// Autogenerated by Thrift Compiler (0.17.0)

// DO NOT EDIT UNLESS YOU ARE SURE THAT YOU KNOW WHAT YOU ARE DOING

#![allow(unused\_imports)]

#![allow(unused\_extern\_crates)]

#![allow(clippy::too\_many\_arguments, clippy::type\_complexity, clippy::vec\_box)]

#![cfg\_attr(rustfmt, rustfmt\_skip)]

use std::cell::RefCell;

use std::collections::{BTreeMap, BTreeSet};

use std::convert::{From, TryFrom};

use std::default::Default;

use std::error::Error;

use std::fmt;

use std::fmt::{Display, Formatter};

use std::rc::Rc;

use thrift::OrderedFloat;

use thrift::{ApplicationError, ApplicationErrorKind, ProtocolError, ProtocolErrorKind, TThriftClient};

use thrift::protocol::{TFieldIdentifier, TListIdentifier, TMapIdentifier, TMessageIdentifier, TMessageType, TInputProtocol, TOutputProtocol, TSerializable, TSetIdentifier, TStructIdentifier, TType};

use thrift::protocol::field\_id;

use thrift::protocol::verify\_expected\_message\_type;

use thrift::protocol::verify\_expected\_sequence\_number;

use thrift::protocol::verify\_expected\_service\_call;

use thrift::protocol::verify\_required\_field\_exists;

use thrift::server::TProcessor;

use crate::data;

//

// PredictionServiceException

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct PredictionServiceException {

pub description: Option<String>,

}

impl PredictionServiceException {

pub fn new<F1>(description: F1) -> PredictionServiceException where F1: Into<Option<String>> {

PredictionServiceException {

description: description.into(),

}

}

}

impl TSerializable for PredictionServiceException {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionServiceException> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<String> = Some("".to\_owned());

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = i\_prot.read\_string()?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

let ret = PredictionServiceException {

description: f\_1,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionServiceException");

o\_prot.write\_struct\_begin(&struct\_ident)?;

if let Some(ref fld\_var) = self.description {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("description", TType::String, 1))?;

o\_prot.write\_string(fld\_var)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

impl Default for PredictionServiceException {

fn default() -> Self {

PredictionServiceException{

description: Some("".to\_owned()),

}

}

}

impl Error for PredictionServiceException {}

impl From<PredictionServiceException> for thrift::Error {

fn from(e: PredictionServiceException) -> Self {

thrift::Error::User(Box::new(e))

}

}

impl Display for PredictionServiceException {

fn fmt(&self, f: &mut Formatter) -> fmt::Result {

write!(f, "remote service threw PredictionServiceException")

}

}

//

// PredictionRequest

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct PredictionRequest {

pub features: data::DataRecord,

}

impl PredictionRequest {

pub fn new(features: data::DataRecord) -> PredictionRequest {

PredictionRequest {

features,

}

}

}

impl TSerializable for PredictionRequest {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionRequest> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<data::DataRecord> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("PredictionRequest.features", &f\_1)?;

let ret = PredictionRequest {

features: f\_1.expect("auto-generated code should have checked for presence of required fields"),

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionRequest");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("features", TType::Struct, 1))?;

self.features.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?;

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// PredictionResponse

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct PredictionResponse {

pub prediction: data::DataRecord,

}

impl PredictionResponse {

pub fn new(prediction: data::DataRecord) -> PredictionResponse {

PredictionResponse {

prediction,

}

}

}

impl TSerializable for PredictionResponse {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionResponse> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<data::DataRecord> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("PredictionResponse.prediction", &f\_1)?;

let ret = PredictionResponse {

prediction: f\_1.expect("auto-generated code should have checked for presence of required fields"),

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionResponse");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("prediction", TType::Struct, 1))?;

self.prediction.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?;

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// BatchPredictionRequest

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct BatchPredictionRequest {

pub individual\_features\_list: Vec<data::DataRecord>,

pub common\_features: Option<data::DataRecord>,

}

impl BatchPredictionRequest {

pub fn new<F2>(individual\_features\_list: Vec<data::DataRecord>, common\_features: F2) -> BatchPredictionRequest where F2: Into<Option<data::DataRecord>> {

