ZW3D永久命名（下）

主讲人：翁亦旸

C O N T E N T S

旧命名有什么问题？

- 六宗罪

新永久命名对象

- Entity Ref
- Tracking Info
- Split & Merge Data

C O N T E N T S

EntRef基本规则

- Face Ref
- Shell Ref
- Edge Ref

新永久命名流程

- 匹配
- 获取追溯信息
- 解析追溯信息
- 命名层流程

说在前面

新的永久命名有一篇非常详细的文档用于介绍，链接在这：

Labeling - 平台架构设计部 - ZWiki (zwcax.com)

这是万字血书，大家一定要看。

接下来这篇ppt会大量使用其中的文字，如果有对概念存在疑惑可以随时举手提问打断。

旧命名有什么问题

旧命名问题

一、语义缺失

旧的命名是非常缺乏语义的，这来源于两个方面：label结构本身欠缺拓展性，以及内核无法将完整的信息传递到对象上。

结构问题：如先前ppt所述，每个label中只有一个字段用于记录其参考对象。这个字段既没能充分记录其参考信息，也没能标明其参考对象是什么内容。对于额外需要区分开的信息，只能使用位置码进行无意义的差异化。

> 比如在草图孔命令中，草图孔做出来的面既需要记录它是由哪一条线扫出的，也需要记录它是打在哪个位置的。这个结构旧无法将两个信息全部记录进来。

内核问题：先前的内核也没有记录对象的变化过程，以及来源。只有利用XRefTable去表示对象的分裂/合并过程，以及在特征命令中添加特殊代码将所需要的信息加入其中。这一整套机制都没有完善的流程进行管理。当然，这部分内容更多的是内核改动部分，我们就不在这篇文章里赘述了。

旧命名问题

二．结构冗余

旧的永久命名全部是使用int数组进行记录的，结构则是依照一套“规则式”的访问顺序进行访问。这个结构的拓展性与访问开销都很大。

三．聚合混乱

旧的永久命名是隶属于brep的，而最关键的聚合信息—特征，却没有被考量。

局部匹配尝试过使用快速回滚信息进行特征修改对象永久命名的聚合，但是这还远远不够。

聚合方式的混乱导致我们在需要永久命名时只能将所有的命名一视同仁全部取出。

四．分裂/合并辨别能力低下

旧的永久命名在分裂对象上的表现是将邻面信息添加在原有label之后，在合并对象上的表现则是将所有的被合并对象的label拼凑成为一个compound。

这里问题很多，但是最大的点则是在于compound这个结构上。

对于这种将所有对象组合在一起的行为然后根据相似条目数量进行评分的结构而言，他如果无法将所有的相关信息进行保存而是挑选部分内容，这个行为从严格意义上来讲是错误的。

在这种情况下，合并和分裂都没有办法有区分性的分辨出他们和原本父对象的关系。

旧命名问题

五．匹配标准不明确

旧的永久命名匹配规则是一个进行过无数次压缩的评分规则，其中有很多case by case 的比较方法（比如compound相似11%获得一个分数，相似34%获得一个分数等等P），最终变成一个系数，表明二者之间的相似度，通过累加这个相似度完成匹配。

