app.cpp

包含

```
ThreadPool.h              -------//线程池
basePerson.hpp            -------//识别人
safeArea.hpp              -------//安全空间
window.hpp                -------//输出界面
SmokePhone.hpp
firesmog.hpp              -------//多个场景
helmet.hpp
xunjian.hpp
uptruck.hpp
zhuangxieyou.hpp          -------//本次的主要场景


firesmog.hpp
xunjian.hpp
convery.hpp
uptruck.hpp
```

ThreadPool.h

```
class ThreadPool {
public:
    ThreadPool(size_t);  //指定线程池的线程数量
    template<class F, class... Args>
    auto enqueue(F&& f, Args&&... args) //添加任务到线程池中
        -> std::future<typename std::result_of<F(Args...)>::type>;
    //用于获得任务的返回值
    ~ThreadPool();
private:
    // need to keep track of threads so we can join them
    std::vector< std::thread > workers;//存取线程池中的线程对象的向量
    // the task queue
    std::queue< std::function<void()> > tasks;//任务队列

    // synchronization
    std::mutex queue_mutex;//互斥量
    std::condition_variable condition;//条件变量
    bool stop;//是否停止执行任务
};



//构造函数
inline ThreadPool::ThreadPool(size_t threads)
    :   stop(false)
{
```

```
    for(size_t i = 0;i<threads;++i)
        workers.emplace_back(//创建一个线程
            [this]
            {
                for(;;)
                {
                    std::function<void()> task;

                    std::thread::id =id std::this_thread::get_id();//获取线程id
                    {
                        #ifdef ANNIWO_INTERNAL_DEBUG
                        std::cout<<"ThreadPool worker calling lock "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                        #endif
                        std::unique_lock<std::mutex> lock(this->queue_mutex);

                        //创建一个变量lock，使用queue_mutex互斥锁进行加锁操作。
                        #ifdef ANNIWO_INTERNAL_DEBUG
                        std::cout<<"ThreadPool worker called lock "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                        #endif

                        if(this->stop )
                            //检查stop变量的值，如果为true，则说明线程池已经停止，直接返回。
                        {
                            #ifdef ANNIWO_INTERNAL_DEBUG
                            std::cout<<"ThreadPool worker returned "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                            #endif
                            return;
                        }
                        //xiangbin:也就是说，一旦带有pred的wait被notify的时候，它会去检查谓词对象的bool返回值是否是true，如果是true才真正唤醒，否则继续block
                        //当销毁threadpool时候此处可能造成死锁，因为当tasks.empty为true时候
                        #ifdef ANNIWO_INTERNAL_DEBUG
                        std::cout<<"ThreadPool worker calling condition.wait "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                        #endif

                        this->condition.wait(lock,
                            [this]{ return this->stop || !this->tasks.empty(); });
                        // 调用condition的wait函数，等待条件满足。条件是stop为true或者tasks不为空。当条件不满足时，线程会被阻塞。
                        #ifdef ANNIWO_INTERNAL_DEBUG
                        std::cout<<"ThreadPool worker called condition.wait "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                        #endif

                        if(this->stop )
                        {
                            #ifdef ANNIWO_INTERNAL_DEBUG
                            std::cout<<"ThreadPool worker returned "<<this->stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
```

```
                            #endif

                            return;
                        }
                        task = std::move(this->tasks.front());
                        this->tasks.pop();//出队列
                    }

                    #ifdef ANNIWO_INTERNAL_DEBUG
                    std::cout<<"ThreadPool worker calling task "<<this-
>stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                    #endif

                    task();//执行任务

                    #ifdef ANNIWO_INTERNAL_DEBUG
                    std::cout<<"ThreadPool worker called task "<<this-
>stop<<this->tasks.size()<<",th:"<<std::hash<std::thread::id>()(id)<<std::endl;
                    #endif


                }
            }
        );
}


template<class F, class... Args>
auto ThreadPool::enqueue(F&& f, Args&&... args)
    -> std::future<typename std::result_of<F(Args...)>::type>
{
    using return_type = typename std::result_of<F(Args...)>::type;//获得f的返回值
//然后，将任务添加到任务队列中。这里使用了std::bind和std::forward来绑定可调用对象f和参数
args，并将其作为一个新的可调用对象添加到任务队列中。
//std::bind函数就像一个函数适配器，接受一个可调用对象，生成一个新的可调用对象来“适应”原对象的
参数列表。
//std::forward<T>(u)当T为左值引用类型时，u将被转换为T类型的左值，否则u将被转换为T类型右值。
    auto task = std::make_shared< std::packaged_task<return_type()> >(
            std::bind(std::forward<F>(f), std::forward<Args>(args)...)
        );//创建一个shared_ptr指向packaged_task<return_type()>的任务task。
    //packaged_task是一个可以将调用的对象封装成一个可调用的对象，并且可以获得返回值的类的模
板。


    //future对象可以用于等待任务的完成，并获取任务的返回值。通过调用future对象的get方法，可以
阻塞当前线程，直到任务完成并返回其结果。如果任务还没有完成，get方法将会阻塞，直到任务完成为止。
    std::future<return_type> res = task->get_future();//获得任务的future对象res。
    {
        std::unique_lock<std::mutex> lock(queue_mutex);//加锁互斥

        // don't allow enqueueing after stopping the pool
        if(stop)//检查线程池是否停止，如果停止抛出异常
            throw std::runtime_error("enqueue on stopped ThreadPool");

        //    std::queue< std::function<void()> > tasks;
        tasks.emplace([task](){ (*task)(); });//insert操作
    }
    condition.notify_one();//通知下一个对象
```

```
    return res;
}



inline ThreadPool::~ThreadPool()
{
    {
        std::unique_lock<std::mutex> lock(queue_mutex);//加锁
        stop = true;//线程池停止
#ifdef ANNIWO_INTERNAL_DEBUG
        std::cout<<"ThreadPool entered ~ this->tasks.size:"<<this->tasks.size()
<<"workers.size"<<workers.size()<<std::endl;
#endif
    }
    condition.notify_all();//通知结束

#ifdef ANNIWO_INTERNAL_DEBUG
    std::cout<<"ThreadPool ~ this->tasks.size:"<<this->tasks.size()
<<"workers.size"<<workers.size()<<std::endl;
#endif

    for(std::thread &worker: workers)//遍历工作线程,
    {
// #ifdef ANNIWO_INTERNAL_DEBUG
        std::cout<<"ThreadPool ~ joining th:"<<std::hash<std::thread::id>()
(worker.get_id())<<" worker this->tasks.size:"<<this->tasks.size()
<<"workers.size"<<workers.size()<<std::endl;
// #endif

        worker.join();   //调用join方法等待执行完成
// #ifdef ANNIWO_INTERNAL_DEBUG
        std::cout<<"ThreadPool ~ joined  th:"<<std::hash<std::thread::id>()
(worker.get_id())<<"  worker this->tasks.size:"<<this->tasks.size()
<<"workers.size"<<workers.size()<<std::endl;
// #endif


    }
}
```

basePerson.hpp

```
utils_intersection.hpp"
subUtils.hpp"
object_detector.h"
yolo_common.hpp"

------
```

zhuangxieyou.hpp

```
utils_intersection.hpp   (✓)
yolo_common.hpp          (✓)      -----之前的部分
//提供了初始化目标跟踪和进行目标检测的功能
class ZxyDetection  {
public:
      ZxyDetection() ;
      ~ZxyDetection() ;
//初始化追踪
      void initTracks(
          const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj);

        //todo:polygonSafeArea
//进行目标检测
      static void detect(  int camID, int instanceID,cv::Mat& img,  const
Polygon* polygonSafeArea_ptr);

};
```

utils_intersection.hpp

```
float distPoint( cv::Point p1, cv::Point p2 ) ;
float distPoint(cv::Point2f p1,cv::Point2f p2) ;
bool segementIntersection(cv::Point p0_seg0,cv::Point p1_seg0,cv::Point
p0_seg1,cv::Point p1_seg1,cv::Point * intersection) ;
bool segementIntersection(cv::Point2f p0_seg0,cv::Point2f p1_seg0,cv::Point2f
p0_seg1,cv::Point2f p1_seg1,cv::Point2f * intersection) ;

bool pointInPolygon(cv::Point p,const cv::Point * points,int n) ;
bool pointInPolygon(cv::Point2f p,const cv::Point2f * points,int n) ;


#define MAX_POINT_POLYGON 64
struct Polygon {//表示多边形
    cv::Point pt[MAX_POINT_POLYGON];
    int n;
```

```
    Polygon(int n_ = 0 ) { assert(n_>= 0 && n_ < MAX_POINT_POLYGON); n = n_;}
    virtual ~Polygon() {}

    void clear() { n = 0; }//清除
    void add(const cv::Point &p) {if(n < MAX_POINT_POLYGON) pt[n++] = p;}//添加
    void push_back(const cv::Point &p) {add(p);}
    int size() const { return n;}
    cv::Point getCenter() const ;
    const cv::Point & operator[] (int index) const { assert(index >= 0 && index
< n); return pt[index];}//重构运算符[]
    cv::Point& operator[] (int index) { assert(index >= 0 && index < n); return
pt[index]; }
    void pointsOrdered() ;
    float area() const ;
    bool pointIsInPolygon(cv::Point p) const ;
};


void intersectPolygon( const cv::Point * poly0, int n0,const cv::Point *
poly1,int n1, Polygon & inter ) ;
void intersectPolygon( const Polygon & poly0, const Polygon & poly1, Polygon &
inter ) ;
void intersectPolygonSHPC(const Polygon * sub,const Polygon* clip,Polygon* res)
;
void intersectPolygonSHPC(const Polygon & sub,const Polygon& clip,Polygon& res)
;



//时间日志
struct AnniwoTimeLog
{
    double configInterval; //ms//它可以用来设置时间日志的记录间隔，即每隔多少时间记录一次
时间点。
    std::chrono::steady_clock::time_point lastTP;//记录最后一次记录的时间点。
};

struct AnniwoStay
{
    bool isMotionless; //isMotionless:是否从静止开始计算停留时间
    int staySec;  //要求停留秒数，超过该时间再报警
};

struct AnniwoSafeEreaConcernTypes
{
    std::vector<std::string> validtypes;//关注类型
    std::vector<std::string> excludeTypes;。。排除类型
};



struct ANNIWO_JSON_CONF_CLASS
{
//id对应识别功能
std::unordered_map<int, std::vector<std::string> > id_func_cap;
//摄像头，用于存储时间间隔信息
```

```
std::unordered_map<int,std::unordered_map<std::string, AnniwoTimeLog > >
interval_conf_map;
//对应停留时间配置信息
std::unordered_map<int,std::unordered_map<std::string, AnniwoStay> >
stay_conf_map;
//用于存储不同功能对应的有效区域的信息
std::unordered_map<int,std::unordered_map<std::string, Polygon>  >
validArea_conf_map;
//用于不同功能
std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >  validtypes_map;
//对应不同功能对应每个小时有效的时间段配置信息
std::unordered_map<int,std::unordered_map<std::string, std::vector<Polygon>>  >
 validperoids_hour_map;
    //对应每周有效的时间段配置信息
std::unordered_map<int,std::unordered_map<std::string, std::vector<std::string>>
 >  validperoids_week_map;

//对应不同功能任务id信息
//camId,{func,stringtaskid}
std::unordered_map<int,std::unordered_map<std::string, std::string>  >
taskid_conf_map;
//不同功能人脸数据采集路径
std::unordered_map<int,std::unordered_map<std::string, std::string>  >
facedatasetpath_conf_map;
//camId,{func,eventUrl}
 //不用功能对应事件URL
std::unordered_map<int,std::unordered_map<std::string, std::string>  >
eventUrl_conf_map;
//存储不同功能对应缺席功能的启动条件。
std::unordered_map<int,std::unordered_map<std::string, std::string>  >
absenceStartConition_conf_map;
};

//获取相机ID和任务ID之间的映射关系，并根据给定的函数名称找到对应的任务ID
inline void getTaskId(const ANNIWO_JSON_CONF_CLASS* globalJsonConfObjPtr,int
camID,const std::string strFuncname,/*out*/std::string& taskIdstr)
{
    //camId,{func,stringtaskid}
    const std::unordered_map<int,std::unordered_map<std::string, std::string>
 >* taskid_conf_map_ptr= &globalJsonConfObjPtr->taskid_conf_map;

    //取得taskId设置
    //camId,{func,taskId}
    //寻找相机ID对应的任务ID映射关系。
    std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::const_iterator got_id_func_cap = taskid_conf_map_ptr->find(camID);

    if (got_id_func_cap == taskid_conf_map_ptr->end())
    {//未找到
        ANNIWOLOG(INFO) << "not found in taskid_conf_map,camID:" <<camID;
    }
    else
    {//存储到conf_map中
        const std::unordered_map<std::string, std::string>& conf_map
=got_id_func_cap->second;
        //在conf_map中寻找对应任务的ID。
```

```
        std::unordered_map<std::string, std::string>::const_iterator
got_id_func_cap2 = conf_map.find(strFuncname);
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "not found "<< strFuncname<<" in
taskid_conf_map,camID:" <<camID;
        }
        else
        {
            taskIdstr = got_id_func_cap2->second ;
        }
    }
}

//strFuncname:查找配置关键字
//suffix:添加在配置之后
//submitUrl:输出

//- "camID" 是一个整数，表示相机ID。
//- "strFuncname" 是一个字符串，表示函数名称。
//- "suffix" 是一个字符串，表示URL的后缀部分。
//- "submitUrl" 是一个传引用的字符串，用于存储获取到的事件URL。

//从全局配置对象中获取相机ID和事件URL之间的映射关系，并根据给定的函数名称找到对应的事件URL，并
将其与后缀部分拼接，最后将结果存储在 "submitUrl" 中。
inline void getEventUrl(const ANNIWO_JSON_CONF_CLASS* globalJsonConfObjPtr,int
camID,const std::string strFuncname,const std::string suffix,/*out*/std::string&
submitUrl)
{
    //camId,{func,stringenventurl}
    const std::unordered_map<int,std::unordered_map<std::string, std::string>
 >* eventUrl_conf_map_ptr= &globalJsonConfObjPtr->eventUrl_conf_map;

    //取得eventUrl设置
    //camId,{func,eventUrl}
    std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::const_iterator got_id_func_cap = eventUrl_conf_map_ptr->find(camID);

    if (got_id_func_cap == eventUrl_conf_map_ptr->end())
    {
        ANNIWOLOG(INFO) << "not found in eventUrl_conf_map,camID:" <<camID;
    }
    else
    {
        const std::unordered_map<std::string, std::string>& conf_map
=got_id_func_cap->second;
        std::unordered_map<std::string, std::string>::const_iterator
got_id_func_cap2 = conf_map.find(strFuncname);
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "not found "<< strFuncname<<" in
eventUrl_conf_map,camID:" <<camID;
        }
        else
        {
            submitUrl = got_id_func_cap2->second + suffix;
        }
    }
```

```
}
```

utils_intersection.cpp

```
float distPoint(Point2f v,Point2f w) {
    return sqrtf((v.x - w.x)*(v.x - w.x) + (v.y - w.y)*(v.y - w.y)) ;
}
//计算两个二维点之间的欧氏距离的函数。
//函数先计算两个点在x轴上的差值和y轴上的差值，然后分别平方并相加，最后取平方根，得到两个点之间的
欧氏距离。
//用于特征匹配。
float distPoint(Point p1,Point p2) {
    int x = p1.x - p2.x;
    int y = p1.y - p2.y;
    return sqrt((float)x*x+y*y);
}//相同用于计算欧氏距离。

//有什么实质的区别吗?

static Point2f Point32f( Point p) {
    Point2f s;
    s.x= (float)p.x;
    s.y= (float)p.y;
    return s;
}//转化为float类型的点，计算数据，返回时使用int?

bool segementIntersection(Point p0,Point p1,Point p2,Point p3,Point *
intersection) {
    Point2f p;
    bool ok =
segementIntersection(Point32f(p0),Point32f(p1),Point32f(p2),Point32f(p3),&p);
    if( ok )
        *intersection = Point((int)p.x,(int)p.y);
    return ok;
}
//计算两条线段是否相交，并求出交点坐标。
bool segementIntersection(Point2f p0,Point2f p1,Point2f p2,Point2f p3,Point2f *
intersection) {
    Point2f s1, s2;
    s1 = Point2f(p1.x - p0.x, p1.y - p0.y);//计算方向向量
    s2 = Point2f(p3.x - p2.x, p3.y - p2.y);

    float s10_x = p1.x - p0.x;
    float s10_y = p1.y - p0.y;
    float s32_x = p3.x - p2.x;
    float s32_y = p3.y - p2.y;
    //计算两条线段的差值和乘积//如果为0平行
    float denom = s10_x * s32_y - s32_x * s10_y;

    if(denom == 0) {
        return false;
    }

    bool denom_positive = denom > 0;
//判断是否有交点
    float s02_x = p0.x - p2.x;
    float s02_y = p0.y - p2.y;
```

```
    float s_numer = s10_x * s02_y - s10_y * s02_x;
//如果小于0为没有交点
    if((s_numer < 0.f) == denom_positive) {
        return false;
    }

    float t_numer = s32_x * s02_y - s32_y * s02_x;
    if((t_numer < 0) == denom_positive) {
        return false;
    }
//最后，根据s_numer、denom和t_numer的值判断交点是否在两条线段的范围内。
    if((s_numer > denom) == denom_positive || (t_numer > denom) ==
denom_positive) {
        return false;
    }

    float t = t_numer / denom;

    *intersection = Point2f(p0.x + (t * s10_x), p0.y + (t * s10_y) );//把交点赋值
到指针空间中
    return true;
}
//判断一个点是否在多边形内部。
//函数通过遍历多边形的每条边，计算点p与当前边的距离，根据交点个数的奇偶性来判断点p是否在多边形内
部。
bool pointInPolygon(Point2f p,const Point2f * points,int n) {
    int i, j ;
    bool c = false;
    for(i = 0, j = n - 1; i < n; j = i++) {
        if( ( (points[i].y >= p.y ) != (points[j].y >= p.y) ) &&
            (p.x <= (points[j].x - points[i].x) * (p.y - points[i].y) /
(points[j].y - points[i].y) + points[i].x)
            )
            c = !c;
    }
    return c;
}

bool pointInPolygon(Point p,const Point * points,int n) {
    int i, j ;
    bool c = false;
    for(i = 0, j = n - 1; i < n; j = i++) {
        if( ( (points[i].y >= p.y ) != (points[j].y >= p.y) ) &&
            (p.x <= (points[j].x - points[i].x) * (p.y - points[i].y) /
(points[j].y - points[i].y) + points[i].x)
            )
            c = !c;
    }
    return c;
}

//计算多边形面积的函数
//函数通过遍历多边形的每条边，计算每条边与x轴的有向面积之和，最后取绝对值并除以2得到多边形的面
积。
float computeArea(const Point * pt,int n ) {
    float area0 = 0.f;
    for (int i = 0 ; i < n ; i++ ) {
        int j = (i+1)%n;
```

```
        area0 += pt[i].x * pt[j].y;
        area0 -= pt[i].y * pt[j].x;
    }
    return 0.5f * fabs(area0);
}

struct PointAngle{
    Point p;
    float angle;
};
//得到中心点
Point Polygon::getCenter() const {
    Point center;
    center.x = 0;
    center.y = 0;
    for (int i = 0 ; i < n ; i++ ) {
        center.x += pt[i].x;
        center.y += pt[i].y;
    }
    center.x /= n;
    center.y /= n;
    return center;
}
//比较角度大小
static int comp_point_with_angle(const void * a, const void * b) {
    if ( ((PointAngle*)a)->angle <  ((PointAngle*)b)->angle ) return -1;
    else if ( ((PointAngle*)a)->angle > ((PointAngle*)b)->angle ) return 1;
    else //if ( ((PointAngle*)a)->angle == ((PointAngle*)b)->angle ) return 0;
        return 0;
}
//面积
float Polygon::area() const {
    return computeArea(pt,n);
}
//角度进行排序
void Polygon::pointsOrdered() {
    if( n <= 0) return;
    Point center = getCenter();//中心点
    PointAngle pc[MAX_POINT_POLYGON];
    for (int i = 0 ; i < n ; i++ ) {
        pc[i].p.x = pt[i].x;
        pc[i].p.y = pt[i].y;
        //计算出角度
        pc[i].angle = atan2f((float)(pt[i].y - center.y),(float)(pt[i].x -
center.x));
    }
    //排序
    qsort(pc,n,sizeof(PointAngle),comp_point_with_angle);
    for (int i = 0 ; i < n ; i++ ) {
        pt[i].x = pc[i].p.x;
        pt[i].y = pc[i].p.y;
    }
}

bool Polygon::pointIsInPolygon(Point p) const {
    return pointInPolygon(p,pt,n);
}
```

```
//计算两个多边形的交集
void intersectPolygon( const Point * poly0, int n0,const Point * poly1,int n1,
Polygon & inter )  {
    inter.clear();
    for (int i = 0 ; i < n0 ;i++) {
        //判断点是否在里面
        if( pointInPolygon(poly0[i],poly1,n1) ) {
            inter.add(poly0[i]);
        }
    }

    for (int i = 0 ; i < n1 ;i++) {
        if( pointInPolygon(poly1[i],poly0,n0) )
            inter.add(poly1[i]);
    }

    for (int i = 0 ; i < n0 ;i++) {
        Point p0,p1,p2,p3;
        p0 = poly0[i];//获得一条边
        p1 = poly0[(i+1)%n0];
        for (int j = 0 ; j < n1 ;j++) {
            p2 = poly1[j];//获得另一条边
            p3 = poly1[(j+1)%n1];
            Point pinter;
            //
            if(segementIntersection(p0,p1,p2,p3,&pinter)) {//寻找交点
                inter.add(pinter);
            }
        }
    }
    inter.pointsOrdered();//对角度进行排序
}

void intersectPolygon( const Polygon & poly0, const Polygon & poly1, Polygon &
inter ) {
    intersectPolygon(&poly0[0],poly0.size(),&poly1[0],poly1.size(),inter);
}


//-------------------------------------------------------------------------------
------------------------------
//SHPC :  Sutherland-Hodgeman-Polygon-Clipping Algorihtm
//Sutherland—Hodgeman多边形裁剪算法具有一般性，被裁剪多边形可以是任意凸多边形或凹多边形，裁
剪窗口不局限于矩形，可以是任意凸多边形。
//-------------------------------------------------------------------------------
------------------------------

//返回计算得到的叉积值。
static inline int cross(const Point* a,const Point* b) {
    return a->x * b->y - a->y * b->x;
}
//方向向量
static inline Point* vsub(const Point* a,const Point* b, Point* res) {
    res->x = a->x - b->x;
    res->y = a->y - b->y;
    return res;
}
//求线段的交点
```

```
static int line_sect(const Point* x0,const Point* x1,const Point* y0,const
Point* y1, Point* res) {
    Point dx, dy, d;
    vsub(x1, x0, &dx);
    vsub(y1, y0, &dy);
    vsub(x0, y0, &d);
    float dyx = (float)cross(&dy, &dx);
    if (!dyx) return 0;
    dyx = cross(&d, &dx) / dyx;
    if (dyx <= 0 || dyx >= 1) return 0;
    res->x = int(y0->x + dyx * dy.x);
    res->y = int(y0->y + dyx * dy.y);
    return 1;
}
//判断点c相对于线段ab的位置关系
static int left_of(const Point* a,const Point* b,const Point* c) {
    Point tmp1, tmp2;
    int x;
    vsub(b, a, &tmp1);
    vsub(c, b, &tmp2);
    x = cross(&tmp1, &tmp2);
    return x < 0 ? -1 : x > 0;//小于0左侧，大于0右侧，等于0线上
}
//对多边形进行裁剪
//两个指向cv::Point类型的指针x0和x1，表示裁剪边界的起点和终点；一个int类型的变量left，表示裁
剪边界的方向；以及一个指向Polygon结构体的指针res，用于存储裁剪后的多边形。
//???
static void poly_edge_clip(const Polygon* sub,const  Point* x0,const  Point* x1,
int left, Polygon* res) {
    int i, side0, side1;
    Point tmp;
    const Point* v0 = sub->pt+ sub->n - 1;
    const Point* v1;
    res->clear();

    side0 = left_of(x0, x1, v0);
    if (side0 != -left) res->add(*v0);

    for (i = 0; i < sub->n; i++) {
        v1 = sub->pt + i;
        side1 = left_of(x0, x1, v1);
        if (side0 + side1 == 0 && side0)
            /* last point and current straddle the edge */
            if (line_sect(x0, x1, v0, v1, &tmp))
                res->add(tmp);
        if (i == sub->n - 1) break;
        if (side1 != -left) res->add(*v1);
        v0 = v1;
        side0 = side1;
    }
}
//用于判断多边形的绕向
static int poly_winding(const Polygon* p) {
    return left_of(p->pt, p->pt + 1, p->pt + 2);
}
//用于计算两个多边形的交集
void intersectPolygonSHPC(const Polygon * sub,const Polygon* clip,Polygon* res)
{
```

```
    int i;
    Polygon P1,P2;
    Polygon * p1 = &P1;
    Polygon * p2 = &P2;
    Polygon * tmp ;

    int dir = poly_winding(clip);
    poly_edge_clip(sub, clip->pt + clip->n - 1, clip->pt, dir, p2);
    for (i = 0; i < clip->n - 1; i++) {
        tmp = p2; p2 = p1; p1 = tmp;
        if(p1->n == 0) {
            p2->n = 0;
            break;
        }
        poly_edge_clip(p1, clip->pt + i, clip->pt + i + 1, dir, p2);
    }
    res->clear();
    for (i = 0 ; i < p2->n ; i++) res->add(p2->pt[i]);
}

void intersectPolygonSHPC(const Polygon & sub,const Polygon& clip,Polygon& res)
{
    intersectPolygonSHPC(&sub,&clip,&res);
}
```

