

iOS 点播回放 Core SDK

- [BJVideoPlayerCore SDK](#) 支持百家云 点播、回放 的 播放及 下载 功能，不包含 UI 资源，包含 UI 的 SDK 请参考 [iOS 点播 UI SDK](#) 及 [iOS 回放 UI SDK](#)。SDK 4.x 版本最低支持 iOS 11.0 及以上系统，旧版的 3.x 支持 iOS 10.0 以上系统。
- git 链接: <https://git2.baijiashilian.com/open-ios/BaijiaYun.git>
- App 下载:
<https://itunes.apple.com/app/id1146697098?ls=1&mt=8>
- 旧版SDK: [iOS 点播回放 Core SDK 2.0](#)
- 变更记录: [changelog](#)

Demo

点播、回放的 UI SDK 是开源的，BJVideoPlayerCore 的集成可参考 [BJVideoPlayerUI SDK demo](#)中 Core SDK 的集成方式。

引入 SDK

SDK 支持 iOS 11.0 及以上 的系统，iPhone、iPad 等设备，集成 2.0 或以上版本的 SDK 要求 Xcode 的版本至少为 9.0，集成 2.2.0 或以上版本的 SDK 要求 Xcode 的版本至少为 11.0，

由于 SDK 依赖关系复杂、手动配置繁琐，建议使用 CocoaPods 方式引入。存在 Cocoapods 导入失败的 spec 仓库时，通常可以查找镜像源，替换本地的 Cocoapods 中对应 spec 仓库的 source 的指向来解决。

Podfile 配置

- Podfile 中设置 source

```
1. source
   'https://github.com/CocoaPods/Specs.git'
2. source 'https://git2.baijiashilian.com/open-
   ios/specs.git'
```

- Podfile 中引入 BJVideoPlayerCore :

```
1. pod 'BaijiaYun/BJVideoPlayerCore',
   '~>3.24.0'
```

项目全局设置

1. 配置ATS

需要在 info.plist 里面增加 NSAllowsArbitraryLoads = true

2. （可选）开启后台音频

如果有后台播放的需求，则必须开启此项：在 Project > Target > Signing & Capabilities 中打开 Background Modes 开关，选中 Audio, Airplay, and Picture in Picture。

Modes ☒ Audio, AirPlay, and Picture in Picture

- ☐ Location updates
- ☐ Voice over IP
- ☐ External accessory communication
- ☐ Uses Bluetooth LE accessories
- ☐ Acts as a Bluetooth LE accessory
- ☐ Background fetch
- ☐ Remote notifications
- ☐ Background processing

主要功能

SDK 功能主要分为点播、回放及下载三个部分。

要点说明

1. 设置专属域名前缀

例如专属域名为“demo123.at.baijiayun.com”，则前缀为“demo123”，参考[专属域名说明](#)。

```
1. #import <BJVideoPlayerCore/BJVAppConfig.h>
2.
3. ...
4.
5. /** 设置专属域名前缀 */
6. [[BJVAppConfig sharedInstance]
   setPrivateDomainPrefix:loginUser.privateDomainPre
```

2. （可选）设置 BJVRequestTokenDelegate

SDK 的点播、回放及下载功能都需要集成方传入视频 token 来向服务器请求数据，数据对应的 token 由集成方后台调用百度云 API 获取：点播 token、回放 token，然后通知给 SDK，SDK 使用 token 等参数调用 SDK 的相关 API 实现功能。

集成方可配置 `token` 的有效期，当 SDK 发现点播、回放、下载的连接过期或不可用时，会通过

`BJVRequestTokenDelegate` 的方法回调，要求给对应的数据传入一个可用的 `token`。此时如果集成方在 `delegate` 的对应方法里重新获取一个可用的 `token` 返回给 SDK，SDK 就能继续正常工作，这样就实现了自动更新 `token` 的效果。如果集成方没有 `token` 过期和自动更新的需求，可以不设置此 `delegate`，SDK 会在发现链接过期或不可用时尝试使用旧的 `token` 重新请求服务器，请求失败将抛出

`BJVErrorCode_invalidToken` 错误，APP 执行错误处理的业务逻辑。

```
1. /** 用于在 点播播放/下载 中，集成方需要调用
    completion 提供 videoID 对应的 token */
2. - (void)requestTokenWithVideoID:(NSString
    *)videoID
3.      completion:(void (^)(NSString *
    _Nullable token, NSError * _Nullable
    error))completion {
4.     // 示例：集成方获取点播 token 的方法
5.     [self getTokenWithVideoID:videoID
6.      completion:^(NSString *token,
7.      NSError *error) {
8.          completion(token, error);
9.      }];
10. }
11. /** 用于在 回放播放/下载 中，客户需要调用
    completion 提供 classID、sessionID 对应的 token
    */
12. - (void)requestTokenWithClassID:(NSString
    *)classID
13.      sessionID:(nullable NSString
    *)sessionID
```

```

14.         completion:(void (^)(NSString *
_Nullable token, NSError * _Nullable
error))completion {
15.     // 示例：集成方获取回放 token 的方法
16.     [self getTokenWithClassID:classID
17.         sessionID:sessionID
18.         completion:^(NSString *token,
NSError *error) {
19.             completion(token, error);
20.         }];
21. }

```

3. 属性、方法监听方式：Block

demo 及示例代码中大量使用特定的 block 进行属性、方法监听，可参考 [直播文档](#)中的对监听 block 的相关介绍。

点播功能说明

建议： 自主集成点播功能之前，可参考 [BJVideoPlayerUI SDK](#) 已经开源的代码，该 UI SDK 提供了清晰的点播模块 API 调用方式。

1. 引入头文件

```

1. #import
<BJVideoPlayerCore/BJVideoPlayerCore.h>

```

2. 设置专属域名

参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

3. （可选）设置

`BJVRequestTokenDelegate`，管理 token

参考[要点说明](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

4. 初始化播放器管理类 `BJVPlayerManager`

4.1 指定播放器类型，创建点播 manager 实例

点播播放器分为 `AVPlayer` 与 `IJKPlayer` 两种，参考 `BJVPlayerType`。

`AVPlayer` 是 iOS 系统的播放器，支持 m3u8 格式的视频，加载比较快，`IJKPlayer` 不支持。

`IJKPlayer` 是第三方开源播放器，基于 `FFmpeg`，支持加密和 flv 格式的视频，`AVPlayer` 不支持。

1. **/** 初始化 manager 实例 */**
2. **- (instancetype)initWithPlayerType:**
(BJVPlayerType)playerType;
- 3.
4. **/****
5. **播放器类型：AVPlayer(不支持加密视频播放)、**
IJKPlayer
6. **#discussion** 当播放器初始设置为 **AVPlayer** 时，但是播放的视频实际为加密格式，
7. **#discussion** SDK 将会切换播放器的类型为 **IJKPlayer** 来尽量保证可以正常播放视频
8. **#discussion** 因此，在获取播放器的属性时，不能作为一个属性存储，需要实时获取，
9. **#discussion** 或者监听播放器类型的变化，及时获取到新的属性。

```
10. */
11. @property (nonatomic, readonly) BJVPlayerType
    playerType;
```

```
1. // example: 创建使用 AVPlayer 的 playerManager
2. self.playerManager = [[BJVPlayerManager alloc]
    initWithPlayerType:BJVPlayerType_AVPlayer];
```

4.2 初始化点播

- 初始化在线点播

```
1. /**
2.  初始化在线视频
3.  #param videoID 视频 ID
4.  #param token 需要集成方后端调用百家云后端的API
    获取，传给移动端
5.  #discussion 调用此方法初始化在线视频时，默认不加
    密、不使用集成方鉴权
6.  */
7. - (void)setupOnlineVideoWithID:(NSString
    *)videoID
8.      token:(NSString *)token;
9.
10. /**
11.  初始化在线视频，设置是否加密、集成方鉴权
12.  #param videoID 视频 ID
13.  #param token 需要集成方后端调用百家云后端的API
    获取，传给移动端
14.  #param encrypted 是否加密，「仅在使用 IJKPlayer
    时有效」，参考 initWithPlayerType: 方法
15.  #param accessKey 集成方鉴权，视频如果需要请求第
    三方服务器查看是否有权限，可设置该参数。鉴权验证请
    求需要与百家云后台沟通。
```

```

16. */
17. - (void)setupOnlineVideoWithID:(NSString
    *)videoID
18.         token:(NSString *)token
19.         encrypted:(BOOL)encrypted
20.         accessKey:(nullable NSString
    *)accessKey;

```

- 初始化本地点播

```

1. /**
2.  初始化本地视频
3.
4.  #param downloadItem 本地视频文件类型，通过下
    载模块获得
5.  */
6. [self.playerManager
    setupLocalVideoWithDownloadItem:downloadItem];

```

5. 播放设置

播放器相关设置在 `BJVPlayProtocol.h` 文件中定义。

5.1 用户信息设置

```

1. /**
2.  第三方用户名和编号
3.  #discussion 需要在调用 setupOnlineVideoWithID:
    之前设置
4.  */
5. @property (nonatomic, strong, nullable) NSString
    *userName;
6. @property (nonatomic, strong, nullable) NSString
    *userNumber;