这个匹配结果是非常难以预测的，不同的对象来到这里获得到的相同的结果可能是拥有完全不同的语义，比如最常见的分裂孙代面评分不如拓扑邻面。

六．匹配重复性过高

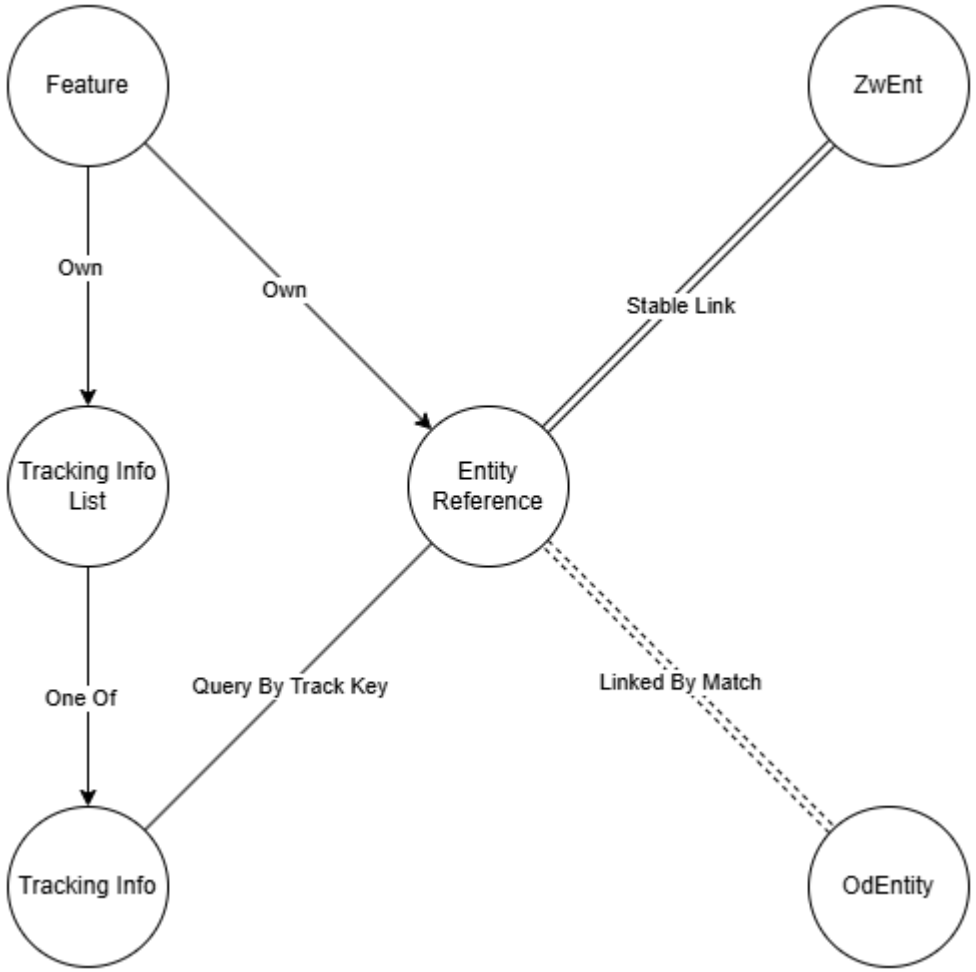
旧的永久命名系统，是记录一个名字，每一次在使用的时候去检索符合这个名字的对象。

这样的话，如果一个对象被反复引用了多次，那么在每一次使用到的时候都需要对他进行一次匹配，产生一次开销。

新的永久命名对象

新的永久命名对象

关系图



新的永久命名对象---Entity Ref

- 由于旧命名问题7重复匹配以及旧命名问题2聚合混乱，我们引入了Entity Reference机制。
- 从属于特征。关联一个底层对象
- 这个实体的对象关系和作用与字典是非常像的。
- 在Face ref和Shell ref中，我们使用xx_track_key字段来记录对应的tracking info在Feature tracking info list中的序号，以此完成二者间的关联。
- 关于为什么是需要使用track key而不是直接携带在ent ref对象身上，有两个原因：
  - 1，track key并非所有的ent ref都有，只有face ref和shell ref有，edge ref就没有。
  - 2，多个ref可能会关联同一份tracking info。比如分裂行为。
- 删除机制：Entity Reference不应当被自行删除。当它没有关联到任意一个OdEntity时，它会进入Sleep状态，而不是被删除。只有当特征被删除时，它所拥有的所有的Ent Ref才会被删除。

新的永久命名对象---Tracking Info

- 由于旧命名问题1，语义拓展性很差。以及问题3，结构冗余。基于这两点，我们添加了Tracking Info结构：FmfBaseTrackingInfo。

- Tracking Info的含义就是记录我们需要记录的语义，这个是由特征定义的。如下图SideFaceTrackingInfo，其中就将drive curve记录了进来。

```
class ZS_DB_API FmfProfileSideFaceTrackingInfo : public FmfCreatedTrackingInfo
{
    DB_OBJECT_DECLARE_DERIVED(FmfProfileSideFaceTrackingInfo, FmfCreatedTrackingInfo);
    DB_OBJECT_VERSION_DEFINE(1); // 3000

    DBOBJECT_MEMBER_FUNCTION_DECLARE;

public:
    FmfProfileSideFaceTrackingInfo() = default;
    ~FmfProfileSideFaceTrackingInfo() = default;

    virtual CompareResult compare(const ZwRef<FmfBaseTrackingInfo> target_tracking) const final;


private:
    OM_XN_MEMBER_PUBLIC(int, m_drive_curve); //own
    OM_XN_MEMBER_PUBLIC(int, m_pos_code);
};
```