BatchPredictionRequest {

individual\_features\_list,

common\_features: common\_features.into(),

}

}

}

impl TSerializable for BatchPredictionRequest {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<BatchPredictionRequest> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<Vec<data::DataRecord>> = None;

let mut f\_2: Option<data::DataRecord> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let list\_ident = i\_prot.read\_list\_begin()?;

let mut val: Vec<data::DataRecord> = Vec::with\_capacity(list\_ident.size as usize);

for \_ in 0..list\_ident.size {

let list\_elem\_0 = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

val.push(list\_elem\_0);

}

i\_prot.read\_list\_end()?;

f\_1 = Some(val);

},

2 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_2 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("BatchPredictionRequest.individual\_features\_list", &f\_1)?;

let ret = BatchPredictionRequest {

individual\_features\_list: f\_1.expect("auto-generated code should have checked for presence of required fields"),

common\_features: f\_2,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("BatchPredictionRequest");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("individualFeaturesList", TType::List, 1))?;

o\_prot.write\_list\_begin(&TListIdentifier::new(TType::Struct, self.individual\_features\_list.len() as i32))?;

for e in &self.individual\_features\_list {

e.write\_to\_out\_protocol(o\_prot)?;

}

o\_prot.write\_list\_end()?;

o\_prot.write\_field\_end()?;

if let Some(ref fld\_var) = self.common\_features {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("commonFeatures", TType::Struct, 2))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// BatchPredictionResponse

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct BatchPredictionResponse {

pub predictions: Vec<data::DataRecord>,

}

impl BatchPredictionResponse {

pub fn new(predictions: Vec<data::DataRecord>) -> BatchPredictionResponse {

BatchPredictionResponse {

predictions,

}

}

}

impl TSerializable for BatchPredictionResponse {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<BatchPredictionResponse> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<Vec<data::DataRecord>> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let list\_ident = i\_prot.read\_list\_begin()?;

let mut val: Vec<data::DataRecord> = Vec::with\_capacity(list\_ident.size as usize);

for \_ in 0..list\_ident.size {

let list\_elem\_1 = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

val.push(list\_elem\_1);

}

i\_prot.read\_list\_end()?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("BatchPredictionResponse.predictions", &f\_1)?;

let ret = BatchPredictionResponse {

predictions: f\_1.expect("auto-generated code should have checked for presence of required fields"),

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("BatchPredictionResponse");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("predictions", TType::List, 1))?;

o\_prot.write\_list\_begin(&TListIdentifier::new(TType::Struct, self.predictions.len() as i32))?;

for e in &self.predictions {

e.write\_to\_out\_protocol(o\_prot)?;

}

o\_prot.write\_list\_end()?;

o\_prot.write\_field\_end()?;

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// DataRecordPair

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct DataRecordPair {

pub first: Option<data::DataRecord>,

pub second: Option<data::DataRecord>,

}

impl DataRecordPair {

pub fn new<F1, F2>(first: F1, second: F2) -> DataRecordPair where F1: Into<Option<data::DataRecord>>, F2: Into<Option<data::DataRecord>> {

DataRecordPair {

first: first.into(),

second: second.into(),

}

}

}

impl TSerializable for DataRecordPair {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<DataRecordPair> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<data::DataRecord> = None;

let mut f\_2: Option<data::DataRecord> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

2 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_2 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

let ret = DataRecordPair {

first: f\_1,

second: f\_2,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("DataRecordPair");

o\_prot.write\_struct\_begin(&struct\_ident)?;

if let Some(ref fld\_var) = self.first {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("first", TType::Struct, 1))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.second {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("second", TType::Struct, 2))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

impl Default for DataRecordPair {

fn default() -> Self {

DataRecordPair{

first: None,

second: None,

}

}

}

//

// PredictionTrainingExample

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

pub struct PredictionTrainingExample {

pub features: Option<data::DataRecord>,

pub features\_for\_pairwise\_learning: Option<DataRecordPair>,

pub compact\_features: Option<data::CompactDataRecord>,

pub compressed\_data\_record: Option<Vec<u8>>,

}

impl PredictionTrainingExample {

pub fn new<F1, F2, F3, F4>(features: F1, features\_for\_pairwise\_learning: F2, compact\_features: F3, compressed\_data\_record: F4) -> PredictionTrainingExample where F1: Into<Option<data::DataRecord>>, F2: Into<Option<DataRecordPair>>, F3: Into<Option<data::CompactDataRecord>>, F4: Into<Option<Vec<u8>>> {