```

5.2 视频信息设置和读取

assign属性的信息需要在调用setupOnlineVideoWithID之前设置。

```
1. /**
2. 视频的起始播放时间
3. #discussion 需要在实例化 playerManager 对象后设置
4. #discussion 设置为大于 0 的值时，记忆播放 playTimeRecordEnabled 无效
5. #discussion !!!: 使用同一个 playerManager 播放不同视频时，该值不会自动重置，需要根据实际情况重新设置
6. */
7. @property (nonatomic, assign) NSTimeInterval
   initialPlayTime;
8.
9. /**
10. 播放倍速
11. #discussion 有效区间: [0.5, 2.0]
12. */
13. @property (nonatomic, assign) CGFloat rate;
14.
15. /**
16. 偏好清晰度列表
17.
18. #discussion 优先使用此列表中的清晰度播放在线视频，优先级按数组元素顺序递减
19. #discussion preferredDefinitionList 列表元素为清晰度的标识字符串，现有标识符: low（标清），
   high（高清），superHD（超清），720p，1080p，
   audio（纯音频），可根据实际情况动态扩展
20. #discussion 此设置对播放本地视频无效
21. */
```

```

22. @property (nonatomic, strong)
    NSArray<NSString *> *preferredDefinitionList;
23.
24. /**
25.  进入后台时，是否继续播放音频，必须在工程的
    background modes 中添加 audio 才会生效
26.  note: 当设置 backgroundAudioEnabled 为 YES 时，
    内部会把 pictureInPictureEnabled 置为 NO
27. */
28. @property (nonatomic, assign) BOOL
    backgroundAudioEnabled;
29.
30. /** 是否开启记忆播放 */
31. @property (nonatomic, assign) BOOL
    playTimeRecordEnabled;
32.
33. /** 记忆播放时间，当播放器设置正确的播放数据源后
    才能返回正常数据 */
34. @property (nonatomic, readonly) NSTimeInterval
    playTimeRecord;
35.
36. /** 清除播放记录 */
37. - (void)clearPlayTimeRecords;
38.
39. /**
40.  开启记忆播放时，视频剩余时间不足
    ignorableRemainingTimeInterval 秒时，
41.  视为播放结束，清除该视频的播放进度纪录，默认 5s，
    可根据需要自行设置
42. */
43. @property (nonatomic, assign) NSTimeInterval
    ignorableRemainingTimeInterval;

```

5.3 视频播放信息

```
1. /** 当前播放状态 */
2. @property (nonatomic, readonly) BJVPlayerStatus
   playStatus;
3.
4. /** 视频的时长 */
5. @property (nonatomic, readonly) NSTimeInterval
   duration;
6.
7. /** 视频缓存时长 */
8. @property (nonatomic, readonly) NSTimeInterval
   cachedDuration;
9.
10. /** 正片的当前的播放时间 */
11. @property (nonatomic, readonly) NSTimeInterval
   currentTime;
12.
13. /** 视频信息 */
14. @property (nonatomic, readonly, nullable)
   BJVPlayInfo *playInfo;
15.
16. /** 当前播放清晰度 */
17. @property (nonatomic, readonly, nullable)
   BJVDefinitionInfo *currDefinitionInfo;
18.
19. /** 当前使用的字幕信息 */
20. @property (nonatomic, readonly, nullable)
   BJVSubtitleInfo *currentSubtitleInfo;
21.
22. /** 当前时间点的字幕内容 */
23. @property (nonatomic, readonly, nullable)
   BJVSubtitle *currentSubtitle;
24.
25. /** 进入后台或锁屏时，是否支持开启远程控制，仅在使用
    后台播放时有效，默认关闭 */
```

```

26. @property (nonatomic, assign) BOOL
    remoteControlEnabled;
27.
28. /** 设置远程控制状态下的封面 */
29. @property (nonatomic, strong) UIImage
    *remoteControllImage;

```

```

1. // example: 监听播放状态变化
2. bjl_weakify(self);
3. [self bjl_kvo:BJLMakeProperty(self.playerManager,
    playerStatus) filter:^(NSNumber * _Nullable
    now, NSNumber * _Nullable old,
    BJLPropertyChange * _Nullable change) {
4.     return now.integerValue != old.integerValue;
5. } observer:^(NSNumber * _Nullable now,
    NSNumber * _Nullable old, BJLPropertyChange *
    _Nullable change) {
6.     bjl_strongify(self);
7.     BJVPlayerStatus status = now.integerValue;
8.     // 播放状态变化的自定义处理
9.     [self playerStatusDidChangeTo:status];
10.    return YES;
11. }];

```

5.4 视频画面

```

1. /** 播放视图 */
2. @property (nonatomic, readonly, nullable) UIView
    *playerView;
3.
4. /** 播放视图显示模式 */
5. @property (nonatomic)
    BJVPlayerViewScalingMode scalingMode;

```

5.5 视频控制

```
1. - (void)play;
2. - (void)pause;
3. - (void)seek:(NSTimeInterval)time;
4. /**
5.  切换清晰度
6.  #param index 目标清晰度 在 BJVPlayInfo 的
    definitionList 数组中的序号
7. */
8. - (void)changeDefinitionWithIndex:
    (NSUInteger)index;
9. -
10. /** 切换字幕文件
11.  #param index 目标字幕 在 BJVPlayInfo 的
    subtitleInfo 数组中的序号
12. */
13. - (void)changeSubtitleWithIndex:
    (NSUInteger)index;
14.
15. /**
16.  清理播放器，重置状态、清空数据
17. */
18. - (void)reset;
19. -
20. /**
21.  销毁播放器
22.  #discussion 调用后彻底销毁播放器，不可恢复
23.  #discussion 可在退出播放界面前调用
24. */
25. - (void)destroy;
```

5.6. 视频播放中的错误监听

```

1. /*
2.  视频播放出错
3.  @param playInfo 出错视频信息
4.  @param error 错误信息
5.  */
6. - (BJLObservable)video:(BJVPlayInfo *)playInfo
   playFailedWithError:(NSError *)error;
7.
8. // example: 监听视频播放出错
9. [self
   bjl_observe:BJLMakeMethod(self.room.playerManag
   video:playFailedWithError:)
   observer:^(BOOL(BJVPlayInfo *playInfo, NSError
   *error) {
10.     bjl_strongify(self);
11.     // 自定义播放错误处理方法
12.     [self video:playInfo playFailedWithError:error];
13.     return YES;
14. }]];

```

- 错误码信息

```

1. typedef NS_ENUM(NSInteger, BJVErrorCode) {
2.     BJVErrorCode_unknown      = -999,    // 未知错误
3.     BJVErrorCode_requestFailed = 1000,    // 网络请求失败
4.     BJVErrorCode_invalidToken  = 1010,    // token 参数错误
5.     BJVErrorCode_invalidAccessKey = 1020,  // 鉴权验证失败
6.     BJVErrorCode_invalidPlayInfo = 1030,  // 播放信息解析错误
7.     BJVErrorCode_invalidVideoURL = 1040,  // 在线播放, 视频地址失效

```

```

8.   BJVErrorCode_fileLost      = 1050,    // 本地
    播放，播放文件不存在
9.   BJVErrorCode_playFailed    = 1060     // 视频
    播放失败
10. };

```

5.7 画中画

画中画功能仅支持 `BJVPlayerType_AVPlayer` 类型的播放器。

```

1. /**
2.  开启画中画
3.  note: 当前的播放器类型为 BJVPlayerType_AVPlayer
    且 backgroundAudioEnabled == NO 时, 设置
    pictureInPictureEnabled 才有效
4.  note: 当设置 backgroundAudioEnabled 为 YES 时,
    内部会把 pictureInPictureEnabled 置为 NO
5.  */
6. @property (nonatomic, assign) BOOL
    pictureInPictureEnabled;
7.
8. /** 画中画视图控制器, 可能为空 */
9. @property (nonatomic, strong, nullable,
    readonly) AVPictureInPictureController
    *pipViewContrller;
10.
11. /** 启动画中画, 仅 AVPlayer 有效 */
12. - (void)startPictureInPicture;
13.
14. /** 关闭画中画, 仅 AVPlayer 有效 */
15. - (void)stopPictureInPicture;

```

5.8 点播打点

```

1.
2. /**
3. 点播：请求视频打点数据
4.
5. #param fids      视频fid
6. #param completion 请求完成回调
7. #return          请求的 task
8. */
9. - (NSURLSessionTask
   *)getVideoCatalogueInfoWithVideoVID:(NSString
   *)vid
10.                                token:(NSString
   *)token
11.                                completion:(nullable
   void (^)(NSURLSessionDataTask *_Nullable task,
12.                                NSArray<BJVVideoCatalogueItem *> *_Nullable
   list,
13.                                NSError *_Nullable error))completion;

```

6. 添加播放视图

播放视图通过 `BJVPlayerManager` 的 `playerView` 属性获取，布局自定。

6.1 添加视图

```

1. [self.view
   addSubview:self.playerManager.playerView];

```

6.2 布局示例：根据当前清晰度获取视频宽高比，使视频画面充满父视图 `superView`

```
1. bjl_weakify(self);
2. [self
    bjl_kvo:BJLMakeProperty(self.room.playerManager,
        currDefinitionInfo)
3.     observer:^(BOOL(id _Nullable now, id
        _Nullable old, BJLPropertyChange * _Nullable
        change) {
4.         bjl_strongify(self);
5.         // 根据当前清晰度获取视频宽高比
6.         BJVDefinitionInfo *currDefinitionInfo =
            self.room.playerManager.currDefinitionInfo;
7.         CGFloat width = currDefinitionInfo.width;
8.         CGFloat height = currDefinitionInfo.height;
9.         CGFloat videoRatio = height > 0.0 ? width /
            height : 0.0;
10.
11.         // 更新播放视图布局
12.         [self.playerManager.playerView
            bjl_remakeConstraints:^(BJLConstraintMaker
            *make) {
13.             if (videoRatio > 0) {
14.
15.                 make.edges.equalTo(superview).priorityHigh();
16.                 make.center.equalTo(superview);
17.
18.                 make.top.left.greaterThanOrEqualTo(superview);
19.
20.                 make.bottom.right.lessThanOrEqualTo(superview);
21.
22.                 make.width.equalTo(playerView.bjl_height).multipliedBy(videoRatio);
23.             }
24.             else {
25.                 make.edges.equalTo(superview);
26.             }
27.         }
28.     }];
```

```
24.  
25.     return YES;  
26. };
```