新的永久命名对象---Tracking Info

- 关于匹配：
- 由于旧命名问题5，那么相应的，在遇到匹配行为时，匹配行为的相同，不同，相似度，是由自身的compare函数给出的，compare函数需要定义自己的每一个成员的比较结果，然后返回其相似性。

- 代码太长了，放不下，详见文档中TrackingInfo章节。

- 这里的compare是FmfSideFaceTrackingInfo的compare函数。逻辑为所在体不同就完全不同，在此前提下，邻面相同的越多相似度越高，如果所有的邻面都相同则是完全相同。

- 关于存储：
- tracking info是存储在对应feature 的brep solver下的 tracking table中的。这个table是一个<int, FmfBaseTrackingInfo>的map，即每一份tracking info有一个自己对应的track key。

- 总结一下，Tracking Info是记录一个对象的追溯信息的，对于每一个对象而言他们需要是独一无二的，以和其他对象进行区分。

新的永久命名对象---Split Data

- Split Data：
- 这个对象是属于EntRef，用来表示记录其与分裂兄弟对象之间互相区分的。

- 值得注意的是，它在EntRef中被记录为了一个List
- 这是因为一个EntRef会被分裂多次，在每一次的分裂中，它都会有相应的分裂信息出现。所以Split Data需要被保存为一个List，以记录每一次的分裂。分裂面在匹配的时候需要使用由当前特征创建的split data。

- 对于每一个字段，其内容为：

m_feature：产生这份split data的特征。
m_split_info：这份split data包含的分裂追溯信息。
m_split_goup：产生这次分裂的父对象的split count。

新的永久命名对象---Merge Data

- Merge Data：
- 虽然拥有相似的名字，但是要注意一点，<u>这个信息跟匹配是没有任何关系的</u>。这个信息是用于维护被合并对象与合并对象之间的关联的。

- 依然是以List方式存在于EntRef当中，因为一个对象可以被合并多次。

- 关于Merge Info中的内容，有两层含义。二者是互斥的，即它有一种就没有另一种，所以会被保存在一个位置中。两个含义为：

Merge Into：在这一次的合并中，这个ref被合并进了哪个ref。
Merge Info：在这一次的合并中，有哪些ref被合并进了这个ref。

Ent Ref基本规则

一些基本法

- 在开启这个章节之前，我们先介绍一下新旧label中最大的差别，即合并继承与分裂继承规则。

- 分裂继承
- 分裂行为是我们新label中修改的一大核心改变。在旧label体系中，我们的分裂label是在其父对象label的基础上添加一些信息来进行新label的命名。而在新体系下，我们的分裂是使用了继承+复制的方式

- 继承关系
- 继承，是指分裂出来的子对象中会有一个直接继承父对象的entref。那么结合上文我们提到的上层不关注od对象，只关注entref。结合这个就可得知，这个被继承的对象，会被上层直接认定为是源对象，所有作用在源对象上的行为都会直接作用到继承对象上。

- 那么这就涉及到一个问题，我们需要合理的选择一个被继承的对象。目前的机制为：
- 如果在这个分裂中，体也发生了分裂，那么我们按照体的继承结果，继承体继承的分裂体上的分裂面。
- 此外，按照按照分裂子对象Bounding Box的左下角，优先进行继承。

- 此外，每一个特征还可以选择继承方式。

- 合并继承

一些基本法

- 合并继承

- 合并继承关系也是一个核心改动。旧的永久命名体系中，合并label会将所有的合并对象组合在一起加以筛选后成为一个compound label。这个行为在实质上是错误的。

- 在新的永久命名体系中，合并label会直接继承某一个parent ref，然后给所有parent ref添加merge data。

- 继承哪一个：最老的一个。即将所有ref按照feature，track key， split count三个属性进行排序，选择最靠前的那个。当然，这个也会改进，提供能力给特征让他选择继承方案。

- merge data：
- 对于继承的ref，记录哪些ref被合并进了当前ref
- 对于没有被继承的ref，记录在这一次行为中，它被合并进了哪个ref。
- 再次重申，merge data不是参与匹配用的，只是记录这些信息，然后在后匹配中使用这些信息进行一些检索。