basePerson.hpp

```
#include "../utils/utils_intersection.hpp"    (✓)
#include "../utils/subUtils.hpp"               ()
#include "../common/ppyoloe/object_detector.h"(来自PPUOLOE深度解析)
#include "../common/yolo/yolo_common.hpp"      (YOLOV4)



extern const std::vector<std::string> person_base_class_namesJYZ ;
extern const std::vector<std::string> person_base_class_namesCOCO ;

//识别类型是否正确
inline bool isVehicle(int inlabel)
{
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_LIANGKU)
    {
        if(
           person_base_class_namesCOCO[inlabel] == std::string("bicycle")
        || person_base_class_namesCOCO[inlabel] == std::string("car")
        || person_base_class_namesCOCO[inlabel] == std::string("motorcycle")
        || person_base_class_namesCOCO[inlabel] == std::string("bus")
        || person_base_class_namesCOCO[inlabel] == std::string("truck")
        )
        {
            return true;
        }
    }
    else if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
```

```
        if(  person_base_class_namesJYZ[inlabel] == std::string("car")
            || person_base_class_namesJYZ[inlabel] == std::string("tank_truck")
            || person_base_class_namesJYZ[inlabel] == std::string("truck")
            || person_base_class_namesJYZ[inlabel] == std::string("motor")
            || person_base_class_namesJYZ[inlabel] == std::string("unloader")
            || person_base_class_namesJYZ[inlabel] ==
std::string("cement_truck")
        )
        {
            return true;
        }


    }
    else
    {
        ANNIWOCHECK(false);
    }

    return false;

}
//判断是否为people
inline bool isPerson(int inlabel)
{
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_LIANGKU)
    {
        if(person_base_class_namesCOCO[inlabel]  == std::string("person"))
        {
            return true;
        }
    }
    else if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        if(person_base_class_namesJYZ[inlabel]  == std::string("person"))
        {
            return true;
        }
    }
    else
    {
        ANNIWOCHECK(false);
    }
    return false;

}
//获得不同场景下的person的类型名
inline std::string getPersonCarbaseClassName(int inlabel)
{
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_LIANGKU)
    {
        return person_base_class_namesCOCO[inlabel];
    }
    else if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        return person_base_class_namesJYZ[inlabel];
```

```
    }
    else
    {
        ANNIWOCHECK(false);
    }
    return std::string("");
}
//人员追踪类
class BasePersonDetection  {
    public:
        BasePersonDetection() ;
        ~BasePersonDetection() ;

        void initTracks(const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj,const
std::unordered_set<std::string>&  person_base_functionsIn);

        //todo:polygonSafeArea
        static int detect(  int camID, int instanceID, cv::Mat& img,
std::vector<Object>& objects);

    private:
        static  std::unordered_map<int, std::vector<float> >  m_input_datas;
};

//临时记录结构，用于检查该trackid对应的目标停留时间是否已经达到阈值
// {camID,{trackid,stayStart}}
struct AnniwoTrackRecord
{
    Object detresult; //
    std::chrono::system_clock::time_point startPoint;  //stayStart时间
};

#endif // YOLOX_DEMO_H
```

basePerson.cpp

```
#include "../utils/httpUtil.hpp"

#include "../utils/rapidjson/writer.h"
#include "../utils/rapidjson/stringbuffer.h"
#include "../utils/subUtils.hpp"

#include "../common/yolo/yolo_common.hpp"

#include <uuid/uuid.h>


#include "../common/mot/include/deepsort.h"



//COCO数据集中类别的数量
static const int NUM_CLASSESCOCO = 80;
```

```
// static const float BBOX_CONF_THRESHCOCO = 0.3;  //yolox
//边界框置信度阈值
static const float BBOX_CONF_THRESHCOCO = 0.5999;   //coco pretrain

//识别类型
const std::vector<std::string> person_base_class_namesCOCO = {
    "person", //0
    "bicycle",//1
     "car",//2
     "motorcycle",//3
     "airplane",
     "bus", //5
     "train",
     "truck",//7
     "boat", "traffic light",
    "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat", "dog",
"horse", "sheep", "cow",
    "elephant", "bear", "zebra", "giraffe", "backpack", "umbrella", "handbag",
"tie", "suitcase", "frisbee",
    "skis", "snowboard", "sports ball", "kite", "baseball bat", "baseball
glove", "skateboard", "surfboard",
    "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon",
"bowl", "banana", "apple",
    "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut",
"cake", "chair", "couch",
    "potted plant", "bed", "dining table", "toilet", "tv", "laptop", "mouse",
"remote", "keyboard", "cell phone",
    "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock",
"vase", "scissors", "teddy bear",
    "hair drier", "toothbrush"
};


//基础类型数据集的个数
static const int NUM_CLASSESJYZCAR = 11;
//置信度
static const float BBOX_CONF_THRESHJYZCAR = 0.5999;

const std::vector<std::string> person_base_class_namesJYZ = {
"work_clothe_blue",
"person",
"car",
"work_clothe_yellow",
"tank_truck",
"truck",
"motor",
"reflective_vest",
"rider",
"cement_truck",
"reflective_vest_half"
};


//模型
const static std::string model_file_jyz_weilan=
{"../models/safearea/jyz_car/jyz_weilan_y4_sim.trt"};
// const static std::string ppyoloe_config_file_path_in =
{"../models/safearea/jyz_car/infer_cfg.yml"};
```

```
static DeepSort* DS = nullptr;

//引擎
static nvinfer1::IRuntime* runtime{nullptr};
static nvinfer1::ICudaEngine* engine{nullptr};

//TensorRT推理引擎的执行上下文，用于执行推理任务。
static std::unordered_map<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext>>
executionContexts;
//表示互斥锁
static std::unordered_map<int, std::unique_ptr<std::mutex> > contextlocks;
//表示当前使用的GPU设备的编号
static int gpuNum=0;



//yolov4 ouput flat size
//yolov4输出尺寸
static const int YOLO4_OUTPUT_SIZE_COCO = 1*7581*NUM_ANCHORS* (NUM_CLASSESCOCO +
5);

//模型文件地址
const static std::string model_file=
{"../models/safearea/personbase/yolov4personbase_sim.trt"};

//深度搜索模型文件地址
const static std::string deepsort_model_file_path_model=
{"../models/safearea/dstk_model/deepsort_sim.trt"};


//todo:用自己训练的加油站车辆reid
const static std::string deepsort_model_file_path_model2=
{"../models/safearea/dstk_model2/carRreid_sim.trt"};

//reid模型批处理大小
const int reidbatchsize=8;

//批处理大小
const int batch_size = 1;

//输入形状
static std::vector<int> input_shape = {batch_size, INPUT_H, INPUT_W, 3};
 //yolov4 keras/tf

//设置一个哈希表静态变量
std::unordered_map<int, std::vector<float> >
 BasePersonDetection::m_input_datas;

//构造函数
BasePersonDetection::BasePersonDetection () {

//根据不同的配置选择对应的模型文件
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_LIANGKU)
    {
            //初始化推理的上下文。
        gpuNum = initInferContext(
```

```
                        model_file.c_str(),
                        &runtime,
                        &engine);




    }
    else if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {

        gpuNum = initInferContext(
                model_file_jyz_weilan.c_str(),
                &runtime,
                &engine);


    }


   ANNIWOLOG(INFO) << "BasePersonDetection(): initilized!";

}

// Destructor//析构函数
BasePersonDetection::~BasePersonDetection ()
{

    // destroy the engine
    delete engine;
    delete runtime;

    engine=nullptr;
    runtime=nullptr;

}


//objects:是遵循person_base_class_namesJYZ的
//输出结果//加油站
static int mainfunc2(int camID,int instanceID,const cv::Mat& bgr,
std::vector<Object>& objects )
{

    std::vector<DetectBox> track_results;
    cv::Mat image = bgr.clone();


    int jsonObjCnt=0;
    int img_w = image.cols;//宽度
    int img_h = image.rows;//高度




    //过滤有用类别与准备track结构体
    for (size_t i = 0; i < objects.size(); i++)
    // for(auto iter=objects.begin(); iter!=objects.end();iter++)  //list
```

```
    {
        Object& obj = objects[i];
        //初始化id
        obj.trackID=-1;

        ANNIWOLOG(INFO) <<
        "BasePersonDetection: mainfunc2. input box:"<<obj.rect.x<<","<<
obj.rect.y<<","
        <<obj.rect.width<<","<<obj.rect.height<<","
        << "score:"<<obj.prob<<"class:"
<<person_base_class_namesJYZ[obj.label].c_str()<<",camID:"<<camID;



        if(    person_base_class_namesJYZ[obj.label]  == std::string("person")
            || person_base_class_namesJYZ[obj.label] == std::string("car")
            || person_base_class_namesJYZ[obj.label] ==
std::string("tank_truck")
            || person_base_class_namesJYZ[obj.label] == std::string("truck")
            || person_base_class_namesJYZ[obj.label] == std::string("motor")
            || person_base_class_namesJYZ[obj.label] == std::string("rider")
            || person_base_class_namesJYZ[obj.label] ==
std::string("cement_truck")
            || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_blue")
            || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_yellow")
            || person_base_class_namesJYZ[obj.label] ==
std::string("reflective_vest")
            || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_wathet")
        )
        {
            ANNIWOLOG(INFO) <<
            "BasePersonDetection: mainfunc2. box:"<<obj.rect.x<<","<<
obj.rect.y<<","
            <<obj.rect.width<<","<<obj.rect.height<<","
            << "score:"<<obj.prob<<"class:"
<<person_base_class_namesJYZ[obj.label].c_str()<<",camID:"<<camID;

            if(person_base_class_namesJYZ[obj.label] == std::string("car")
            || person_base_class_namesJYZ[obj.label] ==
std::string("tank_truck")
            || person_base_class_namesJYZ[obj.label] == std::string("truck")
            || person_base_class_namesJYZ[obj.label] == std::string("motor")
            || person_base_class_namesJYZ[obj.label] == std::string("rider")
            || person_base_class_namesJYZ[obj.label] ==
std::string("cement_truck")
            )
            {
                if(obj.rect.width < 80. || obj.rect.height < 80.)
                {
                    ANNIWOLOG(INFO) <<"Ignore small truck/car";
                    continue;
                }
            }

            float x1=obj.rect.x;
```

```
            float y1 = obj.rect.y;
            //右下角的坐标
            float x2=(obj.rect.x+obj.rect.width) > img_w ? img_w :
(obj.rect.x+obj.rect.width) ;
            float y2 =(obj.rect.y+obj.rect.height) > img_h ? img_h:
(obj.rect.y+obj.rect.height);




            DetectBox detBoxObj;
            detBoxObj.x1 = x1;
            detBoxObj.y1= y1;
            detBoxObj.x2= x2;
            detBoxObj.y2= y2;
            detBoxObj.confidence = obj.prob;

            //区分人，车。因为需要用不同的reid encoder.
            //0:为人，1为车
            //工作服无法跟踪，在各自功能自行处理
            //0:"person"
            if(    person_base_class_namesJYZ[obj.label]  ==
std::string("person")
                || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_blue")
                || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_yellow")
               //反光背心
                || person_base_class_namesJYZ[obj.label] ==
std::string("reflective_vest")
               //
                || person_base_class_namesJYZ[obj.label] ==
std::string("work_clothe_wathet")
            )
            {
                detBoxObj.classID = 0;
            }
            else
            {
                detBoxObj.classID = 1;
            }
            //
            uuid_generate_random(detBoxObj.uuid);
            track_results.push_back(detBoxObj);

            obj.trackID=-1;

            strncpy((char*)obj.uuid, (char*)detBoxObj.uuid, sizeof(uuid_t));


            jsonObjCnt++;

        }

    }
```

```
//进行跟踪
/////////////TRACKING PART//////////////////////////////

    if(track_results.size() > 0)
    {
        //static DeepSort* DS = nullptr;
        if(DS)
        {
            cv::Mat img_rgb;
            //转化为RGB
            cv::cvtColor(image, img_rgb, cv::COLOR_BGR2RGB);
            //分类
            DS->sort(img_rgb, track_results,camID,instanceID);
        }else
        {
            ANNIWOLOG(INFO) <<"BasePersonDetection:mainfunc2 FATAL ERROR! DS is
null!\n";
        }

    }

    //匹配track与det框。
    for (auto & personvehicel_det : objects) {
        int trackID = -1;

// "work_clothe_blue",
// "person",
// "car",
// "work_clothe_yellow",
// "tank_truck",
// "truck",
// "motor",
// "reflective_vest",
// "rider",
// "cement_truck"

        if(     person_base_class_namesJYZ[personvehicel_det.label]  ==
std::string("person")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("car")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("motor")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("tank_truck")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("truck")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("rider")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("cement_truck")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("work_clothe_blue")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("work_clothe_yellow")
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("reflective_vest")
```

```
            || person_base_class_namesJYZ[personvehicel_det.label] ==
std::string("work_clothe_wathet")
        )
        {
            for (auto & box : track_results ) {
                // if(box.uuid == personvehicel_det.uuid)
                //查看跟踪的是否相同
                if(0 == strncmp( (char *)box.uuid,  (char
*)personvehicel_det.uuid, sizeof(uuid_t)))
                {
                    trackID = (int)box.trackID;
                    personvehicel_det.trackID = trackID;

                }
            }

            ANNIWOLOG(INFO) <<"BasePersonDetection:mainfunc2:"
<<person_base_class_namesJYZ[personvehicel_det.label].c_str()<<","<<
personvehicel_det.prob<<","<< "tid:"<<trackID<<  "  x, y, w,  h:"
<<personvehicel_det.rect.x<<","<<personvehicel_det.rect.y<<","
<<personvehicel_det.rect.width<<","<<personvehicel_det.rect.height<<","<<"
camID:"<<camID;


        }

    }



    return jsonObjCnt;

}



//粮仓
static int mainfunc(int camID,int instanceID,const cv::Mat& bgr,
std::vector<Object>& objects )
{

    std::vector<DetectBox> track_results;
    cv::Mat image = bgr.clone();


    int jsonObjCnt=0;
    int img_w = image.cols;
    int img_h = image.rows;

    //过滤有用类别与准备track结构体
    for (size_t i = 0; i < objects.size(); i++)
    {
        Object& obj = objects[i];
        obj.trackID=-1;


        if(person_base_class_namesCOCO[obj.label]  == std::string("person")
        || person_base_class_namesCOCO[obj.label] == std::string("bicycle")
```

```
        || person_base_class_namesCOCO[obj.label] == std::string("car")
        || person_base_class_namesCOCO[obj.label] == std::string("motorcycle")
        || person_base_class_namesCOCO[obj.label] == std::string("bus")
        || person_base_class_namesCOCO[obj.label] == std::string("truck")
        )
        {

            ANNIWOLOG(INFO) <<
            "BasePersonDetection: detect. box:"<<obj.rect.x<<","<<
obj.rect.y<<","
            <<obj.rect.width<<","<<obj.rect.height<<","
            << "score:"<<obj.prob<<"class:"
<<person_base_class_namesCOCO[obj.label].c_str()<<" camID:"<<camID;


            if(obj.rect.width < 5 || obj.rect.height < 5 ||
(obj.rect.x+obj.rect.width) > img_w || (obj.rect.y+obj.rect.height) > img_h)
            {
                ANNIWOLOG(INFO) <<"Ignore invalid box, to check!"<<" camID:"
<<camID;
                continue;
            }

            if(person_base_class_namesCOCO[obj.label] == std::string("bus")
            || person_base_class_namesCOCO[obj.label] == std::string("truck"))
            {
                if(obj.rect.width < 80 || obj.rect.height < 80)
                {
                    ANNIWOLOG(INFO) <<"Ignore small truck"<<" camID:"<<camID;

                    continue;

                }
            }


            float x1=obj.rect.x;
            float y1 = obj.rect.y;
            float x2=(obj.rect.x+obj.rect.width) > img_w ? img_w :
(obj.rect.x+obj.rect.width) ;
            float y2 =(obj.rect.y+obj.rect.height) > img_h ? img_h:
(obj.rect.y+obj.rect.height);




            DetectBox detBoxObj;
            detBoxObj.x1 = x1;
            detBoxObj.y1= y1;
            detBoxObj.x2= x2;
            detBoxObj.y2= y2;
            detBoxObj.confidence = obj.prob;
            detBoxObj.trackID = -1;

            //todo:区分人，车。因为需要用不同的reid encoder.
            //0:为人，1为车
            if(person_base_class_namesCOCO[obj.label]  == std::string("person"))
            {
```

```
                detBoxObj.classID = 0;
            }
            else
            {
                detBoxObj.classID = 1;
            }

            uuid_generate_random(detBoxObj.uuid);
            track_results.push_back(detBoxObj);


            obj.trackID=-1;

            strncpy((char*)obj.uuid, (char*)detBoxObj.uuid, sizeof(uuid_t));


            jsonObjCnt++;

        }

    }




/////////////TRACKING PART//////////////////////////////////

    if(track_results.size() > 0)
    {
        //如下使用cpu
        if(DS)
        {
            cv::Mat img_rgb;
            cv::cvtColor(image, img_rgb, cv::COLOR_BGR2RGB);

            DS->sort(img_rgb, track_results,camID,instanceID);
        }else
        {
           ANNIWOLOG(INFO) <<"BasePersonDetection:FATAL ERROR! DS is null!"<<","
<<" camID:"<<camID;
        }

    }

    //匹配track与det框。
    for (auto & personvehicel_det : objects) {
        int trackID = -1;

        if(person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("person")
        || person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("bicycle")
        || person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("car")
        || person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("motorcycle")
        || person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("bus")
```

```
        || person_base_class_namesCOCO[personvehicel_det.label] ==
std::string("truck")
        )
        {
            bool isInTrackResults=false;
            ANNIWOLOG(INFO) <<"BasePersonDetection:track_results.size:"
<<track_results.size()<<","<<" camID:"<<camID;

            for (auto & box : track_results ) {
                // if(box.uuid == personvehicel_det.uuid)
                if(0 == strncmp( (char *)box.uuid,  (char
*)personvehicel_det.uuid, sizeof(uuid_t)))
                {
                    trackID = (int)box.trackID;
                    personvehicel_det.trackID = trackID;
                    isInTrackResults=true;

                }
            }

            if(!isInTrackResults)
            {
                ANNIWOLOG(INFO) <<"BasePersonDetection: NOT IN track Resuslts:"
<<person_base_class_namesCOCO[personvehicel_det.label].c_str()<<","<<
personvehicel_det.prob<<","<< "tid:"<<trackID<<  "x, y, w,  h:"
<<personvehicel_det.rect.x<<","<<personvehicel_det.rect.y<<","
<<personvehicel_det.rect.width<<","<<personvehicel_det.rect.height<<","<<"
camID:"<<camID;
            }else
            {
                ANNIWOLOG(INFO) <<"BasePersonDetection:"
<<person_base_class_namesCOCO[personvehicel_det.label].c_str()<<","<<
personvehicel_det.prob<<","<< "tid:"<<trackID<<  "x, y, w,  h:"
<<personvehicel_det.rect.x<<","<<personvehicel_det.rect.y<<","
<<personvehicel_det.rect.width<<","<<personvehicel_det.rect.height<<","<<"
camID:"<<camID;

            }

        }

    }



    return jsonObjCnt;

}



//person_base_functionsIn:所有需要先检测人、车的其他功能列表
//初始化
void BasePersonDetection::initTracks(const ANNIWO_JSON_CONF_CLASS&
globalJsonConfObj,const std::unordered_set<std::string>&
 person_base_functionsIn)
{
```

```
//ANNIWO_JSON_CONF_CLASS  JSON配置文件
    std::unordered_set<int> setcamIDs ;

    executionContexts.clear();
    contextlocks.clear();

    cudaSetDevice(gpuNum);

    for (auto iter = globalJsonConfObj.id_func_cap.begin(); iter !=
globalJsonConfObj.id_func_cap.end(); ++iter) {
        //ID
        int camID= iter->first ;
        //遍历哈希表中的功能
        for(auto& f : iter->second)
        {//对于每个摄像头ID，遍历其对应的功能列表。
            std::unordered_set<std::string>::const_iterator
got_person_base_functions = person_base_functionsIn.find(f);

            if (got_person_base_functions == person_base_functionsIn.end())
            {
                continue;
            }
            else
            {//设置
                setcamIDs.insert(camID);
                break;
            }
        }
    }
    if(DS)//释放
    {
        delete DS;
    }

    std::vector<int> camIDs;

    int cntID=0;
    //只生成ANNIWO_NUM_INSTANCE_PERSONBASE个实例
    while(cntID < globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE)
    {
        ANNIWOLOG(INFO) << "BasePersonDetection::initTracks: insert instance"
<<"cntID:"<<cntID<<" ";

        std::vector<float> input_data(batch_size * CHANNELS * INPUT_H *
INPUT_W,0);
        std::pair<int, std::vector<float> >
itempair2(cntID,std::move(input_data));
        m_input_datas.insert( std::move(itempair2) );

        cntID++;
    }



    for (auto& camID : setcamIDs)
    {
        camIDs.push_back(camID);//添加
    }
```

```
    for(int i=0;i<globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE;i++)
    {
        //推理方面初始化
        TrtSampleUniquePtr<nvinfer1::IExecutionContext>  context4thisCam(engine->createExecutionContext());
        std::pair<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext> > tmpitem{i,std::move(context4thisCam)};

        executionContexts.insert(std::move(tmpitem));

          //创建一个互斥锁
        std::unique_ptr<std::mutex>    newmutexptr(new std::mutex);
        std::pair<int, std::unique_ptr<std::mutex> > tmplockitem{i,std::move(newmutexptr)};

        contextlocks.insert(std::move(tmplockitem));
    }

    //人和车分开用reid,例如加油站情况。
    //类型为加油站
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
//模型地址，第二个模型地址，Reid参数，线程数
        DS = new DeepSort(deepsort_model_file_path_model,&deepsort_model_file_path_model2, reidbatchsize, camIDs,globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE);

    }else
    {

        //人和车混用一个reid,车的效果差。
        DS = new DeepSort(deepsort_model_file_path_model,NULL, reidbatchsize, camIDs,globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE);
    }

}

//跟踪检测
int BasePersonDetection::detect(  int camID, int instanceID, cv::Mat& img, /*out*/std::vector<Object>& objects )
{
    int objCnt = 0;
    cudaSetDevice(gpuNum);//设置GPU设备


    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_LIANGKU)//判断是否为粮库地区
    {


        int img_w = img.cols;//宽
        int img_h = img.rows;//高

        int choiceIntVal = randIntWithinScale(globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE);//选择personbase_num
        std::unordered_map<int, std::unique_ptr<std::mutex> >::iterator iterCamInstancelock =  contextlocks.find(choiceIntVal);//找到对应的上下文锁
```

```
        std::unordered_map<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext>
>::iterator iterCamInstance =  executionContexts.find(choiceIntVal);//找到执行上下
文

        if (iterCamInstance != executionContexts.end())
        {//找到执行上下文就进行检测
//输入数据，摄像头id,实例id,图片，跑的时间，引擎，TrtSampleUniquePtr，GPUid,锁
            yolov4_detection_staff(m_input_datas,camID,instanceID,img,
                runtime,engine,
                iterCamInstance->second,//smart pointer context for this func-
cam
                gpuNum,
                iterCamInstancelock->second,//smart pointer context LOCK for
this func-cam

 YOLO4_OUTPUT_SIZE_COCO,INPUT_W,INPUT_H,objects,BBOX_CONF_THRESHCOCO,NUM_CLASSES
COCO,
                "BasePersonDetection");

        }else
        {
            ANNIWOLOG(INFO) <<"Not found the context for camId:"<<camID;
            ANNIWOCHECK(false);
        }




        //先删除无用类别
        std::vector<Object>::iterator it=objects.begin();
        while(it != objects.end() )
        {
            bool isDel=false;
            // ANNIWOLOG(INFO)  << "BasePersonDetection debug 1:size: "
<<objects.size() <<",camID:"<<camID ;

            if(person_base_class_namesCOCO[it->label] == std::string("bus")
            || person_base_class_namesCOCO[it->label] == std::string("truck"))
            {
                if(it->rect.width < 80 || it->rect.height < 80)
                {
                    isDel=true;

                }else
                {
                    isDel=false;
                }
            }
            else  if(person_base_class_namesCOCO[it->label]  ==
std::string("person")
            || person_base_class_namesCOCO[it->label] == std::string("bicycle")
            || person_base_class_namesCOCO[it->label] == std::string("car")
            || person_base_class_namesCOCO[it->label] ==
std::string("motorcycle")
            )
            {
```

```
                if(it->rect.width < 5 || it->rect.height < 5 || (it->rect.x+it-
>rect.width) > img_w || (it->rect.y+it->rect.height) > img_h)
                {
                    ANNIWOLOG(INFO) <<"BasePersonDetection:Ignore invalid box,
to check!";
                    isDel=true;
                }else
                {
                    isDel=false;
                }

            }else
            {
                isDel=true;
            }

            if (isDel)//无效
            {
                it = objects.erase(it); //vector的erase会返回一个自动指向下一个元素，
必须赋值！
            }
            else
            {
                it++;
            }

        }

        ANNIWOLOG(INFO)  << "BasePersonDetection debug 2:size:"<<objects.size()
<<",camID:"<<camID ;


        if(objects.size() > 0)
        {
            objCnt = mainfunc(camID,instanceID, img, /*out*/objects);//跟踪
        }
        else
        {
            ANNIWOLOG(INFO) << "BasePersonDetection:no objects." <<"camID:"
<<camID;
        }
    }

    //加油站的人车,目前就用coco的结果。等新模型好了再说。
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)//加油场
    {

        //yolov4 ouput flat size
        static const int YOLO4_OUTPUT_SIZE = 1*7581*NUM_ANCHORS*
(NUM_CLASSESJYZCAR + 5);


        int choiceIntVal =
randIntWithinScale(globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE);
        std::unordered_map<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext>
>::iterator iterCamInstance =  executionContexts.find(choiceIntVal);
```

```
        std::unordered_map<int, std::unique_ptr<std::mutex> >::iterator
iterCamInstancelock =  contextlocks.find(choiceIntVal);

        if (iterCamInstance != executionContexts.end())
        {
            yolov4_detection_staff(m_input_datas,camID,instanceID,img,
                runtime,engine,
                iterCamInstance->second,//smart pointer context for this cam
                gpuNum,
                iterCamInstancelock->second,//smart pointer context LOCK for
this func-cam

 YOLO4_OUTPUT_SIZE,INPUT_W,INPUT_H,objects,BBOX_CONF_THRESHJYZCAR,NUM_CLASSESJYZ
CAR,
                "BasePersonDetection");
        }else
        {
            ANNIWOLOG(INFO) <<"Not found the context for camId:"<<camID;
            ANNIWOCHECK(false);
        }



        if(objects.size() > 0)
        {
            objCnt = mainfunc2(camID,instanceID, img, /*out*/objects);//
            if(objCnt > 0)
            {
                ANNIWOLOG(INFO) << "BasePersonDetection: objects cnt:"<<objCnt
<<"camID:"<<camID;
            }else
            {
                ANNIWOLOG(INFO) << "BasePersonDetection:no objects." <<"camID:"
<<camID;
            }
        }
        else
        {
            ANNIWOLOG(INFO) << "BasePersonDetection:no objects." <<"camID:"
<<camID;
        }
    }

    ANNIWOLOG(INFO) << "BasePersonDetection: objects size."<<objects.size()
<<"camID:"<<camID;
    ANNIWOLOG(INFO) << "BasePersonDetection:exit detect()" <<"camID:"<<camID ;

    return objCnt;
}
```