7. 播放器清理、销毁

- 清理

调用点播初始化方法 `setupVideo` 时将自动调用，其它情况可根据实际需要调用。

```
1. // 清理播放器，重置状态、清空数据  
2. [self.playerManager reset];
```

- 销毁

```
1. /**  
2.  销毁播放器  
3.  #discussion 调用后彻底销毁播放器，不可恢复  
4.  #discussion 可在退出播放界面前调用  
5.  */  
6. [self.playerManager destroy];
```

8. 部分功能点说明

8.1 记忆播放

- 设置 `BJPlayerManager` 的 `playTimeRecordEnabled` 属性开关记忆播放。开启后记录 vid 对应的播放进度，每 5 秒【更新】一次播放进度，seek 时、停止播放时立即【更新】一次；
- 如果 视频总时长 减去 当前进度 小于 N 秒，则视为播放结束，只移除记录、不添加，N 默认 5 秒，可自定义，参

考 `BJPlayerManager` 的 `ignorableRemainingTimeInter`

`val` 属性;

- 调用 `clearPlayTimeRecords` 方法清除播放纪录;
- 最多存 100 条, 超过 100 条时顶掉最早的;
- 设置视频初始播放时间为大于 0 的值时, 记忆播放无效。

8.2 纯音频播放

- 点播需要在后台配置转码纯音频才有纯音频
- 播在线播放的默认清晰度可以在后台配置清晰度列表, SDK 根据配置的清晰度优先级列表来匹配第一个清晰度, 如果后台配置了纯音频最高优先级, 就可以进入回放和点播的时候直接播放纯音频, 否则就通过改变清晰度的方式播放纯音频
- 视频清晰度 `PMVideoDefinitionType` 增加纯音频枚举
- 纯音频播放

```
1. NSArray *definitionList =
    [self.playerManager.playInfo.definitionList copy];
2. for (NSInteger index = 0; index <
    definitionList.count; index++) {
3.     BJVDefinitionInfo *definitionInfo =
        [[definitionList bjl_objectAtIndex:index] bjl_as:
        [BJVDefinitionInfo class]];
4.     if ([definitionInfo.definitionKey
        isEqualToString:@"audio"]) {
5.         [self.playerManager
            changeDefinitionWithIndex:index];
6.     }
7. }
```

8.3 视频清晰度标识

`BJVDefinitionInfo` 的 `definitionKey` 表示清晰度标识符, 与 `definitionName` 一一对应。可以随视频的实际清晰度动态扩展。

definitionKey	definitionName
low	标清
high	高清
superHD	超清
720p	720p
1080p	1080p
audio	纯音频
...	...

回放功能集成

建议：自主集成回放功能之前，可参考 [BJPlaybackUI SDK](#) 已经开源的代码，该 UI SDK 提供了清晰的回放模块 API 调用方式。

1. 引入头文件

```
1. #import
   <BJVideoPlayerCore/BJVideoPlayerCore.h>
```

2. 设置专属域名前缀

参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

3. （可选）设置

`BJVRequestTokenDelegate`，管理 token

参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

4. 创建、进入回放房间

创建、进入回放房间的整体流程：

- 按需要设置专属域名前缀
- 定义一个 `BJVRoom` 的属性 `room`，用于管理回放房间
- 使用回放相关信息将 `room` 属性实例化
- 为回放的进入，退出，聊天消息更新等事件添加监听
- 调用 `room` 的 `enter` 方法进入房间，监听到进入房间成功之后，即可开始观看回放

4.1 定义回放房间属性

```
1. @property (nonatomic) BJVRoom *room;
```

- 以下为 `BJVRoom` 详细属性说明

```
1. /** 是否是本地视频 */
2. @property (nonatomic, readonly) BOOL
   isLocalVideo;
3.
4. /** 是否是小班课 */
5. @property (nonatomic, readonly) BOOL
   isInteractiveClass;
6.
7. /** 是否是合并回放房间 */
8. @property (nonatomic, readonly) BOOL
   isMixPlaybackRoom;
9.
10. /** 是否正在加载 */
11. @property (nonatomic, readonly) BOOL
    loading;
```

```
12.
13. /** 回放房间信息 */
14. @property (nonatomic, readonly, nullable)
    BJVRoomVM *roomVM;
15.
16. /** 聊天消息 */
17. @property (nonatomic, readonly, nullable)
    BJVMessageVM *messageVM;
18.
19. /** 在线用户 */
20. @property (nonatomic, readonly, nullable)
    BJVOnlineUserVM *onlineUsersVM;
21.
22. /** 播放控制器 */
23. @property (nonatomic, readonly, nullable)
    id<BJVPlayProtocol> playerManager;
24.
25. /** 文档显示，尺寸、位置随意设定，需要设置为
    整数的 pt 值 */
26. @property (nonatomic, readonly, nullable)
    UIViewController<BJLSlideshowUI>
    *slideshowViewController;
27.
28. /**
29. 裁剪后的回放视频的version, 需要在调用enter之
    前设置
30. #discussion -1: 裁剪后的主版本, 0: 裁剪前的
    原始视频.
31. #discussion 默认 clippedVersion 为 -1
32. */
33. @property (nonatomic) NSInteger
    clippedVersion;
34.
35. /**
36. 分组, 设置后, 返回对应分组的聊天信息, 需要在调
    用enter之前设置
```

```

37. #discussion 默认 group 为 -1, 即返回所有分组的聊天信息
38. #discussion group, 0: 大班主讲和助教, 当 group > 0 时, 也会返回大班主讲和助教的信息
39. */
40. @property (nonatomic) NSInteger groupId;
41.
42. /** 是否禁用动态文档, 默认启用动态文档, 为 NO, 但是在无网络状态下进入离线回放时, 将会自动切换成静态文档来保证能够正常观看 */
43. @property (nonatomic) BOOL disablePPTAnimation;
44.
45. /** 是否存在 h5 文档 (指除了 ppt, pptx 类型的动画文档
46. 如果存在, 不能设置成禁用动画文档, 并且建议联网观看, 否则无网络的离线回放将无法加载 */
47. @property (nonatomic, readonly) BOOL existWebDocument;
48.
49. /** 回放房间的相关信息 */
50. @property (nonatomic, readonly) BJVPlaybackInfo *playbackInfo;
51.
52. /** 本地房间的相关信息 */
53. @property (nonatomic, readonly) BJVDownloadItem *downloadItem;

```

4.2 创建回放房间

- 创建在线回放房间

1. /**
2. 创建在线回放房间, 配置项采用默认值
- 3.

```

4. #discussion 具体参考
   \onlinePlaybackRoomWithClassID:sessionID:token:token
   方法
5. #discussion 视频不加密
6. #discussion 使用 AVPlayer
7.
8. @param classID 课程 ID
9. @param sessionID 课节 ID
10. @param token 需要集成方后端调用百家云后端的
    API获取，传给移动端
11. @return 回放房间实例
12. */
13. self.room = [BJVRoom
    onlinePlaybackRoomWithClassID:classID
14.                  sessionID:sessionID
15.                  token:token];

```

```

1. /**
2.  创建在线回放房间，支持多项配置
3.
4.  #param classID 课程 ID
5.  #param sessionID 课节 ID
6.  #param token 需要集成方后端调用百家云后端的
    API获取，传给移动端
7.  #param encrypted 是否加密，「仅在使用
    IJKPlayer 时有效」，playerType 参数传
    BJVPlayerType_IJKPlayer
8.  #param accessKey 集成方鉴权，回放如果需要请求
    第三方服务器查看是否有权限，可设置该参数。鉴权验证
    请求需要与百家云后台沟通。
9.  #param playerType 播放器类型：AVPlayer、
    IJKPlayer
10. #return 回放房间实例
11. */

```

```

12. self.room = [BJVRoom
    onlinePlaybackRoomWithClassID:classID
13.                                sessionID:sessionID
14.                                token:token
15.
    encrypted:encryptEnabled
16.                                accessKey:accessKey
17.                                playerType:playerType];

```

- 创建本地回放房间

创建本地回放的 `downloadItem` 参数通过[下载功能](#)获得

```

1. /**
2.  创建本地回放房间
3.
4.  @param downloadItem 本地回放文件类型，通过
    下载模块获得
5.  @param playerType 播放器类型：AVPlayer、
    IJKPlayer
6.  @return 回放房间实例
7.  */
8. self.room = [BJVRoom
    localPlaybackRoomWithDownloadItem:downloadItem
9.
    playerType:BJVPlayerType_AVPlayer];