Face Ref

- 结构（主要成员）

ZsFaceRef

```
private:
    OM_XN_MEMBER_PUBLIC(ZwRef<ZwFeature>, m_feature); //owned by this feature
    OM_XN_MEMBER_PUBLIC(int, m_face_track_key, {-1});
    OM_XN_MEMBER_PUBLIC(int, m_split_count, {0});
    OM_XN_MEMBER_PUBLIC(om_vector<ZwRef<SplitData>>, m_split_data_list);
    OM_XN_MEMBER_PUBLIC(om_vector<ZwRef<MergedData>>, m_merged_data_list);
```

- m_feature：创建特征。是实体的创建特征而不是创建该face ref对象的特征。也就是说，即使是因分裂等行为而新建的对象，其feature也是其源对象的create feature。值得注意的是，这里还有一个含义，也就是这个face ref是own by what的。
- m_face_track_key：一个索引，关联到其对应的tracking。这是一个索引，是一个key，而不是什么db index。它获取的方式是去对应的特征下的track table中进行检索，获取其中的对象。
- m_split_count：表明分裂数。稍后在分裂继承关系中进行详细描述。
- m_split_data_list： split data，如上述split data。
- m_merged_data_list： merge data，如上文merge data。

Face Ref

- 创建
- 这里指的是创建型行为，而不只是生成一个face ref的行为。

- 由LabelingMachine创建于face labeling过程中。会有两种情况进行创建：

- 没有待比对对象的话，给所有新对象进行创建。
- 匹配流程结束后，所有匹配失败的对象。

- **LabelingMachine::CreateCreatedFaceRef。**代码详见wiki

- 该接口需要一份tracking info以及一个od ent来进行创建。

- 首先将m_feature设置为当前特征，然后将tracking放入对应的brepSolver中，获取其track key，将其设置为该ref的track key。最后绑定od entity与 face ref之间的关联。

Shell Ref

- 几乎完全同Face Ref

Edge Ref

- 结构（主要成员）

```
private:
    OM_XN_MEMBER(ZwRef<ZwFeature>, m_first_feature);
    OM_XN_MEMBER(ZwRef<ZwFeature>, m_second_feature);
    OM_XN_MEMBER(int, m_first_face_track_key, {-1});
    OM_XN_MEMBER(int, m_second_face_track_key, {-1});
    OM_XN_MEMBER(ZwRef<ZwFeature>, m_created_feature);
    OM_XN_MEMBER(int, m_split_count, {0});
```

- 与上述二者不同在于，edge 是没有tracking的，它的label是基于邻面信息来做的。所以在Edge分裂与合并的处理中，也会相应的与face有不同。
- 但是Edge记录的并不是完整的Face Ref，而是其Ftr和Track Key，也就是说Edge 不关注face 的分裂，其分裂由自己来记录。

- 为什么要做这一点，主要是基于两个原因。

  因为Edge Split实际上和Face Split并不完全是一个事情。比如很常见的Block挖槽，这就是Edge分裂而Face没变的场景。

  因为有Relabel这个容易破坏语义的存在，所以如果Face的信息做的过于详细，relabel的场景会变多非常多。

Edge Ref

- 分裂
- 这里的分裂，则是靠split事件完成的分裂。不是relabel同名label 的分裂。
- 其大致功能都与split face相同，但是需要提前进行一步筛选，那就是child edge label 的合法性。
- 我们在分裂事件中需要遍历所有的子边，检查其现在关联的face label是否与parent edge一致。由于edge ref是优先忠于face而不是tracking的，所以不符合条件的child edge会从split product list中被移除，转而变为创建行为。
- 继承关系
- 同face。继承对象的选择是bounding box左下角
- 分裂信息
- 使用顶点面信息，即该边端点上关联的面。

- 合并
- 基本逻辑同face，但是也需要进行筛选。
- 继承关系
- 在merge继承对象选择中会进行筛选，如果其parent ref与当前edge相关面的label不符，则不可继承。
- 如果所有的parent ref都不符，那么会创建新的edge ref，并且将其他的边作为merge info关联在其中。