zhuangxieyou.cpp

```
#include "zhuangxieyou.hpp"
#include "zxy_absence.hpp"
#include "zxy_items.hpp"
#include "../utils/httpUtil.hpp"

//读写文件
#include "../utils/rapidjson/writer.h"
#include "../utils/rapidjson/stringbuffer.h"
#include "../utils/subUtils.hpp"

#include "../personbase/basePerson.hpp"



//配置文件
static const ANNIWO_JSON_CONF_CLASS* globalJsonConfObjPtr;




////////////
//历史报告
static std::unordered_map<int, std::unordered_map<int, int > >
reporthistoryArray ;
//停留配置信息
static const std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>
 >* stay_conf_map_ptr;
//存储有效类型
static const std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >*  validtypes_map_ptr;
//存储上一帧的目标对象
static std::unordered_map<int, std::vector<Object> > allLastObjects ; //防止低级错
误的iou过滤
//camID, isStart
static std::unordered_map<int, bool > thisTimeReportArray ; //记录本次检测是否报警
static std::unordered_map<int, bool > iSStayConfigCheckOKArray ; //记录是否有检测到
油罐车停止满足条件，满足后启动报警;无油罐车时候置为false



static ZxyItems *zxyItemsDet=nullptr;
static ZxyAbsence *absenceObjPtr=nullptr;

//临时记录结构，用于检查该trackid对应的目标停留时间是否已经达到阈值
// {camID,{trackid,stayStart}}


//记录停留时间
//struct AnniwoTrackRecord
//{
//    Object detresult; //
//    std::chrono::system_clock::time_point startPoint;  //stayStart时间
//};
```

```
static  std::unordered_map<int,std::unordered_map<int, AnniwoTrackRecord>  >
trackStayMap;


const static std::string deepsort_model_file_path_model=
{"../models/safearea/dstk_model/deepsort_sim.trt"};

const static std::string deepsort_model_file_path_model2=
{"../models/safearea/dstk_model2/carRreid_sim.trt"};

//REID模型的批处理大小
const int reidbatchsize=8;

static DeepSort* DS = nullptr;



// Default constructor
ZxyDetection::ZxyDetection () {
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        zxyItemsDet= new ZxyItems();
        absenceObjPtr=new ZxyAbsence();
    }else
    {
        ANNIWOLOG(INFO) << "ZxyDetection():Not use zhuangxieyou in other than
jiayouzhan domain!" ;
        ANNIWOCHECK(false);
    }
    ANNIWOLOG(INFO) << "ZxyDetection(): Success initialized!" ;
}



// Destructor
ZxyDetection::~ZxyDetection ()
{

    if(zxyItemsDet)
    {
        delete zxyItemsDet;
    }
    if(absenceObjPtr)
    {
        delete absenceObjPtr;
    }

}

//通过trackid报警历史，停留时间来判断新出现的需要报警的人/车数目.与类型无关。
//卸油区和卸油区离岗：检查tank_truck停留时间

static int deDuplicateTrackResults(int camID, cv::Mat& img,std::vector<Object>&
offendPersonVehicleNoTypefilter, std::vector<Object>& offend_boxes_det,int&
outDurn)
{

    int orig_img_w = img.cols;
```

```
    int orig_img_h = img.rows;
//用于记录新的人车目标数
    int newpersonvehicleCnt=0;
//记录停留时间
    AnniwoStay stayconfig;
    stayconfig.isMotionless=false;
    stayconfig.staySec=-1;

    //取得停留时间设置
    //stayconfig
    //camId,{func,<isMotionless,stay in seconds>}
    std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>
>::const_iterator got_id_func_cap = stay_conf_map_ptr->find(camID);

    if (got_id_func_cap == stay_conf_map_ptr->end())
    {
        ANNIWOLOG(INFO) << "not found in stay_conf_map,camID:" <<camID;
    }
    else
    {
        const std::unordered_map<std::string, AnniwoStay>& conf_map
=got_id_func_cap->second;
        std::unordered_map<std::string, AnniwoStay>::const_iterator
got_id_func_cap2 = conf_map.find("zhuangxieyou");
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "not found zhuangxieyou in stay_conf_map,camID:"
<<camID;
        }
        else
        {
            stayconfig = got_id_func_cap2->second ;
        }
    }


    int durn=-1;
    outDurn=-1;
    //遍历每一结果
    for (auto& obj : offend_boxes_det) {

        int trackID = (int)obj.trackID;
        //未追踪过
        if(trackID == -1)
        {
            newpersonvehicleCnt++;

            if(class_names_xieyouitems[obj.label] ==
std::string("tank_truck"))//装卸油仅油罐车检查停留设置
            {
                //停留时间:首次进入不报警，除非当无逗留时间配置的时候
                if (stayconfig.staySec <= 0)
                {
                    isStayConfigCheckOKArray[camID] = true;
                    thisTimeReportArray[camID]=true;
                }
            }
```

```
        }else//对于已经有id的对象需要看是否已经报警过。如果已经报警过则忽略。否则认为是新的并记录
        {

            std::unordered_map<int, std::unordered_map<int, int > >::iterator got_it = reporthistoryArray.find(camID);

            if (got_it == reporthistoryArray.end())
            {
                ANNIWOLOG(INFO) <<"ZxyDetection: camID Not in history map!!!" <<"camID:"<<camID<<std::endl;
            }
            else
            {
                std::unordered_map<int, int >& perCamIDhistory = got_it->second;
                std::unordered_map<int, int >::iterator got_it2 = perCamIDhistory.find(trackID);

                if (got_it2 == perCamIDhistory.end())//Not reported to this camID
                {
                    float inter_area =0.0;
                    float union_area =0.0;

                    if(class_names_xieyouitems[obj.label] == std::string("tank_truck"))//卸油区准备仅油罐车检查停留设置
                    {
                        //停留时间:在区域内trackid是新的，相当于第一次跟踪上,没有配置逗留时间;
                            if (stayconfig.staySec <= 0)
                            {
                                ANNIWOLOG(INFO) <<
                                    "ZxyDetection.detect no stayconfig,has trackID but new. camID:"<<camID;
                                 isStayConfigCheckOKArray[camID] =true;
                            }
                            else
                            {//查看停留时间
                                std::unordered_map<int,std::unordered_map<int, AnniwoTrackRecord>  >::iterator got_id_func_cap = trackStayMap.find(camID);

                                if (got_id_func_cap == trackStayMap.end())
                                {
                                    ANNIWOLOG(INFO) << "ZxyDetection.detect WARN: trackStayMap,camID:" <<camID;
                                }
                                else
                                {
                                    std::unordered_map<int, AnniwoTrackRecord>& track_stay_map =got_id_func_cap->second;
                                    std::unordered_map<int, AnniwoTrackRecord>::iterator got_id_func_cap2 = track_stay_map.find(trackID);
                                    if (got_id_func_cap2 == track_stay_map.end())
                                    {
                                        //map中未记录该track_id
                                        ANNIWOLOG(INFO) << "ZxyDetection.detect not found in track_stay_map,new add trackID:"<<trackID<<"camID:" <<camID;
```

```
                                        //todo:何时清空？？
                                        //没有找到为新，添加到当中
                                        track_stay_map.insert(std::pair<int,
AnniwoTrackRecord>(trackID,{obj,std::chrono::system_clock::now()}));
                                    }
                                    else
                                    {
                                        // got_id_func_cap2->second 是开始时间
                                        //如果开始时间是设置的…0…值,是已经报过了则不报。
                                        if(got_id_func_cap2->second.startPoint >
std::chrono::system_clock::from_time_t(0))
                                        {
                                            if(stayconfig.isMotionless)//要检测到
静止之后开始计时
                                            {
                                                //与box_det求ioa    求交并比
                                                // intersection over union
//struct AnniwoTrackRecord
//{
//    Object detresult; //上一次结果
//    std::chrono::system_clock::time_point startPoint;  //stayStart时间
//};
                                                inter_area =
intersection_area(obj, got_id_func_cap2->second.detresult);
                                                union_area = obj.rect.area() +
got_id_func_cap2->second.detresult.rect.area() - inter_area;


                                                //本次检测框与上一次IOU > 0.85 认为是
静止
                                                if(inter_area / union_area >
0.85  ) //认为是静止了，那么上次的startPoint OK.
                                                {
                                                    //计算停留时间
                                                    durn =
std::chrono::duration_cast<std::chrono::seconds>
(std::chrono::system_clock::now() - got_id_func_cap2-
>second.startPoint).count();

                                                }else
                                                {
                                                    got_id_func_cap2-
>second.startPoint=std::chrono::system_clock::now();//非静止，更新上次的startPoint.
                                                }

                                                got_id_func_cap2-
>second.detresult=obj;


                                            }else
                                            {
                                                durn =
std::chrono::duration_cast<std::chrono::seconds>
(std::chrono::system_clock::now() - got_id_func_cap2-
>second.startPoint).count();
                                            }
```

```
                                                ANNIWOLOG(INFO)
<<"ZxyDetection.detect:isMotionless:"<<stayconfig.isMotionless<<" trackID "
<<trackID<<" durn:"<<durn<<" camID:"<<camID;

                                                if(durn > stayconfig.staySec)
                                                {
                                                     isStayConfigCheckOKArray[camID]
 = true;
                                                //#对一个trackid对应的物体，如超时后
报警过一次，
                                                //deepsort逻辑:\出了跟踪区域(以miss
100次为准)后删除track
                                                got_id_func_cap2-
>second.startPoint=std::chrono::system_clock::from_time_t(0);
                                                ANNIWOLOG(INFO)
<<"ZxyDetection.detect:isStayConfig triggered:"<<isStayConfigCheckOKArray[camID]
<<" trackID "<<trackID<<" durn:"<<durn<<" camID:"<<camID;


                                                }

                                            }

                                        }
                                    }
                                }
                                //////////////
                    }
//要报警
                    if( isStayConfigCheckOKArray[camID]  )
                    {
                        if(class_names_xieyouitems[obj.label] ==
std::string("tank_truck"))
                        {

                            thisTimeReportArray[camID]=true;

                            newpersonvehicleCnt++;
                            //保存
                            perCamIDhistory.insert(std::pair<int,int>(trackID,1)
);
                        }else
                        {
                            if(thisTimeReportArray[camID]==true)
                            {
                                newpersonvehicleCnt++;
                                perCamIDhistory.insert(std::pair<int,int>
(trackID,1) );
                            }
                        }
                    }
                }
                else//有记录了
                {
                    if(class_names_xieyouitems[obj.label] ==
std::string("tank_truck"))//卸油区准备仅油罐车在的时候报警
                    {
                        thisTimeReportArray[camID]=true;
```

```
                    }
                    ANNIWOLOG(INFO) <<"ZxyDetection: found
tracked&reported..trackID:"<<trackID<<"camID:"<<camID<<"thisTimeReport:"
<<thisTimeReportArray[camID];
                }
            }

        }
    }

    if(thisTimeReportArray[camID]==false)//本次无油罐车
    {
        isStayConfigCheckOKArray[camID] = false;//因为没有检测到油罐车，开始报警开关重
置
        absenceObjPtr->stop(camID);//停止离岗计时检查
        ANNIWOLOG(INFO) <<"ZxyDetection.detect:isStayConfig untriggered:"
<<isStayConfigCheckOKArray[camID]<<" camID:"<<camID;
        return 0;
    }

//没有新的目标
    if(newpersonvehicleCnt <= 0 )
    {
        ANNIWOLOG(INFO) <<"ZxyDetection: No new person, camID:"<<camID
<<"thisTimeReport:"<<thisTimeReportArray[camID]<<"stayCheckOK:"
<<isStayConfigCheckOKArray[camID];
        return newpersonvehicleCnt;
    }
    if( ! isStayConfigCheckOKArray[camID] )
    {
        ANNIWOLOG(INFO) <<"ZxyDetection:Not report because stay config not ok,
camID:"<<camID;
        return 0;
    }else
    {
        //达到报警条件(是油罐车达到超时条件isStayConfigCheckOKArray)之后才开始进行人员离岗
判断
        absenceObjPtr->startOrcheck(camID,img,offendPersonVehicleNoTypefilter);
//检查是否有离岗,传入对象是仅仅做了valid area过滤的，未进行类型过滤
    }


    if(offend_boxes_det.size() <= 0)
    {
        ANNIWOLOG(INFO) <<"ZxyDetection:filteredObjects size 0,camID:"<<camID;
    }else
    {
        if(isResultDuplicated(allLastObjects[camID],offend_boxes_det))
        {
            ANNIWOLOG(INFO) <<"ZxyDetection:Duplicated results.Ignored.camID:"
<<camID<<std::endl;
            allLastObjects[camID]=offend_boxes_det;

            return 0;
        }
    }
    //update history results
    allLastObjects[camID]=offend_boxes_det;
```

```
    for (auto& obj : offend_boxes_det) {
        int trackID = (int)obj.trackID;
        int x = obj.rect.x;
        int y = obj.rect.y;
        int width = obj.rect.width;
        int height = obj.rect.height;


        ANNIWOLOG(INFO) << "ZxyDetection:DEBUG: obj x1:" << x << "y1:" << y <<
"width:" << width << "height:" << height << "camID:" << camID;
        ANNIWOLOG(INFO) << "ZxyDetection:DEBUG: obj orig_img_w:" << orig_img_w
<< "orig_img_h:" << orig_img_h << "camID:" << camID;




    }


    outDurn=durn;
    return newpersonvehicleCnt;
}

//objects是基于class_names_xieyouitems
//分析
void mainfuncZXY(int camID,int instanceID, cv::Mat& bgr, const
std::vector<Object>& objects, const Polygon* polygonSafeArea_ptr)
{

    //
    std::vector<Object> offendPersonVehicleObjects;
    std::vector<Object> offendPersonVehicleNoTypefilter; //不进行concern type和
exclude type过滤，进行valid area过滤的

    cv::Mat image = bgr.clone();
    Polygon _inter;
    Polygon box_poly;

    rapidjson::StringBuffer jsonstrbuf;
    rapidjson::Writer<rapidjson::StringBuffer> writer(jsonstrbuf);

    int offendnewPersonCnt=0;
    writer.StartArray();



    int img_w = image.cols;
    int img_h = image.rows;


    //todo:以后优化这一块。
    std::set<std::string> concern_classes;
    std::set<std::string> exclude_classes;
    //取得关注类型
    //valid types
    //camId,{func,Vector<String>}
```

```
    std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >::const_iterator got_id_func_cap =
validtypes_map_ptr->find(camID);


    if (got_id_func_cap == validtypes_map_ptr->end())
    {
        ANNIWOLOG(INFO) << "ZxyDetection:not set in validtypes_conf_map for
camID:" <<camID<<"use default classes";
        //use default all
        //使用默认
    }
    else
    {
        const std::unordered_map<std::string, AnniwoSafeEreaConcernTypes >&
conf_map =got_id_func_cap->second;
        std::unordered_map<std::string, AnniwoSafeEreaConcernTypes
>::const_iterator got_id_func_cap2 = conf_map.find("zhuangxieyou");
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "ZxyDetection:not set safeErea in
validtypes_conf_map for camID:" <<camID<<"use default classes";

            //use default all
        }
        else
        {
            const AnniwoSafeEreaConcernTypes& typesST = got_id_func_cap2-
>second;
            if(typesST.validtypes.size() == 0 )
            {
                ANNIWOLOG(INFO) << "ZxyDetection:empty in validtypes_conf_map
for camID:" <<camID<<"at least tank_truck should be there!";

                ANNIWOCHECK(false);

            }
            else if(typesST.validtypes.size() >= 1)
            {

                ANNIWOLOG(INFO) << "ZxyDetection:used concern_classes from
config for camID:" <<camID ;


                concern_classes.clear();
                for(auto& onetype:typesST.validtypes)
                {
                    concern_classes.insert(onetype);
                }

                exclude_classes.clear();
                for(auto& onetype:typesST.excludeTypes)
                {
                    exclude_classes.insert(onetype);
                }

            }
```

```
        }
    }

    //日志打印
    for(auto& onetype:concern_classes)
    {
       ANNIWOLOG(INFO) << "ZxyDetection:concern_classes for camID:" <<camID
<<","<< onetype;
    }

    for(auto& onetype:exclude_classes)
    {
       ANNIWOLOG(INFO) << "ZxyDetection:exclude_classes for camID:" <<camID
<<","<< onetype;
    }



////////////////////////////////////////////////////////////////////////////////
//////////////////////////
    float  inter_area=0.;//交集面积
    float  gzf_area=0.;//油罐车面积
    float  obj_area=0.;//物体区域面积
    std::vector<int> clothes_classes;


    bool isClassToReport=false;//类型判断用

    for (size_t i = 0; i < objects.size(); i++)
    {
        const Object& obj = objects[i];

        ANNIWOLOG(INFO) <<
        "ZxyDetection: detect. box:"<<obj.rect.x<<","<< obj.rect.y<<","
        <<obj.rect.width<<","<<obj.rect.height<<","
        << "score:"<<obj.prob<<"class:"
<<class_names_xieyouitems[obj.label].c_str()<<","<<"trackid:"<<obj.trackID <<"
camID:"<<camID;

        //人下部
        float x1=obj.rect.x;
        float y1 = obj.rect.y;
        float x2=(obj.rect.x+obj.rect.width) > img_w ? img_w :
(obj.rect.x+obj.rect.width) ;
        float y2 =(obj.rect.y+obj.rect.height) > img_h ? img_h:
(obj.rect.y+obj.rect.height);

        float y1_ = y1;
        float y2_ = y2;
        if(class_names_xieyouitems[obj.label] == "person"
        || class_names_xieyouitems[obj.label] == std::string("work_clothe_blue")
        || class_names_xieyouitems[obj.label] ==
std::string("work_clothe_yellow")
        )  //人员要求小腿到脚部分在设定区域内
        {
            y1_ = (obj.rect.y+obj.rect.height/4.0*3.0) > y2 ? y2:
(obj.rect.y+obj.rect.height/4.0*3.0);
        }
```

```
        if(class_names_xieyouitems[obj.label] == "tank_truck")  //油罐车要求上部2/3
在设定区域内
        {
            y2_ = (obj.rect.y+obj.rect.height/3.0*2.0) > y2 ? y2:
(obj.rect.y+obj.rect.height/3.0*2.0);
        }


        float area = 0.0;
        if(polygonSafeArea_ptr && polygonSafeArea_ptr->size() >= 3)
        {
            box_poly.clear();
            box_poly.add(cv::Point(int(x1),int(y1_)));
            box_poly.add(cv::Point(int(x2),int(y1_)));
            box_poly.add(cv::Point(int(x2),int(y2_)));
            box_poly.add(cv::Point(int(x1),int(y2_)));
            _inter.clear();
            //用于计算两个多边形的交集
            intersectPolygonSHPC(*polygonSafeArea_ptr,box_poly,_inter);
            //此处无交集时候size==0?
            if( _inter.size() ) {
                area = _inter.area();
                ANNIWOLOG(INFO) <<"ZxyDetection: Area half object intersected =
"<<area<<"camID:"<<camID;

                if(area > 768.0)
                {
                    //通过，在下面执行正常逻辑
                    ANNIWOLOG(INFO) <<
                        "ZxyDetection: detect.checkSafeArea box:"
<<obj.rect.x<<","<< obj.rect.y<<","
                        <<obj.rect.width<<","<<obj.rect.height<<","
                        << "score:"<<obj.prob<<"class:"
<<class_names_xieyouitems[obj.label].c_str()<<","<<"area:"<<area<<"trackid:"
<<obj.trackID <<" camID:"<<camID;

                }else
                {
                    ANNIWOLOG(INFO) <<
                        "ZxyDetection: detect.Ignore checkSafeArea box:"
<<obj.rect.x<<","<< obj.rect.y<<","
                        <<obj.rect.width<<","<<obj.rect.height<<","
                        << "score:"<<obj.prob<<"class:"
<<class_names_xieyouitems[obj.label].c_str()<<","<<"area:"<<area<<"trackid:"
<<obj.trackID <<" camID:"<<camID;
                    //忽略这个对象!
                    continue;
                }

            }else
            {
                ANNIWOLOG(INFO) <<"ZxyDetection: Area intersected None: "
<<area<<"camID:"<<camID;
                continue;

            }
        }else
```

```
        {
            ANNIWOLOG(INFO) <<"ZxyDetection: Area not setting. "<<area<<"camID:"
<<camID;
        }


//************************
        Object objPersonNoTypefilter;
        objPersonNoTypefilter.rect.x=obj.rect.x;
        objPersonNoTypefilter.rect.y=obj.rect.y;
        objPersonNoTypefilter.rect.width =obj.rect.width;
        objPersonNoTypefilter.rect.height=obj.rect.height;
        objPersonNoTypefilter.trackID=obj.trackID;
        objPersonNoTypefilter.prob= obj.prob;
        objPersonNoTypefilter.label = obj.label;
        strncpy((char*)objPersonNoTypefilter.uuid, (char*)obj.uuid,
sizeof(uuid_t));


        offendPersonVehicleNoTypefilter.emplace_back(objPersonNoTypefilter);
//************************




        //关注类别过滤,目前是传什么就报什么！！！
        isClassToReport=false;
        for(auto& onetype:concern_classes)
        {//检测到的物品为关注的物品
            if(onetype == class_names_xieyouitems[obj.label])
            {
                isClassToReport=true;
                break;
            }

        }

        if(!isClassToReport)
        {
            ANNIWOLOG(INFO) <<"ZxyDetection: Ignored by concern types setting."
<<"camID:"<<camID;
            continue;
        }


        clothes_classes.push_back(obj.label);

        //check exclude types
        isClassToReport=true;

        if(exclude_classes.size() > 0 )
        {
            for(int i=0; i < clothes_classes.size(); i++)
            {
                if(isClassToReport)
                {
                    for(auto& onetype:exclude_classes)
                    {
```

```
                        if( onetype  ==
 class_names_xieyouitems[clothes_classes[i]] )
                        {
                            isClassToReport=false;
                            ANNIWOLOG(INFO) <<"ZxyDetection: found in exclude
types setting:"<<onetype<<" camID:"<<camID;

                            break;
                        }

                    }
                }else
                {
                    break;
                }
            }
        }

        if(!isClassToReport)
        {
            ANNIWOLOG(INFO) <<"ZxyDetection: Ignored by exclude types setting."
<<"camID:"<<camID;
            continue;
        }

        Object objPerson;
        objPerson.rect.x=obj.rect.x;
        objPerson.rect.y=obj.rect.y;
        objPerson.rect.width =obj.rect.width;
        objPerson.rect.height=obj.rect.height;
        objPerson.trackID=obj.trackID;
        objPerson.prob= obj.prob;
        objPerson.label = obj.label;
        strncpy((char*)objPerson.uuid, (char*)obj.uuid, sizeof(uuid_t));


        offendPersonVehicleObjects.emplace_back(objPerson);


    }



//////////////TRACKING PART//////////////////////////////////
    int durn=-1;
    if(offendPersonVehicleObjects.size() > 0 )
    {
        ////卸油区和卸油区离岗：检查tank_truck停留时间
        offendnewPersonCnt =
deDuplicateTrackResults(camID,image,offendPersonVehicleNoTypefilter,offendPerson
VehicleObjects,/*out*/durn);
    }else
    {
        ANNIWOLOG(INFO) <<"ZxyDetection:no offendPersonVehicleObjects"<<"camID:"
<<camID<<" "<<std::endl;
        return;

    }
```

```
    //////////////TRACKING PART END//////////////////////////////////


    if(offendnewPersonCnt > 0)
    {

        ////////////////////////////////////

        ANNIWOLOG(INFO) << "ZxyDetection: zxy_items. itemsResults size:"
<<offendPersonVehicleObjects.size()  <<"camID:"<<camID<<" ";
        isClassToReport=false;
        for(auto& offendObj:offendPersonVehicleObjects)
        {
            for(auto& onetype:concern_classes)
            {
                if(onetype == class_names_xieyouitems[offendObj.label])
                {
                    isClassToReport=true;

                    float x1=offendObj.rect.x;
                    float y1 = offendObj.rect.y;
                    float x2=(offendObj.rect.x+offendObj.rect.width) > img_w ?
img_w : (offendObj.rect.x+offendObj.rect.width) ;
                    float y2 =(offendObj.rect.y+offendObj.rect.height) > img_h ?
img_h: (offendObj.rect.y+offendObj.rect.height);

//写出到文件中
                    writer.StartObject();               // Between
StartObject()/EndObject(),

                    writer.Key("y1");
                    writer.Int(y1);
                    writer.Key("x1");
                    writer.Int(x1);
                    writer.Key("y2");
                    writer.Int(y2);
                    writer.Key("x2");
                    writer.Int(x2);
                    writer.Key("classItem");                // output a key,

 writer.String(class_names_xieyouitems[offendObj.label].c_str());
                    writer.Key("staytime");                // output a key,
                    writer.Int(durn);
                    writer.EndObject();


                    ANNIWOLOG(INFO) << "ZxyDetection: ." <<"camID:"<<camID<<" ";
                    break;
                }

            }

        }
```

```
        writer.EndArray();


        std::string imagename=getRandomName();
        std::string imgPath = ANNIWO_LOG_IMAGES_PATH + "/zhuangxieyou/" +
imagename;

        std::string taskIdstr={"00000"};
        std::string submitUrl={"http://localhost:7008/safety-event-
local/socketEvent/zhuangxieyou"};

        getTaskId(globalJsonConfObjPtr,camID,"zhuangxieyou",taskIdstr);

 getEventUrl(globalJsonConfObjPtr,camID,"zhuangxieyou","/zhuangxieyou",submitUrl
);

        ANNIWOLOG(INFO) <<"ZxyDetection:save file name drawed is:"<<"camID:"
<<camID<<" "<<imgPath<<std::endl;
        pool-
>enqueue(saveImgAndPost,camID,taskIdstr,imgPath,image,std::chrono::system_clock:
:from_time_t(0),
        std::string(jsonstrbuf.GetString()),jsonstrbuf.GetLength(), submitUrl);
    }else
    {
        writer.EndArray();
        ANNIWOLOG(INFO) <<"ZxyDetection:no offendnewPersonCnt"<<"camID:"
<<camID<<" "<<std::endl;
        return;


    }
}




//和之前的相似
void ZxyDetection::initTracks(
    const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj)
{
    stay_conf_map_ptr=&globalJsonConfObj.stay_conf_map;
    validtypes_map_ptr=&globalJsonConfObj.validtypes_map;

    globalJsonConfObjPtr=&globalJsonConfObj;

    allLastObjects.clear();

    // static const std::unordered_map<int,std::unordered_map<int,
std::chrono::system_clock::time_point>  > trackStayMap;
    trackStayMap.clear();
    reporthistoryArray.clear();

    thisTimeReportArray.clear();
    iSStayConfigCheckOKArray.clear();
```

```
    if(DS)
    {
        delete DS;
    }


    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        zxyItemsDet->initTracks(globalJsonConfObj);
        absenceObjPtr->init(globalJsonConfObj);


    }else
    {
        ANNIWOLOG(INFO) << "ZxyDetection::initTracks NOT SUPPORT on non-
jiayouzhan mode";
    }

    std::unordered_set<int> setcamIDs ;
    for (auto iter = globalJsonConfObj.id_func_cap.begin(); iter !=
globalJsonConfObj.id_func_cap.end(); ++iter)
    {
        int camID= iter->first ;
        for(auto& f : iter->second)
        {
            if (f == std::string("zhuangxieyou") )
            {
                std::unordered_map<int, int > trackidhits;
                ANNIWOLOG(INFO) << "ZxyDetection::initTracks:reporthistory
insert" <<"camID:"<<camID<<" ";
                reporthistoryArray.insert(std::pair<int, std::unordered_map<int,
int >  >(camID,trackidhits) );
                thisTimeReportArray.insert(std::pair<int, bool >(camID,false) );
                iSStayConfigCheckOKArray.insert(std::pair<int, bool >
(camID,false) );


                std::unordered_map<int, AnniwoTrackRecord > emptymp;
                trackStayMap.insert(std::pair<int,std::unordered_map<int,
AnniwoTrackRecord >  >(camID,emptymp) );

                std::vector<Object> emptyObjArray;
                ANNIWOLOG(INFO) << "ZxyDetection::initTracks:allLast insert"
<<"camID:"<<camID<<" ";
                allLastObjects.insert(std::pair<int, std::vector<Object>  >
(camID,emptyObjArray) );

                setcamIDs.insert(camID);
                break;
            }
            else
            {
                continue;
            }
        }
    }

    std::vector<int> camIDs;
```

```
    /////内置一个跟踪
    for (auto& camID : setcamIDs)
    {
        camIDs.push_back(camID);
    }

    //人和车分开用reid,加油站情况。
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        DS = new
DeepSort(deepsort_model_file_path_model,&deepsort_model_file_path_model2,
reidbatchsize, camIDs,globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY);
    }


}

//判断图像的亮度
static int isLight(cv::Mat& src, int camID)
{
    cv::Mat gray;
    //转为灰度图片
    cv::cvtColor(src, gray, cv::COLOR_BGR2GRAY);
    // 将输入图像转换为灰度图像，保存在变量gray中
    float sum = 0;
    float avg = 0;
    //用于计算图像像素值与128之间的差值的总和和平均值
    cv::Scalar scalar;
    int ls[256];
    int size = gray.rows * gray.cols;
    //计算图像的总像素数量

    //遍历图像中的每个像素，获取像素的灰度值，并更新sum和ls数组
    for (int i = 0; i < 256; i++)
        ls[i] = 0;

    for (int i = 0; i < gray.rows; i++)
    {
        for (int j = 0; j < gray.cols; j++)
        {
            //scalar = cvGet2D(gray, i, j);
            scalar = gray.at<uchar>(i, j);
            sum += (scalar.val[0] - 128);
            int x = (int)scalar.val[0];
            ls[x]++;
        }
    }
    avg = sum / size;
    float total = 0;
    float mean = 0;
    //计算像素值与平均值之间的差值的绝对值乘以对应像素值的数量的总和
    for (int i = 0; i < 256; i++)
    {
        total += abs(float(i - 128) - avg) * ls[i];
    }
    mean = total / size;
    //计算平均值除以平均差值的绝对值
```

```
    float cast = abs(avg / mean);
    //- 如果cast大于1，则判断avg的值：
    //- 如果avg大于0，则表示过亮，返回1。
    //- 如果avg小于0，则表示过暗，返回-1。
    //- 如果cast小于等于1，则表示正常亮度，返回0。
    ANNIWOLOG(INFO) <<  "light cast:" << cast << "camID:"<<camID;
    if (cast > 1)
    {
        if (avg > 0)
        {
            ANNIWOLOG(INFO) << "over light:" << avg << "camID:"<<camID;
            return 1;
        }
        else {
            ANNIWOLOG(INFO) << "over dark:" << avg << "camID:"<<camID;
            return -1;
        }
    }
    else
    {
        ANNIWOLOG(INFO) << "normal:" << "camID:"<<camID;
        return 0;
    }
}

//todo:polygonSafeArea
//检测
void ZxyDetection::detect(  int camID, int instanceID, cv::Mat& img,const
Polygon* polygonSafeArea_ptr)
{
    cv::Mat image = img.clone();

    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {
        ANNIWOLOG(INFO) << "ZxyDetection: domain is jiayouzhan" <<"camID:"
<<camID<<" ";

        ////////////////////////////////////////////////////////////////opencv过滤过
白坏图
        //排除坏图
        if(isLight(image,camID) != 0)
        {
            ANNIWOLOG(INFO) << "ZxyDetection: ignored by over light/dark."
<<"camID:"<<camID<<" ";
            return;
        }

        ////////////////////////////////////////////////////////////


        std::vector<Object> itemsResults;//检测出的结果
        //物品检测
        zxyItemsDet->detect( camID,  instanceID, image, itemsResults );
        int img_w = image.cols;
        int img_h = image.rows;
```

```
        if(itemsResults.size() > 0)
        {
            thisTimeReportArray[camID]=false;  //正式判断前重置该值，这样只有当本次检测到tank_truck的时候才报警

            std::vector<DetectBox> track_results;
            for (size_t i = 0; i < itemsResults.size(); i++)
            {
                //封装结果
                Object& obj = itemsResults[i];

                float x1=obj.rect.x;
                float y1 = obj.rect.y;
                float x2=(obj.rect.x+obj.rect.width) > img_w ? img_w : (obj.rect.x+obj.rect.width) ;
                float y2 =(obj.rect.y+obj.rect.height) > img_h ? img_h: (obj.rect.y+obj.rect.height);

                DetectBox detBoxObj;
                detBoxObj.x1 = x1;
                detBoxObj.y1= y1;
                detBoxObj.x2= x2;
                detBoxObj.y2= y2;
                detBoxObj.confidence = obj.prob;

                //区分人，车。因为需要用不同的reid encoder.
                //0:为人，1为车
                //0:"person"
                if(    class_names_xieyouitems[obj.label]  == std::string("person")
                    || class_names_xieyouitems[obj.label] == std::string("work_clothe_blue")
                    || class_names_xieyouitems[obj.label] == std::string("work_clothe_yellow")
                )
                {
                    detBoxObj.classID = 0;
                }
                else
                {
                    detBoxObj.classID = 1;
                }

                uuid_generate_random(detBoxObj.uuid);
                track_results.push_back(detBoxObj);


                obj.trackID=-1;

                strncpy((char*)obj.uuid, (char*)detBoxObj.uuid, sizeof(uuid_t));

            }

            if(track_results.size() > 0)
            {//分类
                if(DS)
                {
                    cv::Mat img_rgb;
```

```
                    cv::cvtColor(image, img_rgb, cv::COLOR_BGR2RGB);

                    DS->sort(img_rgb, track_results,camID,instanceID);
                }else
                {
                    ANNIWOLOG(INFO) <<"ZxyDetection::detect FATAL ERROR! DS is
null!\n";
                }

            }

            //uuid匹配track与det框。
            for (auto & personvehicel_det : itemsResults)
            {
                int trackID = -1;

                for (auto & box : track_results ) {
                    // if(box.uuid == personvehicel_det.uuid)
                    if(0 == strncmp( (char *)box.uuid,  (char
*)personvehicel_det.uuid, sizeof(uuid_t)))
                    {
                        trackID = (int)box.trackID;
                        personvehicel_det.trackID = trackID;

                    }
                }

                ANNIWOLOG(INFO) <<"ZxyDetection:detect:"
<<class_names_xieyouitems[personvehicel_det.label].c_str()<<","<<
personvehicel_det.prob<<","<< "tid:"<<trackID<<  "  x, y, w,  h:"
<<personvehicel_det.rect.x<<","<<personvehicel_det.rect.y<<","
<<personvehicel_det.rect.width<<","<<personvehicel_det.rect.height<<","<<"
camID:"<<camID;

            }

            mainfuncZXY(camID,instanceID, image,
itemsResults,polygonSafeArea_ptr);
        }
        else
        {
            ANNIWOLOG(INFO) << "ZxyDetection:no objects" <<"camID:"<<camID<<" ";
        }


    }else
    {
        ANNIWOLOG(INFO) << "ZxyDetection:NOT SUPPORT ON non-jiayouzhan mode"
<<"camID:"<<camID<<" ";

    }



    ANNIWOLOG(INFO) << "ZxyDetection:exit detect()" <<"camID:"<<camID<<" ";

    return;
}
```