PredictionTrainingExample {

features: features.into(),

features\_for\_pairwise\_learning: features\_for\_pairwise\_learning.into(),

compact\_features: compact\_features.into(),

compressed\_data\_record: compressed\_data\_record.into(),

}

}

}

impl TSerializable for PredictionTrainingExample {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionTrainingExample> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<data::DataRecord> = None;

let mut f\_2: Option<DataRecordPair> = None;

let mut f\_3: Option<data::CompactDataRecord> = None;

let mut f\_4: Option<Vec<u8>> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = data::DataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

2 => {

let val = DataRecordPair::read\_from\_in\_protocol(i\_prot)?;

f\_2 = Some(val);

},

3 => {

let val = data::CompactDataRecord::read\_from\_in\_protocol(i\_prot)?;

f\_3 = Some(val);

},

4 => {

let val = i\_prot.read\_bytes()?;

f\_4 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

let ret = PredictionTrainingExample {

features: f\_1,

features\_for\_pairwise\_learning: f\_2,

compact\_features: f\_3,

compressed\_data\_record: f\_4,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionTrainingExample");

o\_prot.write\_struct\_begin(&struct\_ident)?;

if let Some(ref fld\_var) = self.features {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("features", TType::Struct, 1))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.features\_for\_pairwise\_learning {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("featuresForPairwiseLearning", TType::Struct, 2))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.compact\_features {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("compactFeatures", TType::Struct, 3))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.compressed\_data\_record {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("compressedDataRecord", TType::String, 4))?;

o\_prot.write\_bytes(fld\_var)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

impl Default for PredictionTrainingExample {

fn default() -> Self {

PredictionTrainingExample{

features: None,

features\_for\_pairwise\_learning: None,

compact\_features: None,

compressed\_data\_record: Some(Vec::new()),

}

}

}

//

// PredictionService service client

//

pub trait TPredictionServiceSyncClient {

fn get\_prediction(&mut self, request: PredictionRequest) -> thrift::Result<PredictionResponse>;

fn get\_batch\_prediction(&mut self, batch\_request: BatchPredictionRequest) -> thrift::Result<BatchPredictionResponse>;

}

pub trait TPredictionServiceSyncClientMarker {}

pub struct PredictionServiceSyncClient<IP, OP> where IP: TInputProtocol, OP: TOutputProtocol {

\_i\_prot: IP,

\_o\_prot: OP,

\_sequence\_number: i32,

}

impl <IP, OP> PredictionServiceSyncClient<IP, OP> where IP: TInputProtocol, OP: TOutputProtocol {

pub fn new(input\_protocol: IP, output\_protocol: OP) -> PredictionServiceSyncClient<IP, OP> {

PredictionServiceSyncClient { \_i\_prot: input\_protocol, \_o\_prot: output\_protocol, \_sequence\_number: 0 }

}

}

impl <IP, OP> TThriftClient for PredictionServiceSyncClient<IP, OP> where IP: TInputProtocol, OP: TOutputProtocol {

fn i\_prot\_mut(&mut self) -> &mut dyn TInputProtocol { &mut self.\_i\_prot }

fn o\_prot\_mut(&mut self) -> &mut dyn TOutputProtocol { &mut self.\_o\_prot }

fn sequence\_number(&self) -> i32 { self.\_sequence\_number }

fn increment\_sequence\_number(&mut self) -> i32 { self.\_sequence\_number += 1; self.\_sequence\_number }

}

impl <IP, OP> TPredictionServiceSyncClientMarker for PredictionServiceSyncClient<IP, OP> where IP: TInputProtocol, OP: TOutputProtocol {}

impl <C: TThriftClient + TPredictionServiceSyncClientMarker> TPredictionServiceSyncClient for C {

fn get\_prediction(&mut self, request: PredictionRequest) -> thrift::Result<PredictionResponse> {

(

{

self.increment\_sequence\_number();

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Call, self.sequence\_number());

let call\_args = PredictionServiceGetPredictionArgs { request };

self.o\_prot\_mut().write\_message\_begin(&message\_ident)?;

call\_args.write\_to\_out\_protocol(self.o\_prot\_mut())?;

self.o\_prot\_mut().write\_message\_end()?;

self.o\_prot\_mut().flush()

}

)?;

{

let message\_ident = self.i\_prot\_mut().read\_message\_begin()?;

verify\_expected\_sequence\_number(self.sequence\_number(), message\_ident.sequence\_number)?;

verify\_expected\_service\_call("getPrediction", &message\_ident.name)?;

if message\_ident.message\_type == TMessageType::Exception {

let remote\_error = thrift::Error::read\_application\_error\_from\_in\_protocol(self.i\_prot\_mut())?;

self.i\_prot\_mut().read\_message\_end()?;

return Err(thrift::Error::Application(remote\_error))