```

4.3 回放房间状态监听

在创建回放房间之后、进入回放房间（调用 `BJVRoom` 的 `enter` 方法）之前设置。

可参考 [BJPlaybackUI SDK](#) 中

`BJPRoomViewController+observer.m` 文件的代码。

- 监听进入房间

```

1. /**
2.  进入房间
3.  #param error 错误信息，成功进入时返回空值，错误
    码参考 BJVErrorCode
4.  */
5. - (BJLObservable)roomDidEnterWithError:(nullable
    NSError *)error;
6.
7. // 示例代码
8. bjl_weakify(self);
9. [self bjl_observe:BJLMakeMethod(self.room,
    roomDidEnterWithError:)
10.     observer:^(BOOL(BJLError *error) {
11.         // 此处建议在监听到进入房间成功之后，再
            对 BJVRoom 的 view models进行监听
12.         bjl_strongify(self);
13.         // example: 添加对 BJVRoom 的各个 VM
            的监听
14.         [self
            addObserverForPlaybackViewModels];
15.         // example: 添加对 BJVRoom 的播放器的
            相关监听
16.         [self addObserverForVideoPlayer];
17.         return YES;
18.     }];

```

- 监听退出房间

```

1. /**
2.  退出房间
3.  #param error 错误信息，成功退出时返回空值，错误
    码参考 BJVErrorCode
4.  */
5. - (BJLObservable)roomDidExitWithError:(NSError
    *)error;

```

```

6.
7. // 示例代码
8. bjl_weakify(self);
9. [self bjl_observe:BJLMakeMethod(self.room,
    roomDidExitWithError:)
10.     observer:^(BOOL(BJLError *error) {
11.         bjl_strongify(self);
12.         // 退出房间之后的自定义操作
13.         [self playbackEndWithError:error];
14.         return YES;
15.     }];

```

- 监听加载状态

```

1. /**
2.  是否正在加载
3.  */
4. @property (nonatomic, readonly) BOOL loading;
5.
6. // 示例代码
7. bjl_weakify(self);
8. [self bjl_kvo:BJLMakeProperty(self.room, loading)
9.     observer:^(BOOL(NSNumber * _Nullable value,
    NSNumber * _Nullable oldValue,
    BJLPropertyChange * _Nullable change) {
10.         bjl_strongify(self);
11.         // example: 根据 loading 值显示/隐藏加载动画
12.         if (value.boolValue) {
13.             [self showLoading];
14.         }
15.         else {
16.             [self stopLoading];
17.         }
18.         return YES;
19.     }];

```

- 错误码信息

```
1. typedef NS_ENUM(NSInteger, BJVErrorCode) {
2.     BJVErrorCode_unknown      = -999,    // 未知错误
3.     BJVErrorCode_requestFailed = 1000,    // 网络请求失败
4.     BJVErrorCode_invalidToken  = 1010,    // token 参数错误
5.     BJVErrorCode_invalidAccessKey = 1020,  // 鉴权验证失败
6.     BJVErrorCode_invalidPlayInfo = 1030,  // 播放信息解析错误
7.     BJVErrorCode_invalidVideoURL = 1040,  // 在线播放，视频地址失效
8.     BJVErrorCode_fileLost      = 1050,    // 本地播放，播放文件不存在
9.     BJVErrorCode_playFailed    = 1060    // 视频播放失败
10. };
```

4.4 合并回放房间

- 创建在线合并回放房间：

```
1. /**
2.  创建在线合并回放房间，配置项采用默认值
3.
4.  #discussion 具体参考
   \onlinePlaybackRoomWithClassID:sessionID:token:enableDiscussion方法
5.  #discussion 视频不加密
6.
7.  #param mixID      课程 ID
```

```

8.  #param mixToken 需要集成方后端调用百家云后端的API获取，传给移动端
9.  #param playerType 播放器类型：AVPlayer、IJKPlayer
10. #return 回放房间实例
11. */
12. self.room = [BJVRoom BJVRoom
    onlineMixPlaybackRoomWithMixID:mixID
13.
    mixToken:mixToken
14.
    playerType:playerType];

```

- 合并回放房间的监听

```

1.  /**
2.  合并回放房间即将播放新片段
3.
4.  #discussion 在获取到必要元数据后，会更新room内的playback info数据再调用该回调
5.  可监听此方法来更新和特定playback info相关的UI组件
6.  */
7.  -
    (BJLObservable)mixPlaybackRoomWillLoadSlice:
    (BJVPlaybackInfo*)slice;
8.
9.  /**
10. 合并回放房间的新片段加载结束
11.
12.  #discussion 如果加载失败，可以调用`reload`方法来重试
13.  可监听此方法来更新和特定playback info相关的UI组件
14.

```

```
15.     #param error 为nil表示加载成功。否则 error 表示
      具体原因, 错误码参考 BJVErrorCode
16.     */
17.     - (BJLObservable)mixPlaybackRoomSlice:
      (BJVPlaybackInfo*)slice didLoadWithError:
      (nullable NSError *)error;
```

5. 进入、退出、刷新回放房间

```
1. /** 进入房间 */
2. - (void)enter;
3.
4. /** 退出房间 */
5. - (void)exit;
6.
7. /** 重新加载房间内容
8.     #discussion 用于回放过程中出错后重试。
9.     */
10. - (void)reload;
```

6. 回放视频播放管理

与点播的 `BJVPlayerManager` 类似, 回放房间 `BJVRoom` 的 `playerManager` 遵循 `BJVPlayProtocol` 协议, 因此回放的视频播放管理可以参考点播的[播放设置](#)、[添加播放视图](#)、[视频播放控制](#)、[视频播放信息](#)、[视频播放中的错误监听](#) 等部分, 使用 `self.room.playerManager` 调用相关方法。

6.1 视频窗口占位图更新

当前回放如果播放时未推送视频流, 可以选择监听主讲人变化, 然后选择展示或者隐藏一张占位图:

```
1. // 主讲
```

```

2.     [self
    bjl_kvo:BJLMakeProperty(self.room.onlineUsersVM,
        currentPresenter)
3.     observer:^(BOOL(id _Nullable now, id
        _Nullable old, BJLPropertyChange *_Nullable
        change) {
4.         bjl_strongify(self);
5.         BJVMediaUser *presenter =
        self.room.onlineUsersVM.currentPresenter;
6.         // 更新视频占位图 显示/隐藏 状态
7.         [self updateAudioOnlyImageViewHidden];
8.         return YES;
9.     }];

```

1. 当前播放纯音频流，可考虑隐藏播放画面；
2. 当前视频为合流录制，会一直有播放画面；
3. 播放视频时，可参考主讲人的音频是否打开，当前是否在推媒体流，当前是否在屏幕共享，当前小班课是否为信令录制。

```

1. - (void)updateAudioOnlyImageViewHidden {
2.     if
    (self.room.playerManager.currDefinitionInfo.isAudio
    {
3.         // 播放纯音频时，显示占位图
4.         self.audioOnlyImageView.hidden = NO;
5.         return;
6.     }
7.
8.     if
    (self.room.playerManager.playInfo.recordType ==
    BJRecordType_Mixed
9.     ||
    self.room.playerManager.playInfo.recordType ==
    BJRecordType_CompositeVideo) {

```

```

10. // 合流视频 和 录制视频 一直不显示占位图
11. self.audioOnlyImageView.hidden = YES;
12. return;
13. }
14.
15. // 播放视频时，根据老师是否打开摄像头来显示占位图
16. self.audioOnlyImageView.hidden =
self.room.onlineUsersVM.currentPresenter.videoOn
|| self.room.roomVM.isMediaPlaying ||
self.room.roomVM.isDesktopSharing ||
self.room.playbackInfo.isInteractiveClass1v1Signal
17. }

```

注意：调用 `self.room.playerManager` 的 `destroy` 方法将导致回放房间不可用，且不可恢复。

7. 回放文档管理

回放房间文档的白板、PPT、画笔的展示都来自 `BJVRoom` 的 `slideshowViewController`。用户根据需要
将 `slideshowViewController.view` 添加到需要展示的父 `view` 上即可。文档的尺寸和位置必须设置为整数的 **pt** 值。

7.1 文档设置

- 交互设置：为保证文档内容与视频同步，需要设置回放的文档不支持交互

```

1. self.room.slideshowViewController.view.userInteractionEnabled
= NO;

```

- 显示设置：通过 `BJLSSlideshowUI` 中的相关属性设置，回放的文档均为静态文档

1. **/** 静态文档显示模式**
2. 只支持 `BJLContentMode_scaleAspectFit`、`BJLContentMode_scaleAspectFill`
3. 只对静态文档生效，参考 ``BJLRoom`` 的 ``disablePPTAnimation``
4. ***/**
5. `@property (nonatomic) BJLContentMode`
`contentMode;`
- 6.
7. **/** 静态文档尺寸**
8. 加载文档图片时对图片做等比缩放，长边小于/等于 ``imageSize``，放大时加载 1.5 倍尺寸的图片
9. 单位为像素，默认初始加载 720、放大加载 1080，取值在 ``BJLAllIMGMinSize`` 到 ``BJLAllIMGMaxSize`` 之间 (1 ~ 4096)
10. 不建议进回放房间成功后设置此参数，因为会导致已经加载过的图片缓存失效
11. 只对静态文档生效，参考 ``BJLRoom`` 的 ``disablePPTAnimation``
12. ***/**
13. `@property (nonatomic) NSInteger` `imageSize;`
- 14.
15. **/** 静态文档占位图**
16. 只对静态文档生效，参考 ``BJLRoom`` 的 ``disablePPTAnimation``
17. ***/**
18. `@property (nonatomic) UIImage`
`*placeholderImage;`

