BaseSectionQuickAdapter

说明：快速实现带头部的 Adapter，由于本质属于多布局，所以继承自 `BaseMultiItemQuickAdapter`。（只有一种布局类型的头部！）

数据实体类不再是实现 **`MultiItemEntity`** 接口！！！

为了方便快速使用，对 `MultiItemEntity` 进行了再次封装。

情况分别为下：

1、`Kotlin`实现`SectionEntity`接口！

1、`Java`则继承`JSectionEntity`抽象类！

`JSectionEntity`是实现于`SectionEntity`接口，java进了再一次的封装是因为：

`SectionEntity`接口是使用`Kotlin`编写的，里面对接口方法进行了默认实现，一般情况下，使用者的数据实体类不需要重写此方法，而`java`7 无法具有*接口默认实现*的特性，这就导致使用者必须自己重写此方法。所以 java 使用抽象类进行了再次封装，在抽象类中进行了默认实现。

此种处理处理措施，必定影响使用者 java 数据类的类继承问题，对此表示抱歉，由于`java`特性的缺失，不得不做出的妥协。如果实在无法继承`JSectionEntity`抽象类，请自己再次封装一下，或者请使用`BaseDelegateMultiAdapter`自行参考实现Section。

> 如果需要更灵活`Section`功能的，可以使用`BaseNodeAdapter`

1、快速使用

`Adapter`代码如下：

```
public class SectionQuickAdapter extends BaseSectionQuickAdapter<MySection, BaseViewHolder> {
    /**
     * 构造方法里， super()必须设置 header layout
     * data可有可无
     */
    public SectionQuickAdapter(int layoutResId, int sectionHeadResId, List<MySection> data) {
        super(sectionHeadResId, data);
        // 设置普通item布局（如果item类型只有一种，使用此方法）
        setNormalLayout(layoutResId);
        // 注册需要点击的子view id
```

```
        addChildClickViewIds(R.id.more);
    }

    /**
     * 设置header数据
     */
    @Override
    protected void convertHeader(@NotNull BaseViewHolder helper, @NotNull MySection item) {
        if (item != null && item.getObject() instanceof String) {
            helper.setText(R.id.header, (String) item.getObject());
        }
    }

    @Override
    protected void convert(@NotNull BaseViewHolder helper, MySection item) {
        //设置item数据
        Video video = (Video) item.getObject();
        helper.setText(R.id.tv, video.getName());
    }
}
```

实体类的写法：

```
/**
 * java 写法
 */
public class MySection extends JSectionEntity {
    // 你的数据内容
    private boolean isHeader;
    private Object  object;

    public MySection(boolean isHeader, Object object) {
```

```
        this.isHeader = isHeader;
        this.object = object;
    }

    public Object getObject() {
        return object;
    }

    /**
     * 重写此方法，返回 boolen 值，告知是否是header
     */
    @Override
    public boolean isHeader() {
        return isHeader;
    }
}
```

```
/**
 * kotlin 写法
 */
class DataBean: SectionEntity {
                // 你的数据实体内容
    ...
    ...

    /**
     * 重写此方法，返回 boolen 值，告知是否是header
     */
    override val isHeader: Boolean
        get() {
            // 根据数据实体内的数据，判断是否是header
            return false
        }
}
```

2、多布局类型 BaseSectionQuickAdapter

`Adapter` 代码如下：

```
public class SectionQuickAdapter extends BaseSectionQuickAdapter<MySection, BaseViewHolder> {
    /**
     * 构造方法里， super()必须设置 header layout
     * data可有可无
     */
    public SectionQuickAdapter(int layoutResId, int sectionHeadResId, List<MySection> data) {
        super(sectionHeadResId, data);
        //  (只有这里有变化)
        // 添加 item 布局、类型
        addItemType(类型, 你的布局id_1)
                                addItemType(类型2, 你的布局id_2)
    }

    /**
     * 设置header数据
     */
    @Override
    protected void convertHeader(@NotNull BaseViewHolder helper, @NotNull MySection item) {
    }

    @Override
    protected void convert(@NotNull BaseViewHolder helper, MySection item) {
        //设置item数据
    }
}
```

数据实体类，多了重写 `itemType` 方法：

```
/**
 * java 写法
 */
public class MySection extends JSectionEntity {
    private boolean isHeader;
    private Object  object;

    public MySection(boolean isHeader, Object object) {
        this.isHeader = isHeader;
        this.object = object;
    }

    public Object getObject() {
        return object;
    }

    /**
     * 重写此方法，返回 boolen 值，告知是否是header
     */
    @Override
    public boolean isHeader() {
        return isHeader;
    }

    /**
     * 重写此方法，返回你的item类型
     */
        @Override
    public int getItemType() {
        if (isHeader()) {
            return SectionEntity.Companion.HEADER_TYPE;
        } else {
```

```
            // 重写此处，返回自己的多布局类型
            return ...;
        }
    }
}
```

```
/**
 * kotlin
 */
class DataBean : SectionEntity {
    // 你的数据实体内容
    ...
    ...

    /**
     * 重写此方法，返回 boolen 值，告知是否是header
     */
    override val isHeader: Boolean
        get() {
            return false
        }

    /**
     * 重写此方法，返回你的item类型
     */
    override val itemType: Int
        get() {
            if (isHeader) {
                return SectionEntity.HEADER_TYPE
            } else {
                // 根据数据内容，返回你自己的类型
                return ...
            }
```

```
        }
}
```