**Τεχνολογία Συστημάτων Λογισμικού**

**Τίτλος εργασίας: «Εντοπισμός Code Smells, Refactorings, Git»**

**Επιμέλεια εργασίας: Μαρία Βασιλική Σαρρή(mai23036), Ευδοκία Στεφανίδου(mai23040)**

<https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY>

ΕΝΤΟΠΙΣΜΟΣ ΟΣΜΩΝ ΚΩΔΙΚΑ:

Ξεκινώντας, με οδηγό τον προσδιορισμό διαφόρων τιμών μετρικών για την τελευταία έκδοση του hprose, εντοπίζουμε αμέσως το εξής πρόβλημα: η κλάση **hprose.server.HproseService.java** έχει το μεγαλύτερο LCOM, πράγμα που σημαίνει ότι η κλάση επιτελεί περισσότερες λειτουργίες απ’ ότι πρέπει, και επιπλέον είναι πάρα πολύ μεγάλη (966 LOC σύμφωνα με την μέτρηση του Metrics Calculator). Όλα αυτά καθιστούν δύσκολη αρχικά την κατανόησή της, και συνηγορούν στο γεγονός πώς τελικά μάλλον θα πρέπει να χωριστεί.

Στην κλάση **Hprose.io.access.Accessors** παρατηρούμε ότι υπάρχουν μεγάλες μέθοδοι όπως getProperties(89-122) και getMembers(175-218) που έχουν μερικά κομμάτια κώδικα ίδια. Θέλουμε να μικρύνουμε αυτές τις μεθόδους, εξάγοντας κομμάτια τους σε άλλες πιο καλά οργανωμένες μεθόδους, ώστε ο κώδικας να διαβάζεται το ίδιο εύκολα με το να ήταν απλά σχόλια.

Στην κλάση **hprose.util.DateTime** παρατηρείται η επανάληψη του ίδιου κώδικα(duplicate code). Πιο συγκεκριμένα υπάρχουν πολλοί DateTime constructors με duplication μέσα τους.

Είναι καλό να αποφεύγεται η επανάληψη του ίδιου κώδικα μέσα στους constructors γιατί στο μέλλον όταν κάποιος θα θέλει να ενημερώσει την εφαρμογή θα πρέπει να ενημερώσει έναν από αυτούς και ίσως ξεχάσει τους υπόλοιπους δημιουργώντας έτσι διάφορα προβλήματα στην λειτουργικότητα του κώδικα.

Επιπρόσθετα, στην κλάση **hprose.io.unserialize.ValueReader.java** , η μέθοδος readTime() παίρνει ως παραμέτρους αντικείμενα τύπου Reader και DateTime. Στο σώμα της μεθόδου, προσπελαύνονται κυρίως πεδία της DateTime (τα οποία έχουν οριστεί ως public🡪κακή πρακτική, παραβαίνει την αρχή της ενσωμάτωσης), οπότε υποπτευόμαστε ότι αυτή η μέθοδος, όπως και αντίστοιχα οι μέθοδοι readTime(Reader reader) και readDateTime(Reader reader), πρέπει να μετακινηθούν στην κλάση DateTime.

Παρατηρούμε επίσης, πως στην κλάση **hprose.server.HproseService.java** η μέθοδος doInvoke() είναι αρκετά μεγάλη και παίρνει σαν παραμέτρους αντικείμενα των κλάσεων ByteBufferStream και ServiceContext. Επί αυτών καλούνται μέθοδοι των αντίστοιχων κλάσεων. Επίσης μέσα στο σώμα της μεθόδου δημιουργούνται αντικείμενα των κλάσεων HproseMethods και Reader. Όλα αυτά συνεπάργονται μια ισχυρή συσχέτιση/σύνδεση όλων των εμπλεκόμενων κλάσεων, γεγονός που δυσκολεύει αρκετά την επέκταση και την συντήρηση του κώδικα.