- 文档动效设置（仅支持在线回放）
 - `BJVRoom` 的 `disablePPTAnimation` 属性表示是否禁止文档动效，在线回放默认禁止，可设置为 `NO` 以开启动效；
 - 本地回放不支持文档动效。

7.2 文档显示

```
1. // example: 添加文档视图到当前的视图控制器
2. UIViewController *pptViewController =
   self.room.slideshowViewController;
3.
4. [self addChildViewController:pptViewController];
5. [self.view addSubview:pptViewController.view];
6. [pptViewController
   didMoveToParentViewController:self];
```

7.3 文档大纲

大纲数据参考[文档大纲](#)属性。

8. 回放房间信息

回放房间信息由 `BJVRoomVM` 管理，在 `BJVRoom` 的实例创建成功之后，可以对 `self.room.roomVM` 添加监听。

```
1. /** 当前是否正在播放媒体文件 */
2. @property (nonatomic, readonly) BOOL
   isMediaPlaying;
3.
4. /** 当前是否正在共享屏幕 */
5. @property (nonatomic, readonly) BOOL
   isDesktopSharing;
6.
7. /** 用户是否在录制屏幕，iOS11以上生效，iOS11以下
   一直是NO */
8. @property (nonatomic, readonly) BOOL
   screenCaptured;
```

8.1 公告

1. `/** 回放房间公告 */`
2. `@property (nonatomic, readonly) BJVNotice *notice;`

8.2 跑马灯

1. `/** 回放房间跑马灯 */`
2. `@property (nonatomic, readonly, nullable) BJVLamp *lamp;`

8.3 文档大纲

1. `/** 当前回放房间文档的大纲信息 */`
2. `@property (nonatomic, readonly, nullable) NSArray <BJPDocumentCatalogueModel *> *documentCatalogueList;`
- 3.
4. `/** 当前回放房间内实时的最新一条翻页数据 */`
5. `@property (nonatomic, readonly, nullable) BJPPageChangeModel *lastPageChangeSignal;`

8.4 定制广播回调

1. `/**`
 2. `收到定制广播信令`
 3. `#param key 信令类型`
 4. `#param value 信令内容，类型可能是字符串或者字典等 JSON 数据类型`
 5. `#param isCache 是否为缓存`
 6. `*/`
 7. `-`
- `(BJLObservable)didReceiveCustomizedBroadcast:`

```
(NSString *)key value:(nullable id)value isCache:
(BOOL)isCache;
```

8.5 测验

```
1. /**
2.  用户: 收到新题目
3.  #param survey 题目
4. */
5. - (BJLObservable)didReceiveSurvey:(BJVSurvey
6.    *)survey;
7. /** 用户: 收到答题结束 */
8. - (BJLObservable)didFinishSurvey;
```

8.6 答题器

```
1. /** 收到答题器开始信息 */
2. -
3.    (BJLObservable)didReceiveQuestionAnswerSheet:
4.    (BJVAnswerSheet *)answerSheet;
5.
6. /**
7.  收到答题结束信息
8.  #param endTime 答题结束时间戳
9. */
10. -
11.    (BJLObservable)didReceiveEndQuestionAnswerWith
12.    (NSTimeInterval)endTime;
13.
14. /**
15.  收到答题器撤销信息
16.  #param endTime 答题结束时间戳
17. */
```

```
14. -  
    (BJLObservable)didReceiveRevokeQuestionAnswerV  
    (NSTimeInterval)endTime;
```

8.7 问答

```
1. /** 清空间答 */  
2. - (BJLObservable)didResetQuestion;  
3.  
4. /**  
5. 问答发布成功  
6. #discussion 只有在发布没有回答的问答时才会回调此  
   方法  
7. #param question 问答全部内容  
8. */  
9. - (BJLObservable)didPublishQuestion:  
   (BJVQuestion *)question;  
10.  
11. /**  
12. 收到问答回复  
13. #discussion 问答如果已经有了回复，取消发布之后重  
   新发布的情况，只回调此方法，不回调发布成功  
14. #param question 问答全部内容  
15. */  
16. - (BJLObservable)didReplyQuestion:(BJVQuestion  
   *)question;  
17.  
18. /**  
19. 取消发布问答成功  
20. #discussion 任何有回答或者没有回答的问题取消发布  
   成功都回调此方法  
21. #param questionID 问答 ID  
22. */  
23. -  
   (BJLObservable)didUnpublishQuestionWithQuestion
```

```
(NSString *)questionID;
```

9. 回放聊天消息

9.1 聊天消息监听

回放聊天消息由 `BJVMessageVM` 管理，在 `BJVRoom` 的实例创建成功之后，可以对 `self.room.messageVM` 添加监听。

Core SDK 不维护聊天消息的列表，而是通过 消息覆盖更新、增量更新 的回调通知 UI 进行维护。

- 消息覆盖更新，重置列表

```
1. // 消息覆盖更新
2. - (BJLObservable)receivedMessagesDidOverwrite:
   (nullable NSArray<BJLMessage *> *)messages;
3.
4. // example: 监听消息覆盖更新
5. bjl_weakify(self);
6. [self
   bjl_observe:BJLMakeMethod(room.messageVM,
   receivedMessagesDidOverwrite:)
7.     observer:^(BOOL(NSArray<BJLMessage *>
   *messageArray)){
8.         bjl_strongify(self);
9.         // !!!: 回放消息数有限，且 key 是唯一的，重
           置列表时不需要清除高度缓存
10.        self.allMessages = [messageArray
   mutableCopy];
11.        [self.tableView reloadData];
12.        return YES;
13.    }];
```

- 消息增量更新，扩展列表

```
1. // 消息增量更新
2. - (BJLObservable)didReceiveMessages:(nullable
   NSArray<BJLMessage *> *)messages;
3.
4. // example: 监听消息增量更新
5. bjl_weakify(self);
6. [self
   bjl_observe:BJLMakeMethod(room.messageVM,
   didReceiveMessages:)
7.     observer:^(BOOL(NSArray<BJLMessage
   *> *messageArray)){
8.         bjl_strongify(self);
9.         if (!messageArray.count) {
10.             return YES;
11.         }
12.         [self.allMessages
   addObjectFromArray:messageArray];
13.         [self.tableView reloadData];
14.         return YES;
15.     }];
```

9.2 （重要）聊天列表界面性能优化

显示聊天消息的 UI 性能是需要重点优化的，大幅跳转回放视频进度会导致聊天数据的大量更新，此时回放聊天界面性能消耗过高会导致 UI 卡顿、音视频和文档不同步，严重时卡死主线程。优化方式主要有：使用高度缓存、控制列表刷新频率、分页加载等等。具体实现可以参考 [BJPlaybackUI SDK](#) 的 [BJPChatMessageViewController.m](#) 文件。如果成功优化之后依旧存在卡顿，可以使用 `profile` 调试工具针对性能消耗较大的功能进行针对性的优化。

10. 用户列表

回放用户列表由 `BJVOnlineUserVM` 管理，在 `BJVRoom` 的实例创建成功之后，可以对 `self.room.onlineUsersVM` 添加监听。

- 房间用户列表

注意：长期课程的回放，除第1节之外的后续课节，房间用户列表信息不准确

```
1. /** 在线人数 */
2. @property (nonatomic, readonly) NSInteger
   onlineUsersTotalCount;
3.
4. /** 在线用户 */
5. @property (nonatomic, readonly, nullable, copy)
   NSArray<BJVUser *> *onlineUsers;
```

```
1. // example: 监听房间用户列表的变化
2. bjl_weakify(self);
3. [self
   bjl_kvo:BJLMakeProperty(self.room.onlineUsersVM,
   onlineUsers)
4.   observer:^(BOOL(id _Nullable now, id
   _Nullable old, B JLPropertyChange * _Nullable
   change) {
5.       bjl_strongify(self);
6.       self.userList =
   [self.room.onlineUsersVM.onlineUsers copy];
7.       [self.tableView reloadData];
8.       return YES;
9.   }];
```

- 用户进出房间

```
1. /** 在线用户列表覆盖更新
```

```

2. 同时更新 `onlineUsers` */
3. - (BJLObservable)onlineUsersDidOverwrite:
    (NSArray<BJVUser *> *)users;
4.
5. /** 有用户进入房间
6. 同时更新 `onlineUsers` */
7. - (BJLObservable)onlineUsersDidEnter:
    (NSArray<BJVUser *> *)users;
8.
9. /** 有用户退出房间
10. 同时更新 `onlineUsers` */
11. - (BJLObservable)onlineUsersDidExit:
    (NSArray<BJVUser *> *)users;

```

- 当前主讲人

```

1. /** 当前主讲 */
2. @property (nonatomic, readonly, nullable, copy)
    BJVUser *currentPresenter;

```

```

1. // example: 监听房间主讲人变化
2. bjl_weakify(self);
3. [self
    bjl_kvo:BJLMakeProperty(self.room.onlineUsersVM,
        currentPresenter)
4.     observer:^(BOOL(id _Nullable now, id
        _Nullable old, BJLPropertyChange * _Nullable
        change) {
5.         bjl_strongify(self);
6.         BJVUser *presenter =
            self.room.onlineUsersVM.currentPresenter;
7.         NSLog(@"当前主讲为:%@", presenter.name);
8.         return YES;
9.     }];