}

verify\_expected\_message\_type(TMessageType::Reply, message\_ident.message\_type)?;

let result = PredictionServiceGetPredictionResult::read\_from\_in\_protocol(self.i\_prot\_mut())?;

self.i\_prot\_mut().read\_message\_end()?;

result.ok\_or()

}

}

fn get\_batch\_prediction(&mut self, batch\_request: BatchPredictionRequest) -> thrift::Result<BatchPredictionResponse> {

(

{

self.increment\_sequence\_number();

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Call, self.sequence\_number());

let call\_args = PredictionServiceGetBatchPredictionArgs { batch\_request };

self.o\_prot\_mut().write\_message\_begin(&message\_ident)?;

call\_args.write\_to\_out\_protocol(self.o\_prot\_mut())?;

self.o\_prot\_mut().write\_message\_end()?;

self.o\_prot\_mut().flush()

}

)?;

{

let message\_ident = self.i\_prot\_mut().read\_message\_begin()?;

verify\_expected\_sequence\_number(self.sequence\_number(), message\_ident.sequence\_number)?;

verify\_expected\_service\_call("getBatchPrediction", &message\_ident.name)?;

if message\_ident.message\_type == TMessageType::Exception {

let remote\_error = thrift::Error::read\_application\_error\_from\_in\_protocol(self.i\_prot\_mut())?;

self.i\_prot\_mut().read\_message\_end()?;

return Err(thrift::Error::Application(remote\_error))

}

verify\_expected\_message\_type(TMessageType::Reply, message\_ident.message\_type)?;

let result = PredictionServiceGetBatchPredictionResult::read\_from\_in\_protocol(self.i\_prot\_mut())?;

self.i\_prot\_mut().read\_message\_end()?;

result.ok\_or()

}

}

}

//

// PredictionService service processor

//

pub trait PredictionServiceSyncHandler {

fn handle\_get\_prediction(&self, request: PredictionRequest) -> thrift::Result<PredictionResponse>;

fn handle\_get\_batch\_prediction(&self, batch\_request: BatchPredictionRequest) -> thrift::Result<BatchPredictionResponse>;

}

pub struct PredictionServiceSyncProcessor<H: PredictionServiceSyncHandler> {

handler: H,

}

impl <H: PredictionServiceSyncHandler> PredictionServiceSyncProcessor<H> {

pub fn new(handler: H) -> PredictionServiceSyncProcessor<H> {

PredictionServiceSyncProcessor {

handler,

}

}

fn process\_get\_prediction(&self, incoming\_sequence\_number: i32, i\_prot: &mut dyn TInputProtocol, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

TPredictionServiceProcessFunctions::process\_get\_prediction(&self.handler, incoming\_sequence\_number, i\_prot, o\_prot)

}

fn process\_get\_batch\_prediction(&self, incoming\_sequence\_number: i32, i\_prot: &mut dyn TInputProtocol, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

TPredictionServiceProcessFunctions::process\_get\_batch\_prediction(&self.handler, incoming\_sequence\_number, i\_prot, o\_prot)

}

}

pub struct TPredictionServiceProcessFunctions;

impl TPredictionServiceProcessFunctions {

pub fn process\_get\_prediction<H: PredictionServiceSyncHandler>(handler: &H, incoming\_sequence\_number: i32, i\_prot: &mut dyn TInputProtocol, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let args = PredictionServiceGetPredictionArgs::read\_from\_in\_protocol(i\_prot)?;

match handler.handle\_get\_prediction(args.request) {

Ok(handler\_return) => {

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Reply, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

let ret = PredictionServiceGetPredictionResult { result\_value: Some(handler\_return), prediction\_service\_exception: None };

ret.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

Err(e) => {

match e {

thrift::Error::User(usr\_err) => {

if usr\_err.downcast\_ref::<PredictionServiceException>().is\_some() {

let err = usr\_err.downcast::<PredictionServiceException>().expect("downcast already checked");

let ret\_err = PredictionServiceGetPredictionResult{ result\_value: None, prediction\_service\_exception: Some(\*err) };

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Reply, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

ret\_err.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

} else {

let ret\_err = {

ApplicationError::new(

ApplicationErrorKind::Unknown,

usr\_err.to\_string()