新永久命名流程

如何确定odentity与entref的关联关系？永久命名流程----前匹配

- 在旧永久命名中，我们在实体创建时对其生成label，并将这个label保存在vdata中。后续特征在解析vdata时，遍历所有的brep对象进行检索，检查是否存在拥有相同label的实体，来使用。

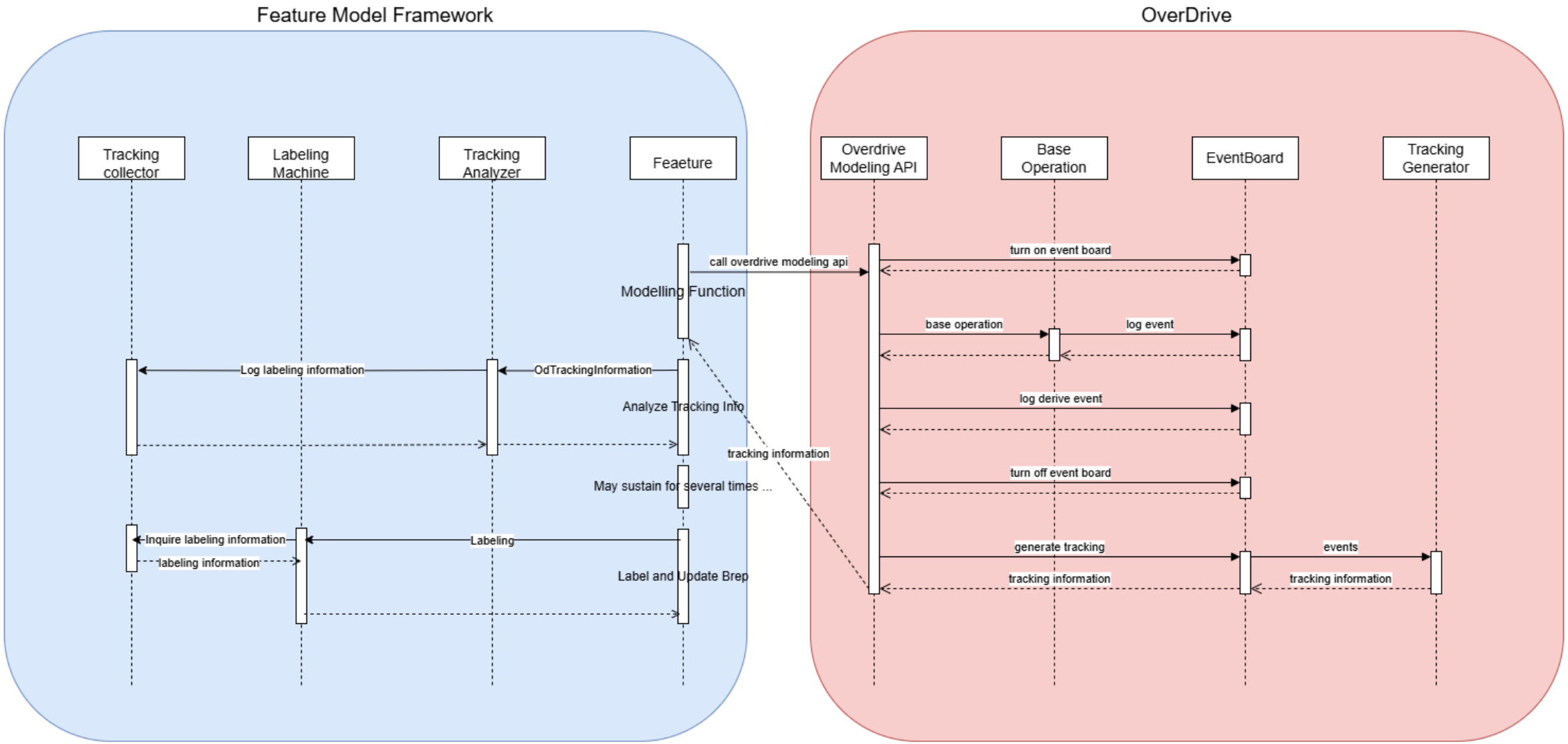
- 在新的永久命名体系中，我们将匹配这个行为定义为命令执行结束时，ent ref与od ent建立关联的过程。为了方便区分，我们在这里将其命名为前匹配，与之对应的则是后匹配，其定义是在特征重生成时，解析vdata通过相关信息检索实体的过程。

