```
import { AbilityConstant, UIAbility, Want } from '@kit.AbilityKit';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { window } from '@kit.ArkUI';

export default class EntryAbility extends UIAbility {
  onCreate(want: Want, launchParam: AbilityConstant.LaunchParam): void {
    hilog.info(0x0000, 'testTag', '%{public}s', 'Ability onCreate');
  }

  onDestroy(): void {
    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.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 {
    // 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-ad
aptation-V5#section1421172621111

this.context.getApplicationContext().setColorMode(ConfigurationConstant.ColorMode.COLOR_M
ODE_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 + '-book',
      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.Books) {
            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.Books]
    })
    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 loadBook(id: string): Promise<BaseItemDto> {
        return this.jellyfin.getUserLibraryApi().getItem({
            userId: this.activeInfo!.userInfo.userId,
            itemId: id
        })
    }

    public getAuth(): string {
        return this.jellyfin.apiClient.getAuthorizationHeader()
    }

    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.getVideosApi().getVideoStreamUrl({
            itemId: id,
            static: true,
            mediaSourceId: sourceId
        })
    }
    public loadBookUrl(id: string, sourceId: string): Promise<string> {
        return this.jellyfin.getLibraryApi().getFile({
            itemId: id,
        })
    }

    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 { 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=1574188982
982436096")
        })
    }
    .alignItems(HorizontalAlign.Start)
    .width(CommonConstants.FULL_PERCENT)
    .height(CommonConstants.FULL_PERCENT)
  }
}
/**
 * @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({space: CommonConstants.SPACE_8}) {
            Image(this.repository?.buildImage(this.item!, ImageType.Primary))
                .alt($r('app.media.alt'))
                .objectFit(ImageFit.Cover)
                .autoResize(true)
                .borderRadius(CommonConstants.SPACE_16)
                .width('100%')
                .aspectRatio(0.67)

            Text(this.item?.Name)
                .width('100%')
                .maxLines(1)
        }
        .onClick(() => {
            let args: MediaArgs = {
                id: this.item?.Id!,
                name: this.item?.Name!,
        }
        router.pushUrl({url: 'pages/read/ReaderPage', 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.Book], 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 router from '@ohos.router'
import { window } from '@kit.ArkUI'
import { ScrollRefreshView, ToolBar } from '@jellyfin-harmony/core'
import { CommonConstants } from '../../common/CommonConstants'
import { ReaderViewModel } from './ReaderViewModel'
import { MediaArgs } from '../detail/MediaArgs'
import { PDFView } from '../../widget/pdf/PDFView'

@Entry
@Component
struct ReaderPage {
    private args: MediaArgs = router.getParams() as MediaArgs

    private viewModel = new ReaderViewModel(getContext(), this.args)

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

    aboutToAppear(): void {
    }

    @Builder
    bookView(path: string) {
        PDFView({
                pdfPath: path,
        })
```

```
  }

  build() {
    Column() {
      ToolBar({title: this.args.name})
      ScrollRefreshView({
        source: this.viewModel,
        itemLayout: (url: string) => {
          this.bookView(url)
        }
      })
        .layoutWeight(1)
        .width(CommonConstants.FULL_PERCENT)
    }
    .width(CommonConstants.FULL_PERCENT)
    .height(CommonConstants.FULL_PERCENT)
  }

  aboutToDisappear(): void {
  }

}
import { LoadType, SimpleRepository } from "@jellyfin-harmony/core";
import { AppViewModel } from "../../lifecycle/AppViewModel";
import { MediaArgs } from "../detail/MediaArgs";
import { request } from "@kit.BasicServicesKit";
import fs from '@ohos.file.fs';
/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/12/7 12:56
 * @Version V1.0
 * @Description
 */
export class ReaderViewModel extends AppViewModel implements SimpleRepository {

  private readonly args: MediaArgs

  private downloadTask?: request.agent.Task

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

```
async loadData(type: LoadType): Promise<string> {
    let item = await this.repository.loadBook(this.args.id)
    let url = await this.repository.loadBookUrl(item.Id!, item.Id!)
    let filePath = this.context.cacheDir + "/" + item.Id!;
    let exists = await fs.access(filePath, fs.AccessModeType.EXIST)
    if (exists) {
        return filePath
    } else {
        // download
        return this.download(url, filePath)
    }
}

private async download(url: string, path: string): Promise<string> {
    if (this.downloadTask) {
        await this.downloadTask.stop()
    }
    let tmpPath = path + ".tmp"
    let tmpExists = await fs.access(tmpPath, fs.AccessModeType.EXIST)
    if (tmpExists) {
        console.log("FinBook download clear tmp.")
        await fs.unlink(tmpPath)
    }
    this.downloadTask = await request.agent.create(this.context, {
        action: request.agent.Action.DOWNLOAD,
        url: url,
        mode: request.agent.Mode.FOREGROUND,
        headers: {
            "Accept": "*/*",
            "Accept-Encoding": "gzip, deflate",
            "Accept-Language": "zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7",
            "Authorization": this.repository.getAuth()
        },
        overwrite: true,
        saveas: tmpPath
    })
    let promise = new Promise<string>((resolve, reject) => {
        this.downloadTask?.on('response', (response) => {
            console.log("FinBook download response: " + JSON.stringify(response))
        })
        this.downloadTask!.on('completed', () => {
            console.log("FinBook download complete.")
            fs.renameSync(tmpPath, path)
```