#####

zxy_items.hpp

```
extern const std::vector<std::string> class_names_xieyouitems;

class ZxyItems  {
    public:
        ZxyItems() ;
        ~ZxyItems() ;

        void initTracks(const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj);
        //todo:polygonSafeArea
        static void detect(  int camID,int instanceID, cv::Mat& img,
std::vector<Object>& objects);

    private:
        static  std::unordered_map<int, std::vector<float> >  m_input_datas;

};
```

zxy_items.cpp

```
//设置一个全局的JSON配置文件
static const ANNIWO_JSON_CONF_CLASS* globalJsonConfObjPtr;
//类型数量
static const int NUM_CLASSES = 12;
//置信度
static const float BBOX_CONF_THRESH = 0.29;
//检测中的目标类型数量
static const int NUM_CLASSES_INUSE = 12;

//yolov4 ouput flat size
static const int YOLO4_OUTPUT_SIZE = 1*7581*NUM_ANCHORS* (NUM_CLASSES + 5);


//识别类型名
const std::vector<std::string>  class_names_xieyouitems = {
"Oil_port_pipeline",
"car_pipeline",
"car_electrostatic",
"ground_electrostatic",
"triangular_pyramid",
"tank_truck",
"fire_extinguisher_35",
"fire_extinguisher_5",
"person",
```

```
"work_clothe_blue",
"work_clothe_yellow",
"Oil_start"
};

//初始化引擎
static nvinfer1::IRuntime* runtime{nullptr};
static nvinfer1::ICudaEngine* engine{nullptr};
static std::unordered_map<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext>>
executionContexts;
//锁
static std::unordered_map<int, std::unique_ptr<std::mutex> > contextlocks;


static int gpuNum = 0;


//todo
const static std::string model_file={"../models/jyz_zxy/jyz_xieyou_sim.trt"};



const int batch_size = 1;
// static std::vector<int> input_shape = {batch_size, 3, INPUT_H, INPUT_W};
//yolox pytorch
static std::vector<int> input_shape = {batch_size, INPUT_H, INPUT_W, 3};
 //yolov4 keras/tf


std::unordered_map<int, std::vector<float> >  ZxyItems::m_input_datas;




ZxyItems::ZxyItems () {

    ANNIWOLOG(INFO) << "ZxyItems(): call initInferContext!" ;
//初始化推理
    gpuNum = initInferContext(
                    model_file.c_str(),
                    &runtime,
                    &engine);

    ANNIWOLOG(INFO) << "ZxyItems(): Success initialized!" ;
}

// Destructor
ZxyItems::~ZxyItems ()
{
    // destroy the engine
    delete engine;
    delete runtime;
}
```

```
//初始化跟踪，与人物跟踪相似
void ZxyItems::initTracks(const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj)
{
    executionContexts.clear();

    contextlocks.clear();

    cudaSetDevice(gpuNum);

    globalJsonConfObjPtr=&globalJsonConfObj;

    for (auto iter = globalJsonConfObj.id_func_cap.begin(); iter !=
globalJsonConfObj.id_func_cap.end(); ++iter)
    {//寻找id对应的功能
        int camID= iter->first ;
        for(auto& f : iter->second)
        {
            if (f == std::string("zhuangxieyou"))
            {

                // std::vector<Object> emptyObjArray;
                ANNIWOLOG(INFO) << "ZxyItems::initTracks: insert" <<"camID:"
<<camID<<" ";

                break;
            }
            else
            {
                continue;
            }
        }
    }


    for(int i=0;i<globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY;i++)
    {
        //
        TrtSampleUniquePtr<nvinfer1::IExecutionContext>  context4thisCam(engine-
>createExecutionContext());
        std::pair<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext> >
tmpitem{i,std::move(context4thisCam)};

        executionContexts.insert(std::move(tmpitem));


        std::unique_ptr<std::mutex>    newmutexptr(new std::mutex);
        std::pair<int, std::unique_ptr<std::mutex> >
tmplockitem{i,std::move(newmutexptr)};

        contextlocks.insert(std::move(tmplockitem));
    }


    int cntID=0;
    //只生成ANNIWO_NUM_INSTANCE_ZXY个实例
    while(cntID < globalINICONFObj.ANNIWO_NUM_THREAD_ZXY)
    {
```

```
        ANNIWOLOG(INFO) << "ZxyItems::initTracks: insert instance" <<"cntID:"
<<cntID;

//////////////////Predictor clone to each camID

        std::vector<float> input_data(batch_size * CHANNELS * INPUT_H *
INPUT_W);
        std::pair<int, std::vector<float> >
itempair2(cntID,std::move(input_data));
        m_input_datas.insert( std::move(itempair2) );

////////////////////////

        cntID++;
    }

 }
//todo:polygonSafeArea
//进行目标的检测
void ZxyItems::detect(  int camID,int instanceID, cv::Mat& img,
std::vector<Object>& objects)
{

    cudaSetDevice(gpuNum);//设定gpunum.
    //生成一个随机整数?
    int choiceIntVal =
randIntWithinScale(globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY);
    //找到和匹配的上下文对象
    std::unordered_map<int, TrtSampleUniquePtr<nvinfer1::IExecutionContext>
>::iterator iterCamInstance =  executionContexts.find(choiceIntVal);
    //找到和匹配的上下文锁
    std::unordered_map<int, std::unique_ptr<std::mutex> >::iterator
iterCamInstancelock =  contextlocks.find(choiceIntVal);

    if (iterCamInstance != executionContexts.end())
    {
    //执行目标检测操作
        yolov4_detection_staff(m_input_datas,camID,instanceID,img,
            runtime,engine,
            iterCamInstance->second,//smart pointer context for this cam
            gpuNum,
            iterCamInstancelock->second,//smart pointer context LOCK for this
func-cam

 YOLO4_OUTPUT_SIZE,INPUT_W,INPUT_H,objects,BBOX_CONF_THRESH,NUM_CLASSES,
            "ZxyItems");
    }else
    {
        ANNIWOLOG(INFO) <<"Not found the context for camId:"<<camID;
        ANNIWOCHECK(false);
    }

    if(objects.size() > 0 && objects.size() < 100)
    {
        //找到
        //nothing
    }
    else
```

```
    {
        //失败
        ANNIWOLOG(INFO) << "ZxyItems:no objects.objects.size():" <<
objects.size() <<"camID:"<<camID <<"instanceID:"<<instanceID ;
    }


    ANNIWOLOG(INFO) << "ZxyItems:exit detect()" <<"camID:"<<camID
<<"instanceID:"<<instanceID ;


    return;
}
```

zxy_absence.hpp

```
#include <opencv2/opencv.hpp>
#include "../utils/utils_intersection.hpp"
#include "../common/yolo/yolo_common.hpp"

//用于处理检测离岗的功能
class ZxyAbsence  {
    public:
        ZxyAbsence() ;
        ~ZxyAbsence() ;
//初始化
        void init(const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj);
//开始或者检查离岗
        static void startOrcheck(int camID, cv::Mat& bgr, std::vector<Object>&
offend_boxes_det);
        static void stop(int camID);
};
```