```

- 发言用户列表变化

```
1. /**
2.  音视频用户列表变更
3.  @param mediaUsers 音视频用户列表
4.  */
5. - (BJLObservable)mediaUsersDidUpdate:
    (NSArray<BJVUser *> *)mediaUsers;
```

```
1. // example: 监听房间发言用户列表的变化
2. bjl_weakify(self);
3. [self
    bjl_observe:BJLMakeMethod(self.room.onlineUsersV
    mediaUsersDidUpdate:)
4.     observer:^(BOOL(NSArray<BJVUser *>
    *mediaUsers) {
5.         bjl_strongify(self);
6.         self.mediaUsers = [mediaUsers copy];
7.         [self.tableView reloadData];
8.         return YES;
9.     }];
```

下载功能集成

建议： 自主集成下载功能之前，可参考 [BJVideoPlayerUI SDK demo](#) 的代码，demo 提供了清晰的下载模块 API 调用方式。

1. 功能点说明

1.1 后台下载

1. 2.x 的下载通过 `NSURLSession` 实现，支持后台下载——App 被杀死之后在电量充足、接入 Wi-Fi 的情况下仍可继续完成下载，参考[苹果官方关于后台下载的文档](#)；

2. 由于 `NSURLSession` 限制，不建议做下载队列管理，等待中的队列实际上是被暂停，后台下载会失效，建议使用 `priority` 属性设置下载任务的优先级；
3. 对于 `NSURLSession` 的设置，例如是否支持 4G 下载等，需要在 `application:will/didFinishLaunchingWithOptions:` 中第一时间 设置 `BJLDownloadManager` 的 `classDelegate` 属性，并实现 `BJLDownloadManagerClassDelegate` 中定义的方法；
4. 如果 `AppDelegate` 中实现了这个方法 - `handleEventsForBackgroundURLSession:completionHandler:`，需要在方法的实现中调用 `BJLDownloadManager` 的 `+handleEventsForBackgroundURLSession:completionHandler:`，否则后台下载任务无法正常恢复；

2. 引入头文件

```
1. #import  
    <BJVideoPlayerCore/BJVideoPlayerCore.h>
```

3. 设置专属域名

参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

4. （可选）设置

`BJVRequestTokenDelegate`，管理 token

参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

5. 初始化下载管理类

BJVDownloadManager

`BJVDownloadManager` 的具体用法可以参考 [BJVideoPlayerUI SDK demo](#) 中 `BJVDownloadViewController.m` 文件。

5.1 定义下载 manager 属性

1. `@property (nonatomic) BJVDownloadManager *downloadManager;`

5.2 创建 manager 实例

5.2.1 初始化

创建 `BJVDownloadManager` 支持设置多个参数，`identifier` 是每个实例的唯一标识，可传入集成方的用户账号，用来区分不同账号的下载数据。`inCaches` 可以让下载文件保存在 `caches` 目录（系统可能会自动清理下载文件，但会保留下载记录，支持重新下载文件），参数默认为 `NO`。

1. `// 指定标识，不保存在 cache 目录`
2. `+ (instancetype)downloadManagerWithIdentifier:(NSString *)identifier;`
- 3.
4. `// 指定标识、配置是否保存在 cache 目录`
5. `+ (instancetype)downloadManagerWithIdentifier:(NSString *)identifier inCaches:(BOOL)inCaches;`
- 6.
7. `// example`

```
8. self.manager = [BJVDownloadManager
    downloadManagerWithIdentifier:@"user.identifier"
    inCaches:NO];
```

5.2.2 下载 `NSURLSessionConfiguration` 配置

对于 `NSURLSession` 的设置，例如是否支持 4G 下载等，需要：

- 在 `application:will/didFinishLaunchingWithOptions:` 中 第一时间 设置 `BJLDownloadManager` 的 `classDelegate` 属性

```
1. // 设置 Class 代理
2. + (nullable
    id<BJLDownloadManagerClassDelegate>)classDele
3. + (void)setClassDelegate:(nullable
    id<BJLDownloadManagerClassDelegate>)classDele
4.
5. // example
6. [BJVDownloadManger setClassDelegate:self];
```

- 实现 `BJLDownloadManagerClassDelegate` 协议中的 `downloadManager:URLSessionConfiguration:` 方法，在该方法中更改 `NSURLSessionConfiguration` 属性

```
1. // Class 代理方法示例
2. - (void)downloadManager:(BJLDownloadManager
    *)downloadManager URLSessionConfiguration:
    (NSURLSessionConfiguration *)configuration {
3.     // 在这里根据需要修改 configuration 的配置项
4.     // configuration.allowsCellularAccess = YES; //
    The default value is YES.
5.     //
    configuration.HTTPMaximumConnectionsPerHost
```

```
= <#MAX#>; // The default value is 6 in macOS,  
or 4 in iOS.  
6. }
```

程序运行过程中更改 `NSURLSession` 的设置，需要：

- 调用 `BJLDownloadManager` 的
`refreshURLSessionWithCompletion:` 方法，注意在
`refreshURLSessionWithCompletion:` 方法回调
`completion` 之前不要对下载任务进行
`add`、`remove`、`pause`、`resume` 等操作

```
1. [self openLoadingView];  
2. // 开始刷新 NSURLSession  
3. [self.downloadManager  
    refreshURLSessionWithCompletion:^(BJLDownloadM  
        * _Nonnull downloadManager) {  
4.     // 刷新 NSURLSession 完成  
5.     [self closeLoadingView]; //  
        userInteractionEnabled: NO  
6. }];
```

- 然后在 `downloadManager`
`initWithURLSessionConfiguration:` 方法中给
`NSURLSessionConfiguration` 的属性设置不同的值

6. 下载任务管理

下载任务由 `BJVDDownloadManager` 定义的相关方法管理。

6.1 校验下载任务是否可添加

注意：视频下载之前都需要先调用

`validateItemWithVideoID:` 或

者 validateItemWithClassID📺:sessionID: 校验点播/回放
是否可以下载。

```
1. /**
2.  校验下载任务对象是否可以添加
3.
4.  #return 是否可添加
5.  #discussion 已经添加过/已下载完成 的任务，方法返回 NO
6. */
7. - (BOOL)validateItemWithVideoID:(NSString
   *)videoID;
8. - (BOOL)validateItemWithClassID:(NSString
   *)classID sessionID:(nullable NSString
   *)sessionID;
9.
10. // example: 判断点播视频是否可下载
11. BOOL validate = [self.downloadManager
   validateItemWithVideoID:videoID];
12.
13. // example: 判断回放是否可下载
14. BOOL validate = [self.downloadManager
   validateItemWithClassID:classID
   sessionID:sessionID];
```

6.2 添加下载任务

- 添加点播下载任务

```
1. /**
2.  点播
3.  #param videoID  视频 ID
4.  #param encrypted  是否加密
5.  #param preferredDefinitionList 下载清晰度列表，
   按顺序匹配、没有匹配将导致下载失败，传 nil 使用默认
```

清晰度

6. `#discussion preferredDefinitionList` 列表元素为清晰度的标识字符串，现有标识符：`low`（标清），`high`（高清），`superHD`（超清），`720p`，`1080p`，`audio`（纯音频），可根据实际情况动态扩展
7. `#param setting` 添加成功后用于设置 `item` 属性的回调
8. `#return` 添加成功返回 `BJVDownloadItem` 实例，`videoID` 已存在会返回 `nil`
9. `*/`
10. `- (nullable BJVDownloadItem
*)addDownloadItemWithVideoID:(NSString
*)videoID`
11. `encrypted:`
`(BOOL)encrypted`
12. `preferredDefinitionList:`
`(nullable NSArray<NSString *>
*)preferredDefinitionList;`
13. `- (nullable BJVDownloadItem
*)addDownloadItemWithVideoID:(NSString
*)videoID`
14. `encrypted:`
`(BOOL)encrypted`
15. `preferredDefinitionList:`
`(nullable NSArray<NSString *>
*)preferredDefinitionList`
16. `setting:(nullable
void (^)(BJVDownloadItem *item))setting;`

1. `// 示例代码`
2. `if (![self.downloadManager
validateItemWithVideoID:videoID]) {`
3. `NSLog(@"video existed: videoID %@",
videoID);`
4. `return;`

```

5. }
6. BJVDownloadItem *downloadItem =
7. [self.downloadManager
   addDownloadItemWithVideoID:videoID
8.                               encrypted:encrypted
9.
   preferredDefinitionList:preferredDefinitionList
10.
   setting:^(BJVDownloadItem * _Nonnull item) {
11.           // 集成方鉴权，根据需
           要设置
12.           item.accessKey =
           accessKey;
13.           // 集成方自定义信息
14.           item.userInfo =
           userInfo;
15.       }];

```

- 添加回放下载任务

```

1. /**
2.  回放
3.  #param classID    课程 ID
4.  #param sessionId  课节 I
5.  #param encrypted  是否加密
6.  #param preferredDefinitionList  下载清晰度列表，
   列表元素为 NSNumber<BJVDefinitionType> *，按顺
   序匹配、没有匹配将导致下载失败，传 nil 使用默认清晰度
7.  #discussion preferredDefinitionList 列表元素为清
   晰度的标识字符串，现有标识符：low（标清），
   high（高清），superHD（超清），720p，1080p，
   audio（纯音频），可根据实际情况动态扩展
8.  #param setting    添加成功后用于设置 item 属性的
   回调

