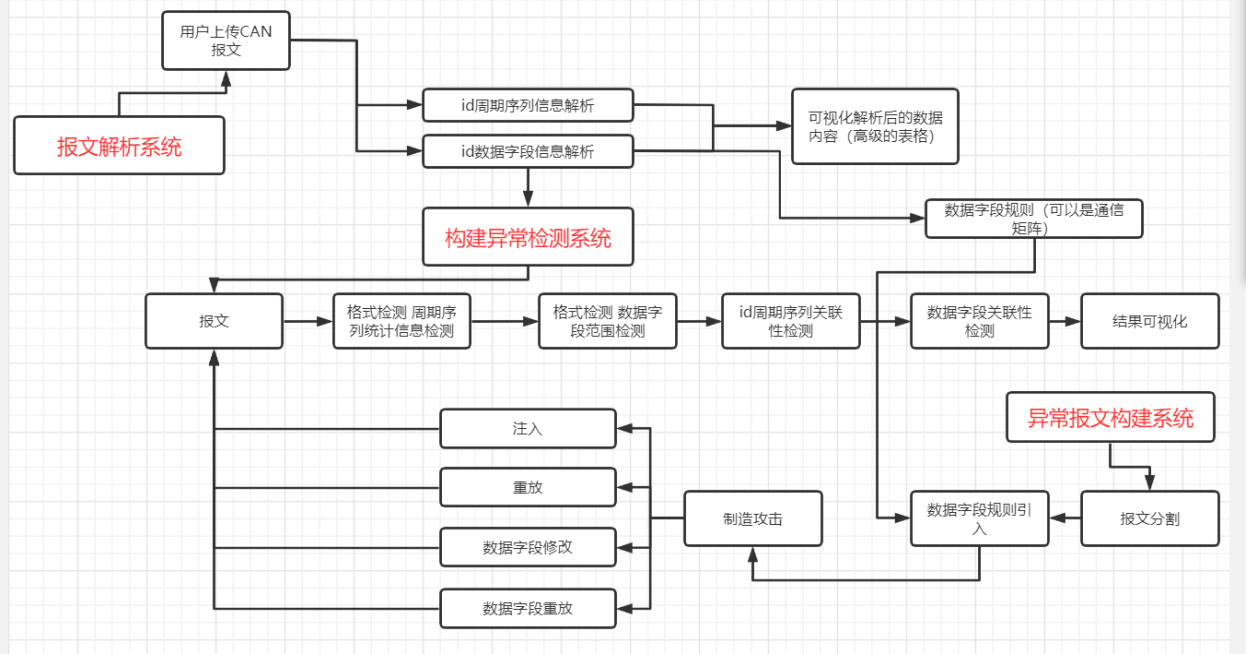
首先做一张流程图，形象展示整个处理过程

认为这个流程图是比较严谨的工作。

想要把整个系统可视化，算是比较大的工作了。



各个系统的API推荐，首先确定接口，再进行编程

暂定的后台整个系统的API函数，为后面的统一开发打下了基础。

```python
# 数据格式统一规定，命名类名为小驼峰，其余为下划线命名方式

# 最底层的数据单元
class SingleData:
    can_id = None
    # 这里的time有点不太清楚
    time = None
    data_in_hex = None
    data_in_binary = None

# 用于枚举数据序列的类型 想要自己定义什么类，可以直接往里面加
class DataListType():
    origin_data_list = 1 # 原始数据
    split_by_canid_origin_data_list = 2 # 按照can_id分割的原始数据

```

```
    not_attacked_data_list = 3 # 按照3s为间隔分割的正常数据
    erase_attack_data_list = 4 # 按照3s为间隔分割的攻击数据
    insert_attack_data_list = 5
    reput_attack_data_list = 6
    changedatafield_attack_data_list = 7

# 用于存储数据序列的类，统一使用的底层类
class DataList:
    # 这个是数据的区分单位，用于读入某一小段数据
    data_list = None # 初始化的data_list 里面存储的应当是SingleData类的一个列表
    data_list_type = None # 专门用于指定类型 self.data_list_type = DataListType.origin_data_list
    # 可以考虑从前台接受一个上传的文件进行初始化
    def __init__(self):
        return None

# 报文解析类，集成解析和数据展示接口，为前端的写法提供基础
class DataParsing:

    # id序列上的规则，统计规则
    period_rule = None
    # 数据字段的多种规则
    data_field_ruel = [] # 存放的类为Rule
    # 此函数专用于id序列周期特性解析，需要自己实现内部内容，或者自己新建类实现
    def id_parsing(self):
        return None
    # 专门用于可视化，需要形象的高级图表
    # 规定返回值为生成图表的绝对路径，方便前台加载
    def id_show_pic(self):
        return None

    # 数据域分界算法，用到先确定传感器，再决定别的字段的手段
    # 返回值为rule类的列表，返回某一个id报文的所有规则串 list Rule
    def data_field_parsing(self, id, datalist):
        return None
    # 要求同上，返回可视化图片的绝对路径
    def data_field_show_pic(self):
        return None
# 图中的异常检测系统，如何与前端对接呢？暂时是不清楚的
class AnomalyDetectionSystem:
    def id_format_detection(self):
        return None
```

```
    def data_format_detection(self):
        return None
    def id_relationship_detection(self):
        return None
    def data_relationship_detection(self):
        return None

class AttackCreate:
    # 这个类我已经写好了，实现暂时都没有写进去
    # 所有方法最终会返回DataList类，为后面的处理打下了基础
    def insert_attack(self, id, normal_T, ratio, exist_time)
    def erase_attack(self, id, exist_time):
    def reput_attack(self, id):
    def get_rule(self, doc_path):
    def changedatafield_attack(self, id, doc_path):
    def change_data_field(self, binary_str, rule_list):

# 专门用于存储规则的类，数据域的内容
class Rule:
    begin_loc = 0
    end_loc = 0
    type_of_class = None
    range = []
    def __init__(self):
        return None
    def init_single_rule(self, begin_loc, end_loc, type_of_class,
range):
        self.begin_loc = begin_loc
        self.end_loc = end_loc
        self.type_of_class = type_of_class
        self.range = range
```