代码框架

1. **`detector.py`**： 定义了 `Detector` 基类，提供了天文探测器数据处理的通用接口，作为所有具体探测器类的父类。
2. **`detector_grid_11b.py`**： 实现了 `GRID11BDetector` 类，继承自 `Detector` ，针对GRID-11B探测器实现了具体的数据处理逻辑，包括数据包提取、解析、ADC校准和能量转换等。
3. **`main.py`**： 程序入口，依次调用四个处理模块。
4. **`packet_extractor.py`**： 从原始数据文件中提取不同类型的数据包。
5. **`data_unpacker.py`**： 解析数据包，提取具体字段。
6. **`event_reconstructor.py`**： 对解析后的数据进行时间和能量校准，生成事件数据。
7. **`file_formatter.py`**： 将事件数据格式化为最终的科学数据产品（FITS文件），包含标准化的时间和能量信息。

数据处理流程是一个线性管道（pipeline），从原始数据到最终科学数据产品，分为以下四个阶段：

1. **数据包提取**（ `packet_extractor.py` ）：从原始 `.dat` 文件中提取数据包，保存为包含原始数据包的FITS文件。
2. **数据包解析**（ `data_unpacker.py` ）：解析数据包，提取具体字段（如时间、通道、ADC值等），保存为解析后的FITS文件。
3. **事件重建**（ `event_reconstructor.py` ）：对解析数据进行时间和能量校准，生成事件数据，保存为初步事件FITS文件。
4. **文件格式化**（ `file_formatter.py` ）：将事件数据格式化为标准化的科学数据产品，包含能量边界（EBOUNDS）、有效时间区间（GTI）和事件数据。

每个阶段的输出作为下一阶段的输入，最终生成可供天文学家分析的FITS文件。

模块功能详解

`detector.py` - 探测器基类

- **作用**：定义了 `Detector` 基类，提供了通用接口，确保所有具体探测器类遵循一致的数据处理方法。
- **关键属性**：
  - `detector_name`： 探测器名称（如"GRID-11B"）。
  - `short_name`： 探测器简称（如"11B"）。

- `source_path_pattern`： 原始数据文件的路径模式。
- `output_path`： 输出文件路径。
- `gagg_channels`： 有效探测通道（如[0, 1, 2]）。

- **抽象方法**（需子类实现）：
  - `extract_packets`： 从原始文件中提取数据包。
  - `unpack_packets`： 解析数据包。
  - `calibrate_adc`： 校准ADC值。
  - `adc_to_energy`： 将校准后的ADC值转换为能量。
- **意义**：通过抽象方法强制子类实现特定逻辑，确保模块化设计和代码的可扩展性。

`detector_grid_11b.py` - GRID-11B探测器实现

- **作用**：实现了 `GRID11BDetector` 类，继承自 `Detector` ，为GRID-11B探测器提供具体的数据处理逻辑。
- **核心方法**：
  i. **`__init__`**：
     - 初始化探测器属性，指定名称、通道和文件模式。
     - 定义正则表达式模式（如 `lvds_pattern` 、 `app_header_pattern` 等）用于识别数据包。
  ii. **`find_and_unpack_lvds_packets`**：
     - 从原始字节数据中提取LVDS（低电压差分信号）数据包。
     - 使用正则表达式匹配帧头帧尾，检查数据长度和校验和。
     - 返回有效数据部分、包计数和错误统计。
  iii. **`find_and_unpack_app_packets`**：
     - 解析应用层数据包，分为科学数据（ `sci` ）和 housekeeping 数据（ `hk` ）。
     - 根据帧类型（如0x03为HK，0x04为科学数据）分类存储。
  iv. **`find_and_classify_tte_frames` / `find_and_classify_spec_frames` / `find_and_handle_hk_frames`**：
     - 分别处理TTE（时间标记事件）、能谱（SPECTRA）和 housekeeping 数据帧。
     - 验证校验和和UTC时间，提取时间戳和数据包。
  v. **`extract_packets`**：
     - 协调上述方法，从原始文件提取并分类数据包，返回包含 `EVENTS` 、 `SPECTRA` 和 `HOUSEKEEPING` 的字典。
  vi. **`unpack_packets`**：
     - 根据数据包类型（EVENTS、SPECTRA、HOUSEKEEPING）解析具体字段。
     - `unpack_event_packets`： 提取UTC、时间戳、通道和ADC值。
     - `unpack_spec_packets`： 提取长能谱（82个通道）和短能谱（20×8矩阵）。
     - `unpack_hk_packets`： 提取温度、电压、电流、星敏数据和经纬度。
  vii. **`calibrate_adc` / `adc_to_energy`**：

