

[离线/端作业]使用文档

- 一、离线相关
 - 1.1 数据同步
 - 1.1.1 OfflineDownload
 - 1.1.2 TypeSerializer
 - 1.2 数据展示
 - 1.2.1 Cacheable
 - 1.2.2 Bind
 - 1.3 数据上行
- 二、端作业相关

一、离线相关

1.1 数据同步

- OfflineDownload: 定义数据同步接口
- TypeSerializer: 定义返回类型与数据存储的映射

1.1.1 OfflineDownload

定义数据同步接口

- 首次调用: 底层在数据获取之后会自动缓存该数据类型
- 下次调用: 先检查缓存是否有效, 如果有效, 直接返回

OfflineDownload

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface OfflineDownload {
    /**
     * Http max-age
     * @return
     */
    long maxAge() default 0L;
}
```

示例:

```
@OfflineDownload(maxAge = 60 * 60 * 1000L)
@GetMapping("/s/movie/popular")
Observable<MovieListResult> getPopularMovies(@NonNull @Query("api_key") String apiKey,
                                             @NonNull @Query("language") String language,
                                             @Query("page") int page);
```

1.1.2 TypeSerializer

定义返回类型与数据存储的映射

- 将数据类型 T 转为数据库存储对象 DBDownload
- 将 DBDownload 转换为数据类型 T

TypeSerializer

```
public interface TypeSerializer<ResponseT> {  
    /**  
     * DBDownload 1 -> 1 / 1 -> N  
     * @param result  
     * @return DBDownload  
     */  
    List<DBDownload> convert(Result<ResponseT> result);  
  
    /**  
     * DBDownload  
     * @param downloads DBDownload  
     * @return  
     */  
    ResponseT reverse(List<DBDownload> downloads);  
}
```

示例：

```
public class MovieListResultSerializer implements OfflineServiceProvider.TypeSerializer<MovieListResult> {  
    @Override  
    public List<DBDownload> convert(Result<MovieListResult> result) {  
        List<DBDownload> caches = new ArrayList<>();  
        MovieListResult movieListResult = result.get();  
  
        DBDownload cacheTotal = DBSynchronizer.serialize(movieListResult);  
        caches.add(cacheTotal);  
  
        for (MovieItem item : movieListResult.getItems()) {  
            DBDownload cacheItem = DBSynchronizer.serialize(item);  
            caches.add(cacheItem);  
        }  
        return caches;  
    }  
  
    @Override  
    public MovieListResult reverse(List<DBDownload> downloads) {  
        if (downloads == null || downloads.size() == 0) {  
            return null;  
        }  
        MovieListResult result = new MovieListResult();  
        List<MovieItem> movieItems = new ArrayList<>();  
        for (DBDownload cache : downloads) {  
            if (cache.getTag().equals(MovieListResult.SCENE_TOTAL_KEY)) {  
                result = (MovieListResult) DBSynchronizer.deserialize(cache, MovieListResult.class);  
            } else if (cache.getTag().equals(MovieItem.SCENE_ITEM_KEY)) {  
                MovieItem item = (MovieItem) DBSynchronizer.deserialize(cache, MovieItem.class);  
                movieItems.add(item);  
            }  
        }  
        result.setItems(movieItems);  
        return result;  
    }  
}
```

1.2 数据展示

- Cacheable：展示的数据如果需要做 DBDownload 同步的需要派生自该基类
- Bind：需要同步的数据字段需要使用该注解做标注

1.2.1 Cacheable

定义需要做数据同步的 Bean 对象，需要实现

- getId：获取对象唯一 ID
- getTag：设置额外参数

两个方法，调用 synchronize 方法做数据同步

Cacheable

```
public abstract class Cacheable {  
    /**  
     * RESERVE_1RESERVE_5  
     */  
    public static final int RESERVE_NULL = 0;  
    public static final int RESERVE_1 = 1;  
    public static final int RESERVE_2 = 2;  
    public static final int RESERVE_3 = 3;  
    public static final int RESERVE_4 = 4;  
    public static final int RESERVE_5 = 5;  
  
    /**  
     *  
     * @return  
     */  
    @NonNull  
    public abstract String getUniqueId();  
  
    /**  
     *  
     * @return  
     */  
    @Nullable  
    public abstract String getTag();  
  
    /**  
     * DB  
     */  
    public void synchronize() {  
        DBSynchronizer.synchronize(this);  
    }  
}
```

1.2.2 Bind

绑定需要同步到 DBDownload 的相关字段，这里可以绑定到 DBDownload 的预留字段上，可以是

- RESERVE_1: 第一个字段
- RESERVE_2: 第二个字段
- RESERVE_3: 第三个字段
- RESERVE_4: 第四个字段
- RESERVE_5: 第五个字段

示例：

```
public class MovieItem extends Cacheable {  
    public static final String NAME_FOR_KEY = "movie_item";  
  
    @SerializedName("id")  
    public long id = 0L;  
  
    @SerializedName("background_image")  
    public String uri = "";  
  
    @SerializedName("title")  
    public String title = "";  
  
    @SerializedName("score")  
    private float score = 0.0f;  
  
    @SerializedName("original_title")  
    public String originalTitle = "";  
  
    @SerializedName("release_date")  
    public String releaseDate = "";  
  
    @SerializedName("overview")  
    public String overview = "";  
  
    public float getScore() { return this.score; }  
    public void setScore(float score, boolean sync) {  
  
        @Override  
        public String getUniqueId() { return String.valueOf(id); }  
  
        @Override  
        public String getTag() { return "movie_item"; }  
    }  
  
    public void setScore(float score, boolean sync) {  
        this.score = score;  
        if (sync) {  
            synchronize();  
        }  
    }  
}
```

1.3 数据上行

- OfflineUpload: 定义数据上行接口

经过 OfflineUpload 定义的接口，框架底层会自动做重试，直到满足重试次数的条件，或者发送成功

OfflineUpload

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface OfflineUpload {
    /**
     *
     * @return
     */
    int retry() default Integer.MAX_VALUE;
}
```

示例:

```
@OfflineUpload(retry = 10)
@POST("/3/movie/{movie_id}/rating")
Observable<RateMovieResult> rateMovie(@Path("movie_id") long movieId,
                                     @NonNull @Query("api_key") String apiKey,
                                     @NonNull @Query("session_id") String sessionId,
                                     @NonNull @Body RateMovieParams params);
```

二、端作业相关

待实现