- 匹配会涉及到一个新的对象，MatchInfo。其功能很简单，就是将一份tracking info暴露出来，和一个对象关联在一起，用来进行匹配。

- 为什么它被定义为了一个模板类，则是因为它所关联的对象可以有很多种。在这里参与匹配的对象可以时一个ent ref，可以是一个odent，还可以是一些类似如wire之类的对象。

- LabelMatcher的职责是将待匹配的对象捉对厮杀完成匹配。具体比较两个tracking info的相似性的事情，还是需要交给在tracking info中定义的compare函数来完成。

Overview


Feature Model Framework
OverDrive
Tracking collector
Labeling Machine
Tracking Analyzer
Feaeture
Overdrive Modeling API
Base Operation
EventBoard
Tracking Generator
call overdrive modeling api
Modelling Function
turn on event board
base operation
log event
Log labeling information
OdTrackingInformation
Analyze Tracking Info
log derive event
tracking information
turn off event board
May sustain for several times ...
Inquire labeling information
Labeling
generate tracking
events
labeling information
Label and Update Brep
tracking information
tracking information

如何获得tracking info？永久命名流程----内核tracking

- 这一段主要讲述我们是如何获取到实体演化的信息的，也就是上图中红色的部分。由于这部分目前是由内核模块来负责的，这里就不做详细的赘述了，只做一个简单的讲解，详情请见
- Base Operation / Tracking Info - 平台架构设计部 - ZWiki (zwcax.com)

- 简而言之，就是在内核中添加了一系列base operation，所有对实体的操作（如创建，分裂等）都会由base operation记录到event board。

- 然后再建模命令执行结束后，调用EventBoard::generate_tracking_info()来对event board中的event hist进行压缩，最后将压缩后的tracking返回上层。这里的压缩行为主要是将无效信息进行删除，比如一个对象被创建后又被删除，这两个事件应该互相抵消掉。

如何整理tracking info？永久命名流程----Analyzer

- 这个结构的目标是为了将特征所给出的信息转换为labeling info，其主流来源则是内核传出的OdTrackInformation。
- 先讲输入，再说输出，至于具体做了什么，各位可以去文档和代码中详细观察。
- **关于输入**：OdTrackingInfoList是一个TrackRecordList，每一条TrackRecord中记录的三个内容：事件类型，源对象，目标对象。
- 事件类型分6种，分别为：分裂，合并，创建，删除，派生，转移。
- **关于输出**：生成TrackingInfo，并将其放入Collector中，以备LabelingMachine取用。
- Collector是一个单例对象，用来存储labeling流程所需要的信息。对象会分类存储在Collection当中，每一个collection中存储的是对应事件类型下需要进行label的对象与其相关信息。

如何完成各种永久命名？永久命名流程----Labeling Machine

- Labeling Machine是为了管理labeling流程而存在的。
- 这里的流程主要有三点，一个是管理需要labeling的对象的先后顺序，二是管理某一类对象的永久命名流程，三则是解除无效对象与EntRef之间的关联。

- **管理需要labeling的对象的先后顺序**
- 因为某些对象的命名过程需要用到其他流程预先完成命名的对象。比如分裂面需要记录其所在的体这个信息，就代表分裂体需要优先于分裂面进行命名。对于目前的LabelingMachine，我们进行命名的流程为：

- 创建面-创建体-分裂体-分裂面-合并面-合并体-边命名-解关联删除对象。

- **管理某一类对象的永久命名流程**
- 这里主要是处理，在一类对象的命名流程中，我们需要做什么事情来完成命名。整体而言，我们的流程属于：
- 收集新对象-收集关联旧对象-匹配-对新匹配成功的对象进行关联。

如何完成各种永久命名？永久命名流程----Labeling Machine

- **解除无效对象与EntRef之间的关联**
- 对于在建模过程中产生的对象，Labeling Machine需要负责对其进行 EntRef attach，那么同样的，对于无效的对象，也要进行detach。

- 有哪些对象会被认为是无效对象及其detach流程：

- 分裂父对象：在split labeling中，进行到当前分组时，先将parent ref detach掉在进行后续行为。
- 合并父对象：在merge labeling中，计算完继承对象后，将所有parent ref进行detach。
- 被删除的对象：在解关联删除对象时，将所有delete事件的对象进行detach。


感谢聆听
Q & A