    - 校准ADC值（目前简单处理，通道3返回NaN）。
    - 使用二次函数将校准后的ADC值转换为能量，基于低能量和高能量校准系数。
  viii. **`read_ec_coef`**：
    - 从JSON文件读取能量校准系数（低能量和高能量）。
  ix. **`correct_temperature`**：
    - 修正异常温度数据，通过比较邻近时间点的数据进行插值或替换。
  x. **`Adc_mapping` 类**：
    - 提供温度-偏压修正和ADC-能量转换的复杂逻辑。
    - 使用校准文件中的系数进行更精确的ADC校准和能量转换。
- **关键特性**：
  - 使用正则表达式高效匹配数据包。
  - 包含错误检查（校验和、长度、时间有效性）。
  - 支持多类型数据（事件、能谱、housekeeping）。

`main.py` - 程序入口

- **作用**：协调整个处理流程，实例化探测器并依次调用四个模块。
- **流程**：
  i. 根据 `payload_number` （如"11"）实例化 `GRID11BDetector` 。
  ii. 依次调用：
    - `pe.extract_packets` ： 提取数据包。
    - `du.unpack_data` ： 解析数据包。
    - `er.reconstruct_events` ： 重建事件。
    - `ff.save_evt_files` ： 格式化最终事件文件。
- **输出**：控制台打印阶段性日志，标记每个处理阶段的开始和结束。

`packet_extractor.py` - 数据包提取

- **作用**：从原始 `.dat` 文件中提取数据包，保存为FITS文件。
- **核心功能**：
  - 使用 `utils.find_unprocessed_files` 查找未处理文件。
  - 调用 `detector.extract_packets` 提取数据包。
  - 使用 `save_packets_tmp` 将数据包保存为FITS文件，包含 `UTC` 、 `TIMESTAMP` 和 `PACKET` 列。
  - 记录已处理文件到日志。
- **输出**：FITS文件，存储在 `path_packets` 目录，文件名格式为 `{short_name}_pac_{原始文件名}.fits` 。

`data_unpacker.py` - 数据包解析

- **作用**：解析提取的数据包，提取具体字段，保存为FITS文件。
- **核心功能**：
  - 读取 `path_packets` 中的FITS文件，提取 `PACKET` 列。
  - 调用 `detector.unpack_packets` 解析数据包，生成字段字典（如UTC、通道、ADC值等）。
  - 使用 `save_unpacked_data` 保存解析结果，处理复杂数据结构（如 `LONG_SPECTRA` 和 `SHORT_SPECTRA` 的多维数组）。
  - 使用 `create_spectrum_column` 为能谱数据生成合适的FITS列格式。
- **输出**：FITS文件，存储在 `path_unpacked_data` 目录，文件名格式为 `{short_name}_unp_{原始文件名}.fits` 。

`event_reconstructor.py` - 事件重建

- **作用**：对解析后的数据进行时间和能量校准，生成事件数据。
- **核心功能**：
  - 读取 `path_unpacked_data` 中的FITS文件，提取 `EVENTS` 扩展的数据。
  - 计算精确UTC时间（ `utc_accurate` ），使用时间戳偏移和晶振频率（ `mean_value` ）。
  - 调用 `detector.calibrate_adc` 和 `detector.adc_to_energy` 进行ADC校准和能量转换。
  - 为每个通道生成事件字典，包含 `UTC` 、 `ENERGY` 、 `ADC_VALUE` 和 `ADC_CALIBRATED` 。
  - 使用 `save_events` 保存事件数据到FITS文件。
- **输出**：FITS文件，存储在 `path_events` 目录，文件名格式为 `{short_name}_tte_{时间范围}_preliminary.fits` 。