)

};

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&ret\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

}

},

thrift::Error::Application(app\_err) => {

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&app\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

\_ => {

let ret\_err = {

ApplicationError::new(

ApplicationErrorKind::Unknown,

e.to\_string()

)

};

let message\_ident = TMessageIdentifier::new("getPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&ret\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

}

},

}

}

pub fn process\_get\_batch\_prediction<H: PredictionServiceSyncHandler>(handler: &H, incoming\_sequence\_number: i32, i\_prot: &mut dyn TInputProtocol, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let args = PredictionServiceGetBatchPredictionArgs::read\_from\_in\_protocol(i\_prot)?;

match handler.handle\_get\_batch\_prediction(args.batch\_request) {

Ok(handler\_return) => {

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Reply, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

let ret = PredictionServiceGetBatchPredictionResult { result\_value: Some(handler\_return), prediction\_service\_exception: None };

ret.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

Err(e) => {

match e {

thrift::Error::User(usr\_err) => {

if usr\_err.downcast\_ref::<PredictionServiceException>().is\_some() {

let err = usr\_err.downcast::<PredictionServiceException>().expect("downcast already checked");

let ret\_err = PredictionServiceGetBatchPredictionResult{ result\_value: None, prediction\_service\_exception: Some(\*err) };

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Reply, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

ret\_err.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

} else {

let ret\_err = {

ApplicationError::new(

ApplicationErrorKind::Unknown,

usr\_err.to\_string()

)

};

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&ret\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

}

},

thrift::Error::Application(app\_err) => {

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&app\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

\_ => {

let ret\_err = {

ApplicationError::new(

ApplicationErrorKind::Unknown,

e.to\_string()

)

};

let message\_ident = TMessageIdentifier::new("getBatchPrediction", TMessageType::Exception, incoming\_sequence\_number);

o\_prot.write\_message\_begin(&message\_ident)?;

thrift::Error::write\_application\_error\_to\_out\_protocol(&ret\_err, o\_prot)?;

o\_prot.write\_message\_end()?;

o\_prot.flush()

},

}

},

}

}

}

impl <H: PredictionServiceSyncHandler> TProcessor for PredictionServiceSyncProcessor<H> {

fn process(&self, i\_prot: &mut dyn TInputProtocol, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let message\_ident = i\_prot.read\_message\_begin()?;

let res = match &\*message\_ident.name {

"getPrediction" => {

self.process\_get\_prediction(message\_ident.sequence\_number, i\_prot, o\_prot)

},

"getBatchPrediction" => {

self.process\_get\_batch\_prediction(message\_ident.sequence\_number, i\_prot, o\_prot)

},

method => {

Err(

thrift::Error::Application(

ApplicationError::new(

ApplicationErrorKind::UnknownMethod,

format!("unknown method {}", method)

)

)

)

},

};

thrift::server::handle\_process\_result(&message\_ident, res, o\_prot)

}

}

//

// PredictionServiceGetPredictionArgs

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

struct PredictionServiceGetPredictionArgs {

request: PredictionRequest,

}

impl PredictionServiceGetPredictionArgs {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionServiceGetPredictionArgs> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<PredictionRequest> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = PredictionRequest::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("PredictionServiceGetPredictionArgs.request", &f\_1)?;

let ret = PredictionServiceGetPredictionArgs {

request: f\_1.expect("auto-generated code should have checked for presence of required fields"),

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("getPrediction\_args");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("request", TType::Struct, 1))?;

self.request.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?;

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// PredictionServiceGetPredictionResult

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

struct PredictionServiceGetPredictionResult {

result\_value: Option<PredictionResponse>,

prediction\_service\_exception: Option<PredictionServiceException>,

}

impl PredictionServiceGetPredictionResult {

fn ok\_or(self) -> thrift::Result<PredictionResponse> {

if self.prediction\_service\_exception.is\_some() {

Err(thrift::Error::User(Box::new(self.prediction\_service\_exception.unwrap())))

} else if self.result\_value.is\_some() {

Ok(self.result\_value.unwrap())

} else {

Err(

thrift::Error::Application(

ApplicationError::new(

ApplicationErrorKind::MissingResult,

"no result received for PredictionServiceGetPrediction"

)

)

)

}

}

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionServiceGetPredictionResult> {

i\_prot.read\_struct\_begin()?;

let mut f\_0: Option<PredictionResponse> = None;

let mut f\_1: Option<PredictionServiceException> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