```
                resolve(path)
            })
            this.downloadTask!.on('failed', (progress) => {
                console.log("FinBook download error: " + JSON.stringify(progress))
                reject(new Error("Download fail " + progress.state))
            })
        })
        this.downloadTask.start()
        return promise
    }

}
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 { preferences } from "@kit.ArkData"

/**
  * @Author peerless2012
```

```
 * @Email peerless2012@126.com
 * @DateTime 2024/11/9 18:36
 * @Version V1.0
 * @Description
 */
export class AppPrefer {

  public static readonly PREFER = "prefer"

  private static readonly KEY_PRIVACY_GRANT = "privacy_grant"

  private readonly preference: preferences.Preferences

  constructor(context: Context) {
    this.preference = preferences.getPreferencesSync(context, { name: "app_prefer" } )
  }

  public isPrivacyGrant(): boolean {
    if (this.preference.hasSync(AppPrefer.KEY_PRIVACY_GRANT)) {
      return this.preference.getSync(AppPrefer.KEY_PRIVACY_GRANT, false) as boolean
    }
    return false
  }

  public setPrivacyGrant(): void {
    this.preference.putSync(AppPrefer.KEY_PRIVACY_GRANT, true)
    this.preference.flush()
  }

}
import { StartupConfig, StartupConfigEntry } from '@kit.AbilityKit';
import { BusinessError } from '@ohos.base';

/**
 * @Author peerless2012
 * @Email peerless2012@126.com
 * @DateTime 2024/11/9 21:20
 * @Version V1.0
 * @Description
 */
export class AppStartupConfig extends StartupConfigEntry {

  private onCompleteCallback = (error: BusinessError<void>) => {
    console.log("AppStartupConfig: onComplete.")
    }

    onConfig(): StartupConfig {
        console.log("AppStartupConfig: onConfig.")
        let config: StartupConfig = {
            timeoutMs: 10000,
            startupListener: {onCompleted: this.onCompleteCallback}
        }
        return config
    }

}
import { common, StartupTask } from "@kit.AbilityKit";
import { Repository } from "../data/Repository";

/**
  * @Author peerless2012
  * @Email peerless2012@126.com
  * @DateTime 2024/11/10 1:04
  * @Version V1.0
  * @Description
  */
@Sendable
export class AppStartupTask extends StartupTask {

    public async init(context: common.AbilityStageContext): Promise<void> {
        console.log("AppStartupTask: init 1.")
        let repository = new Repository(context.getApplicationContext())
        AppStorage.setOrCreate(Repository.REPOSITORY, repository)
        try {
            await repository.init()
        } catch (error) {
            console.error("AppStartupTask: init error " + error['message'])
        }
        console.log("AppStartupTask: init 2.")
    }

}
import { CommonConstants } from "../../common/CommonConstants"
import { router, window } from "@kit.ArkUI"

@Component
export struct HomeToolBar {
```

```
@Require @Prop title: ResourceStr

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

build() {
  Column() {
    Column()
      .width('100%')
      .height(px2vp(this.avoidArea?.topRect.height))

    Row() {
      Text(this.title)
        .fontSize($r('app.float.header_font_size'))
        .fontColor($r('sys.color.ohos_id_color_text_primary'))
        .fontWeight(FontWeight.Bold)
        .textAlign(TextAlign.Start)
        .fontFamily(CommonConstants.HARMONY_HEITI_BOLD_FONT_FAMILY)
        .margin({ right: $r('app.float.sm_padding_margin') })

      // Search({ placeholder: $r('app.string.search') })
      //     .focusable(false)
      //     .fontColor($r('sys.color.ohos_id_color_text_primary'))
      //     .textFont({ size: $r('app.float.large_text_size') })
      //     .width($r('app.float.search_width'))
      //     .height($r('app.float.search_height'))
      //     .onClick(() => {
      //         this.getUIContext().getRouter().pushUrl({
      //             url: "pages/search/SearchPage"
      //         })
      //     })

      SymbolGlyph($r('sys.symbol.more'))
        .fontSize($r('app.float.header_font_size'))
        .fontColor([$r('sys.color.ohos_id_color_text_primary')])
        .onClick(() => {
          router.pushUrl({url: "pages/about/AboutPage"})
        })
    }
    .padding({
      left: $r('app.float.xxl_padding_margin'),
      right: $r('app.float.lg_padding_margin'),
    })
    .justifyContent(FlexAlign.SpaceBetween)
    .width(CommonConstants.FULL_PERCENT)
```

```
            .height($r('app.float.top_navigation_height'))
        }
    }

}
```