大地图相关

大地图城点逻辑里几个比较重要的类：

"world.get"协议，获取大地图上一定范围内的城点数据（即WorldCityInfo）

WorldUpdateController，主要控制大地图城点数据的刷新和解析。

WorldCityInfo，大地图城点数据的基类。

WorldController，大地图控制类。

WorldMapView2，大地图的根节点，主要控制地图所有节点的刷新。

WorldMapView2UI，大地图的ui控制类，主要是控制点击城点选中后的环形按钮。

World3DMapDynamicTiledMap，控制移动大地图后对地块的刷新，以及发送world.get协议刷新城点数据。

World3DMapView2ViewPort，移动大地图的手势控制类，通过手势监听事件来移动、缩放和点击地图。

"world.get"：

协议里几个参数的意义

1. Rect，刷新地图的范围。范围是个矩形
2. Type，只有0和1两个值，为0时，服务器会下发Rect矩形内所有格子的数据，为1时，服务器只会发rect.min这一个点的数据。
3. serverId，对应的服务器id。

WorldUpdateController：

主要是对world.get协议发送接口的封装，以及收到world.get协议

后对收到的数据进行解析。整个数据的解析都是在线程种执行的，最后将解析好的数据覆

盖到本地的时候才转到主线程。

1. doWorldGet()：是world.get协议发送的接口，所有获取地图数据的逻辑都应从该接口获取
2. do\_one\_process()：是对world.get协议回复的地图数据转换成json的方法，如果是二级地图数据，跳转到ParseCityInfoSecond()方法里解析，如果是一级地图数据，跳转到updateCitysByParam()方法解析。
3. ParseCityInfoSecond()：二级地图城点数据解析。原本的设定是服务器仅发送二级地图展示所需要的数据就可以了，但服务器不好做，所以目前服务器发送的仍旧是一级地图里城点的所有数据，只是在解析的时候只解析了二级地图展示所需要的一些数据，同时对一些不应该展示的某些城点数据进行移除。
4. updateCitysByParam(): 一级地图城点数据解析，要注意的是，如果一个城点占多个格子，比如玩家城市，占四个格子，那边这四个格子都会附上解析出来的数据。

WorldCityInfo：

地图城点数据的基类，储存着城点里的所有信息（不包括城战城市和血岩矿）

WorldController：

大地图的全局控制类。

WorldMapView2：

大地图的根节点控制类

1. 一级地图刷新逻辑：在UpdateFirstMap()方法里获取屏幕范围内（实际上会超出屏幕范围一点点）所有的城点的格子坐标缓存到needUpdateKeys列表里，同时遍历同范围里已经创建好的城点，筛选出已被移除了数据但仍未被移除的城点节点，将其移除。然后在UpdateCitys()方法里刷新needUpdateKeys列表里的城点数据（并不是一次全部刷新完，UpdateCitys()是每帧执行，每次执行的时候只刷新10个城点数据）。最后在createCity()方法里刷新每个城点，刷新逻辑是先获取worldCityInfo. m\_jsonToCrc数据跟之前以创建好的城点里缓存的m\_jsonToCrc数据比对，如有变化，将旧的城点删除，然后重新创建新的城点。创建城点的唯一接口是WorldTileFactory. createWorldTile().
2. 二级地图刷新逻辑：与一级地图刷新逻辑类似，在UpdateSecondMap()方法里获取屏幕范围内的二级城点数据，然后UpdateSecondMapItem()方法里刷新二级地图城点。
3. isCreateCity()：一个城点可能会占多个格子，如玩家城市占四个格子，王座占25个格子等，而这些城点里所占的每个格子都会缓存城点数据（worldCityInfo）。在创建城点的时候通过该方法判断，只需要在worldCityInfo.getParentCityIndex()上的格子创建城点，其他格子直接跳出，无需创建。

WorldMapView2UI：

大地图ui控制类，目前这里面的功能基本上只有对城点的环形按钮的控制了。

1. openTilePanel()：每次点击地图城点时都会调用此方法，如果之前已经有环形按钮弹出，则关闭旧的环形按钮，否则通过getPopupViewByType()方法创建新的环形按钮。
2. ClearAllBaseTileInfo()：关闭环形按钮的方法，如果想要关闭环形按钮最好是调用此方法。
3. processDelayEvent()：WorldController.getInstance().openTargetIndex != -1 的时候，即在某次跳转到大地图上某一坐标后并且希望自动选中WorldController.getInstance().openTargetIndex这个城点打开环形按钮， processDelayEvent这个方法就是干这个的。
4. showTroopInfo()：选中行军队列，然后地图视角就会自动跟随行军队列
5. updateDirection()：刷新主界面的那个距离按钮。