zxy_absence.cpp

```
static const ANNIWO_JSON_CONF_CLASS* globalJsonConfObjPtr;

////////////
//配置信息
static const std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>
 >* stay_conf_map_ptr;
static const std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >*  validtypes_map_ptr;


struct AnniwoTrackRecordZxyAbsence
{
    Object detresult; //
    std::chrono::system_clock::time_point startPoint;  //stayStart时间
```

```
    cv::Mat img;
};


static  std::unordered_map<int,std::unordered_map<int,
AnniwoTrackRecordZxyAbsence>  > trackStayMap;


// Default constructor
ZxyAbsence::ZxyAbsence () {
//初始化
    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {

           struct stat st = {0};
           std::string onlineSavePath={"/var/anniwo/images//absence/"};
            //创建目录
            if (stat(onlineSavePath.c_str(), &st) == -1) {
                mkdir(onlineSavePath.c_str(), 0700);//创建一个目录
            }
            //检查是否创建成功
            if (stat(onlineSavePath.c_str(), &st) == -1) {
                ANNIWOLOG(INFO) <<"Unable to create:"<<onlineSavePath;
                ANNIWOCHECK(false);
                exit(-1);
            }


    }else
    {
        ANNIWOLOG(INFO) << "ZxyAbsence():Not use zhuangxieyou in other than
jiayouzhan domain!" ;
        ANNIWOCHECK(false);
    }


    ANNIWOLOG(INFO) << "ZxyAbsence(): Success initialized!" ;

}



// Destructor
ZxyAbsence::~ZxyAbsence ()
{

}


//objects是基于class_names_xieyouitems
//开始检测是否有离岗,此函数自行检测是否有离岗标志
//offend_boxes_det已经在zhuangxieyou主类里边通过了validArea过滤过了
void ZxyAbsence::startOrcheck(int camID, cv::Mat& bgr, std::vector<Object>&
offend_boxes_det)
{

    std::vector<Object> offendPersonVehicleObjects; //装关注类别的，比如蓝色工作服人员
```

```
    Polygon _inter;//处理多边形相关
    Polygon box_poly;

    rapidjson::StringBuffer jsonstrbuf;//生成JSON对象
    rapidjson::Writer<rapidjson::StringBuffer> writer(jsonstrbuf);//生成JSON


    writer.StartArray();    //数组

    int img_w = bgr.cols;//图片宽
    int img_h = bgr.rows;//高


    //todo:以后优化这一块。
    std::set<std::string> concern_classes;//存储关注类型和排除类型
    std::set<std::string> exclude_classes;
    //取得关注类型
    //valid types
    //camId,{func,Vector<String>}
    std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >::const_iterator got_id_func_cap =
validtypes_map_ptr->find(camID);//寻找匹配ID的记录


    if (got_id_func_cap == validtypes_map_ptr->end())
    {
        ANNIWOLOG(INFO) << "ZxyAbsence:not set in validtypes_conf_map for
camID:" <<camID<<"use default classes";
        //use default all
        concern_classes.insert("work_clothe_blue");//使用默认
        concern_classes.insert("work_clothe_yellow");
    }
    else
    {
        //找到关注类型和排除类型
        const std::unordered_map<std::string, AnniwoSafeEreaConcernTypes >&
conf_map =got_id_func_cap->second;
        std::unordered_map<std::string, AnniwoSafeEreaConcernTypes
>::const_iterator got_id_func_cap2 = conf_map.find("absence");//找到对应的类型
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "ZxyAbsence:not set safeErea in
validtypes_conf_map for camID:" <<camID<<"use default classes";

            //use default all
        }
        else
        {
            const AnniwoSafeEreaConcernTypes& typesST = got_id_func_cap2-
>second;
            if(typesST.validtypes.size() == 0 )
            {
                ANNIWOLOG(INFO) << "ZxyAbsence:empty in validtypes_conf_map for
camID:" <<camID<<"at least tank_truck should be there!";

            }
            else if(typesST.validtypes.size() >= 1)
```

```
            {

                ANNIWOLOG(INFO) << "ZxyAbsence:used concern_classes from config
for camID:" <<camID ;


                concern_classes.clear();
                for(auto& onetype:typesST.validtypes)
                {
                    concern_classes.insert(onetype);//加入关注类型
                }

                exclude_classes.clear();
                for(auto& onetype:typesST.excludeTypes)
                {
                    exclude_classes.insert(onetype);//加入排除
                }

            }

        }
    }



    //日志打印
    for(auto& onetype:concern_classes)
    {
       ANNIWOLOG(INFO) << "ZxyAbsence:concern_classes for camID:" <<camID <<","
<< onetype;
    }

    //absence exlude class不需要考虑?
    for(auto& onetype:exclude_classes)
    {
       ANNIWOLOG(INFO) << "ZxyAbsence:exclude_classes for camID:" <<camID <<","
<< onetype;
    }

////////////////////////////////////////////////////////////////////////////////
////


    bool foundConcernObj=false;//是否有关注类型存在
    bool isOilStartExsists=false;//加油牌是否存在
//查找里边有没有关注类型和牌子
    for (size_t i = 0; i < offend_boxes_det.size(); i++)
    {
        const Object& obj = offend_boxes_det[i];


        ANNIWOLOG(INFO) <<
        "ZxyAbsence: detect. box:"<<obj.rect.x<<","<< obj.rect.y<<","
        <<obj.rect.width<<","<<obj.rect.height<<","
        << "score:"<<obj.prob<<"class:"
<<class_names_xieyouitems[obj.label].c_str()<<","<<"trackid:"<<obj.trackID <<"
camID:"<<camID;
```

```
        if( class_names_xieyouitems[obj.label] == std::string("Oil_start") )
        {
            isOilStartExsists = true;
        }

        foundConcernObj=false;
        //关注类别：即关注的离岗人员类别，这些类别没有的时候才报警
        for(auto& onetype:concern_classes)
        {
            if(onetype == class_names_xieyouitems[obj.label])
            {
                foundConcernObj=true;
                break;
            }

        }

        if(!foundConcernObj) //本obj不是关注类型，忽略。
        {
            continue;
        }

        Object objPerson;
        objPerson.rect.x=obj.rect.x;
        objPerson.rect.y=obj.rect.y;
        objPerson.rect.width =obj.rect.width;
        objPerson.rect.height=obj.rect.height;
        objPerson.trackID=obj.trackID;
        objPerson.prob= obj.prob;
        objPerson.label = obj.label;
        strncpy((char*)objPerson.uuid, (char*)obj.uuid, sizeof(uuid_t));


        offendPersonVehicleObjects.emplace_back(objPerson);//里边放关注类型


    }


    //关注类别，这些类别没有的时候才报警
    bool isThisTimeReport=false;

    std::chrono::system_clock::time_point absenceStartTP;

    cv::Mat img_s;

    if(offendPersonVehicleObjects.size() > 0 )
    {
        ANNIWOLOG(INFO) <<"ZxyAbsence:concern Person Objects cnt:"
<<offendPersonVehicleObjects.size()<<"camID:"<<camID;
        return;
    }else if(isOilStartExsists) //有牌子并且没有关注的人
    {
        ANNIWOLOG(INFO) <<"ZxyAbsence:no concern Person Objects.Absence check"
<<"camID:"<<camID<<" ";

        AnniwoStay stayconfig;
        stayconfig.isMotionless=false;
```

```
        stayconfig.staySec=-1;
        //取得离岗时间设置
        //stayconfig
        //camId,{func,<isMotionless,stay in seconds>}
        std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>
>::const_iterator got_id_func_cap = stay_conf_map_ptr->find(camID);

        if (got_id_func_cap == stay_conf_map_ptr->end())
        {
            ANNIWOLOG(INFO) << "not found in stay_conf_map,camID:" <<camID;
        }
        else
        {
            const std::unordered_map<std::string, AnniwoStay>& conf_map
=got_id_func_cap->second;
            std::unordered_map<std::string, AnniwoStay>::const_iterator
got_id_func_cap2 = conf_map.find("absence");
            if (got_id_func_cap2 == conf_map.end())
            {
                ANNIWOLOG(INFO) << "not found absence in stay_conf_map,camID:"
<<camID;
            }
            else
            {
                stayconfig = got_id_func_cap2->second ;
            }
        }

        if (stayconfig.staySec <= 0) //停留时间:当无离岗时间配置的时候,只要无人就报警
        {
            isThisTimeReport=true;
        }else//检查离岗时间
        {
            std::unordered_map<int,std::unordered_map<int,
AnniwoTrackRecordZxyAbsence>  >::iterator got_id_func_cap =
trackStayMap.find(camID);

            if (got_id_func_cap == trackStayMap.end())
            {
                ANNIWOLOG(INFO) << "ZxyAbsence.startOrcheck WARN:
trackStayMap,camID:" <<camID;
            }
            else
            {
                std::unordered_map<int, AnniwoTrackRecordZxyAbsence>&
track_stay_map =got_id_func_cap->second;
                //整图计时，虚拟一个tranckid:0
                int virtualtrackID=0;
                Object virtualobj; //虚拟一个空obj
                std::unordered_map<int, AnniwoTrackRecordZxyAbsence>::iterator
got_id_func_cap2 = track_stay_map.find(virtualtrackID);
                if (got_id_func_cap2 == track_stay_map.end())
                {
                    //anniwoTrackRecordZxyAbsenc
                    //Object detresult; //
                    // std::chrono::system_clock::time_point startPoint;
//stayStart时间
                    // cv::Mat img;
```

```
                    //
                    //???
                    //map中未记录该track_id
                    ANNIWOLOG(INFO) << "ZxyAbsence.startOrcheck not found in
track_stay_map,new add trackID:"<<virtualtrackID<<"camID:" <<camID;
                    cv::Mat tmpMat =   bgr.clone();
                    AnniwoTrackRecordZxyAbsence
tmpRecord{virtualobj,std::chrono::system_clock::now(),tmpMat};
                    ANNIWOCHECK(tmpRecord.img.data != NULL);

                    //pair会调用forward,forward会选择使用移动构造或者拷贝构造
                    std::pair<int, AnniwoTrackRecordZxyAbsence>
tmpPair(virtualtrackID,std::move(tmpRecord));


                    track_stay_map.insert( std::move(tmpPair) );
                    ANNIWOLOG(INFO) << "DEBUG ZxyAbsence.startOrcheck not found
in track_stay_map,added success"<<virtualtrackID<<"camID:" <<camID;

                }
                else if(got_id_func_cap2-
>second.startPoint==std::chrono::system_clock::from_time_t(0))//报过一次，重新计时
                {
                    got_id_func_cap2-
>second.startPoint=std::chrono::system_clock::now();
                    got_id_func_cap2->second.img=bgr.clone();
                }
                else
                {
                    int durn=-1;
                    // got_id_func_cap2->second 是开始时间
                    //如果开始时间是设置的…0…值,是已经报过了则不报。
                    if(got_id_func_cap2->second.startPoint >
std::chrono::system_clock::from_time_t(0))
                    {
                        if(stayconfig.isMotionless)//要检测到静止之后开始计时
                        {
                            //不存在
                        }else
                        {
                            durn =
std::chrono::duration_cast<std::chrono::seconds>
(std::chrono::system_clock::now() - got_id_func_cap2-
>second.startPoint).count();
                        }

                        ANNIWOLOG(INFO)
<<"ZxyAbsence.startOrcheck:isMotionless:"<<stayconfig.isMotionless<<" trackID "
<<virtualtrackID<<" durn:"<<durn<<" camID:"<<camID;

                        if(durn > stayconfig.staySec)
                        {
                            isThisTimeReport=true;
                            absenceStartTP=got_id_func_cap2->second.startPoint;
                            img_s = got_id_func_cap2->second.img;
                            //报过一次之后，将其更新为0
```

```
                            got_id_func_cap2-
>second.startPoint=std::chrono::system_clock::from_time_t(0);
                            ANNIWOLOG(INFO)
<<"ZxyAbsence.startOrcheck:isThisTimeReport triggered:"<<isThisTimeReport<<"
virtualtrackID "<<virtualtrackID<<" durn:"<<durn<<" camID:"<<camID;

                        }

                    }

                }
            }

        }

    }else
    {
        //既无关注物，也无牌子
    }


//已经报过警了，写进JSON中
    if(isThisTimeReport == true)
    {

        float x1=10;
        float y1 = 10;
        float x2=img_w;
        float y2 =img_h;


        writer.StartObject();               // Between
StartObject()/EndObject(),

        writer.Key("y1");
        writer.Int(y1);
        writer.Key("x1");
        writer.Int(x1);
        writer.Key("y2");
        writer.Int(y2);
        writer.Key("x2");
        writer.Int(x2);
        writer.Key("classItem");                // output a key,
        writer.String("absence");
        writer.EndObject();


        writer.EndArray();


        std::string imagename=getRandomName();
        std::string imgPath = ANNIWO_LOG_IMAGES_PATH + "/absence/" + imagename;

        ANNIWOCHECK(img_s.data != NULL);
        std::string imagenameS=imagename + std::string("_s");
        std::string imgPathS = ANNIWO_LOG_IMAGES_PATH + "/absence/" +
imagenameS;
```

```
        ANNIWOLOG(INFO) <<"ZxyAbsence:save _S file name drawed is:"<<"camID:"
<<camID<<" "<<imgPathS;

        cv::imwrite(imgPathS,img_s);


        std::string taskIdstr={"00000"};
        std::string submitUrl={"http://localhost:7008/safety-event-
local/socketEvent/absence"};

        getTaskId(globalJsonConfObjPtr,camID,"absence",taskIdstr);
        getEventUrl(globalJsonConfObjPtr,camID,"absence","/absence",submitUrl);

        ANNIWOLOG(INFO) <<"ZxyAbsence:save file name drawed is:"<<"camID:"
<<camID<<" "<<imgPath;
        cv::Mat image = bgr.clone();
        pool-
>enqueue(saveImgAndPost,camID,taskIdstr,imgPath,image,absenceStartTP,
        std::string(jsonstrbuf.GetString()),jsonstrbuf.GetLength(), submitUrl);
    }else
    {
        writer.EndArray();
        ANNIWOLOG(INFO) <<"ZxyAbsence:PersonCnt"
<<offendPersonVehicleObjects.size()<<",camID:"<<camID<<" ";
        return;
    }
}


//卸油区没有检测到油罐车，停止检测是否有离岗
void ZxyAbsence::stop(int camID)
{
    std::unordered_map<int,std::unordered_map<int, AnniwoTrackRecordZxyAbsence>
 >::iterator got_id_func_cap = trackStayMap.find(camID);//查找有和id的记录。//

    if (got_id_func_cap == trackStayMap.end())
    {//未找到
        ANNIWOLOG(INFO) << "ZxyAbsence.stop WARN: trackStayMap,camID:" <<camID;
    }
    else
    {
        std::unordered_map<int, AnniwoTrackRecordZxyAbsence>& track_stay_map
=got_id_func_cap->second;//开始时间
        //整图计时，虚拟一个trackid:0
        //???
        int virtualtrackID=0;
        Object virtualobj; //虚拟一个空obj
        std::unordered_map<int, AnniwoTrackRecordZxyAbsence>::iterator
got_id_func_cap2 = track_stay_map.find(virtualtrackID);
        if (got_id_func_cap2 == track_stay_map.end())
        {
            //map中未记录该track_id,无需重置
        }
        else
        {
            ANNIWOLOG(INFO) << "ZxyAbsence.stop DEBUG: trackStayMap, erase
camID:" <<camID;
            track_stay_map.erase(virtualtrackID);//删除
```

```
        }
    }

}



void ZxyAbsence::init(
    const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj)
{
    //存储配置
    stay_conf_map_ptr=&globalJsonConfObj.stay_conf_map;
    validtypes_map_ptr=&globalJsonConfObj.validtypes_map;

    globalJsonConfObjPtr=&globalJsonConfObj;


    // static const std::unordered_map<int,std::unordered_map<int,
std::chrono::system_clock::time_point>  > trackStayMap;
    trackStayMap.clear();

    if(globalINICONFObj.domain_config == ANNIWO_DOMANI_JIAYOUZHAN)
    {

    }else
    {
        ANNIWOLOG(INFO) << "ZxyAbsence::initTracks NOT SUPPORT on non-jiayouzhan
mode";
    }

    std::unordered_set<int> setcamIDs ;
    for (auto iter = globalJsonConfObj.id_func_cap.begin(); iter !=
globalJsonConfObj.id_func_cap.end(); ++iter)
    {//查看id对应功能
        int camID= iter->first ;
        for(auto& f : iter->second)
        {
            if (f == std::string("absence") )
            {
                //check absenceStartConition_conf_map
                std::unordered_map<int,std::unordered_map<std::string,
std::string>  >::const_iterator got_id_func_cap =
globalJsonConfObj.absenceStartConition_conf_map.find(camID);
//存储不同功能对应缺席功能的启动条件。
                if (got_id_func_cap ==
globalJsonConfObj.absenceStartConition_conf_map.end())
                {
                 //ignore non-zxy absence
                }else
                {
                    ANNIWOLOG(INFO) << "ZxyAbsence::initTracks:trackStayMap
insert" <<"camID:"<<camID<<" ";

                    std::unordered_map<int, AnniwoTrackRecordZxyAbsence >
emptymp;
                    trackStayMap.insert(std::pair<int,std::unordered_map<int,
AnniwoTrackRecordZxyAbsence >  >(camID,emptymp) );
                }
```

```
                break;
            }
            else
            {
                continue;
            }
        }
    }

}
```