`file_formatter.py` - 文件格式化

- **作用**：将事件数据格式化为标准化的科学数据产品（FITS文件）。
- **核心功能**：
  - 读取 `path_events` 中的FITS文件，提取事件数据。
  - 生成标准FITS头信息，包含任务信息（ `MISSION` 、 `CUBESAT` ）、时间范围等。
  - 创建三个扩展：
    - `EBOUNDS` ： 能量边界表，定义能量通道的上下限。
    - `GTI` ： 有效时间区间表，记录观测开始和结束时间。
    - `EVENTS{channel}` ： 事件表，包含时间（ `TIME` ）、能量通道（ `PI` ）、死时间（ `DEAD_TIME` ）和事件类型（ `EVT_TYPE` ）。
  - 使用 `np.digitize` 将能量值分配到能量通道。
- **输出**：FITS文件，存储在 `path_evt_files` 目录，文件名格式为 `G11_evt_{时间范围}.fits` 。

关于如何将流程用于处理新卫星载荷（如SAT-A-01）数据的说明

要处理SAT-A-01卫星的数据，只需创建一个新的探测器类（如 `SatA_Detector`），继承 `Detector` 基类。

或在SAT-A-01与GRID-11B探测器高度相似的情况下，继承 `GRID11BDetector` 类（定义在 `detector_grid_11b.py` 中）。

步骤1：创建 `SatA_01_Detector` 类

- **继承 `Detector` 基类**：
  - 在 `detectors` 目录下创建新文件（如 `detector_sat_01a.py`）。
  - 定义 `SatA_01_Detector` 类，继承 `Detector`：

```
from detector import Detector

class SatA_01_Detector(Detector):
    def __init__(self):
        super().__init__(
            detector_name='SAT-A',
            short_name='01',
            source_pattern='SAT-A_*',  # SAT-A数据文件命名模式
            valid_channels=[0, 1, 2, 3]  # 根据SAT-A通道调整
        )
```

  - 如果SAT-A-01的数据格式与GRID-11B比较相似，可直接继承 `GRID11BDetector`，省去大量重复性代码工作：

```
from detector_grid_11b import GRID11BDetector

class SatA_Detector(GRID11BDetector):
    def __init__(self):
        super().__init__()
        self.detector_name = 'SAT-A'
        self.short_name = '01'
        self.source_pattern = 'SAT-A_*'
        self.valid_channels = [0, 1, 2, 3]
```

步骤2：实现特定处理方法

- 如果SAT-A与GRID-11B的数据结构类似，可复用 `GRID11BDetector` 的绝大部分方法，仅重写差异部分。
  - **`extract_packets`**： 可能需要匹配新的帧头帧尾。
  - **`unpack_packets`**： 提取时间、通道、ADC值或能谱等字段时，对应于不同的字段。
  - **`calibrate_adc`**： 进行ADC校正时， 使用新的校准文件或公式。
  - **`adc_to_energy`**： 基于新的能量道址对应关系。

步骤3：更新 `main.py`

- 修改 `main.py` 以支持SAT-A-01探测器，添加新的条件分支：

```
# payload_number = '11'
payload_number = 'SAT-A-01'  # 或从配置文件读取

if payload_number == '11':
    from detectors.detector_grid_11b import GRID11BDetector
    detector = GRID11BDetector()

elif payload_number == 'SAT-A-01':
    from detectors.detector_sat_a import SatA_Detector
    detector = SatA01_Detector()
```

- 核心处理流程（四个模块）无需修改，因为它们只调用 `detector` 这个通用接口，而detector已经在初始化对象时完成了所有载荷的个性化流程定义。

步骤4：准备SAT-A数据和校准文件

- 将SAT-A的原始 `.dat` 文件放置在指定目录（如 `paths.source_dir/SAT-A/data/` ），确保文件名匹配 `source_pattern` 。
- 提供SAT-A的校准文件（如能量校准系数JSON文件或能量边界Numpy文件），并在 `SatA01_Detector` 中指定读取路径。