namespace java com.twitter.graph\_feature\_service.thriftjava

#@namespace scala com.twitter.graph\_feature\_service.thriftscala

#@namespace strato com.twitter.graph\_feature\_service.thriftscala

// edge type to differentiate different types of graphs (we can also add a lot of other types of edges)

enum EdgeType {

FOLLOWING,

FOLLOWED\_BY,

FAVORITE,

FAVORITED\_BY,

RETWEET,

RETWEETED\_BY,

REPLY,

REPLYED\_BY,

MENTION,

MENTIONED\_BY,

MUTUAL\_FOLLOW,

SIMILAR\_TO, // more edge types (like block, report, etc.) can be supported later.

RESERVED\_12,

RESERVED\_13,

RESERVED\_14,

RESERVED\_15,

RESERVED\_16,

RESERVED\_17,

RESERVED\_18,

RESERVED\_19,

RESERVED\_20

}

enum PresetFeatureTypes {

EMPTY,

HTL\_TWO\_HOP,

WTF\_TWO\_HOP,

SQ\_TWO\_HOP,

RUX\_TWO\_HOP,

MR\_TWO\_HOP,

USER\_TYPEAHEAD\_TWO\_HOP

}

struct UserWithCount {

1: required i64 userId(personalDataType = 'UserId')

2: required i32 count

}(hasPersonalData = 'true')

struct UserWithScore {

1: required i64 userId(personalDataType = 'UserId')

2: required double score

}(hasPersonalData = 'true')

// Feature Type

// For example, to compute how many of source user's following's have favorited candidate user,

// we need to compute the intersection between source user's FOLLOWING edges, and candidate user's

// FAVORITED\_BY edge. In this case, we should user FeatureType(FOLLOWING, FAVORITED\_BY)

struct FeatureType {

1: required EdgeType leftEdgeType // edge type from source user

2: required EdgeType rightEdgeType // edge type from candidate user

}(persisted="true")

struct IntersectionValue {

1: required FeatureType featureType

2: optional i32 count

3: optional list<i64> intersectionIds(personalDataType = 'UserId')

4: optional i32 leftNodeDegree

5: optional i32 rightNodeDegree

}(persisted="true", hasPersonalData = 'true')

struct GfsIntersectionResult {

1: required i64 candidateUserId(personalDataType = 'UserId')

2: required list<IntersectionValue> intersectionValues

}(hasPersonalData = 'true')

struct GfsIntersectionRequest {

1: required i64 userId(personalDataType = 'UserId')

2: required list<i64> candidateUserIds(personalDataType = 'UserId')

3: required list<FeatureType> featureTypes

4: optional i32 intersectionIdLimit

}

struct GfsPresetIntersectionRequest {

1: required i64 userId(personalDataType = 'UserId')

2: required list<i64> candidateUserIds(personalDataType = 'UserId')

3: required PresetFeatureTypes presetFeatureTypes

4: optional i32 intersectionIdLimit

}(hasPersonalData = 'true')

struct GfsIntersectionResponse {

1: required list<GfsIntersectionResult> results

}

service Server {

GfsIntersectionResponse getIntersection(1: GfsIntersectionRequest request)

GfsIntersectionResponse getPresetIntersection(1: GfsPresetIntersectionRequest request)

}

###################################################################################################

## For internal usage only

###################################################################################################

struct WorkerIntersectionRequest {

1: required i64 userId(personalDataType = 'UserId')

2: required list<i64> candidateUserIds(personalDataType = 'UserId')

3: required list<FeatureType> featureTypes

4: required PresetFeatureTypes presetFeatureTypes

5: required i32 intersectionIdLimit

}(hasPersonalData = 'true')

struct WorkerIntersectionResponse {

1: required list<list<WorkerIntersectionValue>> results

}

struct WorkerIntersectionValue {

1: i32 count

2: i32 leftNodeDegree

3: i32 rightNodeDegree

4: list<i64> intersectionIds(personalDataType = 'UserId')

}(hasPersonalData = 'true')

struct CachedIntersectionResult {

1: required list<WorkerIntersectionValue> values

}

service Worker {

WorkerIntersectionResponse getIntersection(1: WorkerIntersectionRequest request)

}