Αναδομήσεις

Για την απομάκρυση της 1ης οσμής (Large and complex class) <https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/blob/main/After1stRefactoring(extracted%20class)> που αναφέρθηκε πάνω, διαλέγουμε την επιλογή extract class και δημιουργούμε μία κλάση με όνομα HproseServiceData.

public class HproseServiceData {

public ArrayList<HproseFilter> filters;

public *HproseMode* mode;

public boolean debugEnabled;

public int errorDelay;

public HproseServiceEvent event;

public HproseMethods globalMethods;

public int timeout;

public int heartbeat;

public PushEvent pushEvent;

public ConcurrentHashMap<String, ConcurrentHashMap<String, Topic>> allTopics;

Δημιουργούμε πεδίο “mode” στην ByteBufferStream για να φύγει το σφάλμα απ΄ την HproseService. Από την HproseService, μετακινούμε τις μεθόδους beforeInvoke(), multicast() και getGlobalMethods() στην ΗproseServiceData. Για να διορθώσω τα σφάλματα στην HproseService , δημιουργώ μεθόδους add(String, Callable<String>, String,Boolean) και add(String, Func<Object, String>, String) . Τέλος, όποιες μεθόδους χειρίζονται δεδομένα της νέας κλάσης, τις σβήνω απ’ την HproseService.

Για την 2η οσμή: <https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/blob/main/afterLongmethod>

Παρατηρούμε ότι μέσα στη μέθοδο getProperties υπάρχουν πολλές if statements που προσδίδουν πολυπλοκότητα για αυτό θα εξάγουμε ένα μέρος της(γραμμή94-117) στην εξής μέθοδο propertiesCache:

**private** **static** **void** propertiesCache(Type type, LinkedCaseInsensitiveMap<String, MemberAccessor> properties,

Method[] methods) {

**for** (Method setter : methods) {

**if** (Modifier.*isStatic*(setter.getModifiers())) {

**continue**;

}

String name = setter.getName();

**if** (!name.startsWith("set")) {

**continue**;

}

**if** (!setter.getReturnType().equals(**void**.**class**)) {

**continue**;

}

Class<?>[] paramTypes = setter.getParameterTypes();

**if** (paramTypes.length != 1) {

**continue**;

}

String propertyName = name.substring(3);

Method getter = *findGetter*(methods, propertyName, paramTypes[0]);

**if** (getter != **null**) {

PropertyAccessor propertyAccessor = **new** PropertyAccessor(type, getter, setter);

propertyName = *propertyName*(propertyName);

properties.put(propertyName, propertyAccessor);

}

}

***propertiesCache***.put(type, properties);

}

Και αντίστοιχα εφαρμόζουμε παρόμοια διαδικασία στην μέθοδο membersCache(185) δημιουργώντας την membersCache

**private** **static** **void** membersCache(Type type, Class<?> clazz,

LinkedCaseInsensitiveMap<String, MemberAccessor> members, Method[] methods) **throws** SecurityException {

**for** (Method setter : methods) {

**if** (Modifier.*isStatic*(setter.getModifiers())) {

**continue**;

}

String name = setter.getName();

**if** (!name.startsWith("set")) {

**continue**;

}

Class<?> returnType = setter.getReturnType();

**if** (!returnType.equals(**void**.**class**) && !returnType.isAssignableFrom(clazz)) {

**continue**;

}

Class<?>[] paramTypes = setter.getParameterTypes();

**if** (paramTypes.length != 1) {

**continue**;

}

String propertyName = name.substring(3);

Method getter = *findGetter*(methods, propertyName, paramTypes[0]);

**if** (getter != **null**) {

PropertyAccessor propertyAccessor = **new** PropertyAccessor(type, getter, setter);

propertyName = *propertyName*(propertyName);

members.put(propertyName, propertyAccessor);

}

}

Field[] fs = clazz.getFields();

**for** (Field field : fs) {

**int** mod = field.getModifiers();

**if** (!Modifier.*isTransient*(mod) && !Modifier.*isStatic*(mod)) {

String fieldName = field.getName();

members.put(fieldName, *getFieldAccessor*(type, field));

}

}

***membersCache***.put(type, members);

}

Κάνοντας τις παραπάνω αλλαγές παρατηρείται ο εξής διπλότυπος κώδικας μέσα σε αυτές τις δύο μεθόδους

**char**[] cname = propertyName.toCharArray();

cname[0] = Character.*toLowerCase*(cname[0]);

propertyName = **new** String(cname);

Αυτόν θέλουμε να τον εξάγουμε σε μία νέα μέθοδο propertyName:

**private** **static** String propertyName(String propertyName) {

**char**[] cname = propertyName.toCharArray();

cname[0] = Character.*toLowerCase*(cname[0]);

propertyName = **new** String(cname);

**return** propertyName;

}

Για την αντιμετώπιση της 3ης οσμής: <https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/blob/main/afterDuplicatecode>

Αυτή επιλύεται έχοντας μερικούς constructors που καλούν έναν γενικής χρήσης constructor, έτσι θα εξαλείψει όλες τις επικαλύψεις όσον αφορά την εκχώρηση τιμών που ακούν σε διαφορετικά πεδία μέσα στον constructor.

