
}

public getCurrent(): AudioItem | undefined {
    if (this.audioIndex < 0 || this.audioIndex >= this.audioQueue.length) {
        return undefined
    }
    return this.audioQueue[this.audioIndex]
}

private getRandom(): AudioItem | undefined {
    if (this.audioQueue.length > 0) {
        let index = Date.now() % this.audioQueue.length
        return this.getItem(index)
    }
    return undefined
}

public getPre(): AudioItem | undefined{
    if (this.loopMode === avSession.LoopMode.LOOP_MODE_SHUFFLE) {
```

```
            return this.getRandom()
        } else {
            let index = this.audioIndex - 1
            if (index < 0) {
                index = this.audioQueue.length - 1
            }
            return this.getItem(index)
        }
    }

    public getNext(): AudioItem | undefined{
        if (this.loopMode === avSession.LoopMode.LOOP_MODE_SHUFFLE) {
            return this.getRandom()
        } else {
            let index = this.audioIndex + 1
            if (index >= this.audioQueue.length) {
                index = 0
            }
            return this.getItem(index)
        }
    }

}
```