```

```

9. #return 添加成功返回 BJVDDownloadItem 实例，
   classID+sessionID 已存在会返回 nil
10. */
11. - (nullable BJVDDownloadItem
    *)addDownloadItemWithClassID:(NSString
    *)classID
12.                                sessionID:(nullable
    NSString *)sessionID
13.                                encrypted:
    (BOOL)encrypted
14.                                preferredDefinitionList:
    (nullable NSArray<NSString *>
    *)preferredDefinitionList;
15. - (nullable BJVDDownloadItem
    *)addDownloadItemWithClassID:(NSString
    *)classID
16.                                sessionID:(nullable
    NSString *)sessionID
17.                                encrypted:
    (BOOL)encrypted
18.                                preferredDefinitionList:
    (nullable NSArray<NSString *>
    *)preferredDefinitionList
19.                                setting:(nullable
    void (^)(BJVDDownloadItem *item))setting;

```

```

1. // 示例代码
2. if (![self.manager
    validateItemWithClassID:classID
    sessionID:sessionID]) {
3.     NSLog(@"playback existed: %@-%@", classID,
    sessionID);
4.     return;
5. }
6. BJVDDownloadItem *downloadItem =

```

```

7. [self.downloadManager
   addDownloadItemWithClassID:classID
8.   sessionID:sessionID
9.   encrypted:encrypted
10.  preferredDefinitionList:preferredDefinitionList
11.  setting:^(BJVDownloadItem * _Nonnull item) {
12.      // 集成方鉴权，根据
   需要设置
13.      item.accessKey =
   accessKey;
14.      // 集成方自定义信息
15.      item.userInfo =
   userInfo;
16.  }];

```

6.3 纯音频下载

同一个视频只能下载纯音频或其他的一种清晰度，按照传入的清晰度列表匹配第一个满足条件的清晰度。下载纯音频时，调用上面[添加下载任务](#)方法时，`preferredDefinitionList` 参数传入 `@[@"audio"]` 即可。

7. 获取下载任务列表

- 所有已添加的下载任务，包括下载中、下载完成和下载失败的

```

1. @property (nonatomic, readonly, copy)
   NSArray<__kindof BJLDownloadItem *>
   *downloadItems;
2.
3. // example: 获取所有下载任务

```

```
4. NSArray *allDownloadItems =  
    self.downloadManager.downloadItems;
```

- 获取不同下载状态的下载任务

注意: `downloadItemsWithStates:` 方法的参数一定要以 `NSNotFound` 结束

```
1. // !!!: REQUIRES NSNotFound TERMINATION  
2. - (nullable NSMutableArray<__kindof  
    BJLDownloadItem *> *)downloadItemsWithStates:  
    (BJLDownloadItemState)state, ...;  
3.  
4. // example: 获取下载中的任务  
5. NSArray<BJVDownloadItem *> *items =  
    [self.downloadManager  
    downloadItemsWithStates:BJLDownloadItemState_r  
    NSNotFound];
```

8. 下载任务数据类型

BJVDownloadItem

添加下载任务方法返回值为 `BJVDownloadItem` 实例，对应单个下载任务，包含下载任务的各项信息、状态等。

8.1 任务信息

```
1. /** 点播视频 ID*/  
2. @property (nonatomic, readonly, nullable)  
    NSString *videoID;  
3. /** 回放课程、课节 ID*/  
4. @property (nonatomic, readonly, nullable)  
    NSString *classID, *sessionID;  
5. /** 是否为回放 item*/
```

```
6. @property (nonatomic, readonly) BOOL
   isPlayback;
7. /** 文件是否加密*/
8. @property (nonatomic, readonly) BOOL
   isEncrypted;
9. /** 视频播放信息*/
10. @property (nonatomic, readonly, nullable)
    BJVPlayInfo *playInfo;
11. /** 视频、信令文件 */
12. @property (nonatomic, readonly, nullable)
    BJLDownloadFile *videoFile, *signalFile;
13. /** 封面、水印图片*/
14. @property (nonatomic, readonly, nullable)
    BJLDownloadFile *coverImageFile,
    *watermarkImageFile;
15. /** 视频清晰度*/
16. @property (nonatomic, readonly)
    BJVDefinitionInfo *currentDefinitionInfo;
```

8.2 监听下载任务信息变化

对于每一个下载任务，可从 `BJVDownloadItem` 的属性读取下载相关的信息。

属性对应关系如下：

属性名	含义
state	下载状态，参考 BJLDownloadItemState
priority	下载优先级，取值范围 0.0 - 1.0， 参考 NSURLSessionTask.priority
progress	下载进度，

	progress.fractionCompleted 取值范围 0.0 - 1.0，表示进度百分比
downloadFiles	下载文件数组
totalSize	下载文件总大小单位字节
bytesPerSecond	下载速度，单位字节每秒
coverImageFile	封面图片

8.2.1 监听 BJVDownloadItem 变化更新下载列表 UI

- 设置 BJLDownloadManager 的 delegate 属性

```

1. // 设置代理
2. @property (nonatomic, weak, nullable)
   id<BJLDownloadManagerDelegate> delegate;
3.
4. // example
5. self.downloadManger.delegate = self;

```

- 实现 BJLDownloadManagerDelegate 协议中的
downloadManager:downloadItem:didChange: 方法，在该方法中更新视图

```

1. - (void)downloadManager:(BJVDownloadManager
   *)downloadManager
2.     downloadItem:(BJVDownloadItem
   *)downloadItem
3.     didChange:(BJLPropertyChange *)change
   {
4.     NSInteger index =
   [self.downloadManager.downloadItems
   indexOfObject:downloadItem];
5.     if (index == NSNotFound) {
6.         return;

```

```

7.     }
8.
9.     NSIndexPath *indexPath = [NSIndexPath
indexPathForRow:index inSection:0];
10.    BJVDownloadItemCell *cell = [self.tableView
cellForRowAtIndexPath:indexPath];
11.    [cell updateWithItem:downloadItem];
12. }

```

8.2.2 显示下载进度和速度

由于下载进度的变化过于频繁，并且不在主线程回调，需要切换到主线程才能更新 UI，因此会严重影响渲染帧率，多个任务同时下载时更加严重。

另外，当进度没有变化时速度需要用 `timer` 来实现衰减的效果，但是每个 `BJVDownloadItem` 内置一个 `timer` 就有点浪费了。

因此

- 使用一个全局的 `timer` 定时、主动获取下载进度和速度，并且只在必要时才去一次性的重新渲染视图

```

1.    bjl_weakify(self);
2.    self.progressTimer = [NSTimer
bjl_scheduledTimerWithTimeInterval:0.1
repeats:YES block:^(NSTimer * _Nonnull timer) {
3.        // self dealloc 后销毁 timer
4.        bjl_strongify_ifNil(self) {
5.            [timer invalidate];
6.            return;
7.        }
8.        // 视图不显示时跳过
9.        if (!self.isViewLoaded || !self.view.window) {
10.            return;

```

```

11.     }
12.     // 只刷新可见 cell
13.     NSArray<NSIndexPath *> *indexPaths =
14.     [self.tableView indexPathsForVisibleRows];
15.     for (NSIndexPath *indexPath in indexPaths) {
16.         BJVDownloadItem *item =
17.         [self.downloadManager.downloadItems
18.         bjl_objectAtIndex:indexPath.row];
19.         BJVDownloadItemCell *cell =
20.         [self.tableView cellForRowAtIndexPath:indexPath];
21.         // 尝试更新
22.         [cell tryToUpdateWithItem:item];
23.     }
24. }];
25. // 避免滑动时 timer 回调不执行
26. [[NSRunLoop currentRunLoop]
27. addTimer:self.progressTimer
28. forMode:NSRunLoopCommonModes];

```

- 在 `UITableViewCell` 中缓存上一次下载进度和速度，在 `timer` 执行时检查变化，只有变化足够大时才更新视图

```

1. // 缓存进度和速度
2. @property (nonatomic) BJLProgress
3. cachedProgress;
4.
5. @property (nonatomic) long long
6. cachedBytesPerSecond;
7.
8. - (void)prepareForReuse {
9.     [super prepareForReuse];
10.    // prepare 时重置缓存
11.    self.cachedProgress = BJLProgressMake(0, 0);
12.    self.cachedBytesPerSecond = - 1;
13. }

```

```

12. - (void)tryToUpdateWithItem:(BJVDownloadItem
    *)item {
13.     // 尝试更新时与缓存比较，这里
    `progress.fractionCompleted` 变化达到 0.1% 或者
    下载速度有变化时才会更新视图
14.     if ([item
        compareWithProgress:self.cachedProgress
        bytesPerSecond:self.cachedBytesPerSecond]) {
15.         [self updateWithItem:item];
16.     }
17. }
18.
19. - (void)updateWithItem:(BJVDownloadItem *)item
    {
20.     // 更新视图时更新缓存
21.     self.cachedProgress = item.progress;
22.     self.cachedBytesPerSecond =
        item.bytesPerSecond;
23.     // 更新视图
24.     ...
25. }

```

8.3 下载状态说明

需要注意的是：在拿到 `BJVDownloadItem` 更新UI时，需要结合 `BJVDownloadItem` 的 `state` 和 `error` 对当前下载任务作出正确的判断。

- `BJLDownloadItemState_paused` && `error` 表示下载中出错
- `BJLDownloadItemState_completed` && `error` 表示下载完文件丢失

```

1. typedef NS_ENUM(NSInteger,
    BJVDownloadItemState) {

```

```

2.   BJLDownloadItemState_invalid = - 1, // NOT be
    added to any BJLDownloadManager, or be
    invalidated
3.   BJLDownloadItemState_running,
4.   BJLDownloadItemState_paused, // paused +
    error = error occurred
5.   BJLDownloadItemState_completed, //
    completed + error = file lost
6.   BJLDownloadItemState_suspended =
    BJLDownloadItemState_paused // 与
    BJLDownloadItemState_paused 等效，将废弃
7. };