Δεν παρατηρείται σημαντική αλλαγή στις υπολογισμένες μετρικές πέραν του size2, παρόλα αυτά μειώνονται οι γραμμές κώδικα και αυτός είναι πιο εύκολα επεκτάσιμος και κατανοήσιμος.

//generalcostructor

public DateTime(int year, int month, int day,int hour,int minute,int second,int nanosecond,boolean utc) {

this.year = year;

this.month = month;

this.day = day;

this.hour = hour;

this.minute = minute;

this.second = second;

this.nanosecond = nanosecond;

this.utc = utc;

}

//otherconstructors

public DateTime(int year, int month, int day) {

this(year,month,day,0,0,0,0,false);

}

public DateTime(int year, int month, int day,boolean utc) {

this(year,month,day,0,0,0,0,utc);

}

public DateTime(int hour,int minute,int second,int nanosecond) {

this(1970,1,1,hour,minute,second,nanosecond,false);

}

public DateTime(int hour,int minute,int second,int nanosecond,boolean utc) {

this(1970,1,1,hour,minute,second,nanosecond,utc);

}

public DateTime(int year,int month,int day,int hour,int minute,int second) {

this(year,month,day,hour,minute,second,0,false);

}

public DateTime(int year, int month, int day,int hour,int minute,int second,boolean utc) {

this(year,month,day,hour,minute,second,0,utc);

}

public DateTime(int year, int month, int day,int hour,int minute,int second,int nanosecond) {

this(year,month,day,hour,minute,second,nanosecond,false);

}

Για την 4η οσμή (Feature envy) :

<https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/commit/e99c57f5c152ac1f1d8c48a7e3899ba4c31fbfd3>

<https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/blob/main/DateTime.java>

Μετακινώ την τις μεθόδους readTime(), readDateTime(), read2Digit(), read4Digit() της ValueReader στην DateTime και ταυτόχρονα τις σβήνω φυσικά απ΄την ValueReader.

public final static int readTime(Reader reader, DateTime dt) throws IOException {

InputStream stream = reader.stream;

dt.hour = read2Digit(stream);

dt.minute = read2Digit(stream);

dt.second = read2Digit(stream);

int tag = stream.read();

if (tag == TagPoint) {

dt.nanosecond = stream.read() - '0';

dt.nanosecond = dt.nanosecond \* 10 + (stream.read() - '0');

dt.nanosecond = dt.nanosecond \* 10 + (stream.read() - '0');

dt.nanosecond = dt.nanosecond \* 1000000;

tag = stream.read();

if (tag >= '0' && tag <= '9') {

dt.nanosecond += (tag - '0') \* 100000;

dt.nanosecond += (stream.read() - '0') \* 10000;

dt.nanosecond += (stream.read() - '0') \* 1000;

tag = stream.read();

if (tag >= '0' && tag <= '9') {

dt.nanosecond += (tag - '0') \* 100;

dt.nanosecond += (stream.read() - '0') \* 10;

dt.nanosecond += stream.read() - '0';

tag = stream.read();

}

}

}

return tag;

}

public final static DateTime readTime(Reader reader) throws IOException {

DateTime dt = new DateTime();

dt.utc = (readTime(reader, dt) == TagUTC);

return dt;

}

public final static DateTime readDateTime(Reader reader) throws IOException {

InputStream stream = reader.stream;

DateTime dt = new DateTime();

dt.year = read4Digit(stream);

dt.month = read2Digit(stream);

dt.day = read2Digit(stream);

int tag = stream.read();

if (tag == TagTime) {

tag = readTime(reader, dt);

}

dt.utc = (tag == TagUTC);

return dt;

}

private static int read4Digit(InputStream stream) throws IOException {

int n = stream.read() - '0';

n = n \* 10 + stream.read() - '0';

n = n \* 10 + stream.read() - '0';

return n \* 10 + stream.read() - '0';