World3DMapDynamicTiledMap：

大地图可移动的节点，大地图的移动和缩放都是通过对该节点的移动和缩放来控制的。

1. 大地图移动并不是移动摄像机，而是移动World3DMapDynamicTiledMap这个节点。
2. setPosition() 设置大地图移动的接口。
3. setScale() 设置大地图缩放的接口。
4. updateDynamicMap() 发送world.get协议刷新大地图数据。如果参数forceType = 1时，只刷新参数里的point这一个城点的数据，否则刷新以point为中心点，刷新屏幕范围内（实际上会超过屏幕范围一点）的城点数据。

注意的一些点：

每次刷新大地图发送world.get协议后，都会缓存下这次刷新的范围，即rangeRect，该范围会超出摄像机范围rangeOffset个标准格子范围大小，意思是，如果在一倍地图的情况下，刷新的范围会超过摄像机范围rangeOffset个格子，但如果地图缩放到了0.5倍，刷新的范围会超过摄像机范围rangeOffset \* 2个格子。玩家下次移动地图的时候，会对比上次刷新的范围，当地图移动到上次刷新范围的边界后，重新发送world.get协议刷新。

大地图目前每隔mapSecondUpdateInterval秒后都会发送world.get协议重新刷新地图。该逻辑在Update() 方法里控制

大地图上的格子编号坐标、格子实际坐标和屏幕坐标的相互转换工具类基本上都在这个类里面。这些工具类上都有相关的注释。

World3DMapView2ViewPort：HF3DMapViewport

大地图上的手势监听类

1. 地图的移动范围设置方法是HF3DMapViewport. setMoveRange()。
2. 手势对地图的移动和缩放都是通过HF3DMapViewport类里直接对m\_TargetNode节点（实际上就是World3DMapDynamicTiledMap节点）进行操作的。至于各种手势的监听都是通过调用I3DMapTouchDelegate接口来传递到外界的。

WorldCityInfoTable

解析城点数据时需要用到这个类里的一张映射表infokeys，

serverType：代表城点类型，

serverId： 代表服务器下发的key

clientId： 代表客户端本地缓存服务器下发的数据时对应的key。

valueType：代表该数据的类型，只在解析数据时用到，目前只有string，int，double，time四种

defaultValue: 如果服务器没有下发该key值所对应的数据时，客户端就会直接使用defaultValue这个值。

比如 现在需要解析一个玩家城市的城点json数据，解析时会先从映射表里取到所有

类型serverType为 -1 和(int)WorldCityType.CityTile 的WorldCityInfoKey数据，再根据json数据里的key值和WorldCityInfoKey数据里的serverId值进行比较，如果是一样的话，就代表json的value值需要以valueType类型来解析并且缓存到WorldCityInfo的datas\_字典里，同时，缓存的key值是clientId。如果在解析完毕后 客户端取了一个json里没有的值，则客户端就会直接使用defaultValue这个默认值。

因此如果有需求对某个城点需要新增数据或者需要增加新的城点类型时，都需要扩充这个infokeys映射表，否则无法解析新增加的地图数据。

要注意的几个点，大地图上的城点数据大部分都是从world.get协议下发下来的，然后再根据infokeys映射表进行解析缓存到WorldCityInfo里面取。但还有少量的城点数据是通过其他的协议（比如"world.get.detail"协议）下发并且会缓存到WorldCityInfo里面去。

这样就会有个问题，当某个城点里缓存了其他协议下发的一些数据后，下次再收到world.get协议会将城点数据WorldCityInfo删除，然后重新new一个新的WorldCityInfo并将协议下发的json数据解析到里面，这样就会导致通过 "world.get.detail"等其他协议获取的城点数据都会被覆盖消失。

现在该bug的修改逻辑是，在WorldCityInfoTable里新增加一张表replaceDataKey，每当收到world.get协议并生成新的WorldCityInfo后，会先查找旧的WorldCityInfo里是否缓存了replaceDataKey这个表里对应key值的数据，如果有，并且新的WorldCityInfo里没有这些数据，则将旧的WorldCityInfo里的这些数据复制到新的WorldCityInfo数据里，以保证通过非world.get协议获取的数据不会被覆盖。

城战城市和血岩矿，这两个类型的城点是后来新加的逻辑，World.get协议只会下发它们的坐标。其他的数据是通过其他协议获取的（但不会缓存到WorldCityInfo里）。

城战城市数据控制类是WorldCityController，血岩矿的数据控制类是WorldBloodShardController，这两个城点的数据均通过这两个个类获取。

新增加个城点类型的步骤：

1. 在WorldCityInfoTable的映射表里infokeys添加好新增城点的数据。
2. 在WorldTileFactory. createWorldTile() 方法里实现城点的创建。
3. 在WorldMapView2UI. getPopupViewByType() 方法里实现环形按钮的创建
4. 如果城点占多个格子，则需要在WorldController. WorldController() 方法里添加该城点占格子的数量，在WorldMapView2.isCreateCity() 方法里添加某个WorldCityInfo是否需要创建城点的判断。
5. 如果城点占的格子数量是偶数，则还需要在WorldController. isEvenNumGrid() 方法里添加该城点占地大小为偶数的判断。