```

9. 下载任务控制

- 开始：调用 `BJVDownloadItem` 的 `resume` 方法

```
1. [item resume];
```

- 暂停：调用 `BJVDownloadItem` 的 `pause` 方法

`pause` 方法用于替代 `suspend` (`suspend` 现在仍然可用)，解决了频繁调用 `suspend` 、 `resume` 可能出现的下载进度不动的问题，因此 `restart` 方法也不需要了

```
1. [item pause];
```

- 删除：调用 `BJVDownloadManager` 的 `removeDownloadItemWithIdentifier:` 方法

```

1. [self.downloadManager
    removeDownloadItemWithIdentifier:item.itemIdentifi

```

10. 旧版点播回放 SDK 下载数据迁移

PMDownloadManager 下载数据迁移，可参考

[BJVideoPlayerUI SDK demo](#) 中的

`BJVDownloadViewController.m` 文件

- 规则：
 - 未下载完成的任务 - 已下载部分数据只能丢弃，创建新的下载任务重新下载；
 - 已下载完成的任务 - 已下载文件将被保留，迁移完成后需要通过网络请求获取新下载记录所需的基本信息；
- 步骤：
 - !!!: 迁移过程需要访问网络，但网络请求可能因各种原始而失败、App 也有可能因各种原因被杀死或崩溃，因此需要设计一个入口、能让用户随时触发迁移，确保迁移成功才关闭入口；
 - a. 实现 协议，可根据视频 `videoID` 或 `classID`、`sessionID` 返回相应的 `token`；
 - a. App 启动后调用 `+[PMDownloadManager downloadManagerWithRootPath:]` 设置旧的下载根目录，然后调用 `PMDownloadManager.downloadingList` 和 `PMDownloadManager.finishedList` 分别获取下载中和下载完成的任务 - `PMDownloadModel` 的实例；
 - a. 如果 `PMDownloadModel.readyForMigration` 是 NO，需要调用 `[PMDownloadModel prepareForMigrationWithCompletion:]` 完成迁移前准备；
 - a. 对每个任务调用 `BJLDownloadManager` 的 `migratePMDownloadModel` 或 `migrateCompletedPMDownloadModel` 方法进行迁移；

- a. 迁移方法返回的 `downloadItem` 不为 `nil`、且 `code` 为 `BJVDownloadMigrationCode_added` 时表示添加成功；
- a. 添加成功后状态为 `paused`，需要 `resume` 来补齐视频/回放的基本信息（`autoResume` 设置 `YES`，或调用 `[BJVDownloadItem resume]`），此步骤需要联网完成，出错可通过 `BJVDownloadItem.error` 查看，再次调用 `[BJVDownloadItem resume]` 可重试；
- a. 迁移完成后调用 `[PMDownloadManager removePMDownloadModel:]` 方法删除旧的下载记录。

10.1 （必须）设置

`BJVRequestTokenDelegate`，管理 `token`

SDK 需要旧版数据对应的 `token` 才能完成迁移，因此必须实现 `BJVRequestTokenDelegate` 中的方法：参考[全局设置](#)部分，点播、回放、下载通用，项目工程内只需要设置一次。

10.2 使用旧版数据存放路径创建下载管理类 `PMDownloadManager` 实例

可参考 [BJVideoPlayerUI SDK demo](#) 中的

`PUAppDelegate.m` 文件

```
1. PMDownloadManager *pmDownloadManager =  
   [PMDownloadManager  
    downloadManagerWithRootPath:yourPath];
```

10.3 获取旧版 SDK 下载数据

- 获取下载中的旧数据列表

```
1. NSArray <PMDownloadModel *>  
   *downloadingModels =
```

```
pmDownloadManager.downloadingList;
```

- 获取下载完成的旧数据列表

```
1. NSArray <PMDownloadModel *> *finishedModels  
   = pmDownloadManager.finishedList;
```

10.4 准备迁移

如果 `PMDownloadModel` 的 `readyForMigration` 是 NO，需要调用 `prepareForMigrationWithCompletion:` 完成迁移前准备。

```
1. // example: 迁移旧版的 `PMDownloadModel` 实例  
   `model` 到新版下载  
2. // 实例可从上面获取的  
   `pmDownloadManager.downloadingList` 和  
   `pmDownloadManager.finishedList` 中获取  
3. BOOL completed = (model.state ==  
   PMDownloadState_Completed);  
4. if (completed && !model.readyForMigration) {  
5.     bjl_weakify(self);  
6.     [model  
   prepareForMigrationWithCompletion:^(NSError *  
   _Nullable error) {  
7.         bjl_strongify(self);  
8.         if (error || !model.readyForMigration) {  
9.             [self showHUDWithMessage:[NSString  
   stringWithFormat:@"prepare error - %@: %@",  
   model.videoFileName, error]];  
10.        }  
11.        else {  
12.            // 准备完成，开始迁移  
13.            [self migrateDownloadModel:model];  
14.        }
```

```
15.     }];  
16.     return;  
17. }
```

10.5 开始迁移旧版下载数据

下载中的 `model` 调用 `[BJLDownloadManager migratePMDownloadModel:code:]` 方法迁移；下载完成的 `model` 调用 `[BJLDownloadManager migrateCompletedPMDownloadModel:code:]` 方法进行迁移。

迁移方法返回的 `downloadItem` 不为 `nil`、且 `code` 为 `BJVDownloadMigrationCode_added` 时表示添加成功；否则抛出错误，错误码参考 `BJVDownloadMigrationCode`。

```
1. // example: model 为 `PMDownloadModel` 实例  
2. BJVDownloadMigrationCode code =  
   BJVDownloadMigrationCode_added;  
3. BJVDownloadItem *item = nil;  
4. BOOL completed = (model.state ==  
   PMDownloadState_Completed);  
5. if (!completed) {  
6.     item = [self.manager  
   migratePMDownloadModel:model code:&code];  
7. }  
8. else {  
9.     item = [self.manager  
   migrateCompletedPMDownloadModel:model  
   code:&code];  
10. }
```

10.6 旧版下载数据迁移结果处理

在上面 开始迁移旧版数据 的示例代码中，调用

`BJLDownloadManager` 的 `migratePMDownloadModel`
或 `migrateCompletedPMDownloadModel`
方法将返回 `BJVDownloadItem` 类型的新版下载
数据实例 `item`，并给 `code` 赋值。通过 `item` 和
`code` 判断迁移结果。

```
1. // example: 开始迁移 -> 处理迁移结果 的完整示例，  
   `model` 为 `PMDownloadModel` 实例  
2. BJVDownloadMigrationCode code =  
   BJVDownloadMigrationCode_added;  
3. BJVDownloadItem *item = nil;  
4.  
5. if (!completed) {  
6.     item = [self.manager  
   migratePMDownloadModel:model code:&code];  
7. }  
8. else {  
9.     item = [self.manager  
   migrateCompletedPMDownloadModel:model  
   code:&code];  
10. }  
11.  
12. if (item) {  
13.     [self observeItem:item];  
14.     [self showHUDWithMessage:@"添加成功"];  
15.     [self.tableView reloadData];  
16. }  
17. else {  
18.     switch (code) {  
19.         case  
   BJVDownloadMigrationCode_invalidPMDownloadModel  
20.         [self showHUDWithMessage:[NSString  
   stringWithFormat:@"invalidPMDownloadModel -  
   %@ ", model.videoFileName]];
```

```

21.         break;
22.     case
        BJVDDownloadMigrationCode_uncompletedPMDownload:
23.         [self showHUDWithMessage:[NSString
        stringWithFormat:@"uncompletedPMDownloadMode
        - %@", model.videoFileName]];
24.         break;
25.     case BJVDDownloadMigrationCode_fileLost:
26.         [self showHUDWithMessage:[NSString
        stringWithFormat:@"fileLost - %@",
        model.videoFileName]];
27.         break;
28.     case
        BJVDDownloadMigrationCode_notReadyForMigration:
29.         [self showHUDWithMessage:[NSString
        stringWithFormat:@"notReadyForMigration -
        %@", model.videoFileName]];
30.         break;
31.     case
        BJVDDownloadMigrationCode_failedWithError:
32.         [self showHUDWithMessage:[NSString
        stringWithFormat:@"failedWithError - %@",
        model.videoFileName]];
33.         break;
34.     default:
35.         [self showHUDWithMessage:[NSString
        stringWithFormat:@"unknownError - %@",
        model.videoFileName]];
36.         break;
37.     }
38. }

```

10.7 补全新版下载数据

旧版下载数据迁移成功后获得的 `BJVDownloadItem` 实例 `item` 的状态为 `paused`，需要通过 `resume` 来补齐视频/回放的基本信息（设置 `BJVDownloadManager` 的 `autoResume` 为 YES，或调用 `[item resume]`），此步骤需要联网完成，出错可通过监听 `item.error` 查看 (参考[监听下载任务信息变化](#))，再次调用 `[item resume]` 可重试。

```
1. [item resume];
```

或者

```
1. self.downloadManager.autoResume = YES;
```

10.8 迁移成功后删除旧版数据

迁移成功后调用 `PMDownloadManager` 的 `removePMDownloadModel:` 方法删除旧的下载记录。迁移失败时建议保留数据，方便重试。

```
1. // example: pmDownloadManager 由 9.2 部分的示例代码获取；model 由 9.3 部分的示例代码获取
2. [pmDownloadManager
   removePMDownloadModel:model];
```