}

private static int read2Digit(InputStream stream) throws IOException {

int n = stream.read() - '0';

return n \* 10 + stream.read() - '0';

}

Πηγαίνω στην κλάση ReferenceReader στο ίδιο πακέτο, και όπου υπάρχει κλήση των μεθόδων readDateTime και readTime, αλλάζω τον τύπο αντικειμένου από ValueReader σε DateTime:

public final class ReferenceReader {

public final static DateTime readDateTime(Reader reader) throws IOException {

DateTime datetime = DateTime.*readDateTime*(reader);

reader.setRef(datetime);

return datetime;

}

public final static DateTime readTime(Reader reader) throws IOException {

DateTime datetime = DateTime.*readTime*(reader);

reader.setRef(datetime);

return datetime;

}

Για την 5η οσμή (intensive coupling) : <https://github.com/maribilisarri/TECHLOGIASISTIMATONLOGISMIKOY/blob/main/afterIntensiveCouplingFix>

Μετακινώ όλη την μέθοδο doInvoke() στην κλάση hprose.io.ByteBufferStream σύμφωνα με τις οδηγίες του Eclipse (διατηρώ το όνομά της). Φυσικά, την σβήνω από την κλάση ΗproseService.

public Promise<ByteBuffer> doInvoke(HproseService hproseService, ServiceContext context) throws IOException {

HproseMethods methods = context.getMethods();

Reader reader = new Reader(getInputStream(), hproseService.data.mode);

ArrayList<Object> results = new ArrayList<Object>();

int tag;

do {

reader.reset();

String name = reader.readString();

String aliasname = name.toLowerCase();

HproseMethod remoteMethod = null;

Object[] args;

tag = reader.checkTags((char) ***TagList*** + "" +

(char) ***TagEnd*** + "" +

(char) ***TagCall***);

if (tag == ***TagList***) {

reader.reset();

int count = reader.readInt(***TagOpenbrace***);

if (methods != null) {

remoteMethod = methods.get(aliasname, count);

}

if (remoteMethod == null) {

args = reader.readArray(count);

}

else {

args = new Object[count];

reader.readArray(remoteMethod.paramTypes, args, count);

}

tag = reader.checkTags((char) ***TagTrue*** + "" +

(char) ***TagEnd*** + "" +

(char) ***TagCall***);

if (tag == ***TagTrue***) {

context.setByref(true);

tag = reader.checkTags((char) ***TagEnd*** + "" +

(char) ***TagCall***);

}

}

else {

if (methods != null) {

remoteMethod = methods.get(aliasname, 0);

}

args = new Object[0];

}

if (remoteMethod == null) {

if (methods != null) {

remoteMethod = methods.get("\*", 2);

}

context.setMissingMethod(true);

}

else {

context.setMissingMethod(false);

}

if (remoteMethod == null) {

results.add(hproseService.sendError(new NoSuchMethodError("Can't find this method " + name), context));

}

else {

context.setRemoteMethod(remoteMethod);

results.add(hproseService.data.beforeInvoke(hproseService, name, args, context));

}

} while (tag == ***TagCall***);

return Promise.*reduce*(results.toArray(),

new Reducer<ByteBufferStream, ByteBuffer>() {

public ByteBufferStream call(ByteBufferStream output, ByteBuffer result, int index) throws Throwable {

output.write(result);

return output;

}

},

new ByteBufferStream()).then(new Func<ByteBuffer, ByteBufferStream>() {

public ByteBuffer call(ByteBufferStream data) throws Throwable {

data.write(***TagEnd***);

return data.buffer;

}

}

);

}

Για περαιτέρω βελτίωση του κώδικα, στην ίδια κλάση (HproseService) παρατηρώ ότι οι μέθοδοι getTimeout(), setTimeout(), getHeartbeat(), setHeartbeat(), getPushEvent(), setPushEvent() χειρίζονται δεδομένα της hprose.server.HproseServiveData.java άρα τις μετακινώ εκεί και τις διαγράφω απ΄την ΗproseService. Παρατηρώ όμως ότι αν τις διαγράψω όλες αυτές εκτός της getTimeout() , δεν δημιουργείται κάποιο σφάλμα, γεγονός που μας δείχνει ότι δεν καλούνται πουθενά στο πρόγραμμα. Συνεπώς τις σβήνω απ΄ την HproseServiceData:

public int getTimeout(HproseService hproseService) {

return timeout;

}