0 => {

let val = PredictionResponse::read\_from\_in\_protocol(i\_prot)?;

f\_0 = Some(val);

},

1 => {

let val = PredictionServiceException::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

let ret = PredictionServiceGetPredictionResult {

result\_value: f\_0,

prediction\_service\_exception: f\_1,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionServiceGetPredictionResult");

o\_prot.write\_struct\_begin(&struct\_ident)?;

if let Some(ref fld\_var) = self.result\_value {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("result\_value", TType::Struct, 0))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.prediction\_service\_exception {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("predictionServiceException", TType::Struct, 1))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// PredictionServiceGetBatchPredictionArgs

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

struct PredictionServiceGetBatchPredictionArgs {

batch\_request: BatchPredictionRequest,

}

impl PredictionServiceGetBatchPredictionArgs {

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionServiceGetBatchPredictionArgs> {

i\_prot.read\_struct\_begin()?;

let mut f\_1: Option<BatchPredictionRequest> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

1 => {

let val = BatchPredictionRequest::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

verify\_required\_field\_exists("PredictionServiceGetBatchPredictionArgs.batch\_request", &f\_1)?;

let ret = PredictionServiceGetBatchPredictionArgs {

batch\_request: f\_1.expect("auto-generated code should have checked for presence of required fields"),

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("getBatchPrediction\_args");

o\_prot.write\_struct\_begin(&struct\_ident)?;

o\_prot.write\_field\_begin(&TFieldIdentifier::new("batchRequest", TType::Struct, 1))?;

self.batch\_request.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?;

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}

//

// PredictionServiceGetBatchPredictionResult

//

#[derive(Clone, Debug, Eq, Hash, Ord, PartialEq, PartialOrd)]

struct PredictionServiceGetBatchPredictionResult {

result\_value: Option<BatchPredictionResponse>,

prediction\_service\_exception: Option<PredictionServiceException>,

}

impl PredictionServiceGetBatchPredictionResult {

fn ok\_or(self) -> thrift::Result<BatchPredictionResponse> {

if self.prediction\_service\_exception.is\_some() {

Err(thrift::Error::User(Box::new(self.prediction\_service\_exception.unwrap())))

} else if self.result\_value.is\_some() {

Ok(self.result\_value.unwrap())

} else {

Err(

thrift::Error::Application(

ApplicationError::new(

ApplicationErrorKind::MissingResult,

"no result received for PredictionServiceGetBatchPrediction"

)

)

)

}

}

fn read\_from\_in\_protocol(i\_prot: &mut dyn TInputProtocol) -> thrift::Result<PredictionServiceGetBatchPredictionResult> {

i\_prot.read\_struct\_begin()?;

let mut f\_0: Option<BatchPredictionResponse> = None;

let mut f\_1: Option<PredictionServiceException> = None;

loop {

let field\_ident = i\_prot.read\_field\_begin()?;

if field\_ident.field\_type == TType::Stop {

break;

}

let field\_id = field\_id(&field\_ident)?;

match field\_id {

0 => {

let val = BatchPredictionResponse::read\_from\_in\_protocol(i\_prot)?;

f\_0 = Some(val);

},

1 => {

let val = PredictionServiceException::read\_from\_in\_protocol(i\_prot)?;

f\_1 = Some(val);

},

\_ => {

i\_prot.skip(field\_ident.field\_type)?;

},

};

i\_prot.read\_field\_end()?;

}

i\_prot.read\_struct\_end()?;

let ret = PredictionServiceGetBatchPredictionResult {

result\_value: f\_0,

prediction\_service\_exception: f\_1,

};

Ok(ret)

}

fn write\_to\_out\_protocol(&self, o\_prot: &mut dyn TOutputProtocol) -> thrift::Result<()> {

let struct\_ident = TStructIdentifier::new("PredictionServiceGetBatchPredictionResult");

o\_prot.write\_struct\_begin(&struct\_ident)?;

if let Some(ref fld\_var) = self.result\_value {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("result\_value", TType::Struct, 0))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

if let Some(ref fld\_var) = self.prediction\_service\_exception {

o\_prot.write\_field\_begin(&TFieldIdentifier::new("predictionServiceException", TType::Struct, 1))?;

fld\_var.write\_to\_out\_protocol(o\_prot)?;

o\_prot.write\_field\_end()?

}

o\_prot.write\_field\_stop()?;

o\_prot.write\_struct\_end()

}

}