app.cpp(未)

```
#include "crow_all.h"


#include "core/personbase/basePerson.hpp"
#include "core/safeErea/safeArea.hpp"
#include "core/window/window.hpp"
#include "core/smoke/SmokePhone.hpp"
#include "core/firesmog/firesmog.hpp"
#include "core/helmet/helmet.hpp"
#include "core/xunjian/xunjian.hpp"
#include "core/uptruck/uptruck.hpp"
#include "core/zhuangxieyou/zhuangxieyou.hpp"

#ifdef __aarch64__
#else
    #include "core/convery/convery.hpp"
#endif

#include "core/chepai/chepai.hpp"
#include "core/mask/Mask.hpp"


#include "core/common/yolo/yolo_common.hpp"


#include "framing/Rectangle.h"
#include "core/utils/utils_intersection.hpp"

#include "core/utils/utils_intersection.hpp"
#include "core/utils/rapidjson/document.h"
#include "core/utils/rapidjson/writer.h"
#include "core/utils/rapidjson/stringbuffer.h"
#include "core/utils/cycrptUtil.hpp"
#include "core/utils/subUtils.hpp"
#include "core/utils/ini.h"



#include "ThreadPool.h"
```

```
#include <iostream>
#include <thread>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <string>
#include <sstream>

#include <unordered_set>
#include <unordered_map>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>


const std::string VERSION=std::string(__DATE__)+std::string("
")+std::string(__TIME__);

    bool isVideoEnd5=false;
    bool isVideoEnd23=false;
    bool isVideoEnd64=false;

std::mutex mtx4GlobalVariables;
std::mutex mtx4SubmitThread; //Notice:submitThread只能单线程执行，多线程需要上锁。


bool istodoInitTracks=false; //engineStart时候需要重新初始化
bool isFirstimeInitFramer=true;//首次初始化framer
bool tidayUpNow=false;//凌晨整理时间到了,看现在是否做重清
bool tidiedOnce=false; //控制不重复做凌晨整理

std::vector<std::string> rtmpStrs;
std::vector<int> rtmpIds;



safeEreaDetection* safeEreaDetect = NULL;
WindowDetection* windowDetect = NULL;
FireDetection* fireDetect = NULL;
HelmetDetection* helmetDetect = NULL;

SmokePhoneDetection* smokePhoneDetect = NULL;

#ifdef __aarch64__
void* converyDetect = NULL;
#else
ConveryDetection* converyDetect = NULL;
#endif

BasePersonDetection* basePersonDetect = NULL;

XunjianDetection* xunjianDetect = NULL;
ZxyDetection* zxyDetect = NULL;
ChepaiDetection* chepaiDetect = NULL;
MaskDetection* maskDetect = NULL;
UptruckDetection* uptruckDetect= NULL;
```

```
static std::unordered_set<std::string> person_base_functions={"safeErea"
,"helmet" ,"phone" ,"smoke" , "convery", "chepai","mask","uptruck"};
const static std::unordered_set<std::string> all_conf_functions = {
"fire","smoke","helmet","safeErea","window","convery","uptruck","phone",
"xunjian",

 "absence","zhuangxieyou","face","chepai","mask"};



ThreadPool *pool=nullptr;
ThreadPool *poolpersonbase=nullptr;


//map(map)不能直接初始化
std::unordered_map<std::string,std::unordered_map<int, std::future<void>>  >
tasks;

std::unordered_map<int, std::future<void>>  pGroupTasks;


//////////////////////json 配置信
息////////////////////////////////////////////////////

static ANNIWO_JSON_CONF_CLASS globalJsonConfObj;
static ANNIWO_JSON_CONF_CLASS globalJsonConfObjNotApplied;

//////////ini 配置信息
struct ANNIWO_INI_CONF_CLASS globalINICONFObj;

static std::string currentInJsonAll;

//////////////////////配置信息结
束////////////////////////////////////////////////////////




std::thread* threadDetect=NULL;

static const int tidyuptimeS= 1438; //23:58
static const int tidyuptimeE= 1440; //24:00
static int log_suffix_cnt=1;

class ExampleLogHandler : public crow::ILogHandler {
    public:
        void log(std::string /*message*/, crow::LogLevel /*level*/) override {
//            cerr << "ExampleLogHandler -> " << message;
        }
};

struct ExampleMiddleware
```

```
{
    std::string message;

    ExampleMiddleware()
    {
        message = "foo";
    }

    void setMessage(std::string newMsg)
    {
        message = newMsg;
    }

    struct context
    {
    };

    void before_handle(crow::request& /*req*/, crow::response& /*res*/, context&
/*ctx*/)
    {
        CROW_LOG_DEBUG << " - MESSAGE: " << message;
    }

    void after_handle(crow::request& /*req*/, crow::response& /*res*/, context&
/*ctx*/)
    {
        // no-op

    }
};




class PERSONBASE_DET {
public:
        PERSONBASE_DET(const std::unordered_map<int, std::vector<std::string> >&
id_func_mapIn, const std::unordered_set<std::string>&  person_base_functionsIn)

:id_func_map(id_func_mapIn),mPerson_base_functions(person_base_functionsIn)
        {

            for (auto iter = globalJsonConfObj.id_func_cap.begin(); iter !=
globalJsonConfObj.id_func_cap.end(); ++iter) {
                int camId= iter->first ;
                for(auto& f : iter->second)
                {
                    std::unordered_set<std::string>::const_iterator
got_person_base_functions = this->mPerson_base_functions.find(f);

                    if (got_person_base_functions == this-
>mPerson_base_functions.end())
                    {
                        continue;
                    }
                    else
                    {
                        objCntArray.insert(std::pair<int, int >(camId,-1) );
```

```
                        std::vector<Object> results;
                        det_results.insert(std::pair<int, std::vector<Object> >
(camId,results)) ;

                        break;
                    }
                }
            }
        }

        //多线程调用
        void do_detect(int camId, int instanceID, cv::Mat& img)  {
            if(objCntArray[camId] == -1)
            {
                det_results[camId].clear();
                objCntArray[camId] = 0;
                //针对一个摄像头进行预检测，检测人.检测结果送入各个相关功能后续检测。
                //do person detection
                ANNIWOLOG(INFO) <<"PERSONBASE_DET:do person base
detection.camId: "<<camId ;
                objCntArray[camId]  = basePersonDetect->detect(
camId,instanceID,img,this->det_results[camId] );
                ANNIWOLOG(INFO) <<"PERSONBASE_DET:do person base detection
exited.camId: "<<camId ;

            }
    }

    bool hasperson(int camId) {return objCntArray[camId] >= 0; }
    void clearResults(int camId)
    {
        std::unordered_map<int, std::vector<Object> >::iterator got_det = this-
>det_results.find(camId);

        if (got_det == this->det_results.end())
        {
            return;
        }
        else
        {
            got_det->second.clear();
            std::unordered_map<int, int>::iterator got_ObjCnt = this-
>objCntArray.find(camId);
            if (got_ObjCnt == this->objCntArray.end())
            {
                ANNIWOLOG(INFO) <<"PERSONBASE_DET:FATAL!!!! NOT IN objCntArray "
<<camId;
                return;
            }else
            {
                got_ObjCnt->second=-1;
            }

        }

    }

    const std::unordered_set<std::string>&  mPerson_base_functions;
```

```
    const std::unordered_map<int, std::vector<std::string> >& id_func_map;

    std::unordered_map<int, std::vector<Object> > det_results ;
    std::unordered_map<int, int> objCntArray ;

};
static PERSONBASE_DET* personbase_detObjPtr=nullptr;


inline void resetIntervalRecord(const std::string& f, int camId)
{
        //时间间隔记录
    //camId,{func,interval}
    std::unordered_map<int,std::unordered_map<std::string, AnniwoTimeLog >
 >::iterator got_intervalIter =
 globalJsonConfObj.interval_conf_map.find(camId);
    if (got_intervalIter == globalJsonConfObj.interval_conf_map.end())
    {
        ANNIWOLOG(INFO) << "resetIntervalRecord:not found in
interval_conf_map,camId:" <<camId;
    }
    else
    {
        std::unordered_map<std::string, AnniwoTimeLog >& intervalFuncMap  =
got_intervalIter->second;
        std::unordered_map<std::string, AnniwoTimeLog >::iterator
gotIntervalFuncMapIter = intervalFuncMap.find(f);
        if (gotIntervalFuncMapIter == intervalFuncMap.end())
        {
            ANNIWOLOG(INFO) << "resetIntervalRecord:not found in
intervalFuncMap,camId:" <<camId<<" func:"<<f;
        }
        else
        {
            AnniwoTimeLog& vecValues =  gotIntervalFuncMapIter->second;
            const double confInterval = vecValues.configInterval;
            auto& prevTP = vecValues.lastTP;


            prevTP=std::chrono::steady_clock::time_point{}; //设置为最小时间，下次必
然到期保证被提交
            ANNIWOLOG(INFO) << "resetIntervalRecord - camId:" <<camId<<" func:"
<<f;

        }
    }
}

inline bool isTimeIntervalOK(const std::string& f, int camId,bool bRecord=true)
{
    //进行时间段判断
    //valid peroids_hour_map
    //valid peroids_week_map
    //camId,{func,Vector<Polygon>}
    //camId,{func,Vector<std::string>}
    // std::unordered_map<int,std::unordered_map<std::string,
std::vector<std::string>>  >  validperoids_week_map;
```

```
    // std::unordered_map<int,std::unordered_map<std::string,
std::vector<Polygon>>  >  validperoids_hour_map;
    std::unordered_map<int,std::unordered_map<std::string,
std::vector<std::string> >  >::iterator got_intervalIterWM =
 globalJsonConfObj.validperoids_week_map.find(camId);
    if (got_intervalIterWM == globalJsonConfObj.validperoids_week_map.end())
    {
        ANNIWOLOG(INFO) << "not found in validperoids_week_map,camId:" <<camId;
    }
    else
    {
        std::unordered_map<std::string, std::vector<std::string> >&
intervalFuncMap  = got_intervalIterWM->second;
        std::unordered_map<std::string, std::vector<std::string> >::iterator
gotIntervalFuncMapIter = intervalFuncMap.find(f);
        if (gotIntervalFuncMapIter == intervalFuncMap.end())
        {
            ANNIWOLOG(INFO) << "not found in validperoids_week_map,camId:"
<<camId<<" func:"<<f;
        }
        else
        {
            bool isInCurWeek=false;
            int curWeek = getCurWeekDayXB();
            int weekDayArrIndex=0;

            std::vector<std::string>& vecValues =  gotIntervalFuncMapIter-
>second;
            for(auto& sstr:vecValues )
            {
                int iweek = sstr[0] - '0';

                if(curWeek == iweek)
                {
                    isInCurWeek=true;
                    weekDayArrIndex++;
                    break;
                }else
                {
                    weekDayArrIndex++;
                    continue;
                }
            }
            weekDayArrIndex-=1;//超了1.

            std::stringstream buffer;
            for(int i=0; i < vecValues.size(); i++)
            {
                const std::string& pt=vecValues[i];
                buffer << pt<<" ";
            }
            std::string vecValuestr(buffer.str());

            if(isInCurWeek)
            {
```

```
                ANNIWOLOG(INFO) << "It is in validperoids_week_map,camId:"
<<camId<<" func:"<<f<<"curweek:"<<curWeek<<"configweek:"
<<vecValuestr<<std::endl;


                std::unordered_map<int,std::unordered_map<std::string,
std::vector<Polygon>> >::iterator got_intervalIter =
 globalJsonConfObj.validperoids_hour_map.find(camId);
                if (got_intervalIter ==
globalJsonConfObj.validperoids_hour_map.end())
                {
                    ANNIWOLOG(INFO) << "not found in
validperoids_hour_map,camId:" <<camId;
                }
                else
                {
                    std::unordered_map<std::string, std::vector<Polygon>>&
intervalFuncMap  = got_intervalIter->second;
                    std::unordered_map<std::string,
std::vector<Polygon>>::iterator gotIntervalFuncMapIter =
intervalFuncMap.find(f);
                    if (gotIntervalFuncMapIter == intervalFuncMap.end())
                    {
                        ANNIWOLOG(INFO) << "not found in
validperoids_hour_map,camId:" <<camId<<" func:"<<f;
                    }
                    else
                    {
                        bool isValidPeriod=false;
                        int curDayTime=getCurMinuteDaytimeXB();

                        std::vector<Polygon>& hourvecValues =
 gotIntervalFuncMapIter->second;
                        //注意，与week设置有对应关系.
                        Polygon configHours= hourvecValues[weekDayArrIndex];
                        for(int i=0; i < configHours.size(); i++)
                        {
                            const cv::Point& pt=configHours.pt[i];
                            //start time;end time
                            if(curDayTime >= pt.x && curDayTime < pt.y)
                            {
                                ANNIWOLOG(INFO) << "It is in
validperoids_hour_map,camId:" <<camId<<" func:"<<f<<"curDaytime:"
<<curDayTime<<"configDaytime:"<<pt.x<<","<<pt.y<<std::endl;
                                isValidPeriod=true;
                                break;
                            }else
                            {
                                continue;
                            }
                        }
                        if(!isValidPeriod)
                        {

                            std::stringstream buffer;
                            for(int i=0; i < configHours.size(); i++)
                            {
                                const cv::Point& pt=configHours.pt[i];
```

```
                                buffer << pt.x<<","<<pt.y<<" ";
                            }
                            std::string text(buffer.str());

                            ANNIWOLOG(INFO) << "It is NOT in
validperoids_hour_map,camId:" <<camId<<" func:"<<f<<"curDaytime:"
<<curDayTime<<"configDaytime:"<<text<<std::endl;
                            return false;
                        }


                    }
                }
            }else
            {
                ANNIWOLOG(INFO) << "not in validperoids_week_map,camId:"
<<camId<<" func:"<<f<<"curweek:"<<curWeek<<"configweek:"
<<vecValuestr<<std::endl;
            }
        }
    }




    //进行时间间隔判断
    //camId,{func,interval}
    auto postprocess_start = std::chrono::steady_clock::now();
    std::unordered_map<int,std::unordered_map<std::string, AnniwoTimeLog >
 >::iterator got_intervalIter =
 globalJsonConfObj.interval_conf_map.find(camId);
    if (got_intervalIter == globalJsonConfObj.interval_conf_map.end())
    {
        ANNIWOLOG(INFO) << "not found in interval_conf_map,camId:" <<camId;
    }
    else
    {
        std::unordered_map<std::string, AnniwoTimeLog >& intervalFuncMap  =
got_intervalIter->second;
        std::unordered_map<std::string, AnniwoTimeLog >::iterator
gotIntervalFuncMapIter = intervalFuncMap.find(f);
        if (gotIntervalFuncMapIter == intervalFuncMap.end())
        {
            ANNIWOLOG(INFO) << "not found in intervalFuncMap,camId:" <<camId<<"
func:"<<f;
        }
        else
        {
            AnniwoTimeLog& vecValues =  gotIntervalFuncMapIter->second;
            const double confInterval = vecValues.configInterval;
            auto& prevTP = vecValues.lastTP;

            auto postprocess_end = std::chrono::steady_clock::now();
            std::chrono::duration<float> postprocess_diff = postprocess_end -
prevTP;

            double intervalCnter = confInterval - static_cast<double>
(postprocess_diff.count() * 1000); //ms
```

```
            if(intervalCnter <= 0.0)
            {
                ANNIWOLOG(INFO) << "Interval OK. camId:" <<camId<<"func:"<<f<<"
interval:"<< intervalCnter ;
                if(bRecord)
                {
                    prevTP=std::chrono::steady_clock::now();
                }
            }else
            {
                ANNIWOLOG(INFO) << "IGNORE camId:" <<camId<<"func:"<<f<<"
because interval:"<< intervalCnter ;
                return false;
            }

        }
    }

    return true;
}

//return:
//true: submit success
//false: submit failed
static bool submitThread(const std::string& f, int camId, cv::Mat img,
PERSONBASE_DET& personbaseDetObj)
{
    bool submitSuccess=false;
    if(istodoInitTracks)
    {
        ANNIWOLOG(INFO) <<"submitThread: istodoInitTracks is true. Abort...";
        return submitSuccess;
    }

    std::unique_lock<std::mutex> uniqueLockSubmit(mtx4SubmitThread,
std::defer_lock);
    ANNIWOLOG(INFO) <<"submitThread: waiting lock...";

    uniqueLockSubmit.lock();
    ANNIWOLOG(INFO) <<"submitThread: get into uniqueLock OK";

    //取得有效区域
    const Polygon* polygonSafeArea_ptr={nullptr};

    std::unordered_map<int,std::unordered_map<std::string, Polygon>  >::iterator
got_id_func_cap = globalJsonConfObj.validArea_conf_map.find(camId);

    if (got_id_func_cap == globalJsonConfObj.validArea_conf_map.end())
    {
        ANNIWOLOG(INFO) << "not found in validArea_conf_map,camId:" <<camId;
    }
    else
    {
        std::unordered_map<std::string, Polygon>& conf_map =got_id_func_cap-
>second;
        std::unordered_map<std::string, Polygon>::iterator got_id_func_cap2 =
conf_map.find(f);
```

```
        if (got_id_func_cap2 == conf_map.end())
        {
            ANNIWOLOG(INFO) << "not found in validArea_conf_map,camId:"
<<camId<<"func:"<<f;
        }
        else
        {
            const Polygon& polygonSafeArea = got_id_func_cap2->second;
            polygonSafeArea_ptr=&polygonSafeArea;
        }
    }

    if(f  == std::string("safeErea" ))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> safeEreaTasks;
        bool useExistingInstance=false;


        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 safeEreaTasks.find(tmpInstanceId);
            if (perCamTask == safeEreaTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in safeEreaTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for safeErea
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_SAFEAREA;
            ANNIWOLOG(INFO) <<"use this isntance: for safeErea  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }
```

```
        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 safeEreaTasks.find(instanceCnt);
        if (perCamTask == safeEreaTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for safeErea when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(safeEreaDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            safeEreaTasks.insert(std::move(taskpair));
            submitSuccess=true;

        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for safeErea camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(safeEreaDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            perCamTask->second=std::move(futr);
            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: safeErea group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;
            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end safeErea group camId: "
<<camId<<" instanceID:"<<instanceCnt;


    }
    else if(f ==  std::string("helmet"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> helmetTasks;

        bool useExistingInstance=false;
```

```
        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 helmetTasks.find(tmpInstanceId);
            if (perCamTask == helmetTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in helmetTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for helmet
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_HELMET;
            ANNIWOLOG(INFO) <<"use this isntance: for helmet  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 helmetTasks.find(instanceCnt);
        if (perCamTask == helmetTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for helmet when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(helmetDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            ANNIWOLOG(INFO) <<"submited thread: for helmet when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;


            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            helmetTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
```

```
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for helmet camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(helmetDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            ANNIWOLOG(INFO) <<"submited thread: for helmet when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            submitSuccess=false;

            ANNIWOLOG(INFO) <<"DISCARD img for: helmet group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;
        }
```

```
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end helmet group camId: "<<camId<<"
instanceID:"<<instanceCnt;

    }

    #ifdef __aarch64__
    #else
    else if(f ==  std::string("convery"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> converyTasks;

        bool useExistingInstance=false;


        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 converyTasks.find(tmpInstanceId);
            if (perCamTask == converyTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in converyTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for convery
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_CONVERY;
            ANNIWOLOG(INFO) <<"use this isntance: for convery  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }
```

```
        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 converyTasks.find(instanceCnt);
        if (perCamTask == converyTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for convery when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(converyDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            converyTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for convery camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(converyDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
```

```
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: convery group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;
            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }


    }
    #endif

    else if(f  == std::string("uptruck"))
    {
        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> UptruckTasks;

        bool useExistingInstance=false;


        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 UptruckTasks.find(tmpInstanceId);
            if (perCamTask == UptruckTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in UptruckTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;
```

```
                ANNIWOLOG(INFO) <<"use existing isntance: for uptruck
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_UPTRUCK;
            ANNIWOLOG(INFO) <<"use this isntance: for uptruck  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 UptruckTasks.find(instanceCnt);
        if (perCamTask == UptruckTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for uptruck when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(uptruckDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            UptruckTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for uptruck camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(uptruckDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);
```

```
            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: uptruck group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end uptruck group camId: "<<camId<<"
instanceID:"<<instanceCnt;


    }
    else if(f  == std::string("window"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> windowTasks;

        bool useExistingInstance=false;


        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;
```

```
            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 windowTasks.find(tmpInstanceId);
            if (perCamTask == windowTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in windowTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for window
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_WINDOW;
            ANNIWOLOG(INFO) <<"use this isntance: for window  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 windowTasks.find(instanceCnt);
        if (perCamTask == windowTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for window when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(windowDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            windowTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
```

```
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for window camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(windowDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: window group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;
            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end window group camId: "<<camId<<"
instanceID:"<<instanceCnt;

    }
    else if (f ==  std::string("fire"))
    {
        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> fireTasks;
```

```
        bool useExistingInstance=false;

        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 fireTasks.find(tmpInstanceId);
            if (perCamTask == fireTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in fireTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for fire  instanceID:"
<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_FIRE;
            ANNIWOLOG(INFO) <<"use this isntance: for fire  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 fireTasks.find(instanceCnt);
        if (perCamTask == fireTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for fire when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(fireDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            fireTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
```

```
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for fire camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(fireDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);


            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: fire group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }
```

```
        ANNIWOLOG(INFO) <<"submited thread: end fire group camId: "<<camId<<"
instanceID:"<<instanceCnt;


    }
    else if (f ==  std::string("xunjian"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> xunjianTasks;

        bool useExistingInstance=false;

        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 xunjianTasks.find(tmpInstanceId);
            if (perCamTask == xunjianTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in xunjianTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for xunjian
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_XUNJIAN;
            ANNIWOLOG(INFO) <<"use this isntance: for xunjian  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 xunjianTasks.find(instanceCnt);
        if (perCamTask == xunjianTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for xunjian when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;
```

```
            std::future<void> futr=pool->enqueue(xunjianDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            xunjianTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for xunjian camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(xunjianDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);


            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
```

```
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: xunjian group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end xunjian group camId: "<<camId<<"
instanceID:"<<instanceCnt;

    }
    else if (f ==  std::string("zhuangxieyou"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> zhuangxieyouTasks;

        bool useExistingInstance=false;

        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 zhuangxieyouTasks.find(tmpInstanceId);
            if (perCamTask == zhuangxieyouTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in zhuangxieyouTasks:"<<iterCamInstanceId->second<<" camId:"
<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for zhuangxieyou
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_ZXY;
```

```
            ANNIWOLOG(INFO) <<"use this isntance: for zhuangxieyou  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 zhuangxieyouTasks.find(instanceCnt);
        if (perCamTask == zhuangxieyouTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for zhuangxieyou when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(zxyDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            zhuangxieyouTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for zhuangxieyou camId:"
<<camId<<" instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(zxyDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr);



            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
```

```
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: zhuangxieyou group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end zhuangxieyou group camId: "
<<camId<<" instanceID:"<<instanceCnt;


    }
    else if (f ==  std::string("chepai"))
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> chepaiTasks;

        bool useExistingInstance=false;

        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 chepaiTasks.find(tmpInstanceId);
            if (perCamTask == chepaiTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in chepaiTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
```

```
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for chepai
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_CHEPAI;
            ANNIWOLOG(INFO) <<"use this isntance: for chepai  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 chepaiTasks.find(instanceCnt);
        if (perCamTask == chepaiTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for chepai when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(chepaiDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            chepaiTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
```

```
            ANNIWOLOG(INFO) <<"submiting thread: for chepai camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(chepaiDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);


            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: chepai group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end chepai group camId: "<<camId<<"
instanceID:"<<instanceCnt;


    }
    else if (f ==  std::string("mask"))
    {
        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> maskTasks;

        bool useExistingInstance=false;

        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
```

```
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 maskTasks.find(tmpInstanceId);
            if (perCamTask == maskTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in maskTasks:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for mask  instanceID:"
<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {
            instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_MASK;
            ANNIWOLOG(INFO) <<"use this isntance: for mask  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }



        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 maskTasks.find(instanceCnt);
        if (perCamTask == maskTasks.end())
        {
            ANNIWOLOG(INFO) <<"submiting thread: for mask when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(maskDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            maskTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
```

```
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for mask camId:"<<camId<<"
instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(maskDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);


            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: mask group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;

            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end mask group camId: "<<camId<<"
instanceID:"<<instanceCnt;
```

```
    }
    else if( f ==  std::string("smoke") || f ==  std::string("phone") )
    {

        static int instanceCnt=0;
        //camId, instanceId
        static std::unordered_map<int, int> busyCamIdInstanceMap;
        static std::unordered_map<int, std::future<void>> smokephoneTasks;

        bool useExistingInstance=false;


        std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
        if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
        {
            int tmpInstanceId=iterCamInstanceId->second;

            std::unordered_map<int, std::future<void>>::iterator perCamTask =
 smokephoneTasks.find(tmpInstanceId);
            if (perCamTask == smokephoneTasks.end())
            {
                ANNIWOLOG(INFO) <<"Warining:It in busyCamIdInstanceMap but NO
this instanceID in smokephone:"<<iterCamInstanceId->second<<" camId:"<<camId;
                busyCamIdInstanceMap.erase(camId);

            }else
            {
                useExistingInstance=true;
                instanceCnt=iterCamInstanceId->second;

                ANNIWOLOG(INFO) <<"use existing isntance: for smokephone
instanceID:"<<instanceCnt<<" camId:"<<camId;

            }


        }else
        {

 instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_SMOKEPHONE;
            ANNIWOLOG(INFO) <<"use this isntance: for smokephone  instanceID:"
<<instanceCnt<<" camId:"<<camId;

        }




        std::unordered_map<int, std::future<void>>::iterator perCamTask =
 smokephoneTasks.find(instanceCnt);
        if (perCamTask == smokephoneTasks.end())
        {
```

```
            ANNIWOLOG(INFO) <<"submiting thread: for smokephone when no this
instanceID:"<<instanceCnt<<" camId:"<<camId;

            std::future<void> futr=pool->enqueue(smokePhoneDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(futr));
            smokephoneTasks.insert(std::move(taskpair));

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));

            submitSuccess=true;


        }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
        {
            ANNIWOLOG(INFO) <<"submiting thread: for smokephone camId:"
<<camId<<" instanceID:"<<instanceCnt;
            std::future<void> futr=pool->enqueue(smokePhoneDetect-
>detect,camId,instanceCnt,img,polygonSafeArea_ptr,personbaseDetObj.det_results[c
amId]);

            perCamTask->second=std::move(futr);

            //更新busymap
            int previousCamId=-1;
            for(auto& item:busyCamIdInstanceMap)
            {
                if(item.second == instanceCnt)
                {
                    previousCamId=item.first;
                    break;
                }
            }
            if(previousCamId != -1)
            {
                busyCamIdInstanceMap.erase(previousCamId);
            }
            std::pair<int, int> tmpitem(camId,instanceCnt);
            busyCamIdInstanceMap.insert(std::move(tmpitem));
```

```
            submitSuccess=true;

        }
        else
        {
            ANNIWOLOG(INFO) <<"DISCARD img for: smokephone group camId: "
<<camId<<"because busy. instanceID:"<<instanceCnt;
            submitSuccess=false;

        }
        if(!useExistingInstance)
        {
            instanceCnt++;
        }

        ANNIWOLOG(INFO) <<"submited thread: end smokephone group camId: "
<<camId<<" instanceID:"<<instanceCnt;


    }
    else
    {
        ANNIWOLOG(INFO) <<f<<" is NOT the expect  function!"<<std::endl;
    }

    uniqueLockSubmit.unlock();
    return submitSuccess;

}



static void personbaseGroupFunc(const std::vector<std::string> infunctions, int
camId, int instanceID, cv::Mat img )
{
    PERSONBASE_DET& persondetObj=*personbase_detObjPtr;
    //clear camId base person det result buffer
    persondetObj.clearResults(camId);

    if(istodoInitTracks)
    {
        ANNIWOLOG(INFO) <<"personbaseGroupFunc: istodoInitTracks is true.
Abort...";
        return;
    }
    persondetObj.do_detect(camId,instanceID,img);

    if(istodoInitTracks)
    {
        ANNIWOLOG(INFO) <<"personbaseGroupFunc: istodoInitTracks is true.
Abort...";
        return;

    }

    if(persondetObj.hasperson(camId))
    {
```

```
        std::stringstream buffer;
        for(int i=0; i < infunctions.size(); i++)
        {
            const std::string& pt=infunctions[i];
            buffer << pt<<" ";
        }
        std::string vecValuestr(buffer.str());
        ANNIWOLOG(INFO) <<" personbaseGroupFunc infunctions:"
<<vecValuestr<<",camId"<<camId;


        for(const std::string& f:infunctions)
        {
            if(istodoInitTracks)
            {
                ANNIWOLOG(INFO) <<"personbaseGroupFunc: istodoInitTracks is
true. Abort...";
                return;

            }
            submitThread(f, camId, img,persondetObj);

        }
        ANNIWOLOG(INFO) <<"personbaseGroupFunc: exit...";


    }
}




void initAllTracks(const ANNIWO_JSON_CONF_CLASS& globalJsonConfObj)
{
    if(safeEreaDetect)
    {
        safeEreaDetect->initTracks(globalJsonConfObj);

    }
    if(windowDetect)
    {
        windowDetect->initTracks(globalJsonConfObj);

    }

    if(uptruckDetect)
    {
        uptruckDetect->initTracks(globalJsonConfObj);

    }

    if(fireDetect)
    {
        fireDetect->initTracks(globalJsonConfObj);

    }
    if(helmetDetect)
    {
```

```
        helmetDetect->initTracks(globalJsonConfObj);

    }
    if(smokePhoneDetect)
    {
        smokePhoneDetect->initTracks(globalJsonConfObj);
    }
    if(converyDetect)
    {
        #ifdef __aarch64__
            ANNIWOCHECK(false);
        #else
            converyDetect->initTracks(globalJsonConfObj);
        #endif
    }
    if(xunjianDetect)
    {
        xunjianDetect->initTracks(globalJsonConfObj);
    }
    if(zxyDetect)
    {
        zxyDetect->initTracks(globalJsonConfObj);
    }
    if(chepaiDetect)
    {
        chepaiDetect->initTracks(globalJsonConfObj);
    }
    if(maskDetect)
    {
        maskDetect->initTracks(globalJsonConfObj);
    }
}

//todo:absence是否提交逻辑可以写在这里
inline bool isRequireSubmitThread(const std::unordered_set<std::string>&
