函数调用关系

create_data.py

- nuscenes_data_prep (basic infos, 2d annotations, gt database)
    - converter.create_nuscenes_infos
        - splits.val
        - get_available_scenes(nusc)
        - _fill_trainval_infos
            - lidarpath, boxes = nusc.get_sample_data(lidar_token) # 点云的信息基本齐了
            - obtain_sensor2top()
- update_pkl_infos
- create_groundtruth_database

lidarpath, boxes

lidarpath: LIDAR_TOP一帧点云

box

- label
- score
- xyz
- wlh
- rot axis
- ang(degrees)
- ang(rad)
- vel (vx, vy, vz)
- name
- token

rotation & translation

外参 RT矩阵

```
l2e_t = info['lidar2ego_translation'] # 3
e2g_r = info['ego2global_rotation']   # 4
e2g_t = info['ego2global_translation']
l2e_r_mat = Quaternion(l2e_r).rotation_matrix  # 3*3
e2g_r_mat = Quaternion(e2g_r).rotation_matrix
```

obtain_sensor2top function

Obtain the info with RT matric from general sensor to Top LiDAR.

lidar -> ego -> global

ego vehicle

Vehicle coordinate system

The term ego refers to the vehicle that contains the sensors that perceive the environment around the vehicle.

```
# obtain the RT from sensor to Top LiDAR
# sweep->ego->global->ego'->lidar
```

numpy 中@表示 矩阵乘法

```
def obtain_sensor2top(nusc,
                      sensor_token,
                      l2e_t,
                      l2e_r_mat,
                      e2g_t,
                      e2g_r_mat,
                      sensor_type='lidar'):
    """Obtain the info with RT matric from general sensor to Top LiDAR.

    Args:
        nusc (class): Dataset class in the nuScenes dataset.
        sensor_token (str): Sample data token corresponding to the
            specific sensor type.
        l2e_t (np.ndarray): Translation from lidar to ego in shape (1, 3).
        l2e_r_mat (np.ndarray): Rotation matrix from lidar to ego
            in shape (3, 3).
        e2g_t (np.ndarray): Translation from ego to global in shape (1, 3).
        e2g_r_mat (np.ndarray): Rotation matrix from ego to global
            in shape (3, 3).
        sensor_type (str, optional): Sensor to calibrate. Default: 'lidar'.

    Returns:
        sweep (dict): Sweep information after transformation.
    """
    sd_rec = nusc.get('sample_data', sensor_token)
    cs_record = nusc.get('calibrated_sensor',
                         sd_rec['calibrated_sensor_token'])
    pose_record = nusc.get('ego_pose', sd_rec['ego_pose_token'])
    data_path = str(nusc.get_sample_data_path(sd_rec['token']))
    if os.getcwd() in data_path:  # path from lyftdataset is absolute path
        data_path = data_path.split(f'{os.getcwd()}/')[-1]  # relative path
    sweep = {
        'data_path': data_path,
        'type': sensor_type,
        'sample_data_token': sd_rec['token'],
        'sensor2ego_translation': cs_record['translation'],
        'sensor2ego_rotation': cs_record['rotation'],
        'ego2global_translation': pose_record['translation'],
        'ego2global_rotation': pose_record['rotation'],
```

```
    'timestamp': sd_rec['timestamp']
}
l2e_r_s = sweep['sensor2ego_rotation']
l2e_t_s = sweep['sensor2ego_translation']
e2g_r_s = sweep['ego2global_rotation']
e2g_t_s = sweep['ego2global_translation']

# obtain the RT from sensor to Top LiDAR
# sweep->ego->global->ego'->lidar
l2e_r_s_mat = Quaternion(l2e_r_s).rotation_matrix
e2g_r_s_mat = Quaternion(e2g_r_s).rotation_matrix
R = (l2e_r_s_mat.T @ e2g_r_s_mat.T) @ (
    np.linalg.inv(e2g_r_mat).T @ np.linalg.inv(l2e_r_mat).T)
T = (l2e_t_s @ e2g_r_s_mat.T + e2g_t_s) @ (
    np.linalg.inv(e2g_r_mat).T @ np.linalg.inv(l2e_r_mat).T)
T -= e2g_t @ (np.linalg.inv(e2g_r_mat).T @ np.linalg.inv(l2e_r_mat).T
              ) + l2e_t @ np.linalg.inv(l2e_r_mat).T
sweep['sensor2lidar_rotation'] = R.T  # points @ R.T + T
sweep['sensor2lidar_translation'] = T
return sweep
```