perCamsubmittedfuncs, const std::string& f, int camId)
{

    if(f == std::string("smoke") || f == std::string("phone"))
    {
        if(  ( perCamsubmittedfuncs.find("smoke") != perCamsubmittedfuncs.end()
) || ( perCamsubmittedfuncs.find("phone") != perCamsubmittedfuncs.end() )  )
        {
            return false;
        }
    }

    if(!isTimeIntervalOK(f,  camId))
    {
        ANNIWOLOG(INFO) <<"f is not time OK.ignore "<<f<<" camId:"<<camId;
        return false;
    }

    return true;
}


void* detectFunc(void* param)
```

```
{

    shapes::Rectangle videogpuObj;


    for (auto& f : globalINICONFObj.in_use_conf_functions)
    {
        if(f == std::string("fire"))
        {
            fireDetect = new
FireDetection(globalINICONFObj.onlineCollectionPath);
        }
        if( f == std::string("smoke") || f == std::string("phone") )
        {
            if(!smokePhoneDetect)
            {
                smokePhoneDetect = new SmokePhoneDetection();
            }

        }
        if(f == std::string("helmet") )
        {
            helmetDetect = new HelmetDetection();

        }
        if(f == std::string("safeErea"))
        {
            safeEreaDetect = new safeEreaDetection();

        }
        if(f == std::string("window"))
        {
            windowDetect = new WindowDetection();
        }
        if(f == std::string("convery"))
        {
            #ifdef __aarch64__
                ANNIWOCHECK(false);
            #else
                converyDetect = new ConveryDetection();
            #endif

        }
        if(f == std::string("uptruck"))
        {
            uptruckDetect = new UptruckDetection();
        }
        if(f == std::string("xunjian"))
        {
            xunjianDetect = new XunjianDetection();
        }
        if(f == std::string("zhuangxieyou"))
        {
            zxyDetect = new ZxyDetection();
        }
        if(f == std::string("chepai"))
        {
            chepaiDetect = new ChepaiDetection();
```

```
        }

        if(f == std::string("mask"))
        {
            maskDetect = new MaskDetection();
        }
    }


    basePersonDetect = new BasePersonDetection();

    ANNIWOCHECK(pool==nullptr);

    #ifdef __aarch64__
        // pool = new ThreadPool(100);
        pool = new ThreadPool(50);
        // poolpersonbase = new ThreadPool(28);
        poolpersonbase = new ThreadPool(12);
        std::cout<<"arm 64 machine enable! "<<std::endl;
    #else
        //x86_64 server
        pool = new ThreadPool(865);
        poolpersonbase = new ThreadPool(105);
    #endif

    int requestlogDumpTimeCounter=0;
    while(true)
    {
            // sleep(1);//unsigned int seconds
            // static int useconds=50*1000;
            // static int useconds=10*1000;//10ms 是内存不足会退出!!!!!!!!!!是因为线
程池太大!

            #ifdef __aarch64__
            static int useconds=50*1000;//50ms
            #else
            static int useconds=30*1000;//30ms
            #endif

            usleep(useconds);



            //todo:获取时间，备份日志与每日重新清空track
            int nowMinute=getCurMinuteDaytimeXB();

            if(!tidayUpNow)
            {
                if(nowMinute>tidyuptimeS && nowMinute<tidyuptimeE)
                {
                    //需要做重清并设置标志
                    tidayUpNow = true;
                    ANNIWOLOG(INFO) << "detect: enter set tidayUpNow:"
<<tidayUpNow<< "tidiedOnce:" <<tidiedOnce;
                }else
                {
                }
            }else
```

```
            {
                if(nowMinute>tidyuptimeS && nowMinute<tidyuptimeE)
                {
                }else
                {
                    ANNIWOLOG(INFO) << "detect: enter re-set tidayUpNow:"
<<tidayUpNow<< "tidiedOnce:" <<tidiedOnce;
                    tidayUpNow=false;    //重置时间标识
                    tidiedOnce = false;
                }
            }


            if(istodoInitTracks || ( tidayUpNow && (!tidiedOnce) )  )
            {
                if(istodoInitTracks)
                {
                    ANNIWOLOG(INFO) << "detect: enter istodoInitTracks==True" ;
                }else
                {
                    ANNIWOLOG(INFO) << "detect: enter tidayUpNow:"
<<tidayUpNow<< "tidiedOnce:" <<tidiedOnce;
                }

                if(isFirstimeInitFramer)
                {
                    ANNIWOLOG(INFO) << "detect: enter isFirstimeInitFramer" ;
                    videogpuObj.init_framer(rtmpStrs,rtmpIds);
                    isFirstimeInitFramer = false;
                }
                else
                {
                    ANNIWOLOG(INFO) << "detect: calling
stop2uninitialize_framing" ;
                    videogpuObj.stop2uninitialize_framing();
                    ANNIWOLOG(INFO) << "detect: calling re_init_framer" ;
                    videogpuObj.re_init_framer(rtmpStrs, rtmpIds);
                }
                ANNIWOLOG(INFO) <<"detect: calling start_framing";
                videogpuObj.start_framing();
                ANNIWOLOG(INFO) <<"detect: start_framing OK";

                //must wait all thread done,otherwise may have issue!!! like
deepsort feature extractor
                if(pool)
                {
                    if(poolpersonbase)
                    {
                        ANNIWOLOG(INFO) <<"detect: try to deleted
poolpersonbase...";

                        delete poolpersonbase;
                        poolpersonbase=nullptr;

                        ANNIWOLOG(INFO) <<"detect: poolpersonbase deleted OK";

                    }
                    ANNIWOLOG(INFO) <<"detect: try to deleted main pool";
```

```
                    delete pool;
                    pool=nullptr;


                    ANNIWOLOG(INFO) <<"detect: pool deleted OK";


                    //在这里进行整理
                    if(tidayUpNow )
                    {
                        usleep(useconds);
                        size_t filesize = getFileSize1("../lsm.log");

                        if(filesize > 100*1024*1024)//200M
                        {
                            std::stringstream buffer;

                            buffer <<"../lsm.log_"<<log_suffix_cnt;

                            if(CopyFile("../lsm.log",buffer.str().c_str() ))
                            {
                                if(TrunckFile("../lsm.log"))
                                {
                                    ANNIWOLOG(INFO) <<"detect: TrunckFile
success!";
                                }
                                else
                                {
                                    ANNIWOLOG(INFO) <<"detect: TrunckFile
error!";
                                }

                            }else
                            {
                                ANNIWOLOG(INFO) <<"detect: calling file copy
error!";
                            }

                            log_suffix_cnt += 1;
                        }

                        ANNIWOLOG(INFO) <<"detect: log tidyup OK";

                    }


                    #ifdef __aarch64__
                        pool = new ThreadPool(100);
                        poolpersonbase = new ThreadPool(28);
                    #else
                        //x86_64 server
                        pool = new ThreadPool(768);
                        poolpersonbase = new ThreadPool(125);
                    #endif

                    ANNIWOLOG(INFO) <<"detect: pool created OK";
```

```
                }

                std::unique_lock<std::mutex> uniqueLock(mtx4GlobalVariables,
std::defer_lock);
                uniqueLock.lock();
                ANNIWOLOG(INFO) <<"detect: get into uniqueLock OK";

                //////////

 globalJsonConfObj.id_func_cap=globalJsonConfObjNotApplied.id_func_cap;

 globalJsonConfObj.validtypes_map=globalJsonConfObjNotApplied.validtypes_map;

 globalJsonConfObj.validperoids_hour_map=globalJsonConfObjNotApplied.validperoid
s_hour_map;

 globalJsonConfObj.validperoids_week_map=globalJsonConfObjNotApplied.validperoid
s_week_map;

 globalJsonConfObj.validArea_conf_map=globalJsonConfObjNotApplied.validArea_conf
_map;

 globalJsonConfObj.interval_conf_map=globalJsonConfObjNotApplied.interval_conf_m
ap;

 globalJsonConfObj.stay_conf_map=globalJsonConfObjNotApplied.stay_conf_map;

 globalJsonConfObj.taskid_conf_map=globalJsonConfObjNotApplied.taskid_conf_map;

 globalJsonConfObj.facedatasetpath_conf_map=globalJsonConfObjNotApplied.facedata
setpath_conf_map;

 globalJsonConfObj.eventUrl_conf_map=globalJsonConfObjNotApplied.eventUrl_conf_m
ap;

 globalJsonConfObj.absenceStartConition_conf_map=globalJsonConfObjNotApplied.abs
enceStartConition_conf_map;
                //////////

                std::stringstream buffer;
                buffer << "../requestjsonApplied_"
<<requestlogDumpTimeCounter<<".log";
                std::string nameStr(buffer.str());

                dumpToTxtFile(nameStr.c_str(),currentInJsonAll);
                requestlogDumpTimeCounter++;

                uniqueLock.unlock();

                ANNIWOLOG(INFO) <<"detect: out of  uniqueLock OK";
                ANNIWOLOG(INFO) <<"detect: newly
applied:globalJsonConfObj.validperoids_week_map.size"<<
globalJsonConfObj.validperoids_week_map.size();
                //std::unordered_map<int,std::unordered_map<std::string, Polygon>
>
                for(auto& itempair:globalJsonConfObj.validArea_conf_map)
                {
```

```
                    // cout<<kv.first<<kv.second<<endl;
                    int camId = itempair.first;
                    auto& funcAreaMap = itempair.second;

                    for(auto& funcAreaItem:funcAreaMap)
                    {
                        auto& funcname = funcAreaItem.first;
                        auto& polygonSafeArea = funcAreaItem.second;

                        std::stringstream buffer;
                        for(int i=0; i < polygonSafeArea.size(); i++)
                        {
                            const cv::Point& pt=polygonSafeArea.pt[i];
                            buffer << pt.x<<","<<pt.y<<" ";
                        }
                        std::string text(buffer.str());

                        ANNIWOLOG(INFO) << "detect
json:validArea_conf_map,camId:" <<camId<<" func:"<<funcname<<" validArea:"
<<text;

                    }

                }


                basePersonDetect-
>initTracks(globalJsonConfObj,person_base_functions);

                ANNIWOLOG(INFO) <<"detect: basePersonDetect->initTracks OK";

                if(personbase_detObjPtr)
                {
                    delete personbase_detObjPtr;
                    personbase_detObjPtr=nullptr;
                }
                ANNIWOLOG(INFO) <<"detect: personbase_detObjPtr deleted OK";

                personbase_detObjPtr =  new
PERSONBASE_DET(globalJsonConfObj.id_func_cap,person_base_functions);
                ANNIWOLOG(INFO) <<"detect: personbase_detObjPtr created OK";



                initAllTracks(globalJsonConfObj);

                ANNIWOLOG(INFO) <<"detect: initAllTracks OK";


                istodoInitTracks = false;

                if(tidayUpNow)
                {
                    tidiedOnce=true;
                }

            }
```

```
            ANNIWOLOG(INFO) <<"detect(): calling consume_frame";

            std::pair< int,cv::Mat> out_tuple = videogpuObj.consume_frame();
            int camId = out_tuple.first;
            cv::Mat img = out_tuple.second.clone();



            std::unordered_map<int, std::vector<std::string> >::const_iterator
got_id_func_cap = globalJsonConfObj.id_func_cap.find(camId);

            if (got_id_func_cap == globalJsonConfObj.id_func_cap.end())
            {
                ANNIWOLOG(INFO) << "not found in
globalJsonConfObj.id_func_cap,camId:" <<camId;
            }
            else
            {
                const std::vector<std::string>& func_list=got_id_func_cap-
>second;
                int foundcamID =  got_id_func_cap->first;

                ANNIWOLOG(INFO) <<"camId:" << foundcamID << " has " <<
func_list[0] << "...etc.";



                std::unordered_set<std::string> perCamsubmittedfuncs;

                std::vector<std::string> personfuncsPerCamthis;

                for (auto& f : func_list)
                {
                    // ANNIWOLOG(INFO) <<"detect(): f="<< f ;

                    std::unordered_set<std::string>::const_iterator
got_in_use_conf_functions = globalINICONFObj.in_use_conf_functions.find(f);

                    if (got_in_use_conf_functions ==
globalINICONFObj.in_use_conf_functions.end())
                    {
                        ANNIWOLOG(INFO) <<"f is not use.continue:"<<f;
                        continue;
                    }
                    else
                    {
                        ANNIWOLOG(INFO) <<" f in the list:"<<f<< " camId:"
<<camId;



                        bool isOK2Submit =
isRequireSubmitThread(perCamsubmittedfuncs,f,camId);

                        if(isOK2Submit)
                        {

                            //组控制,比如personbase组
```

```
                            std::unordered_set<std::string>::const_iterator
got_person_base_functions = person_base_functions.find(f);

                            if (got_person_base_functions ==
person_base_functions.end())
                            {
                                bool submitSuccess = submitThread(f, camId,
img,*personbase_detObjPtr);

                                perCamsubmittedfuncs.insert(f);

                                if( ! submitSuccess )
                                {
                                    resetIntervalRecord(f, camId);
                                }

                            }
                            else
                            {
                                personfuncsPerCamthis.push_back(f);
                                perCamsubmittedfuncs.insert(f);

                                ANNIWOLOG(INFO) <<"put:"<<f<<" in array for
later submit. "<<" camId:"<<camId;

                            }


                        }else
                        {
                            ANNIWOLOG(INFO) <<"no submit thread as not
isOK2Submit:"<<f<<" camId:"<<camId<<std::endl;
                        }


                    }



                }



                if(personfuncsPerCamthis.size() > 0)
                {
                    static int instanceCnt=0;

                    //camId, instanceId
                    static std::unordered_map<int, int> busyCamIdInstanceMap;
                    bool useExistingInstance=false;


                    std::unordered_map<int, int>::iterator iterCamInstanceId =
 busyCamIdInstanceMap.find(camId);
                    if (iterCamInstanceId != busyCamIdInstanceMap.end()) //在
busyCamIdInstanceMap中有记录
                    {
```

```
                        int tmpInstanceId=iterCamInstanceId->second;

                        std::unordered_map<int, std::future<void>>::iterator
perCamTask =  pGroupTasks.find(tmpInstanceId);
                        if (perCamTask == pGroupTasks.end())
                        {
                            ANNIWOLOG(INFO) <<"Warining:It in
busyCamIdInstanceMap but NO this instanceID in pGroupTasks:"<<iterCamInstanceId-
>second<<" camId:"<<camId;
                            busyCamIdInstanceMap.erase(camId);

                        }else
                        {
                            useExistingInstance=true;
                            instanceCnt=iterCamInstanceId->second;

                            ANNIWOLOG(INFO) <<"use existing isntance: for
personbaseGroup  instanceID:"<<instanceCnt<<" camId:"<<camId;

                        }


                    }else
                    {

 instanceCnt=instanceCnt%globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE;
                        ANNIWOLOG(INFO) <<"use this isntance: for
personbaseGroup  instanceID:"<<instanceCnt<<" camId:"<<camId;

                    }



                    std::unordered_map<int, std::future<void>>::iterator
perCamTask =  pGroupTasks.find(instanceCnt);
                    if (perCamTask == pGroupTasks.end())
                    {
                        ANNIWOLOG(INFO) <<"submiting thread: for personbaseGroup
when no this instanceID:"<<instanceCnt<<" camId:"<<camId;

                        std::future<void> fur=poolpersonbase-
>enqueue(personbaseGroupFunc,personfuncsPerCamthis,camId,instanceCnt,img);
                        std::pair<int, std::future<void>>
taskpair(instanceCnt,std::move(fur));
                        pGroupTasks.insert(std::move(taskpair));

                                                //更新busymap
                        int previousCamId=-1;
                        for(auto& item:busyCamIdInstanceMap)
                        {
                            if(item.second == instanceCnt)
                            {
                                previousCamId=item.first;
                                break;
                            }
                        }
                        if(previousCamId != -1)
                        {
```

```
                            busyCamIdInstanceMap.erase(previousCamId);
                        }
                        std::pair<int, int> tmpitem(camId,instanceCnt);
                        busyCamIdInstanceMap.insert(std::move(tmpitem));


                    }else if(!perCamTask->second.valid()  || perCamTask-
>second.wait_for(std::chrono::nanoseconds(3))==std::future_status::ready)
                    {
                        ANNIWOLOG(INFO) <<"submiting thread: for personbaseGroup
camId:"<<camId<<" instanceID:"<<instanceCnt;
                        std::future<void> futr=poolpersonbase-
>enqueue(personbaseGroupFunc,personfuncsPerCamthis,camId,instanceCnt,img);

                        perCamTask->second=std::move(futr);

                        //更新busymap
                        int previousCamId=-1;
                        for(auto& item:busyCamIdInstanceMap)
                        {
                            if(item.second == instanceCnt)
                            {
                                previousCamId=item.first;
                                break;
                            }
                        }
                        if(previousCamId != -1)
                        {
                            busyCamIdInstanceMap.erase(previousCamId);
                        }
                        std::pair<int, int> tmpitem(camId,instanceCnt);
                        busyCamIdInstanceMap.insert(std::move(tmpitem));


                    }
                    else
                    {
                        ANNIWOLOG(INFO) <<"DISCARD img for: personbaseGroup
group camId: "<<camId<<"because busy. instanceID:"<<instanceCnt;
                    }

                    if(!useExistingInstance)
                    {
                        instanceCnt++;
                    }

                }


            }

//////////////////////////Main logic end
    }
}


void engineStart()
{
```

```
    ANNIWOLOG(INFO) << "Now (re)start engine!";

    ////////////////////////////
    std::ifstream ifs;
    // 3 打开文件 判断是否打开成功
    ifs.open("../key.txt", std::ios::in);
    if (!ifs.is_open()) {
            ANNIWOLOG(INFO) << "file open ../key.txt failed!"<< endl;
            ANNIWOCHECK(false);
            exit(0);
    }
    string buf;
    string outbuf;
    while (getline(ifs, buf)) {
        outbuf=buf;
        ANNIWOLOG(INFO) << outbuf ;
    }
    bool verifyResult = VerifyKey(outbuf);
    if(verifyResult)
    {
        ANNIWOLOG(INFO) << "KEY OK using ../key.txt"<< endl;
    }
    else
    {
        ANNIWOLOG(INFO) << "KEY Failed using ../key.txt"<< endl;
        ANNIWOCHECK(false);
        exit(0);
    }
    ////////////////////////

    std::unique_lock<std::mutex> uniqueLock(mtx4GlobalVariables,
std::defer_lock);
    uniqueLock.lock();

    //start the detect thread
    if(!threadDetect)
    {
        istodoInitTracks=true;
        threadDetect = new std::thread(detectFunc, (void*)0 );
        ANNIWOLOG(INFO) << "detection thread started" ;

    }
    else
    {
        ANNIWOLOG(INFO) << "detection thread is exsiting...set
istodoInitTracks=True";
        istodoInitTracks=true;
    }


    uniqueLock.unlock();

    return ;
}
```

```
void managerMent(const std::string& inJsonAll)
{
        static int requestlogDumpTimeCounter=0;
        if(geteuid()==0)
        {
            ANNIWOLOG(INFO) << "with root running,OK!"<< endl;
        }
        else
        {
            ANNIWOLOG(INFO) << "Not with root failed!"<< endl;
            ANNIWOCHECK(false);
            exit(-1);
        }
//////////////////////////
        std::ifstream ifs;
        // 3 打开文件 判断是否打开成功
        ifs.open("../key.txt", std::ios::in);
        if (!ifs.is_open()) {
                ANNIWOLOG(INFO) << "file open ../key.txt failed!"<< endl;
                ANNIWOCHECK(false);
                exit(0);
        }
        string buf;
        string outbuf;
        while (getline(ifs, buf)) {
            outbuf=buf;
            ANNIWOLOG(INFO) << outbuf ;
        }
        bool verifyResult = VerifyKey(outbuf);
        if(verifyResult)
        {
            ANNIWOLOG(INFO) << "KEY OK using ../key.txt"<< endl;
        }
        else
        {
            ANNIWOLOG(INFO) << "KEY Failed using ../key.txt"<< endl;
            ANNIWOCHECK(false);
            exit(0);
        }
//////////////////////
        ANNIWOLOG(INFO) <<"request json:"<< inJsonAll ;

        std::stringstream buffer;
        buffer << "../requestjsonNAp_"<<requestlogDumpTimeCounter<<".log";
        std::string nameStr(buffer.str());

        dumpToTxtFile(nameStr.c_str(),inJsonAll);
        requestlogDumpTimeCounter++;

        currentInJsonAll=inJsonAll;



        rapidjson::Document  document;

        if(document.Parse(inJsonAll.c_str()).HasParseError())
        {
```

```
            ANNIWOLOG(INFO) <<"Fatal error json parse!"<<std::endl;
            std::ostringstream os;
            os << "Fatal error json parse! Check the format!!";
            return ;
        }

        assert(document.IsArray());    // Document is a JSON value represents
the root of DOM. Root can be either an object or array.


        std::unique_lock<std::mutex> uniqueLock(mtx4GlobalVariables,
std::defer_lock);
        uniqueLock.lock();

        globalJsonConfObjNotApplied.id_func_cap.clear();
        globalJsonConfObjNotApplied.validtypes_map.clear();
        globalJsonConfObjNotApplied.validperoids_hour_map.clear();
        globalJsonConfObjNotApplied.validperoids_week_map.clear();
        globalJsonConfObjNotApplied.validArea_conf_map.clear();
        globalJsonConfObjNotApplied.interval_conf_map.clear();
        globalJsonConfObjNotApplied.stay_conf_map.clear();
        globalJsonConfObjNotApplied.taskid_conf_map.clear();
        globalJsonConfObjNotApplied.facedatasetpath_conf_map.clear();
        globalJsonConfObjNotApplied.eventUrl_conf_map.clear();
        rtmpIds.clear();
        rtmpStrs.clear();


        for (rapidjson::SizeType i = 0; i < document.Size(); i++) // rapidjson
uses SizeType instead of size_t.
        //todo:解析json
        {
            const rapidjson::Value& q = document[i];

            assert(q.IsObject());


            bool bHascamId = q.HasMember("camId");
            bool bHascamStr =  q.HasMember("camStr");
            bool bHasfuncConf =  q.HasMember("funcConf");
            assert(bHascamId && bHascamStr  );


            int camId = -1;

            // camID必须为int值，保持与c++ videogpu一致
            std::stringstream stream;
            stream << q["camId"].GetString();     //向stream中插入字符串"1234"
            stream >> camId;     //从stream中提取刚插入的字符串"1234


            std::string rmtplink = {q["camStr"].GetString()};
            // cam_map[camId] = rmtplink
            rtmpIds.emplace_back(camId);
            rtmpStrs.emplace_back(rmtplink);

            assert(q.HasMember("func"));
```

```
            const rapidjson::Value& funcq=q["func"];
            assert(funcq.IsArray());
            for (rapidjson::SizeType j = 0; j < funcq.Size(); j++) // rapidjson
uses SizeType instead of size_t.
            {
                const rapidjson::Value& fq = funcq[j];
                std::string  f=fq.GetString();
                std::unordered_map<int, std::vector<std::string> >::iterator
got_id_func_cap = globalJsonConfObjNotApplied.id_func_cap.find(camId);

                if (got_id_func_cap ==
globalJsonConfObjNotApplied.id_func_cap.end())
                {
                    ANNIWOLOG(INFO) << "not found in
globalJsonConfObjNotApplied.id_func_cap,camId:" <<camId<<"new func:"<<f;
                    std::vector<std::string> caps = {f};

 globalJsonConfObjNotApplied.id_func_cap.insert(std::pair<int,
std::vector<std::string> >(camId,caps) );
                }
                else
                {
                    std::vector<std::string>& func_list=got_id_func_cap->second;
                    func_list.emplace_back(f);
                }
            }


            if(q.HasMember("funcConf"))
            {
                const rapidjson::Value&  funcConfq = q["funcConf"];
                assert(funcConfq.IsObject());

                if( funcConfq.IsNull())
                {
                    continue;
                }

                for(rapidjson::Value::ConstMemberIterator confIter =
funcConfq.MemberBegin(); confIter != funcConfq.MemberEnd(); ++confIter){
                {   //"funcConf"

                    const rapidjson::Value& childIter = confIter->value;
                    std::string funcname  = confIter->name.GetString();
                    //"safeErea":/"helmet":/"smoke"
                    ANNIWOLOG(INFO) <<"funcname:"<<funcname<<std::endl;
                    if(! childIter.IsObject() )
                    {//错误处理!
                        const rapidjson::Value& funcValue  = childIter;
                        rapidjson::Type tpValue= funcValue.GetType();
                        ANNIWOLOG(INFO) <<"funcValue type:"<<tpValue<<std::endl;
                        if(tpValue == rapidjson::Type::kNullType)
                        {
                            ANNIWOLOG(INFO) <<"funcValue is null"<<std::endl;
                        }
                        else if(tpValue == rapidjson::Type::kArrayType)
                        {
```

```
                            ANNIWOLOG(INFO) <<"funcValue is Array. Wrong type"
<<std::endl;

                        }
                        else if(tpValue == rapidjson::Type::kNumberType)
                        {
                            ANNIWOLOG(INFO) <<"funcValue is Number. Wrong type"
<<std::endl;

                        }
                        else if (tpValue == rapidjson::Type::kStringType)
                        {
                            ANNIWOLOG(INFO) <<"funcValue is String. Wrong type"
<<std::endl;
                        }
                        else
                        {
                            ANNIWOLOG(INFO) <<"funcValue is UNKOWN type"
<<std::endl;

                        }
                    }else
                    {
                        for(rapidjson::Value::ConstMemberIterator it =
childIter.MemberBegin(); it != childIter.MemberEnd(); ++it)
                        {//功能对应的具体配
置."area"/"interval"/"stay"/"types"/"periods"

                            std::string  paramname=it->name.GetString();

                            ANNIWOLOG(INFO) <<"paramname:"
<<paramname<<std::endl;

                            const rapidjson::Value& paramvalue  = it->value;
                            rapidjson::Type tpValue= paramvalue.GetType();
                            ANNIWOLOG(INFO) <<"paramvalue type:"
<<tpValue<<std::endl;
                            if(tpValue == rapidjson::Type::kNullType)
                            {
                                ANNIWOLOG(INFO) <<"paramvalue is null"
<<std::endl;
                            }
                            else if(tpValue == rapidjson::Type::kArrayType)
                            {//功能对应的具体配
置."area"/"interval"/"stay"/"types"/"periods" 对应的值.

                                if(paramname == std::string("types"))
                                {
                                    std::vector<std::string> validtypes;
                                    for (rapidjson::SizeType i = 0; i <
paramvalue.Size(); i++) // rapidjson uses SizeType instead of size_t.
                                    {
                                        assert(paramvalue[i].IsString());

 validtypes.push_back(paramvalue[i].GetString());
                                    }

                                    //valid types
```

```
                                    //camId,{func,Vector<String>}

 std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.validtypes_map.find(camId);
                                    std::vector<std::string> tmpempty;

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.validtypes_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
validtypes_conf_map,camId:" <<camId<<"new to validtypes_conf_map";
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes> mapp = { {funcname , {validtypes,tmpempty}} };

 globalJsonConfObjNotApplied.validtypes_map.insert(std::pair<int,
std::unordered_map<std::string, AnniwoSafeEreaConcernTypes>  >( camId,
std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>::iterator got_id_func_cap2 =
conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, AnniwoSafeEreaConcernTypes  >(funcname,
{validtypes,tmpempty}));
                                        }
                                        else
                                        {
                                            got_id_func_cap2->second.validtypes
= validtypes;
                                        }
                                    }
                                }

                                if(paramname == std::string("excludeTypes"))
                                {
                                    std::vector<std::string> excludetypes;
                                    for (rapidjson::SizeType i = 0; i <
paramvalue.Size(); i++) // rapidjson uses SizeType instead of size_t.
                                    {
                                        assert(paramvalue[i].IsString());

 excludetypes.push_back(paramvalue[i].GetString());
                                    }

                                    //exclude types
                                    //camId,{func,
{Vector<String>,Vector<String>}}

 std::unordered_map<int,std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>  >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.validtypes_map.find(camId);
```

```
                                    std::vector<std::string> tmpempty;

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.validtypes_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
excludetypes_conf_map,camId:" <<camId<<"new to excludetypes_conf_map";
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes> mapp = { {funcname , {tmpempty,excludetypes}} };

 globalJsonConfObjNotApplied.validtypes_map.insert(std::pair<int,
std::unordered_map<std::string, AnniwoSafeEreaConcernTypes>  >( camId,
std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
AnniwoSafeEreaConcernTypes>::iterator got_id_func_cap2 =
conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, AnniwoSafeEreaConcernTypes  >(funcname,
{tmpempty,excludetypes}));
                                        }
                                        else
                                        {
                                            got_id_func_cap2-
>second.excludeTypes = excludetypes;
                                        }
                                    }
                                }




                                if(paramname == std::string("periods"))
                                {
                                    //注意如下两个有对应关系。
                                    std::vector<Polygon> validPeriods_hour;
                                    std::vector<std::string> validPeriods_week;

                                    Polygon polygon;
                                    for (rapidjson::SizeType i = 0; i <
paramvalue.Size(); i++) // rapidjson uses SizeType instead of size_t.
                                    {

                                        const rapidjson::Value& q =
paramvalue[i];
                                        assert(q.IsObject());
                                        bool bHasHour = q.HasMember("hour");
                                        bool bHasWeek =  q.HasMember("week");
                                        assert(bHasHour && bHasWeek);
                                        const rapidjson::Value&  hourarr =
q["hour"];
```

```
                                        const rapidjson::Value&  weekvalue =
q["week"];



                                        //////////////////////
                                        assert(hourarr.IsArray());
                                        polygon.clear();

                                        for (rapidjson::SizeType i = 0; i <
hourarr.Size(); i++) // rapidjson uses SizeType instead of size_t.
                                        {
                                            assert(hourarr[i].IsArray());

                                            cv::Point p;
                                            for (rapidjson::SizeType j = 0; j <
hourarr[i].Size(); j++) // rapidjson uses SizeType instead of size_t.
                                            {
                                                if(j == 0)
                                                    p.x = hourarr[i]
[j].GetInt();
                                                else
                                                    p.y = hourarr[i]
[j].GetInt();
                                            }
                                            polygon.add(p);
                                        }
                                        //////////////////////
                                        validPeriods_hour.push_back(polygon);

 validPeriods_week.push_back(weekvalue.GetString());

                                    }
                                    ANNIWOLOG(INFO) <<
"validPeriods_hour.size():" <<validPeriods_hour.size();
                                    ANNIWOLOG(INFO) <<
"validPeriods_week.size():" <<validPeriods_week.size();




 std::unordered_map<int,std::unordered_map<std::string, std::vector<Polygon>>
 >::iterator got_id_func_capIter =
globalJsonConfObjNotApplied.validperoids_hour_map.find(camId);

                                    if (got_id_func_capIter ==
globalJsonConfObjNotApplied.validperoids_hour_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
validperoids_hour_map,camId:" <<camId<<"new to validperoids_hour_map";
                                        std::unordered_map<std::string,
std::vector<Polygon>> mapp = { {funcname , validPeriods_hour} };

 globalJsonConfObjNotApplied.validperoids_hour_map.insert(std::pair<int,
std::unordered_map<std::string, std::vector<Polygon>>  >( camId, std::move(mapp)
) );
                                    }
                                    else
```

```
                                    {
                                        std::unordered_map<std::string,
std::vector<Polygon>>& conf_map =got_id_func_capIter->second;
                                        std::unordered_map<std::string,
std::vector<Polygon>>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, std::vector<Polygon> >
(funcname,validPeriods_hour));
                                        }
                                        else
                                        {
                                            got_id_func_cap2->second =
validPeriods_hour;
                                        }
                                    }

                                    ANNIWOLOG(INFO) <<
"globalJsonConfObjNotApplied.validperoids_hour_map.size():"
<<globalJsonConfObjNotApplied.validperoids_hour_map.size();



 std::unordered_map<int,std::unordered_map<std::string,
std::vector<std::string>>  >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.validperoids_week_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.validperoids_week_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
validperoids_week_map,camId:" <<camId<<"new to validperoids_week_map";
                                        std::unordered_map<std::string,
std::vector<std::string>> mapp = { {funcname , validPeriods_week} };

 globalJsonConfObjNotApplied.validperoids_week_map.insert(std::pair<int,
std::unordered_map<std::string, std::vector<std::string>>  >( camId,
std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
std::vector<std::string>>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
std::vector<std::string>>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, std::vector<std::string> >
(funcname,validPeriods_week));
                                        }
                                        else
                                        {
                                            got_id_func_cap2->second =
validPeriods_week;
                                        }
                                    }
```

```
                                }
                                ANNIWOLOG(INFO) <<
"globalJsonConfObjNotApplied.validperoids_week_map.size():"
<<globalJsonConfObjNotApplied.validperoids_week_map.size();



                                if(paramname == std::string("area"))
                                {
                                    Polygon polygonSafeArea;
                                    polygonSafeArea.clear();

                                    for (rapidjson::SizeType i = 0; i <
paramvalue.Size(); i++) // rapidjson uses SizeType instead of size_t.
                                    {
                                        assert(paramvalue[i].IsArray());

                                        cv::Point p;
                                        for (rapidjson::SizeType j = 0; j <
paramvalue[i].Size(); j++) // 2个int值
                                        {
                                            if(j == 0)
                                                p.x = paramvalue[i][j].GetInt();
                                            else
                                                p.y = paramvalue[i][j].GetInt();
                                        }
                                        polygonSafeArea.add(p);
                                    }

                                    if(polygonSafeArea.size() <= 1)
                                    {
                                        ANNIWOLOG(INFO) << "safeArea setting
less than 2 points,ignored.camId:" <<camId;
                                        polygonSafeArea.clear();
                                    }

                                    //valid area
                                    //camId,{func,Polygon}

 std::unordered_map<int,std::unordered_map<std::string, Polygon>  >::iterator
got_id_func_cap = globalJsonConfObjNotApplied.validArea_conf_map.find(camId);


                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.validArea_conf_map.end())
                                    {
                                        std::unordered_map<std::string, Polygon>
mapp = { {funcname , polygonSafeArea} };

 globalJsonConfObjNotApplied.validArea_conf_map.insert(std::pair<int,
std::unordered_map<std::string, Polygon>  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
Polygon>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
Polygon>::iterator got_id_func_cap2 = conf_map.find(funcname);
```

```
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, Polygon  >(funcname,polygonSafeArea));
                                        }
                                        else
                                        {
                                            got_id_func_cap2->second =
polygonSafeArea;
                                        }
                                    }
                                }

                            }
                            else if(tpValue == rapidjson::Type::kNumberType)
                            {
                                ANNIWOLOG(INFO) <<"paramvalue:"
<<paramvalue.GetInt()<<std::endl;
                                if(paramname == std::string("interval"))
                                {
                                    //interval
                                    //camId,{func,{interval,intervalCnter}}

 std::unordered_map<int,std::unordered_map<std::string, AnniwoTimeLog>
 >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.interval_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.interval_conf_map.end())
                                    {
                                        //随机开始时间，错开间隔到期时间
                                        int
configvalue=paramvalue.GetInt()*1000;

                                        int rdint =
randIntWithinScale(configvalue);
                                        ANNIWOLOG(INFO) << "new to
interval_conf_map:funcname:"<<funcname<<" camId:" <<camId<<"configvalue:"
<<configvalue<< "rdint:"<<rdint;

                                        std::chrono::milliseconds one_second(
rdint ); //应该用ms
                                        std::chrono::steady_clock::time_point
next_minute = std::chrono::steady_clock::now() + one_second;


                                        std::unordered_map<std::string,
AnniwoTimeLog> mapp = { {funcname , {paramvalue.GetInt()*1000.0,next_minute}} };

 globalJsonConfObjNotApplied.interval_conf_map.insert(std::pair<int,
std::unordered_map<std::string, AnniwoTimeLog>  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
AnniwoTimeLog>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
AnniwoTimeLog>::iterator got_id_func_cap2 = conf_map.find(funcname);
```

```
                                    if (got_id_func_cap2 == conf_map.end())
                                    {

                                        //随机开始时间，错开间隔到期时间
                                        int
configvalue=paramvalue.GetInt()*1000;
                                        int rdint =
randIntWithinScale(configvalue);
                                        ANNIWOLOG(INFO) <<
"interval_conf_map:funcname:"<<funcname<<" camId:" <<camId<<"configvalue:"
<<configvalue<< "rdint:"<<rdint;

                                        std::chrono::milliseconds
one_second( rdint ); //ms
                                        std::chrono::steady_clock::time_point
 next_minute = std::chrono::steady_clock::now() + one_second;


 conf_map.insert(std::pair<std::string, AnniwoTimeLog  >(funcname,
{paramvalue.GetInt()*1000.0,next_minute}));
                                    }
                                    else
                                    {
                                        AnniwoTimeLog& vecValues =
 got_id_func_cap2->second ;
                                        vecValues.configInterval=
paramvalue.GetInt()*1000.0;

                                        //随机开始时间，错开间隔到期时间
                                        int
configvalue=paramvalue.GetInt()*1000;
                                        int rdint =
randIntWithinScale(configvalue);
                                        ANNIWOLOG(INFO) << "new to
interval_conf_map:funcname:"<<funcname<<" camId:" <<camId<<"configvalue:"
<<configvalue<< "rdint:"<<rdint;

                                        std::chrono::milliseconds
one_second( rdint ); //应该用ms
                                        std::chrono::steady_clock::time_point
 next_minute = std::chrono::steady_clock::now() + one_second;

                                        vecValues.lastTP = next_minute;
                                    }
                                }

                            }

                            if(paramname == std::string("stay"))
                            {
                                //stay
                                //camId,{func,<isMotionless,stay>}

 std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>  >::iterator
got_id_func_cap = globalJsonConfObjNotApplied.stay_conf_map.find(camId);

                                if (got_id_func_cap ==
globalJsonConfObjNotApplied.stay_conf_map.end())
```

```
                                    {
                                        ANNIWOLOG(INFO) << "not found in
stay_conf_map,camId:" <<camId<<"new to stay_conf_map";
                                        //{func,<isMotionless,stay>}
                                        std::unordered_map<std::string,
AnniwoStay> mapp = { {funcname , {false, paramvalue.GetInt()}} };

 globalJsonConfObjNotApplied.stay_conf_map.insert(std::pair<int,
std::unordered_map<std::string, AnniwoStay >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
AnniwoStay>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
AnniwoStay>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, AnniwoStay  >(funcname,{false,
paramvalue.GetInt()}));
                                        }
                                        else
                                        {
                                            int& curValue =  got_id_func_cap2-
>second.staySec ;

                                            curValue = paramvalue.GetInt();
                                        }
                                    }

                                }

                                if(paramname == std::string("motionless"))
                                {
                                    //stay isMotionless
                                    //camId,{func,<isMotionless,stay>}

 std::unordered_map<int,std::unordered_map<std::string, AnniwoStay>  >::iterator
got_id_func_cap = globalJsonConfObjNotApplied.stay_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.stay_conf_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "isMotionless:not
found in stay_conf_map,camId:" <<camId<<"new to stay_conf_map";
                                        //{func,<isMotionless,stay>}
                                        std::unordered_map<std::string,
AnniwoStay> mapp = { {funcname , {paramvalue.GetBool(), 1}} };

 globalJsonConfObjNotApplied.stay_conf_map.insert(std::pair<int,
std::unordered_map<std::string, AnniwoStay >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
AnniwoStay>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
AnniwoStay>::iterator got_id_func_cap2 = conf_map.find(funcname);
```

```
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, AnniwoStay  >(funcname,
{paramvalue.GetBool(), 1}));
                                        }
                                        else
                                        {
                                            bool& curValue =  got_id_func_cap2-
>second.isMotionless ;
                                            curValue = paramvalue.GetBool();
                                        }
                                    }

                                }
                            }
                            else if(tpValue == rapidjson::Type::kStringType)
                            {
                                ANNIWOLOG(INFO) <<"paramvalue:"
<<paramvalue.GetString()<<std::endl;
                                if(paramname == std::string("taskId"))
                                {
                                    //camId,{func,strtaskId}

 std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.taskid_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.taskid_conf_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
taskid_conf_map,camId:" <<camId<<"new to taskid_conf_map";
                                        std::unordered_map<std::string,
std::string> mapp = { {funcname , paramvalue.GetString()} };

 globalJsonConfObjNotApplied.taskid_conf_map.insert(std::pair<int,
std::unordered_map<std::string, std::string >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
std::string>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
std::string>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, std::string  >
(funcname,paramvalue.GetString()));
                                        }
                                        else
                                        {
                                            std::string& curValue =
 got_id_func_cap2->second ;
                                            curValue = paramvalue.GetString();
                                        }
```

```
                                    }

                                }

                                if(paramname == std::string("dataset"))
                                {
                                    //camId,{func,strtaskId}

 std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.facedatasetpath_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.facedatasetpath_conf_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
facedatasetpath_conf_map,camId:" <<camId<<"new to facedatasetpath_conf_map";
                                        std::unordered_map<std::string,
std::string> mapp = { {funcname , paramvalue.GetString()} };

 globalJsonConfObjNotApplied.facedatasetpath_conf_map.insert(std::pair<int,
std::unordered_map<std::string, std::string >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
std::string>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
std::string>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, std::string  >
(funcname,paramvalue.GetString()));
                                        }
                                        else
                                        {
                                            std::string& curValue =
 got_id_func_cap2->second ;

                                            curValue = paramvalue.GetString();
                                        }
                                    }

                                }

                                if(paramname == std::string("eventUrl"))
                                {
                                    //camId,{func,strtaskId}

 std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.eventUrl_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.eventUrl_conf_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
eventUrl_conf_map,camId:" <<camId<<"new to eventUrl_conf_map";
```

```
                                        std::unordered_map<std::string,
std::string> mapp = { {funcname , paramvalue.GetString()} };

 globalJsonConfObjNotApplied.eventUrl_conf_map.insert(std::pair<int,
std::unordered_map<std::string, std::string >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
std::string>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
std::string>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {

 conf_map.insert(std::pair<std::string, std::string  >
(funcname,paramvalue.GetString()));
                                        }
                                        else
                                        {
                                            std::string& curValue =
 got_id_func_cap2->second ;

                                            curValue = paramvalue.GetString();
                                        }
                                    }

                                }

                                if(paramname == std::string("startCondition"))
                                {
                                    //This is for absence function's
startCondition
                                    //camId,{func,startCondition}

 std::unordered_map<int,std::unordered_map<std::string, std::string>
 >::iterator got_id_func_cap =
globalJsonConfObjNotApplied.absenceStartConition_conf_map.find(camId);

                                    if (got_id_func_cap ==
globalJsonConfObjNotApplied.absenceStartConition_conf_map.end())
                                    {
                                        ANNIWOLOG(INFO) << "not found in
absenceStartConition_conf_map,camId:" <<camId<<"new to
absenceStartConition_conf_map";
                                        std::unordered_map<std::string,
std::string> mapp = { {funcname , paramvalue.GetString()} };

 globalJsonConfObjNotApplied.absenceStartConition_conf_map.insert(std::pair<int,
std::unordered_map<std::string, std::string >  >( camId, std::move(mapp) ) );
                                    }
                                    else
                                    {
                                        std::unordered_map<std::string,
std::string>& conf_map =got_id_func_cap->second;
                                        std::unordered_map<std::string,
std::string>::iterator got_id_func_cap2 = conf_map.find(funcname);
                                        if (got_id_func_cap2 == conf_map.end())
                                        {
```

```
 conf_map.insert(std::pair<std::string, std::string  >
(funcname,paramvalue.GetString()));
                                        }
                                        else
                                        {
                                            std::string& curValue =
 got_id_func_cap2->second ;
                                            curValue = paramvalue.GetString();
                                        }
                                    }

                                }




                            }
                            else
                            {
                                ANNIWOLOG(INFO) <<"paramvalue is UNKOWN type"
<<std::endl;

                            }

                        }
                    }


                }


            }
        }


    uniqueLock.unlock();


    ANNIWOLOG(INFO) << "read
json:globalJsonConfObjNotApplied.validperoids_week_map.size():"
<<globalJsonConfObjNotApplied.validperoids_week_map.size();
    ANNIWOLOG(INFO) << "read
json:globalJsonConfObjNotApplied.validArea_conf_map.size():"
<<globalJsonConfObjNotApplied.validArea_conf_map.size();

    //std::unordered_map<int,std::unordered_map<std::string, Polygon>  >
    for(auto& itempair:globalJsonConfObjNotApplied.validArea_conf_map)
    {
        // cout<<kv.first<<kv.second<<endl;
        int camId = itempair.first;
        auto& funcAreaMap = itempair.second;

        for(auto& funcAreaItem:funcAreaMap)
        {
            auto& funcname = funcAreaItem.first;
            auto& polygonSafeArea = funcAreaItem.second;
```

```
            std::stringstream buffer;
            for(int i=0; i < polygonSafeArea.size(); i++)
            {
                const cv::Point& pt=polygonSafeArea.pt[i];
                buffer << pt.x<<","<<pt.y<<" ";
            }
            std::string text(buffer.str());

            ANNIWOLOG(INFO) << "read json:validArea_conf_map,camId:" <<camId<<"
func:"<<funcname<<" validArea:"<<text;

        }

    }




}


int main(int argc, char *argv[])
{

    //////////////Init g3 log custom sink for mine,此处参数未使
用//////////////////
    const std::string path_to_log_file = "../";
    const std::string log_file = "";

    std::unique_ptr<g3::LogWorker> logworker {g3::LogWorker::createLogWorker()};


    // auto handle = logworker->addDefaultLogger(log_file, path_to_log_file);
    // g3::initializeLogging(logworker.get());


   auto sinkHandle = logworker->addSink(std::make_unique<LogRotate>
(log_file,path_to_log_file),
                                        &LogRotate::save);

   // initialize the logger before it can receive LOG calls
   initializeLogging(logworker.get());


 ////////////////////////////////////////////////////////////////////////////
//////
    srand((unsigned)time(NULL));//初始化种子

    ////////////////////////////////////////

    //read the ini file
    mINI::INIFile file("../config.ini");
    mINI::INIStructure ini;

    if(!  file.read(ini))
    {
        ANNIWOLOG(INFO) << "Error no ini file!" ;
```

```
        return -1;
    }

    if(ini.has("config"))
    {
        if(ini["config"].has("inusefunctions"))
        {
            std::string strvalue = ini.get("config").get("inusefunctions");
            std::vector<std::string> strVector = stringSplit(strvalue, ' ');
            for (auto f : strVector)
            {
                std::unordered_set<std::string>::const_iterator
got_conf_functions = all_conf_functions.find(f);

                if (got_conf_functions == all_conf_functions.end())
                {
                    ANNIWOLOG(INFO) <<"ini f is not valid.ignored!"<<f;
                    continue;
                }
                else
                {
                    globalINICONFObj.in_use_conf_functions.emplace(f);
                }
            }

        }else
        {
            ANNIWOLOG(INFO) << "Error no inusefunctions setting in ini file!" ;
            ANNIWOCHECK(false);
            exit(-1);

        }

        globalINICONFObj.domain_config=ANNIWO_DOMANI_LIANGKU;
        if(ini["config"].has("domain"))
        {
            std::string strvalue = ini.get("config").get("domain");
            std::vector<std::string> strVector = stringSplit(strvalue, ' ');
            if(strVector[0]=="jiayouzhan")
            {
                globalINICONFObj.domain_config=ANNIWO_DOMANI_JIAYOUZHAN;
                ANNIWOLOG(INFO) <<"domain_config:  "<<"jiayouzhan"<<std::endl;

            }else
            {
                ANNIWOLOG(INFO) <<"domain_config:  "<<"jiayouzhan"<<std::endl;
            }

        }

    }
    if(ini.has("path"))
    {

        //创建根图片输出目录
        std::string tmpPath=std::string("/var/anniwo/");

        struct stat st = {0};
```

```
        //创建目录
        if (stat(tmpPath.c_str(), &st) == -1) {
            mkdir(tmpPath.c_str(), 0700);
        }
        //检查是否创建成功
        if (stat(tmpPath.c_str(), &st) == -1) {
            ANNIWOLOG(INFO) <<"Unable to create:"<<tmpPath<<std::endl;
            ANNIWOCHECK(false);
            exit(-1);
        }

        ////////////////////////////////////

        if(ini["path"].has("onlineCollectionPath"))
        {
            std::string strvalue = ini.get("path").get("onlineCollectionPath");
            std::vector<std::string> strVector = stringSplit(strvalue, ' ');

            //todo
            globalINICONFObj.onlineCollectionPath=strVector[0];
            if(globalINICONFObj.onlineCollectionPath.length() > 5)
            {
                //标识目录
                if(globalINICONFObj.onlineCollectionPath.back() != '/')
                {
                    globalINICONFObj.onlineCollectionPath.push_back('/');
                }


                struct stat st = {0};
                //创建目录
                if (stat(globalINICONFObj.onlineCollectionPath.c_str(), &st) ==
-1) {
                    mkdir(globalINICONFObj.onlineCollectionPath.c_str(), 0700);
                }
                //检查是否创建成功
                if (stat(globalINICONFObj.onlineCollectionPath.c_str(), &st) ==
-1) {
                    ANNIWOLOG(INFO) <<"Unable to create:"
<<globalINICONFObj.onlineCollectionPath<<std::endl;
                    ANNIWOCHECK(false);
                    exit(-1);
                }



            }

        }
    }

    //阈值配置
    if(ini.has("theshold"))
    {
        //todo:遍历key,value

        if(ini["theshold"].has("helmet"))
        {
```

```
            std::string strvalue = ini.get("theshold").get("helmet");

            std::istringstream isb_str(strvalue);
            float fvalue = 1.0;
            isb_str >> fvalue;

            std::string f("helmet");
            std::unordered_map<std::string, float  >::iterator  got_theshold=
globalINICONFObj.thesholdsetting.find(f);

            if (got_theshold == globalINICONFObj.thesholdsetting.end())
            {
                ANNIWOLOG(INFO) <<"ini:f is not thesholdsetting....make one.f: "
<<f<<" value:"<<fvalue ;
                std::pair<std::string, float> settingpair(f,fvalue);

                globalINICONFObj.thesholdsetting.insert(std::move(settingpair));
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:f is already in thesholdsetting....CHECK
your ini, ignored. "<<f;
            }
        }

        if(ini["theshold"].has("jyzxunjian_jiayouji"))
        {
            std::string strvalue =
ini.get("theshold").get("jyzxunjian_jiayouji");

            std::istringstream isb_str(strvalue);
            float fvalue = 1.0;
            isb_str >> fvalue;

            std::string f("jyzxunjian_jiayouji");
            std::unordered_map<std::string, float  >::iterator  got_theshold=
globalINICONFObj.thesholdsetting.find(f);

            if (got_theshold == globalINICONFObj.thesholdsetting.end())
            {
                ANNIWOLOG(INFO) <<"ini:f is not thesholdsetting....make one.f: "
<<f<<" value:"<<fvalue ;
                std::pair<std::string, float> settingpair(f,fvalue);

                globalINICONFObj.thesholdsetting.insert(std::move(settingpair));
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:f is already in thesholdsetting....CHECK
your ini, ignored. "<<f;
            }
        }

        if(ini["theshold"].has("jyzxunjian_rkj"))
        {
            std::string strvalue = ini.get("theshold").get("jyzxunjian_rkj");

            std::istringstream isb_str(strvalue);
```

```
            float fvalue = 1.0;
            isb_str >> fvalue;

            std::string f("jyzxunjian_rkj");
            std::unordered_map<std::string, float  >::iterator  got_theshold=
globalINICONFObj.thesholdsetting.find(f);

            if (got_theshold == globalINICONFObj.thesholdsetting.end())
            {
                ANNIWOLOG(INFO) <<"ini:f is not thesholdsetting....make one.f: "
<<f<<" value:"<<fvalue ;
                std::pair<std::string, float> settingpair(f,fvalue);

                globalINICONFObj.thesholdsetting.insert(std::move(settingpair));
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:f is already in thesholdsetting....CHECK
your ini, ignored. "<<f;
            }
        }


    }


    //检测线程数目阈值配置
    if(ini.has("threads"))
    {
        //todo:遍历key,value
        globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE=8;

        if(ini["threads"].has("personbaseInstance"))
        {
            std::string strvalue = ini.get("threads").get("personbaseInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.personbaseInstance should
bigger than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_FIRE=8;

        if(ini["threads"].has("fireInstance"))
        {
            std::string strvalue = ini.get("threads").get("fireInstance");
```

```
            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_FIRE=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.fireInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_WINDOW=8;
        if(ini["threads"].has("windowInstance"))
        {
            std::string strvalue = ini.get("threads").get("windowInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_WINDOW=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.windowInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_THREAD_UPTRUCK=1;
        if(ini["threads"].has("uptruck"))
        {
            std::string strvalue = ini.get("threads").get("uptruck");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_UPTRUCK=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.uptruck should bigger than
1...CHECK your ini, ignored. ";
            }
        }
```

```
        globalINICONFObj.ANNIWO_NUM_THREAD_HELMET=8;
        if(ini["threads"].has("helmetInstance"))
        {
            std::string strvalue = ini.get("threads").get("helmetInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_HELMET=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.helmetInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_CONVERY=8;
        if(ini["threads"].has("converyInstance"))
        {
            std::string strvalue = ini.get("threads").get("converyInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_CONVERY=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.converyInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_SMOKEPHONE=8;

        if(ini["threads"].has("smokephoneInstance"))
        {
            std::string strvalue = ini.get("threads").get("smokephoneInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_SMOKEPHONE=ivalue;
```

```
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.smokephoneInstance should
bigger than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_SAFEAREA=8;
        if(ini["threads"].has("safeErea"))
        {
            std::string strvalue = ini.get("threads").get("safeErea");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_SAFEAREA=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.safeErea should bigger than
1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_THREAD_XUNJIAN=8;
        if(ini["threads"].has("xunjian"))
        {
            std::string strvalue = ini.get("threads").get("xunjian");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_XUNJIAN=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.xunjian should bigger than
1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_ZXY=8;
        if(ini["threads"].has("zhuangxieyou"))
        {
            std::string strvalue = ini.get("threads").get("zhuangxieyou");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
```

```
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_ZXY=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.zhuangxieyou should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_THREAD_CHEPAI=8;
        if(ini["threads"].has("chepai"))
        {
            std::string strvalue = ini.get("threads").get("chepai");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_CHEPAI=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.chepai should bigger than
1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_THREAD_MASK=8;
        if(ini["threads"].has("mask"))
        {
            std::string strvalue = ini.get("threads").get("mask");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_THREAD_MASK=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.mask should bigger than
1...CHECK your ini, ignored. ";
            }
        }

    }else
```

```
    {
        globalINICONFObj.ANNIWO_NUM_THREAD_PERSONBASE=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_FIRE=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_WINDOW=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_HELMET=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_CONVERY=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_SMOKEPHONE=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_SAFEAREA=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_XUNJIAN=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_ZXY=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_MASK=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_CHEPAI=8;
        globalINICONFObj.ANNIWO_NUM_THREAD_UPTRUCK=1;

    }


    //检测GPU实例(context)占用数目阈值配置
    if(ini.has("gpuoccupy"))
    {
        //todo:遍历key,value
        globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE=1;

        if(ini["gpuoccupy"].has("personbaseInstance"))
        {
            std::string strvalue =
ini.get("gpuoccupy").get("personbaseInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.personbaseInstance should
bigger than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_FIRE=1;

        if(ini["gpuoccupy"].has("fireInstance"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("fireInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_FIRE=ivalue;
```

```
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.fireInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_WINDOW=1;
        if(ini["gpuoccupy"].has("windowInstance"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("windowInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_WINDOW=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.windowInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_INSTANCE_UPTRUCK=1;
        if(ini["gpuoccupy"].has("uptruck"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("uptruck");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_UPTRUCK=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.uptruck should bigger than
1...CHECK your ini, ignored. ";
            }
        }




        globalINICONFObj.ANNIWO_NUM_INSTANCE_HELMET=1;
        if(ini["gpuoccupy"].has("helmetInstance"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("helmetInstance");
```

```
            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_HELMET=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.helmetInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_CONVERY=1;
        if(ini["gpuoccupy"].has("converyInstance"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("converyInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_CONVERY=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.converyInstance should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_SMOKEPHONE=1;

        if(ini["gpuoccupy"].has("smokephoneInstance"))
        {
            std::string strvalue =
ini.get("gpuoccupy").get("smokephoneInstance");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_SMOKEPHONE=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.smokephoneInstance should
bigger than 1...CHECK your ini, ignored. ";
```

```
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_SAFEAREA=1;
        if(ini["gpuoccupy"].has("safeErea"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("safeErea");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_SAFEAREA=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.safeErea should bigger than
1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_INSTANCE_XUNJIAN=1;
        if(ini["gpuoccupy"].has("xunjian"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("xunjian");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_XUNJIAN=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.xunjian should bigger than
1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY=1;
        if(ini["gpuoccupy"].has("zhuangxieyou"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("zhuangxieyou");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
```

```
                globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.zhuangxieyou should bigger
than 1...CHECK your ini, ignored. ";
            }
        }

        globalINICONFObj.ANNIWO_NUM_INSTANCE_CHEPAI=1;
        if(ini["gpuoccupy"].has("chepai"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("chepai");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_CHEPAI=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.chepai should bigger than
1...CHECK your ini, ignored. ";
            }
        }


        globalINICONFObj.ANNIWO_NUM_INSTANCE_MASK=1;
        if(ini["gpuoccupy"].has("mask"))
        {
            std::string strvalue = ini.get("gpuoccupy").get("mask");

            std::istringstream isb_str(strvalue);
            int ivalue = 0;
            isb_str >> ivalue;


            if (ivalue > 0)
            {
                globalINICONFObj.ANNIWO_NUM_INSTANCE_MASK=ivalue;
            }else
            {
                //duplicated setting in config!
                ANNIWOLOG(INFO) <<"ini:performance.mask should bigger than
1...CHECK your ini, ignored. ";
            }
        }

    }else
    {

        globalINICONFObj.ANNIWO_NUM_INSTANCE_PERSONBASE=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_FIRE=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_WINDOW=1;
```

```
        globalINICONFObj.ANNIWO_NUM_INSTANCE_HELMET=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_CONVERY=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_SMOKEPHONE=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_SAFEAREA=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_XUNJIAN=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_ZXY=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_MASK=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_CHEPAI=1;
        globalINICONFObj.ANNIWO_NUM_INSTANCE_UPTRUCK=1;


    }



    if(globalINICONFObj.in_use_conf_functions.size() > 0)
    {
        ANNIWOLOG(INFO) <<"in use functions:";

        std::string tmpPath;

        //创建根图片输出目录
        tmpPath=std::string("/var/anniwo/images/");

        struct stat st = {0};
        //创建目录
        if (stat(tmpPath.c_str(), &st) == -1) {
            mkdir(tmpPath.c_str(), 0700);
        }
        //检查是否创建成功
        if (stat(tmpPath.c_str(), &st) == -1) {
            ANNIWOLOG(INFO) <<"Unable to create:"<<tmpPath<<std::endl;
            ANNIWOCHECK(false);
            exit(-1);
        }


        for (auto& item:globalINICONFObj.in_use_conf_functions)
        {
            ANNIWOLOG(INFO) <<item<<" ";

            //创建所有图片输出目录
            if(item == "fire")
            {
                tmpPath=std::string("/var/anniwo/images/")+"firesmog";
            }else
            {
                tmpPath="/var/anniwo/images/"+item;
            }

            struct stat st = {0};
            //创建目录
            if (stat(tmpPath.c_str(), &st) == -1) {
                mkdir(tmpPath.c_str(), 0700);
            }
            //检查是否创建成功
            if (stat(tmpPath.c_str(), &st) == -1) {
```

```
                ANNIWOLOG(INFO) <<"Unable to create:"<<tmpPath<<std::endl;
                ANNIWOCHECK(false);
                exit(-1);
            }

        }
        ANNIWOLOG(INFO) <<"  "<<std::endl;

    }
    else
    {
        ANNIWOLOG(INFO) <<"No in use function,ignore!";
        ANNIWOCHECK(false);
    }


    ANNIWOLOG(INFO) << "server start! version:" << VERSION ;


    crow::App<ExampleMiddleware> app;

    crow::logger::setLogLevel(crow::LogLevel::DEBUG);

    app.get_middleware<ExampleMiddleware>().setMessage("hello anniwo");

    // simple json response
    // To see it in action enter {ip}:18080/json
    CROW_ROUTE(app, "/engineStart")
    ([]{

        crow::json::wvalue x;
        x["message"] = "Now (re)start engine!";

        engineStart();

        return x;
    });

        // A simpler way for json example:
    //      * curl -d '{"a":1,"b":2}' {ip}:18080/add_json
    CROW_ROUTE(app, "/managerMent")
        .methods("POST"_method)
    ([](const crow::request& req){



        managerMent(req.body);

        std::ostringstream os;
        os << "Configuration accept!";
        return crow::response{os.str()};
    });

    app.port(5000)
        .multithreaded()
        .run();
}
```